2.2 Recursion

Abstraction involves making reusable modules, ones that can be used for solving other problems. The special case of abstraction where the other problem is a version of the original problem is known as recursion. The term is most common in computing, but recursion is broader than just a computational method – as our first example illustrates.

2.2.1 Coin-flip game

The first example is the following game.

> Two people take turns flipping a (fair) coin. Whoever first turns over heads wins. What is the probability that the first player wins?

As a first approach to finding the probability, get a feel for the game by playing it. Here is one iteration of the game resulting from using a real coin:

```
TH
```

The first player missed the chance to win by tossing tails; but the second player won by tossing heads. Playing many such games may suggest a pattern to what happens or suggest how to compute the probability.

However, playing many games by flipping a real coin becomes tedious. A computer can instead simulate the games using pseudorandom numbers as a substitute for a real coin. Here are several runs produced by a computer program – namely, by a Python script `coin-game.py`. Each line begins with 1 or 2 to indicate which player won the game; then the line gives the coin tosses that resulted in the win.

```
2 TH
2 TH
1 H
2 TH
1 TTH
2 TTTH
2 TH
1 H
1 H
1 H
```


From the game's description it doesn't seem like half the time is a plausible answer. Player 1 definitely has a higer chance of winning...

where did this 1.58 come from?

I was a little confused at this being the school method. Took me a while to realize I do actually calculation complex multiplication like this but vertically. Maybe it would be easier to show it the vertical way. Also every multiplication we do as humans is the recursion or multiplication since build on what we know via our memorization of basic multiplication (ie. 3*5=15 so 50*3=150).

I really don't understand how this was derived. Before I checked it, it seemed like you just split up the numbers and adding/multiplied them around several times. Does this work for any complex multiplication or is the a special case? Please review in class.

I guess this is similar to when you multiple vertically because we do cross multiply digits and them sum the results up. It may just seem cooler when its explicitly written out and shown horizontally.

I don't understand the subtraction portion.

I think the examples in the chapter did a good job in explaining what recursion is because initially the definition just sounded like abstraction and divide and conquer.

2.2 Recursion

Abstraction involves making reusable modules, ones that can be used for solving other problems. The special case of abstraction where the other problem is a version of the original problem is known as recursion. The term is most common in computing, but recursion is broader than just a computational method – as our first example illustrates.

2.2.1 Coin-flip game

The first example is the following game.

> Two people take turns flipping a (fair) coin. Whoever first turns over heads wins. What is the probability that the first player wins?

As a first approach to finding the probability, get a feel for the game by playing it. Here is one iteration of the game resulting from using a real coin:

```
TH
```

The first player missed the chance to win by tossing tails; but the second player won by tossing heads. Playing many such games may suggest a pattern to what happens or suggest how to compute the probability.

However, playing many games by flipping a real coin becomes tedious. A computer can instead simulate the games using pseudorandom numbers as a substitute for a real coin. Here are several runs produced by a computer program – namely, by a Python script `coin-game.py`. Each line begins with 1 or 2 to indicate which player won the game; then the line gives the coin tosses that resulted in the win.

```
2 TH
2 TH
1 H
2 TH
1 TTH
2 TTTH
2 TH
1 H
1 H
1 H
```


Read Section 2.2 (memo due Sunday at 10pm).

This is kind of awkward wording. I had to read the sentence a couple of times to figure out what you mean.

yea, when I think about it I usually think of it as a problem inside a problem, like those wooden dolls that open up and another little replica is inside and it keeps going until you get a tiny doll. That example works for me.

It might be useful to include one or two sentences in the beginning explaining how the coin-flip game applies to recursion. While it is reasonably clear by the end of the section, a few sentences might do a lot to orient the reader.

I disagree, I think the way this section is structured is perfect and does a good job of letting the reader discover the idea of recursion on their own without holding their hand and telling them exactly how the example relates in the beginning.

I disagree and believe it is better this way. It makes the reader think a bit more throughout the section and is better for learning.

I can appreciate both sides of this discussion. There's always a tension between presenting results as in a handbook (i.e. for those who already know a result, and discussing results so as to help readers best learn them. Mostly I've chosen the learning side, but I sometimes wish there could be a two-layered book: Once you read it and learn everything in it, it turns into a handbook!

2.2 Recursion

Abstraction involves making reusable modules, ones that can be used for solving other problems. The special case of abstraction where the other problem is a version of the original problem is known as recursion. The term is most common in computing, but recursion is broader than just a computational method – as our first example illustrates.

2.2.1 Coin-flip game

The first example is the following game.

Two people take turns flipping a (fair) coin. Whoever first turns over heads wins. What is the probability that the first player wins?

As a first approach to finding the probability, get a feel for the game by playing it. Here is one iteration of the game resulting from using a real coin:

```
TH
```

The first player missed the chance to win by tossing tails; but the second player won by tossing heads. Playing many such games may suggest a pattern to what happens or suggest how to compute the probability.

However, playing many games by flipping a real coin becomes tedious. A computer can instead simulate the games using pseudorandom numbers as a substitute for a real coin. Here are several runs produced by a computer program – namely, by a Python script `coin-game.py`. Each line begins with 1 or 2 to indicate which player won the game; then the line gives the coin tosses that resulted in the win.

```
2 TH
2 TH
1 H
2 TH
1 TTH
2 TTTH
2 TH
1 H
1 H
1 H
```

It is worthwhile to note that the condition is based on whether the first player wins. Something I took to mean the first "toss", but it actually refers to the *player* who tosses first. So the condition will still be true if the player wins on the third toss (or fifth, etc.).

I don't really understand what you found confusing in this little paragraph. Is your confusion with "first" meaning toss or player actually farther down?

On first read I too thought it meant "what is the liklihood that the first player wins on the first toss (1/2)" rather than the correct reading of "what is the liklihood that the player who goes first wins at all?" I think its just a priming thing.

I agree - I got a little confused about it the first time, interpreting it as "first player winning the first toss" but after reading it again it's fine.

Yeah I think what this means here is that coins are flipped until someone gets heads. That player is the winner. Then what is the probability the first player is that winner?

just as a counter-perspective, i really didn't have any problems understanding this game set up and all and thought it was very clear.

just as a counter-perspective, i really didn't have any problems understanding this game set up and all and thought it was very clear.

just as a counter-perspective, i really didn't have any problems understanding this game set up and all and thought it was very clear.

just as a counter-perspective, i really didn't have any problems understanding this game set up and all and thought it was very clear.

I also thought it was clearly stated.

this was a little confusing to me. then i realized what it represented.

Agreed. Although I think we've all seen this notation before, if you wanted this book to be accessible by non-math oriented people, then you should probably introduce it as Tails, Heads before using TH.

Or, it could be stated that T means tails and H means heads, so that it is stated for the rest of the document.

2.2 Recursion

Abstraction involves making reusable modules, ones that can be used for solving other problems. The special case of abstraction where the other problem is a version of the original problem is known as recursion. The term is most common in computing, but recursion is broader than just a computational method – as our first example illustrates.

2.2.1 Coin-flip game

The first example is the following game.

> Two people take turns flipping a (fair) coin. Whoever first turns over heads wins. What is the probability that the first player wins?

As a first approach to finding the probability, get a feel for the game by playing it. Here is one iteration of the game resulting from using a real coin:

```
TH
```

The first player missed the chance to win by tossing tails; but the second player won by tossing heads. Playing many such games may suggest a pattern to what happens or suggest how to compute the probability.

However, playing many games by flipping a real coin becomes tedious. A computer can instead simulate the games using pseudorandom numbers as a substitute for a real coin. Here are several runs produced by a computer program – namely, by a Python script `coin-game.py`. Each line begins with 1 or 2 to indicate which player won the game; then the line gives the coin tosses that resulted in the win.

```
2 TH
2 TH
1 H
2 TH
1 TTH
2 TTTH
2 TH
1 H
1 H
1 H
```

Does the first player always have the first flip?

I think so in the case of this example.

It seems that the first player is *defined* by the first toss.

I agree it would be a different probability if he didn't.

Exactly – it doesn't matter whether Alice or Bob wins, just whether the first player (ie the one who tosses first) wins.

this would be the perfect place to have an aside as to what a pseudo-random number is...as a side note, I really like that you actually [and correctly] call it a psuedo-random number [as opposed to a random number]

how is pseudorandom different from random?

pseudorandom means it approximates the properties of random numbers

Taken from wikipedia: "A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The sequence is not truly random in that it is completely determined by a relatively small set of initial values, called the PRNG's state."

Basically, the PRNG is given an initial number (or set of numbers), and by performing certain operations on that number it arrives at a sequence of "random numbers." The pseudo part comes because knowledge of the initial numbers determines the rest.

What about just using math to calculate the probability? Computer simulations provide an estimate of the expected probability, but doesn't really add to the logic behind the theory, right?

Yes, especially since the simulation is technically not entirely random. How bout the course 2 students make a coin flipping robot?

There are also many calculator games which do this. I realize its unrelated, but in high school they made us put them on our calculators to do probability simulations

I think the calculator games are also "pseudorandom number generators". The idea is that any computational device, like a computer or calculator, can't truly pick a random number, and can only simulate a random choice. So I think the calculator games you mentioned are the same as the computer program described here.

2.2 Recursion

Abstraction involves making reusable modules, ones that can be used for solving other problems. The special case of abstraction where the other problem is a version of the original problem is known as recursion. The term is most common in computing, but recursion is broader than just a computational method – as our first example illustrates.

2.2.1 Coin-flip game

The first example is the following game.

> Two people take turns flipping a (fair) coin. Whoever first turns over heads wins. What is the probability that the first player wins?

As a first approach to finding the probability, get a feel for the game by playing it. Here is one iteration of the game resulting from using a real coin:

```
TH
```

The first player missed the chance to win by tossing tails; but the second player won by tossing heads. Playing many such games may suggest a pattern to what happens or suggest how to compute the probability.

However, playing many games by flipping a real coin becomes tedious. A computer can instead simulate the games using pseudorandom numbers as a substitute for a real coin. Here are several runs produced by a computer program – namely, by a Python script `coin-game.py`. Each line begins with 1 or 2 to indicate which player won the game; then the line gives the coin tosses that resulted in the win.

```
2 TH
2 TH
1 H
2 TH
1 TTH
2 TTTH
2 TH
1 H
1 H
1 H
```

Is there any way you could include the script in an appendix or something? I would be interested in seeing it.

I would also be interested in seeing the script. Also, unless the script is included, I'm not sure why it is relevant to include the name of the script.

Sure. I've just posted it on the course website (in the "data/scripts" section).

Awesome, which version of Python is this written in? I believe we used 2.4 for 6.00.

is it really necessary to have this detail in the text then?

I think it would be useful to have an appendix devoted to all the code in the book and maybe a brief description, or if there's code that explains a concept well that does not fit in the flow of the book proper. It would help with understanding for the code-inclined.

I agree – a code appendix would be very nice, and could easily be ignored by someone who wasn't interested in it.

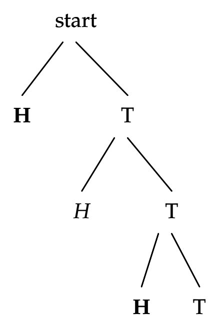
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

shouldn't player 1 have won 50% of the time on the first try? maybe that'll hold true when the simulation is run more times.

what we're calculating isn't who wins the first time, but who ultimately wins. so if the first player didn't win the first time, we go to the second player, and so on and so forth, until one of the two players gets a head and wins the game.

I think you may have been caught in the confusion of the sentence describing the win condition. As the comment before this says, we're calculating how often player 1 is the first to win, not when he wins on the first try.

"Reasonable" conclusion? I don't know that many people would make this assumption.

You might if you knew nothing about the game. If you were only told that there was a game that Player 1 won 5 out of 10 times (without being told how the game worked), you would guess it was fair, or at least close to fair.

Even if I didn't know anything, I'd still be suspicious. 10 throws is certainly better than just 2, but not enough to make any claims of fairness.

Yes, but if someone asked you "which player would you like to be" (again, without explaining the game), you wouldn't have any reason to prefer one or the other, and moreover you wouldn't have any reason to suppose that you /should/ have a preference.

i just don't think it's a big deal. the phrase is fine.

yeah i don't know what you guys are arguing about. the point he's trying to make is obvious.

Hearing that it is fair and looking at the example and who won are totally different. It is obvious player one has an advantage because they have to lose before 2 can win. So how could it be fair?

I think the point of this was just that if you didn't understand the problem the observed data would show you that it was fair. Not that the probability shows that the game is fair.

I agree. If you just think of it as a black box, the output makes it look like a fair game from the outside.

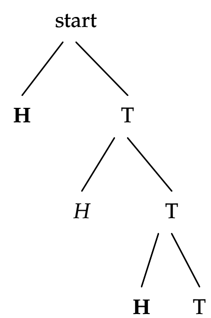
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

I think that a side-bar would be useful here...explain what, exactly, you mean by fair. I mean, I get it, but i'm not sure it's something that would be intuitive to [most] everyone that might read the book

I would think that this might give a slight advantage to the person who goes first

Minor edit, but this line seems a little too wordy on first read through, enough to confuse me (I thought the placement of 'based' was a typo)

A comma after strongly or striking "as it is" might make it clearer.

You are right. Leaving out helpful or even all commas is a confusing habit that I picked up in England (I lived there 30% of my life). [Whoops, I did it again in the preceding sentence.] The habit returns like a retrovirus when I least expect it.

I think the sentence would be clearer if it were slightly reordered to read "the conclusion cannot be believed too strongly, as it is based on only 10 games" (comma debatable).

I would consider jsut moving the word 'based' after 'as it is'. That seems more grammatical in my opinion.

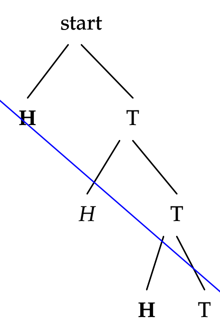
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \quad (2.1)$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and p/2 add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

Is there a way of determining exactly how many games would be necessary? A number that would lead to a 'confidant' result?

I have often wondered about what makes a probability legit. For example, in a lot of papers I read, simulations or experiments will show that "98.9%" of the time something they want happens. However, I was talking to someone who does a lot more probability than I do (in biology), and he said he would only trust something that is 99.9999% or along those lines. Does it just depend on the thing that you are looking at? Such that if you are dealing with living things, that you must have a larger sample to account for variability?

I think this is always a hard question to answer. From what I know about probability and statistics (which isn't much), there are many ways you can go about doing this. t-tests, chi square tests, etc. to guage for how accurate certain values are; and you could always look at standard deviation and z scores as a measure of how far off measurements are from each other–larger sample leads to smaller standard deviation

Well I'm not entirely sure, but I believe you could use the law of large numbers to solve this. The LLN states the lim(n->inf)Prob(|Xn-u|<e)=1 for Xn average of the trials, u expected value of the outcome, and e error. By choosing a small e we can produce a "confident" answer and figure out the number of trials n.

That's a very interesting question. I've been looking for examples for the (only half-written) chapter on "Probabilistic reasoning". After reading your question, I think I'll use this coin-game simulation as one example. We'll figure out how confident one can be about p as a function of the number of games in the simulation.

(For those who already have familiarity with statistical inference: The analysis will be Bayesian.)

central limit theorem says we must approach the true result with more number of trials

I think the theorem says it works fine above n = 30

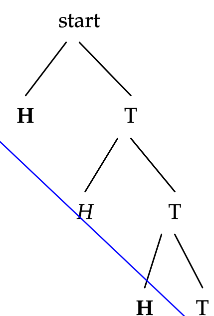
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and p/2 add to unity. Therefore, p = 2/3, as conjectured from the simulation.

Why don't you just build it into the coin.game code?

Abstraction!

Haha, exactly! That's sort of the same idea behind creating functions in programs. When you make a function, you define it and allow it to be called later, which not only saves you time (and redudant typing) but also gives you the ability to access and combine these 'modules'.

It would be significantly faster though...

In case anyone else has to look this up... wc is the command for word count

i don't bother to look them up... i just skim over all the programming and go to the sections where he explains the actual estimation technique.

That's probably where the dividing line is between the course 6 and course 2 majors...

I hope the line gets blurred by the end of the course!

Not extremely important, but "most recent" is somewhat ambiguous here. Should I take it to mean the most recent in terms of when this book was written? Perhaps use "one" instead?

My guess would be that "most recent" refers to one, non-specific pipeline that was run at the time this section was written.

This makes sense as the idea of the inclusion of pseudorandom would cause different experimental results, but the wording is rather awkward. The important part is the 68 to 32 split between the two players and I think this should be the focus.

I think that the wording "most recent" is a little off. I understand that it is meant to place the time of the results, but I'm not sure that's necessary in a book. Maybe list a set of three (or so) results?

Perhaps you could also discuss how large n should be for such results to be significant.

the first player has an inherent advantage, but this doesn't mean it's not fair, right?

"fair" would imply that each player has an equal (1/2) probability of winning each game. If the game is set up such that one player is more likely to win, it doesn't meet the definition of fair.

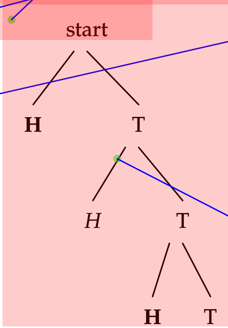
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and p/2 add to unity. Therefore, p = 2/3, as conjectured from the simulation.

Doesn't mean much- In the 10 iterations, it is still more probable it had a ratio closer to 2/3

This kinda makes sense because the first player has to lose for the second player to even have a chance to win. A one time coin toss would be fair but this is a little different.

This tree made it much easier of understanding the premise for the game

Maybe I'm confused now – I thought the premise was you and another person alternate flipping coins until someone gets heads, right?

yes, that is the premise. I think that the tree does nothing for better explaining the game, but it does improve the understanding of the probability calculations

I don't quite understand how this tree is a representation of the game- what happens if the coin has to be flipped three time (tails, tails, head) where does that come in?

So if the boldface represents the first player, I assume the italics represent the second player?

I think that the only italic is the second H down the tree...and i'm pretty sure that's a typo...i do know it should be bold

No, I don't think it's a typo. It should indeed represent the second player winning because the pattern that leads up to it is TH, which means that the first player threw tails, and then the second player threw heads and won.

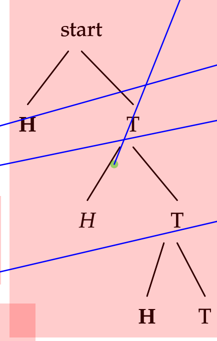
We used this method a lot in high school statistics, but they never introduced the concept of recursion with it.

In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.



To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:

$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

The tree makes it so easy to understand now

I feel this would be the best way to find the anser to the problem

It might be helpful to add a line next to each toss showing which player is tossing. Also, it's a minor point, but the tree doesn't show the TTTTH scenario mentioned in the text.

I agree. The tree makes the answer seem obvious,

i know that this will be better explained later on in this section, but when i first saw this tree, my first reaction was "isn't that from divide and conquer? aren't we supposed to be on abstraction now?" did anyone else do this?

I think maybe the point of using the tree is to show that trees have many useful applications in different areas and in different problem solving methods.... divide and conquer, abstraction, etc.

yeah so if we had this, why did we run a program in the first place?

How did you get these probabilities?

I've never seen this result before and found it pretty interesting and surprising.

I think that this is completely clear, but for someone who doesn't have any probability (given that probability isn't a pre-req for this course), you may want to explain how that sum comes around.

I agree. This conclusion definitely could be explained in greater detail. I'm confused on why the probability doesn't remain 1/2 the whole time or why it doesn't go from 1/2 to 1/4

because the second one P(TTH) = 1/2 * 1/2*1/2 = 1/8, P(TTTTH) = 1/2 ˆ 5 = 1/32. 1/4 (TH) is not what we want, because (TH) means the second person gets a head, but we want to calculate the probability of the 1st person gets the head

I agree... clearly explaining that you are calculating the probability that only the 1st, 3rd, 5th etc (only taking odd terms) would be helpful

or maybe just include exponents. p = 1/2 + 1/2ˆ3 + 1/2ˆ5 ...

is this the sum of 1/2*4ˆn?

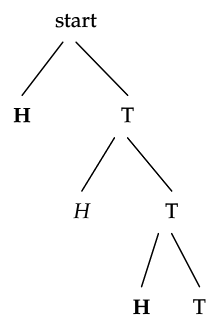
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

I always forget in probability (I never ever have to think about it!) when to add, or multiply, or use exponents. Does anyone have a good way of remembering, so that when I do need to use it once every year or so, I can do it right?

I think if there are n ways to reach the same result, in this case, n ways to win, then you add up the probabilities. however, if a series of things need to happen in order to achieve a certain result, then you multiply, this is just one way to understand it.

In probability, "and" corresponds to multiplication and "or" corresponds to addition of probabilities. in this case, the probability of the first player winning is if the first coin is heads OR if the third coin is heads OR if the fifth coin is heads; thus, the probabilities are added

Thank you!

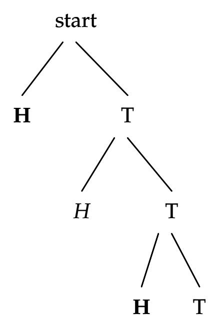
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \qquad (2.1)$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and p/2 add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

What formula is that? I forget...

I agree, the formula should be included.

sum(a*r^n) = a/(1-r) over n=0 to +inf, for |r|<1. Here, r=1/4, and a=1/2. (1/2)/(1-1/4)=2/3

Thank you for the equation! I agree it should probably be included in the reading.

I don't think it has to be. I consider a good textbook one that gives you the basic ideas, and then encourages you to go out and explore things for yourself. Besides, it's pretty fun to figure out the formula yourself if you have forgotten.

oh yeah...fun...if you're a masochist.

Fun exploration is trying out the new knowledge you gained, and playing around with it. Tedious, unnecessary 'exploration' is google searching "geometric series formula" to get an equation that would have taken up less space than the phrase "a familiar formula".

Even if you wanted figure out the sum for yourself, it'd be nice to know if you got it right.

Although I see how it might be nice to see the formula it is somewhat besides the point of this chapter. We are using abstraction in a lot of these examples to prove a point, and going into too much detail in every example just detracts from the actual point.

In the next paragraph, I explain an alternative method that you can use instead. Then you can use the alternative method to derive the geometric-series formula.

when can you use this formula- not for all series? are there a few formulas we should memorize for these types of series?

Equation or problem 2.1? This seems to refer to the equation above, not the problem at the end of the section.

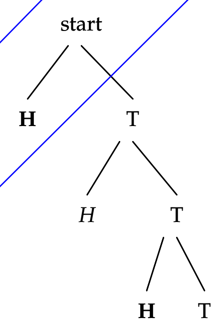
It is indeed Problem 2.1, whose solution gives the familiar formula needed for Equation 2.1 (though until one solves the problem, it's not so evident).

In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.



To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:

$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

so in this instance, the tree method and recursion method are used interchangeably on this problem. i am a more visual person and find the tree more intuitive than the written unix code. is there a rule of thumb for when one is more appropriate than another?

If you consider who wins as part of the tree structure, then it's really sort of 'reversed with respect to player number' one level down... and hence the next result. If it were truly actually repeated one level down, then the game would just be Player 1 flipping a coin until he won, right?

This is clever. I would have easily realized that p1+p2=1 but i would not have so easily realized that p2=1/2*p1. (Where p1=probability of 1 winning and p2=probability of 2 winning)

I agree- this is really awesome. I would never have come up with this.

I agree–this is really clever. The tree diagram helps a lot with visualizing the recursion–every time there's a tails, the same process happens again, and here, for the second player to win, the first player first has to get tails, then the second player has to get heads–1/2 the chance of the first player winning since theres that extra condition of the first player getting a tails

The probability a player wins a toss starts at 1/2. However the second person only gets to go if the first person losses (half the time). Only two people can win so p+p/2 = 1. and now we know p, which is percentage for first player.

This is pretty cool. I wouldn't have guessed this either at first, but after looking at the tree, it makes sense.

I didn't quite get it at first, but the class comments really cleared it up!

Yeah, this is amazingly simple and I didn't get it at first either. Pretty cool.

I am still kinda not completely convinced about all this. It still seems kinda intuitive to me that if this is a recursive example then the probability should be the same every time and thus should add to equal 1. The class comments do help and if I am correct then the reason it is not 1 is because player 1 gets the first flip? I don't really see how that would affect the probability though. (shouldn't it not matter who goes first?)

Would be even more clear if you added these probabilities to the tree and show how it iterates

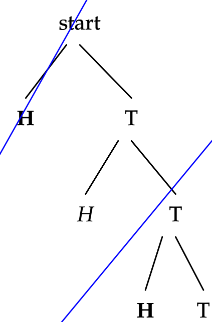
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

This analysis is much easier for me to follow.

I agree, I like this analysis, although it seems a little less intuitive, which makes it a nice addition in the text.

Agreed, this seems very simple and it's very interesting that it really is this intuitive.

I think the effect is even greater given the contrast with the more complicated math of the previous section

I have to agree that it was much easier to follow this when compared to the previous section. It's clear and simple, without much outside knowledge needed.

I also think so, I actually started thinking about it this way first.

Where p is the probability that the first player loses the first round? Because if that's the case how do you get p = 2/3...

I'm not sure I follow what's happening in this paragraph.

Prob that P1 wins = p, Prob that P2 wins = 1/2*p. They add to unity, so p + 1/2*p = 1. This reduces to 3/2*p=1, or p=2/3

i don't understand either. all variables should be really clear, and it's easier to understand math when it's not inside of a paragraph

This seems pretty obvious to me. If the first player always starts, he's going to win more. Am I not getting something?

There is a disconnect for me on this step. I get it from a theoretical standpoint, however, it still seems that the probability should be 50/50.

p = 2/3 is really counter-intuitive. One first pass, I thought it would be 1/2 by symmetry, but I guess since player 1 flips first, it breaks the symmetry argument. I like the p + p/2 = 1 argument a lot though.

I'm still misunderstanding this part.

It might be helpful to just write out explicitly the two equations you use here to solve, as a summary:

p1 + p2 = 1 p2 = (1/2)*p1

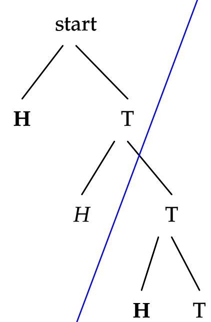
In these ten iterations, each player won five times. A reasonable conclusion, is that the game is fair: Each player has an equal chance to win. However, the conclusion cannot be believed too strongly based as it is on only 10 games.

Let's try 100 games. With only 10 games, one can quickly count the number of wins by each player by scanning the line beginnings. But rather than repeating the process with 100 lines, here is a UNIX pipeline to do the work:

```
coin-game.py | head -100 | grep 1 | wc -l
```

Each run of this pipeline, because `coin-game` uses different pseudorandom numbers each time, produces a different total. The most recent invocation produced 68: In other words, player 1 won 68 times and player 2 won 32 times. The probability of player 1's winning now seems closer to 2/3 than to 1/2.

To find the exact value, first diagram the game as a tree. Each horizontal layer contains H and T, and represents one flip. The game ends at the leaves, when one player has tossed heads. The boldface H's show the leaves where the first player wins, e.g. H, TTH, or TTTTH. The probabilities of each winning way are, respectively, 1/2, 1/8, and 1/32. The infinite sum of these probabilities is the probability p of the first player winning:



$$p = \frac{1}{2} + \frac{1}{8} + \frac{1}{32} + \cdots. \tag{2.1}$$

This series can be summed using a familiar formula.

However, a more enjoyable analysis – which can explain the formula (Problem 2.1) – comes from noticing the presence of recursion: The tree repeats its structure one level down. That is, if the first player tosses tails, which happens with probability 1/2, then the second player starts the game as if he or she were the first player. Therefore, the second player wins the game with probability 1/2 times p (the factor of 1/2 is from the probability that the first player tosses tails). Because one of the two players must win, the two winning probabilities p and $p/2$ add to unity. Therefore, $p = 2/3$, as conjectured from the simulation.

I really like this example. The idea of recursion is a very powerful way to analyze the game's probability. However I think the analysis could be improved by expanding the last paragraph, and possibly including a few more drawings.

Problem 2.1 **Summing a series using abstraction**
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \quad (2.2)$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \quad (2.3)$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \quad (2.4)$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \quad (2.5)$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \quad (2.6)$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \times 2|7 = 3 \times 2 \;\big|\; 3 \times 7 + 5 \times 2 \;\big|\; 5 \times 7. \quad (2.7)$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

This seems a bit out of place...and I'm not sure I know enough to do this!

How does this relate to the examples above? And I think you need to add the condition that abs(r) < 1 for this question to have any kind of meaning.

On the previous page (line 2.1) we compute the probability of the first player winning by adding his chance of winning throughout the inifinite tree. However, there are an infinite number of terms to add, but since the terms are powers of 1/4, (p = (1/2)(1+1/4+1/4^2+...), the formula simply helps you get the answer to the sum of the infinite terms.

Yeah I agree, it should be noted that the condition of abs(r) < 1.

Let Z = 1 + r + r^2 + r^3 + ...

Z = 1 + r(1 + r + r^2 + ...) Z = 1 + r(Z) Z - rZ = 1 Z(1-r) = 1 Z = 1 / (1-r)

Wow, that was helpful! And the simplification from step 1 to 2 is a great example of abstraction. Maybe this could be included as an example before we are prompted with a problem to solve on our own?

You do know enough, but it requires thinking about it in an interesting way (and it's a problem on HW 2).

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \;\times\; 2|7 = 3 \times 2 \;\Big|\; 3 \times 7 + 5 \times 2 \;\Big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

wouldn't this just be infinite if |r|>=1?

yeah.. it's conventional to state a |r|<1 bound, although the answer could include cases, |r|<1 and |r|>=1.

Maybe it should be stated in the problem prompt. Although I hadn't actually considered the above case, I think it would be nice to have the specifcs.i

Maybe it should be stated in the problem prompt. Although I hadn't actually considered the above case, I think it would be nice to have the specifcs.

I think the purpose of this question is not to be mathematically rigorous here, but rather to find interesting, pictorial or abstractional ways to find the sum of this series

but the abstraction falls apart for |r|>=1, and bounds are usually VERY important. If you perform the abstraction, get an equation, and forget to determine and include the bounds, then can run into big trouble. For r=2, in this example, your equation would yield Sum = -1. In this case a sense check can quickly show that something is wrong, but not every situation is so clear.

> On the other hand, there are whole areas of mathematics and physics where one just goes ahead and sums the series, worrying about the rigor later (if ever). Quantum electrodynamics is a good example.
>
> Given the abundance of rigor in most education, it's worthwhile to suspend, for a while, the quest for it.

I agree about focusing on other ways to solve it as opposed to mathematical rigor, however, since recursion was just introduced with only one example, i feel that this problem is a little advance for people to begin drawing on recursion principles. At first look I didn't even really understand how recursion applied until i saw it explained in the comments.

I disagree – I think that not stating that sort of condition in the prompt is appropriate in this circumstance. It's important for us to be able to distinguish between the cases in which the series might converge or not, and a real application might not give us all the restrictions we would like.

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \ \times \ 2|7 = 3 \times 2 \ \Big| \ 3 \times 7 + 5 \times 2 \ \Big| \ 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

How is this useful for us if we're learning to approximate in order to not use a machine?

Think of the goal here not only as learning the approximate (I admit, the course title is misleading), but as learning to use the reasoning tools, like abstraction, across many, many fields. So, here you will see how abstraction leads to recursion which leads to a clever multiplication algorithm.

i'm not sure i understand how this is abstraction though

It's abstraction in that it takes a higher level routine and calls itself to accomplish a smaller task.

What is the standard school method?

But we are humans. So how does this method help us humans approximate? Wouldn't we just use the 1 few or 10 method and do it in a few seconds instead?

The one/few/10 method is good for approximations. I think this is for when we need a more exact answer.

Or for when you want to teach a computer how to do large calculations. (I really need to change the title of the course so that I can indicate that the course is not just approximation...)

Just curious- Pi is an irrational number...so what are computers calculating mathematically when they calculate pi to billions of digits?

"Although practically a physicist needs only 39 digits of Pi to make a circle the size of the observable universe accurate to one atom of hydrogen, the number itself as a mathematical curiosity has created many challenges in different fields."

http://en.wikipedia.org/wiki/Pi#Computation_in_the_computer_age

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \times 2|7 = 3 \times 2 \;\big|\; 3 \times 7 + 5 \times 2 \;\big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

I have no idea what this is. Maybe replace it with a more common knowledge example?

I agree that anyone outside of course 6 wouldn't have any idea what public key cryptography is. But for anyone still confused, wikipedia has a nice explanation: http://en.wikipedia.org/wik key_cryptography

I think public-key cryptography is one of the most important concepts that has emerged in recent decades. Even in its simplest form, it should be taught in any class (not just in course 6) for the simplicity and logic of the idea.

I think this is a good example in an electronic format where you can google something you don't know. Maybe it's not a common example, but he's trying to accommodate multiple audiences as can be seen by the title of the class.

Is this sort of like divide and conquer, but then apply the same operation to each smaller part(recursive step)?

I don't understand this way of computing, it seems somewhat difficult and extensive. Also, how is this useful for us as approximaters?

I've never heard of this

Agreed. Is it just how people are taught to multiply by hand?

I too have never been taught the "school method" shown here, but if the example is only meant to illustrate how multiplication can be tedious without abstraction then it still seems to serve its purpose even if you haven't seen that particular method.

I don't think I was taught how to multiply with this. It was more of a "this is what these numbers can breakdown into."

I'm also confused what is meant by the "school method". The only method I was ever taught in school was where you lined the numbers up vertically and multiplied each digit through, etc.

I think the idea, although it is not exactly like how we solved problems in school, is that we are isolating the various parts of the problem. Aside from the tricks of grouping these numbers, we are still multiplying the same way as always.

Is the school method something you are making up right now, or is it an establishes technique? I have never herad of it before

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

This is a very awkward way of writing this

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \; \times \; 2|7 = 3 \times 2 \;\Big|\; 3 \times 7 + 5 \times 2 \;\Big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \;\times\; 2|7 = 3 \times 2 \;\big|\; 3 \times 7 + 5 \times 2 \;\big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

Where does this "school method" come from? I was taught to do multiplication by hand this way. 35 x27 —- 245 70 945 (7*5)=35, write down the 5, carry the 3 over, 7*3+3=24, write that down. Then repeat for 2. 2*5=10, write 0, carry 1 over, 2*3+1=7, write that down. Then add the two numbers.

I would definitely say that the above would be reflective of my schooling, however I find myself using this other method (outlined in the text) when I need ot multiply larger numbers, although I wouldn't say I was ever formally taught it. And I never took the time to write it all out. Very interesting where you take this.

I agree that this is how I was taught to do multiplication, but most of us wouldn't do it this way anymore - it would probably be 27*30 + 27*10/2 since that's faster and more intuitive for us.

I agree with this method, it's much faster and how I would approach it, although I did (35x30) - (35x3) which came out to a clean 945.

I agree that this is a bit more intuitive however, i think it takes more effort and concentration to do all of this in ones head and remember the numbers too. Since many human beings are lazy it seems that there is a bit of a trade off of effort vs. time (since you can probably do it in your head faster than on paper).

If you look at what you've written out, I think that's the answer to your question. That is school method except it is written in column form instead of side-ways in regular equation form. The general procedure is to take the units, tens, hundreds (27 = 7+20) and multiply by the first number. When you take each digit and multiply by the first number, you do it by breaking down the first number into units, tens, hundreds (35=30+5). In the end, you multiple out everything keeping track of all powers of ten.

That being said, it might be easier to tie introduce this section using the column multiplication that we all learned.

If I am doing the multiplication on paper, I use the column method as explained above. But if for whatever reason I need to do it in my head, I''ll use the method as explained in the text. (assuming the problem is 2*2 or maybe 3*2 max... after that I can't keep the numbers in my memory easily and will resort to paper or calculator)

Remember from algebra: (3x+5)(2x+7)=... He's applying what your teacher's made you drill in abstract

this is a clever insight, it makes it much easier to understand what he is doing in Eg. 2.4. Old FOIL method.

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \;\times\; 2|7 = 3 \times 2 \;\Big|\; 3 \times 7 + 5 \times 2 \;\Big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

school method? did anyone do this in school?

Yes, I did this in school.

I didn't do this in school, but I can see how this method would be taught. I was a bit confused by the name "school method" as well.

it is a horizontal representation of the vertical multiplication that i learned

I didn't realize that. Maybe it would be easier to recognize if it was presented in the vertical form somehow.

I assume the recursion comes in for each n power of 10?

y are we regrouping by the powers of 10? is this in reference to befoe when we estimated large multiplications by separating the power of 10 from the from number?

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \times 2|7 = 3 \times 2 \;\Big|\; 3 \times 7 + 5 \times 2 \;\Big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

This is a really confusing way to write the math out.

I agree... how is this more useful than standard two-number multiplication where you start with the units digit, and carry the factors of 10, etc.?

Yeah, I would think that the school-method would just be the multiplication method just mentioned? I understand that this method represents recursion but it's also making this multiplication so much more difficult than it is. I though recursion was supposed to simplify our calculations?

I don't think so, I mean it's pretty clear what's happening here in terms of grouping factors and dealing with distribution. I think the target audience of this book is college age or older students / teachers, who shouldn't have that much trouble figuring out the math. Also, the point is kind of to show how cumbersome and totally unnecessary the 'school method' is compared to smart recursion.

I don't think so, I mean it's pretty clear what's happening here in terms of grouping factors and dealing with distribution. I think the target audience of this book is college age or older students / teachers, who shouldn't have that much trouble figuring out the math. Also, the point is kind of to show how cumbersome and totally unnecessary the 'school method' is compared to smart recursion.

I don't think so, I mean it's pretty clear what's happening here in terms of grouping factors and dealing with distribution. I think the target audience of this book is college age or older students / teachers, who shouldn't have that much trouble figuring out the math. Also, the point is kind of to show how cumbersome and totally unnecessary the 'school method' is compared to smart recursion.

I don't think so, I mean it's pretty clear what's happening here in terms of grouping factors and dealing with distribution. I think the target audience of this book is college age or older students / teachers, who shouldn't have that much trouble figuring out the math. Also, the point is kind of to show how cumbersome and totally unnecessary the 'school method' is compared to smart recursion.

I don't think so, I mean it's pretty clear what's happening here in terms of grouping factors and dealing with distribution. I think the target audience of this book is college age or older students / teachers, who shouldn't have that much trouble figuring out the math. Also, the point is kind of to show how cumbersome and totally unnecessary the 'school method' is compared to smart recursion.

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x + y$ and $z|y|x$ represent $100z + 10y + x$. Then the school method runs as follows:

$$3|5 \;\times\; 2|7 = 3 \times 2 \;\big|\; 3 \times 7 + 5 \times 2 \;\big|\; 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

While compared to the school method this takes less paper space and the math isnt too hard to figure out, it still comes across as an application of recursion with hard to see benefits. Adding in the following section was very useful though.

could you at least add a few more spaces between the terms being added? i felt myself goign back several times and trying to find the term. it made reading it take way longer than necessary.

Or, as suggested in another comment, I will add parentheses to make the grouping clear.

This totally glitched out and over-posted a bunch of times. Weird.

Maybe add a feature to delete double posts?

is that a reliable definition of an abstraction?

I think so... its reusable and keeps only important details.

This is similar to what he was talking about in class with the tree programming language

Actually, the programming the tree thing makes me wonder – is object oriented coding all about abstraction? You abstract things into objects, classes, etc...?

I don't think that this is meant to be a definition of abstraction, it is only pointing out that commonly convenient notation's are also abstractions.

A notation is not the only kind of abstraction, but it's one of the most useful and easy to recognize (and common).

Think of musical notation, e.g. for piano or guitar music. It is such a good notation that we hardly even realize that it is a notation.

i dont understand this type of notation

10y+x, I think?

agreed!

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \tag{2.2}$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \tag{2.3}$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \tag{2.4}$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \tag{2.5}$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \tag{2.6}$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x+y$ and $z|y|x$ represent $100z+10y+x$. Then the school method runs as follows:

$$3|5 \times 2|7 = 3 \times 2 \mid 3 \times 7 + 5 \times 2 \mid 5 \times 7. \tag{2.7}$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

I believe this should be:

$y|x$ represents 10y+x.

As it is different from the next example and doesn't make sense in equation 2.7

I agree I think this might have been a typo, otherwise 3|5 would yield 53 and not 35

Perhaps a better way to present this abstraction would be to present the calculation graphically, similar to how the alternative "school method" mentioned in the comments above is organized? (I mean, by lining up all the terms that are x1, and all the x10 terms, and all the x100 terms, etc, then simply adding them rather than multiplying them?) It would avoid the cluttering of the | syntax here.

how would this work for fractions?

For decimal expansions you could maybe do: 3|1|.4|1|5 for 31.415? I'm not sure why it would need to work for fractions..

i am honestly only glossing over this page and all the arithmetic manipulations and labeling it as an inefficient method. im not sure if that is the point of this, but this is how it is coming across to me.

This is rather confusing when it's embedded in the text. If it's going to be so crucial for the next part, try separating it out of the text.

How do you know where the |'s are used to separate the numbers you multiply?

How did this come from that description? I would have expected (10*3+5)*(2*10+7) and so on

Problem 2.1 Summing a series using abstraction
Use abstraction to find the sum of the infinite series

$$1 + r + r^2 + r^3 + \cdots. \qquad (2.2)$$

2.2.2 Computational Recursion

The second example of recursion is an algorithm to multiply many-digit numbers much more rapidly than is possible with the standard school method. The school method is sufficient for humans, for we rarely multiply large numbers by hand. However, computers are often called upon to multiply gigantic numbers, whether in computing π to billions of digits or in public-key cryptography. I'll introduce the new method by contrasting it with the school method on the example of 35×27.

In the school method, the product is written as

$$35 \times 27 = (3 \times 10 + 5) \times (2 \times 10 + 7). \qquad (2.3)$$

The product expands into four terms:

$$(3 \times 10) \times (2 \times 10) + (3 \times 10) \times 7 + 5 \times (2 \times 10) + 5 \times 7. \qquad (2.4)$$

Regrouping the terms by the powers of 10 gives

$$3 \times 2 \times 100 + (3 \times 7 + 5 \times 2) \times 10 + 5 \times 7. \qquad (2.5)$$

Then you remember the four one-digit multiplications 3×2, 3×7, 5×2, and 5×7, finding that

$$35 \times 27 = 6 \times 100 + 31 \times 10 + 35 = 945. \qquad (2.6)$$

Unfortunately, the preceding description is cluttered with powers of 10 obscuring the underlying pattern. Therefore, define a convenient notation (an abstraction!): Let $y|x$ represent $10x + y$ and $z|y|x$ represent $100z + 10y + x$. Then the school method runs as follows:

$$3|5 \times 2|7 = 3 \times 2 \;|\; 3 \times 7 + 5 \times 2 \;|\; 5 \times 7. \qquad (2.7)$$

This notation shows how school multiplication replaces a two-digit multiplication with four one-digit multiplications. It would recursively replace

It took me a long time to figure out what this was saying. As written, I read (3x2)x10 + 3x7 + (5x2)x10 + 5x7. I think it'd be much more clear if you used parentheses, i.e. (3x2)|(3x7+5x2)|(5x7). The root of this problem is you never specified where the | feel with regards to order of operations. Also, y|x is meant in discrete mathematics (at least as taught in 6.042) that x divides y (y=ax), so this might be confusing.

I agree. I think a few sentences explaining the exact method of getting to that expression would be helpful. Especially during its first appearance in this chapter. For example the expression in the middle bracket wasn't clearly apparent.

Agreed, I had to read over this section several times in order to understand what was meant by |. Even just a quick explanation of this would and how you came up with the notation would be very helpful.

Need to see this in class. I don't get it.

Need to see this in class. I don't get it.

I'm not sure if I get it, which is also why I would like to see it in class.

I had never thought of multiplication in this manner; this is really useful, and I think after some practice with it, it will be a powerful tool.

I agree - parentheses would've helped emphasize that you are doing (x)|(y)|(z)

I think I'm still a little confused about what is going on here... I can't really see how this method is supposed easier or better than other methods.

It also took me a little while to see what this was saying. But something I noticed is that the expansion of the multiplication is kind of like the FOIL method taught for expanding multiplication of factors.

why is this helpful? it seems like the same thing as above

I need some more time to learn to do this.

a four-digit multiplication with four two-digit multiplications. For example, using a modified | notation where $y|x$ means $100y + x$, the product 3247×1798 becomes

$$32|47 \times 17|98 = 32 \times 17 \mid 32 \times 98 + 47 \times 17 \mid 47 \times 98. \quad (2.8)$$

Each two-digit multiplication (of which there are four) would in turn become four one-digit multiplications. For example (and using the normal $y|x = 10y + x$ notation),

$$3|2 \times 1|7 = 3 \times 2 \mid 3 \times 7 + 2 \times 1 \mid 2 \times 7. \quad (2.9)$$

Thus, a four-digit multiplication becomes 16 one-digit multiplications.

Continuing the pattern, an eight-digit multiplication becomes four four-digit multiplications or, in the end, 64 one-digit multiplications. In general, an n-digit multiplication requires n^2 one-digit multiplications. This recursive algorithm seems so natural, perhaps because we learned it so long ago, that improvements are hard to imagine.

Surprisingly, a slight change in the method significantly improves it. The key is to retain the core idea of recursion but to improve the method of decomposition. Here is the improvement:

$$a_1|a_0 \times b_1|b_0 = a_1b_1 \mid (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \mid a_0b_0.$$

Before analyzing the improvement, let's check that it is not nonsense by retrying the 35×27 example.

$$3|5 \times 2|7 = 3 \times 2 \mid (3+5)(2+7) - 3 \times 2 - 5 \times 7 \mid 5 \times 7.$$

Doing the five one-digit multiplications gives

$$3|5 \times 2|7 = 6 \,|\, 31 \,|\, 35 = 6 \times 100 + 31 \times 10 + 35 = 945, \quad (2.10)$$

just as it should.

At first glance, the method seems like a retrograde step because it requires five multiplications whereas the school method requires only four. However, the magic of the new method is that two multiplications are redundant: a_1b_1 and a_0b_0 are each computed twice. Therefore, the new method requires only three multiplications. The small change from four to three multiplications, when used recursively, makes the new method significantly faster: An n-digit multiplication requires roughly $n^{1.58}$ one-digit

Comments on page 4

I might recommend using a different notation here such that

y||x means 100y + x

Agreed. It would make things more clear so that instead of trying to figure out whether | means 10y+x or 100y+x, we can just focus on the more relevant parts of the problem.

Yeah this is a little confusing.

first thing that entered my mind I was thinking "given" like y given x, since we were just talking about probabilities

I agree. My first reaction was that this must have been a typo, and your change to the notation would make it much clearer that it's intentional.

Looks very cluttered and hard to read

As soon as you understand the method it's really not that cluttered, I don't think there's a better way to illustrate the multiplication.

I feel like this makes the problem even more complicated

I think this looks fine. It flows well. Maybe you could make the bars bolder.

here it is corrected

You could possibly use arrows to make things clearer? (how the multiplying is done etc.

I think that's a typo. Should be 3 x 1|3 x 7...

a four-digit multiplication with four two-digit multiplications. For example, using a modified | notation where $y|x$ means $100y + x$, the product 3247×1798 becomes

$$32|47 \times 17|98 = 32 \times 17 \mid 32 \times 98 + 47 \times 17 \mid 47 \times 98. \quad (2.8)$$

Each two-digit multiplication (of which there are four) would in turn become four one-digit multiplications. For example (and using the normal $y|x = 10y + x$ notation),

$$3|2 \times 1|7 = 3 \times 2 \mid 3 \times 7 + 2 \times 1 \mid 2 \times 7. \quad (2.9)$$

Thus, a four-digit multiplication becomes 16 one-digit multiplications.

Continuing the pattern, an eight-digit multiplication becomes four four-digit multiplications or, in the end, 64 one-digit multiplications. In general, an n-digit multiplication requires n^2 one-digit multiplications. This recursive algorithm seems so natural, perhaps because we learned it so long ago, that improvements are hard to imagine.

Surprisingly, a slight change in the method significantly improves it. The key is to retain the core idea of recursion but to improve the method of decomposition. Here is the improvement:

$$a_1|a_0 \times b_1|b_0 = a_1b_1 \mid (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \mid a_0b_0.$$

Before analyzing the improvement, let's check that it is not nonsense by retrying the 35×27 example.

$$3|5 \times 2|7 = 3 \times 2 \mid (3+5)(2+7) - 3 \times 2 - 5 \times 7 \mid 5 \times 7.$$

Doing the five one-digit multiplications gives

$$3|5 \times 2|7 = 6 \,|\, 31 \,|\, 35 = 6 \times 100 + 31 \times 10 + 35 = 945, \quad (2.10)$$

just as it should.

At first glance, the method seems like a retrograde step because it requires five multiplications whereas the school method requires only four. However, the magic of the new method is that two multiplications are redundant: a_1b_1 and a_0b_0 are each computed twice. Therefore, the new method requires only three multiplications. The small change from four to three multiplications, when used recursively, makes the new method significantly faster: An n-digit multiplication requires roughly $n^{1.58}$ one-digit

Does this method make multiplication easy to do in your head? It seems like 16 1-digit multiplications would be easy to lose track of, since there's a limit to how many digits people can keep in their temporary memory at a time.

I agree that breaking the problem into so many pieces may make it hard to keep track of the entire problem. Though I haven't quite wrapped my head around this method yet, would it be possible to work through the problem as you're breaking it down. So instead of keeping all 16 multiplications in your head, you work through a part at a time (keeping track of what you still have to break down before the next step)?

I would personally break this down further and do it in parts. I agree that it does seem a bit difficult to keep track of everything.

I would personally break this down further and do it in parts. I agree that it does seem a bit difficult to keep track of everything.

I think the point is to create problems which we can solve in our head more easily rather then make it easier to keep track of. Clearly, breaking the numbers down to smaller numbers makes more solutions to keep track of but it's a good balance that we're after.

I found this very very interesting. One of the things I have most enjoyed reading about thus far.

I feel the same way. Wouldn't it be easier to just use divide and conquer?

Pretty neat. But doesn't this make room for a lot of mistakes, going against our idea of intelligent redundancy?

is this faster than the school method?

some sort of comparison?

i never learned multiplication looking like this... doesn't seem as natural, but it definitely makes sense

a four-digit multiplication with four two-digit multiplications. For example, using a modified | notation where $y|x$ means $100y + x$, the product 3247×1798 becomes

$$32|47 \times 17|98 = 32 \times 17 \mid 32 \times 98 + 47 \times 17 \mid 47 \times 98. \tag{2.8}$$

Each two-digit multiplication (of which there are four) would in turn become four one-digit multiplications. For example (and using the normal $y|x = 10y + x$ notation),

$$3|2 \times 1|7 = 3 \times 2 \mid 3 \times 7 + 2 \times 1 \mid 2 \times 7. \tag{2.9}$$

Thus, a four-digit multiplication becomes 16 one-digit multiplications.

Continuing the pattern, an eight-digit multiplication becomes four four-digit multiplications or, in the end, 64 one-digit multiplications. In general, an n-digit multiplication requires n^2 one-digit multiplications. This recursive algorithm seems so natural, perhaps because we learned it so long ago, that improvements are hard to imagine.

Surprisingly, a slight change in the method significantly improves it. The key is to retain the core idea of recursion but to improve the method of decomposition. Here is the improvement:

$$a_1|a_0 \times b_1|b_0 = a_1b_1 \mid (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \mid a_0b_0.$$

Before analyzing the improvement, let's check that it is not nonsense by retrying the 35×27 example.

$$3|5 \times 2|7 = 3 \times 2 \mid (3+5)(2+7) - 3 \times 2 - 5 \times 7 \mid 5 \times 7.$$

Doing the five one-digit multiplications gives

$$3|5 \times 2|7 = 6\,|\,31\,|\,35 = 6 \times 100 + 31 \times 10 + 35 = 945, \tag{2.10}$$

just as it should.

At first glance, the method seems like a retrograde step because it requires five multiplications whereas the school method requires only four. However, the magic of the new method is that two multiplications are redundant: a_1b_1 and a_0b_0 are each computed twice. Therefore, the new method requires only three multiplications. The small change from four to three multiplications, when used recursively, makes the new method significantly faster: An n-digit multiplication requires roughly $n^{1.58}$ one-digit

I don't really see this algorithm as "natural" though, it is more complicated to me than just multiplying via one number on top of the other and carrying the 10s, etc.

> yeah I agree, although this method makes sense, I was definitely not taught to do multiplication this way at school so it doesn't feel "natural" as stated in the paragraph.
>
> Well, I think it feels unnatural because it sounds like most of us never actually learned multiplication this way (this is the first time I've seen this)... so it feels like there is some sort of gap here based on the way people were taught multiplication. I can see where this is going in terms of recursion, but it certainly isn't the way I'd multiply by hand.
>
> Maybe if we were taught multiplication by this recursion method, it would seem more natural. If nothing else, introducing the concept of recursion earlier in schooling would allow people to understand this concept better

How does this improve the method?

visual diagrams are actually a lot more helpful for me rather than writing out the recursion steps

Why would you break it down into this step? It seems more complicated?

I think these examples are interesting and do illustrate the recursion and abstraction principles, but I am still having a hard time relating them to something a person would do rather than a computer. Maybe the section should include an example that is practical for problem solving by the human brain.

> i agree...it's sorta interesting, but i don't think i would ever do this
>
> > I can see how 2.9 is very useful but the improvement seems a lot more complicated to remember for a four or six digit multiplication.

How many orders of computation are being saved?

This is a really cool method.

> When doing quick approximations, could we speed up the process by adding numbers to make "few"?

i like the logic used here. not sure if it is necessary though, as it is assumed that since it is in the book the new method should work by most students.

a four-digit multiplication with four two-digit multiplications. For example, using a modified | notation where $y|x$ means $100y + x$, the product 3247×1798 becomes

$$32|47 \times 17|98 = 32 \times 17 \mid 32 \times 98 + 47 \times 17 \mid 47 \times 98. \tag{2.8}$$

Each two-digit multiplication (of which there are four) would in turn become four one-digit multiplications. For example (and using the normal $y|x = 10y + x$ notation),

$$3|2 \times 1|7 = 3 \times 2 \mid 3 \times 7 + 2 \times 1 \mid 2 \times 7. \tag{2.9}$$

Thus, a four-digit multiplication becomes 16 one-digit multiplications.

Continuing the pattern, an eight-digit multiplication becomes four four-digit multiplications or, in the end, 64 one-digit multiplications. In general, an n-digit multiplication requires n^2 one-digit multiplications. This recursive algorithm seems so natural, perhaps because we learned it so long ago, that improvements are hard to imagine.

Surprisingly, a slight change in the method significantly improves it. The key is to retain the core idea of recursion but to improve the method of decomposition. Here is the improvement:

$$a_1|a_0 \times b_1|b_0 = a_1b_1 \mid (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \mid a_0b_0.$$

Before analyzing the improvement, let's check that it is not nonsense by retrying the 35×27 example.

$$3|5 \times 2|7 = 3 \times 2 \mid (3 + 5)(2 + 7) - 3 \times 2 - 5 \times 7 \mid 5 \times 7.$$

Doing the five one-digit multiplications gives

$$3|5 \times 2|7 = 6|31|35 = 6 \times 100 + 31 \times 10 + 35 = 945, \tag{2.10}$$

just as it should.

At first glance, the method seems like a retrograde step because it requires five multiplications whereas the school method requires only four. However, the magic of the new method is that two multiplications are redundant: a_1b_1 and a_0b_0 are each computed twice. Therefore, the new method requires only three multiplications. The small change from four to three multiplications, when used recursively, makes the new method significantly faster: An n-digit multiplication requires roughly $n^{1.58}$ one-digit

Wow this is crazy how it works, but I still don't totally understand exactly how it works

I'm also a little lost. Is there some way to convey this graphically? Which numbers are being multiplied where and why.

I feel like this relates a lot to the chess game. Yes this is great for a computer because they dont have trouble remembering but for humans it doesnt seem practical. I need more convincing.

It is definitely not practical for a human! The method is used here to illustrate how abstraction, of which recursion is a special case, leads to understanding a very sly algorithm. The goal of the course is to understand lots of natural systems (e.g. blue skies) and to learn tools that help in designing and building person-made systems (e.g. bridges, large software systems).

Wow. That is really impressive. I never would've though of that

Woah, this is also amazing. These alternative methods of using abstraction are very interesting.

I was wondering where we were saving time.

I feel like this is true for programming. The computer will notice that there were a few unnecessary calculations. But for humans to realize that would take more time. Recursion to me doesn't seem like a practical method for humans.

This is awesome. Why don't they teach this to us at school?

Ok this makes way more sense as to why it is "clever redundancy!"

I agree, I wish I had learned this a long time ago. I would assume that it is even more practical for humans because of the need to eliminate calculations while working with mental math.

a four-digit multiplication with four two-digit multiplications. For example, using a modified | notation where $y|x$ means $100y + x$, the product 3247×1798 becomes

$$32|47 \times 17|98 = 32 \times 17 \mid 32 \times 98 + 47 \times 17 \mid 47 \times 98. \quad (2.8)$$

Each two-digit multiplication (of which there are four) would in turn become four one-digit multiplications. For example (and using the normal $y|x = 10y + x$ notation),

$$3|2 \times 1|7 = 3 \times 2 \mid 3 \times 7 + 2 \times 1 \mid 2 \times 7. \quad (2.9)$$

Thus, a four-digit multiplication becomes 16 one-digit multiplications.

Continuing the pattern, an eight-digit multiplication becomes four four-digit multiplications or, in the end, 64 one-digit multiplications. In general, an n-digit multiplication requires n^2 one-digit multiplications. This recursive algorithm seems so natural, perhaps because we learned it so long ago, that improvements are hard to imagine.

Surprisingly, a slight change in the method significantly improves it. The key is to retain the core idea of recursion but to improve the method of decomposition. Here is the improvement:

$$a_1|a_0 \times b_1|b_0 = a_1b_1 \mid (a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0 \mid a_0b_0.$$

Before analyzing the improvement, let's check that it is not nonsense by retrying the 35×27 example.

$$3|5 \times 2|7 = 3 \times 2 \mid (3 + 5)(2 + 7) - 3 \times 2 - 5 \times 7 \mid 5 \times 7.$$

Doing the five one-digit multiplications gives

$$3|5 \times 2|7 = 6 \,|\, 31 \,|\, 35 = 6 \times 100 + 31 \times 10 + 35 = 945, \quad (2.10)$$

just as it should.

At first glance, the method seems like a retrograde step because it requires five multiplications whereas the school method requires only four. However, the magic of the new method is that two multiplications are redundant: a_1b_1 and a_0b_0 are each computed twice. Therefore, the new method requires only three multiplications. The small change from four to three multiplications, when used recursively, makes the new method significantly faster: An n-digit multiplication requires roughly $n^{1.58}$ one-digit

I don't understand how this change would really make a computer faster. The way I see it, a computer would have a database with all one digit multiplications and their results. My sense of the reading is that the time-saving comes from the fact that while there are the same number of 1x1 multiplications, some of them are the duplicates. I don't understand how it would take less time for it to "compute" this number with the duplicates than the case without them, wouldn't it still need to replace the same number of 1x1 calculations with answers?

This is actually really clever. At first it seems very complicated, but if you can follow their recursion, it actually makes it a lot faster

Yeah, once I understood that a1b1 and a0b0 are the same, it made sense.

where does the 1.58 come from? is that the log2(3) as mentioned below?

Why is it log2(3) though?

multiplications (Problem 2.2). In contrast, the school algorithm requires n^2 one-digit multiplications. The small decrease in the exponent from 2 to 1.58 has a large effect when n is large. For example, when multiplying billion-digit numbers, the ratio of n^2 to $n^{\log_2 3}$ is roughly 5000.

Why would anyone multiply billion-digit numbers? One answer is to compute π to a billion digits. Computing π to a huge number of digits, and comparing the result with the calculations of other supercomputers, is the standard way to verify the numerical hardware in a new supercomputer.

The new algorithm is known as the Karatsuba algorithm after its inventor [15]. But even it is too slow for gigantic numbers. For large enough n, an algorithm using fast Fourier transforms is even faster than the Karatsuba algorithm. The so-called Schönhage–Strassen algorithm [27] requires a time proportional to $n \log n \log \log n$. High-quality libraries for large-number multiplication recursively use a combination of regular multiplication, Karatsuba, and Schönhage–Strassen, selecting the algorithm according to the number of digits.

Problem 2.2 Running time of the Karatsuba algorithm
Show that the Karatsuba multiplication method requires $n^{\log_2 3} \approx n^{1.58}$ one-digit multiplications.

2.3 Low-pass filters

2.3.1 RC circuits

2.3.2 Light-bulb flicker

2.3.3 Temperature fluctuations

2.4 Summary and further problems

The diagram for the hiker has two names: a phase-space diagram or a spacetime diagram. Both types are useful in science and engineering. Spacetime diagrams, used in Einstein's theory of relativity, are the subject of the wonderful textbook [30]. They are the essential ingredient in

Very interesting.. Who originally thought of this? Why do we learn the other way?

Yea school way is much less efficient than computational recursion

but it's less complicated to learn! we don't really learn the school method in the same way that it's explained here...we do it vertically. This physical setup is much easier to remember [and learn than] an equation

is there an example of it being used for higher order multiplication?

is there an example of it being used for higher order multiplication?

How does multiplying billion digits allow one to compute pi? How do people calculate pi anyways?

http://en.wikipedia.org/wiki/Numerical_approximations_of_%CF%80

I'm really glad you asked that because that was a really interesting article... I think the question you asked has a million different answers and has been an important questoin for a long time

I like that you explain a concept then tell us why this is important at all.

In the smaller form, this feels useful for humans to possibly use as a way to do 2 digit (maybe 4 digit) multiplication. Even though I think it is not useful for humans (we have calculators) past that, it's nice to now know how this type of thing works. Very informational; I think this section was helpful.

very interesting fact, i never knew why people wanted to compute pi to a more digits than 10

multiplications (Problem 2.2). In contrast, the school algorithm requires n^2 one-digit multiplications. The small decrease in the exponent from 2 to 1.58 has a large effect when n is large. For example, when multiplying billion-digit numbers, the ratio of n^2 to $n^{\log_2 3}$ is roughly 5000.

Why would anyone multiply billion-digit numbers? One answer is to compute π to a billion digits. Computing π to a huge number of digits, and comparing the result with the calculations of other supercomputers, is the standard way to verify the numerical hardware in a new supercomputer.

The new algorithm is known as the Karatsuba algorithm after its inventor [15]. But even it is too slow for gigantic numbers. For large enough n, an algorithm using fast Fourier transforms is even faster than the Karatsuba algorithm. The so-called Schönhage–Strassen algorithm [27] requires a time proportional to $n \log n \log\log n$. High-quality libraries for large-number multiplication recursively use a combination of regular multiplication, Karatsuba, and Schönhage–Strassen, selecting the algorithm according to the number of digits.

Problem 2.2 Running time of the Karatsuba algorithm
Show that the Karatsuba multiplication method requires $n^{\log_2 3} \approx n^{1.58}$ one-digit multiplications.

2.3 Low-pass filters

2.3.1 RC circuits

2.3.2 Light-bulb flicker

2.3.3 Temperature fluctuations

2.4 Summary and further problems

The diagram for the hiker has two names: a phase-space diagram or a spacetime diagram. Both types are useful in science and engineering. Spacetime diagrams, used in Einstein's theory of relativity, are the subject of the wonderful textbook [30]. They are the essential ingredient in

Random, but interesting. I like these little sidenotes, they help me remember what I read.

That's true, it is useful how the random application examples help differentiate topics.

I agree I would encourage including these random interesting facts because the random/interesting nature makes them easy to remember and thus it is easier to remember the material associated with them.

I'm going to have to fourth this. It's a lot of these side notes, ans short purposes for the readings that keeps me engaged and saying "I wonder what this is used for? Ohhh!"

Yeah, these are really cool to read over.

That answers my question..

It's not that clear that the Karatsuba algorithm refers to what we just did above. I had to Wikipedia that...

I don't agree, I think it is clear that the "new algorithm" refers to the algorithm that was just explained in the previous paragraphs.

I'm kind of confused as to how this applies to recursion. Another example after explaining it would be helpful.

I somewhat agree... I thought it was a really interesting side note as someone who is pretty technically competent in these types of things. I could see readers getting pretty lost though.

I was definitely confused a little bit here but I do see how it relates to abstraction/recusrion

multiplications (Problem 2.2). In contrast, the school algorithm requires n^2 one-digit multiplications. The small decrease in the exponent from 2 to 1.58 has a large effect when n is large. For example, when multiplying billion-digit numbers, the ratio of n^2 to $n^{\log_2 3}$ is roughly 5000.

Why would anyone multiply billion-digit numbers? One answer is to compute π to a billion digits. Computing π to a huge number of digits, and comparing the result with the calculations of other supercomputers, is the standard way to verify the numerical hardware in a new supercomputer.

The new algorithm is known as the Karatsuba algorithm after its inventor [15]. But even it is too slow for gigantic numbers. For large enough n, an algorithm using fast Fourier transforms is even faster than the Karatsuba algorithm. The so-called Schönhage–Strassen algorithm [27] requires a time proportional to $n \log n \log \log n$. High-quality libraries for large-number multiplication recursively use a combination of regular multiplication, Karatsuba, and Schönhage–Strassen, selecting the algorithm according to the number of digits.

Problem 2.2 Running time of the Karatsuba algorithm
Show that the Karatsuba multiplication method requires $n^{\log_2 3} \approx n^{1.58}$ one-digit multiplications.

2.3 Low-pass filters

2.3.1 RC circuits

2.3.2 Light-bulb flicker

2.3.3 Temperature fluctuations

2.4 Summary and further problems

The diagram for the hiker has two names: a phase-space diagram or a spacetime diagram. Both types are useful in science and engineering. Spacetime diagrams, used in Einstein's theory of relativity, are the subject of the wonderful textbook [30]. They are the essential ingredient in

Are we going to actually see this later? Otherwise it seems slightly unnecessary.

Agreed - I don't think algorithm names should be thrown around unless they'll come back later.

I think a one liner name doesn't hurt to be put in here, it gives you a bit of insight into this world of algorithms an you can research it if you want on your own.

but it takes away from the message here and distracts the reader (clearly)

I think it's a nice addition to reference additional algorithms at the end of a section. It gives more background on the topic and if someone is interested in learning about more complex/efficient algorithms they can go look them up or at least know about their existence.

Could someone please explain the background on algorithm speeds and the meaning of "n logn loglogn"?

yea i'd like one too...unless it's ridiculously long and complicated, then i guess i'd rather know it's not worth my time

It would be a little difficult to explain under these circumstances, but take a look at the Wikipedia article under "big O Notation" hopefully it would be a bit cleaer from there.

This is unclear where the parentheses are.

n (log n)(log (log n))

The multiple terms probably derive from multiple loops within the algorithm and/or program.

How?? if this SS algorithm is so fast, why isn't the only one used, and maybe some info on how SS works?

multiplications (Problem 2.2). In contrast, the school algorithm requires n^2 one-digit multiplications. The small decrease in the exponent from 2 to 1.58 has a large effect when n is large. For example, when multiplying billion-digit numbers, the ratio of n^2 to $n^{\log_2 3}$ is roughly 5000.

Why would anyone multiply billion-digit numbers? One answer is to compute π to a billion digits. Computing π to a huge number of digits, and comparing the result with the calculations of other supercomputers, is the standard way to verify the numerical hardware in a new supercomputer.

The new algorithm is known as the Karatsuba algorithm after its inventor [15]. But even it is too slow for gigantic numbers. For large enough n, an algorithm using fast Fourier transforms is even faster than the Karatsuba algorithm. The so-called Schönhage–Strassen algorithm [27] requires a time proportional to $n \log n \log \log n$. High-quality libraries for large-number multiplication recursively use a combination of regular multiplication, Karatsuba, and Schönhage–Strassen, selecting the algorithm according to the number of digits.

Problem 2.2 Running time of the Karatsuba algorithm
Show that the Karatsuba multiplication method requires $n^{\log_2 3} \approx n^{1.58}$ one-digit multiplications.

2.3 Low-pass filters

2.3.1 RC circuits

2.3.2 Light-bulb flicker

2.3.3 Temperature fluctuations

2.4 Summary and further problems

The diagram for the hiker has two names: a phase-space diagram or a spacetime diagram. Both types are useful in science and engineering. Spacetime diagrams, used in Einstein's theory of relativity, are the subject of the wonderful textbook [30]. They are the essential ingredient in

Is there a reference to where we can read about this in more detail? it sounds very interesting.

one of my favorite things about this class is that in addition to just "approximation" we get to learn all sorts of interesting things... such as why lectures are the way they are, how CDs work, and how pi can be computed faster :)

I find it very interesting that various programs will look at a problem and decide the best way of computing the answer based on the number of digits. I didn't realize programs are so dynamic.

The GNU Multiple Precision library (http://gmplib.org/) is a very high-quality library that does that. The page also has a link to a program (using the library) that computes pi to one billion digits.

How do I know that it chooses the algorithm by the size of the numbers? I think I remember reading about that in the documentation, but I also experimented with it. Python has an interface (a "binding") to the library, and plotted the running times versus number size, and you can see the breakpoints in the graph as the algorithm changes.

Maybe we can go over Strassen's Algorithm for matrix multiplication? It would be nice to find an abstraction there.

can you please go over this method in class? I am having a really hard time reading about it and understanding what is going on

While it should be clear what n is based on the above discussion, it would still be nice to explicitly say that we are calculating O(n) for a n-digit times a n-digit number.

agreed. does this have to do with abstraction directly as well? or does it have to do with the algorithm that used abstraction a while ago?

I think the examples given above could be stronger by adding a closing paragraph about how it directly relates to abstraction. I feel like the term abstraction was somewhat avoided in this section.

Overall an informative and interesting chapter. It is generally well written (though adding parentheses to the | notation would improve it a lot).

multiplications (Problem 2.2). In contrast, the school algorithm requires n^2 one-digit multiplications. The small decrease in the exponent from 2 to 1.58 has a large effect when n is large. For example, when multiplying billion-digit numbers, the ratio of n^2 to $n^{\log_2 3}$ is roughly 5000.

Why would anyone multiply billion-digit numbers? One answer is to compute π to a billion digits. Computing π to a huge number of digits, and comparing the result with the calculations of other supercomputers, is the standard way to verify the numerical hardware in a new supercomputer.

The new algorithm is known as the Karatsuba algorithm after its inventor [15]. But even it is too slow for gigantic numbers. For large enough n, an algorithm using fast Fourier transforms is even faster than the Karatsuba algorithm. The so-called Schönhage–Strassen algorithm [27] requires a time proportional to $n \log n \log \log n$. High-quality libraries for large-number multiplication recursively use a combination of regular multiplication, Karatsuba, and Schönhage–Strassen, selecting the algorithm according to the number of digits.

Problem 2.2 Running time of the Karatsuba algorithm
Show that the Karatsuba multiplication method requires $n^{\log_2 3} \approx n^{1.58}$ one-digit multiplications.

2.3 Low-pass filters

2.3.1 RC circuits

2.3.2 Light-bulb flicker

2.3.3 Temperature fluctuations

2.4 Summary and further problems

The diagram for the hiker has two names: a phase-space diagram or a spacetime diagram. Both types are useful in science and engineering. Spacetime diagrams, used in Einstein's theory of relativity, are the subject of the wonderful textbook [30]. They are the essential ingredient in

Is section 2.3 not written yet? lol. I guess this would emphasize to everyone that this book is a process in the making and we should try to set aside any frustrations by focusing more on constructive comments.

I was tempted to put a page break in before Section 2.3, to avoid showing my hand. But I resisted. Indeed, in some parts of the course, I have a larger margin of safety than in others.

This unit on abstraction is the one where I have the least margin of safety. I've been thinking about it and trying different versions for a few years, and this year has been the most coherent so far (but I leave to you to decide whether, on an absolute scale, it is actually coherent).

what happened to these guys?

lol don't complain

The posted assignment was to read section 2.2 only so he'll probably add these in before Tuesday

I never thought of light-bulb flicker as a low pass filter. Live and learn, I guess