Problem Set 1 Solution

All parts are due Tuesday, September 27 at 11:59PM.

Name: Jisoo Min

Collaborators:

Part A

Problem 1-1. [18 points] Asymptotic behavior of functions

(a) [6 points] Solution: The correct order of these functions is \( f_5, f_2, f_3, f_1, f_4 \).

By the properties of logarithms, we get \( f_5 = 4 \log^2 n \).

We know that exponential functions grow asymptotically faster than polynomials and polynomials grow asymptotically faster than logarithmic functions as \( n \to \infty \).

Comparing asymptotic growth rates, we get:

\[
\begin{align*}
f_5 &= O(\log^2 n) = O((n^{0.5})^2) = O(f_2) \quad \because \forall \epsilon > 0, \log n = O(n^\epsilon) \\
f_2 &= O(n) = O(n \log n) = O(f_3) \\
f_3 &= O(n \log n) = O(n^2) = O(f_1) \\
f_1 &= O(n^2) = O(2^n) = O(f_4)
\end{align*}
\]

(b) [6 points] Solution: The correct order of these functions is \( f_1, f_6, f_3, f_5, f_2, f_4 \).

By the properties of logarithms, functions can be written as:

\[
\begin{align*}
f_1 &= 3 \log \log n, \\ f_6 &= \log^3 \log n, \\ f_3 &= n^{\log 3} \approx n^{0.4771}, \\ f_5 &= \log 3 + n^3 \log n, \\ f_2 &= (\log n)^{3 \log 3 + 3 \log n}, \\ f_4 &= n^{n^{\log 3}} \approx n^{n^{0.4771}}
\end{align*}
\]

We know that exponential functions grow asymptotically faster than polynomials and polynomials grow asymptotically faster than logarithmic functions as \( n \to \infty \).

Comparing asymptotic growth rates and computing limits, we get:

\[
\begin{align*}
f_1 &= O(\log \log n) = O(\log^3 \log n) = O(f_6) \\
f_6 &= O(\log^3 \log n) = O(\log^2 n) = O(n^{\log 3}) = O(f_3) \quad \because \forall \epsilon > 0, \log n = O(n^\epsilon) \\
f_3 &= O(n) = O(n^{\log 3} \log n) = O(f_5) \\
f_5 &= O(f_2) \quad \because \lim_{n \to \infty} \frac{f_2(n)}{f_5(n)} = \lim_{n \to \infty} \frac{(\log n)^{3 \log 3 + 3 \log n}}{\log 3 + n^3 \log n} = \infty \\
f_2 &= O(\log n) = O(n^{1\log n}) = O(n^{n^{\log 3}}) = O(f_4) \quad \because \forall \epsilon > 0, \log n = O(n^\epsilon)
\end{align*}
\]
(c) [6 points] Solution: The correct order of these functions is \( f_5, f_1, f_2, f_4, f_3 \).

By the properties of exponents, functions can be written as:

\[
\begin{align*}
 f_5 &= 2^{5n}, \\
 f_1 &= 2^{6n}, \\
 f_2 &= 2^{n^4}, \\
 f_4 &= 3^{3n}, \\
 f_3 &= 8^{3n}
\end{align*}
\]

We know that exponential functions grow asymptotically faster than polynomials and polynomials grow asymptotically faster than logarithmic functions as \( n \to \infty \).

Comparing asymptotic growth rates, we get:

\[
\begin{align*}
 f_5 &= O(2^{5n}) = O(2^{6n}) = O(f_1) \\
 f_1 &= O(2^{6n}) = O(2^{n^4}) = O(f_2) \\
 f_2 &= O(2^{n^4}) = O(3^{3n}) = O(f_4) \\
 f_4 &= O(3^{3n}) = O(8^{3n}) = O(f_3)
\end{align*}
\]

**Problem 1-2. [18 points] Recurrences**

(a) [12 points] **Solving recurrences:**

1. \( T(n) = 8T\left(\frac{n}{2}\right) + n^2 \)

   **Solution:** \( T(n) = \Theta(n^2) \)

   We will use the master theorem. We will determine which case of the master theorem applies and write down the answer.

   For this recurrence, we have \( a = 8, b = 3, f(n) = n^2 \), and \( n^\log_b a = n^\log_3 8 = O(n^{1.8928}) \). Since \( f(n) = \Omega(n^{\log_3 8 + \epsilon}) \), where \( \epsilon \approx 0.1 \), case 3 applies if we can show that the regularity condition holds for \( f(n) \). For sufficiently large \( n \),

   \[
   af\left(\frac{n}{3}\right) = 8 f\left(\frac{n}{3}\right) = \frac{8n^2}{9} \leq \frac{9}{10} n^2 = cf(n) \quad \text{for} \quad c = 9/10.
   \]

   Consequently, by case 3, the solution to the recurrence is \( T(n) = \Theta(n^2) \).

2. \( T(n) = 10T\left(\frac{n}{3}\right) + n^2 \)

   **Solution:** \( T(n) = \Theta(n^{\log_3 10}) \)

   We will use the master theorem. We will determine which case of the master theorem applies and write down the answer.

   For this recurrence, we have \( a = 10, b = 3, f(n) = n^2 \), and \( n^\log_b a = n^\log_3 10 = O(n^{2.0960}) \). Since \( f(n) = O(n^{\log_3 10 - \epsilon}) \), where \( \epsilon \approx 0.01 \), we can apply case 1 of the master theorem and conclude that the solution is \( T(n) = \Theta(n^{\log_3 10}) \).

3. \( T(n) = 2T\left(\frac{n}{2}\right) + n \)

   **Solution:** \( T(n) = \Theta(n \lg n) \)

   - The Master Theorem
     We will use the master theorem. We will determine which case of the master theorem applies and write down the answer.

     For this recurrence, we have \( a = 2, b = 2, f(n) = n \), and \( n^\log_b a = n \).

     Case 2 applies, since \( f(n) = \Theta(n^{\log_b a}) = \Theta(n) \), and thus the solution to the recurrence is \( T(n) = \Theta(n \lg n) \).
The Recursion Tree
Cost at each of $\log n$ levels is $n$. \( \therefore T(n) = \Theta(n \log n) \)

\[ T(n) = T\left(\frac{n}{2}\right) + O(n) \]

**Solution:** \( T(n) = \Theta(n) \)

For some constant $c$,

\[
T(n) = T\left(\frac{n}{2}\right) + c \cdot n \\
= T\left(\frac{n}{4}\right) + c \cdot \frac{n}{2} + c \cdot n \\
= T\left(\frac{n}{8}\right) + c \cdot \frac{n}{4} + c \cdot \frac{n}{2} + c \cdot n \\
= \ldots \\
= T\left(\frac{n}{2^\log n}\right) + c \cdot n \cdot \sum_{k=1}^{\log n} \frac{1}{2^k} \\
= T(1) + c \cdot n \cdot (2 - \frac{2}{n}) \\
= \Theta(n)
\]
(b) [6 points] Setting up recurrences:

1. **Solution:** \( T(n) = T(n - 1) + c \) \((c \text{ is a constant})\)

\[
T(n) = T(n - 1) + c \\
= T(n - 2) + c + c \\
= T(n - 3) + c + c + c \\
= \ldots \\
= T(1) + c + \ldots + c \\
= T(1) + c \cdot (n - 1) \\
= \Theta(n)
\]

2. **Solution:** \( T(n) = T(n - 1) + T(n - 2) + c \) \((c \text{ is a constant})\)

**Problem 1-3.** [24 points] Balances and Extremes

(a) [14 points] Peak squares in unbalanced arrays

1. [4 points] **Solution:** There are four cases to this problem. We can find a smaller size unbalanced sub-array of \( A \) in all four cases.

   i) \( A[0][k] \) and \( A[n - 1][k] \) are of different colors.  
      \( A[0][0] \) and \( A[n - 1][0] \) are of the same color.  
      \(\Rightarrow\) A 2d-array with \( A[0][k], A[n - 1][k], A[0][0], \) and \( A[n - 1][0] \) cells as corners is unbalanced.

   ii) \( A[0][k] \) and \( A[n - 1][k] \) are of different colors.  
      \( A[0][n - 1] \) and \( A[n - 1][n - 1] \) are of the same color.  
      \(\Rightarrow\) A 2d-array with \( A[0][k], A[n - 1][k], A[0][n - 1], \) and \( A[n - 1][n - 1] \) cells as corners is unbalanced.

   iii) \( A[0][k] \) and \( A[n - 1][k] \) are of the same color.  
      \( A[0][0] \) and \( A[n - 1][0] \) are of different colors.  
      \(\Rightarrow\) A 2d-array with \( A[0][k], A[n - 1][k], A[0][0], \) and \( A[n - 1][0] \) cells as corners is unbalanced.

   iv) \( A[0][k] \) and \( A[n - 1][k] \) are of the same color.  
      \( A[0][n - 1] \) and \( A[n - 1][n - 1] \) are of different colors.  
      \(\Rightarrow\) A 2d-array with \( A[0][k], A[n - 1][k], A[0][n - 1], \) and \( A[n - 1][n - 1] \) cells as corners is unbalanced.
2. [5 points] **Solution:**

```python
def find_peak(i_min, i_max, j_min, j_max):
    # corner value ranges
    while ((i_max - i_min) != 1 or (j_max - j_min) != 1):
        if (i_max - i_min) != 1:
            k = int((i_max + i_min) / 2)
            check for new unbalanced array # see part 1
        if the new unbalanced array is on the left:
            i_max = k
        elif the new unbalanced array is on the right:
            i_min = k
        if (j_max - j_min) != 1:
            k = int((j_max + j_min) / 2)
            check for new unbalanced array # see part 1
        if the new unbalanced array is on the top:
            j_max = k
        elif the new unbalanced array is on the bottom:
            j_min = k
    return updated (i_min, i_max, j_min, j_max) # peak square
```

3. [5 points] **Solution:**

\[
T((n, n)) = T\left((n/2, n/2)\right) + c \quad (c \text{ is a constant})
\]

\[
T((n, n)) = T((n/2, n/2)) + c
= T((n/4, n/4)) + c + c
= T((n/8, n/8)) + c + c + c
= \ldots
= T\left((n/2^{\log_2 n}, n/2^{\log_2 n})\right) + c \cdot \log_2 n
= T(1) + \Theta(\log n)
= \Theta(\log n)
\]

(b) [10 points] **Peak in circular arrays**

1. [4 points] **Solution:** Consider the maximum value among the four cells. If the cell is a peak, return it. If not, then the sub-array adjacent to the maximum value is guaranteed to have a peak. Say A[0] is the maximum among the four given cells. If A[0] is not a peak, then A[1] > A[0]. If A[1] is not a peak, then A[2] > A[1]. If we continue the process and check values along the circular array, then we may arrive at A[k-1], in the worst case. Then A[k-1] must be a peak. A[k-1] is greater than A[k] (A[0] ≥ A[k]) and all values A[0], A[1], A[2], ..., A[k-2]. Therefore, A[k-2] ≤ A[k-1] ≥ A[k]. So in the case when A[0] was the maximum, a peak is guaranteed among cells between A[0] and A[k]. Similarly, any cell among the given four as a maximum guarantees a sub-array with a peak.
2. [6 points] Solution:

def find_peak(A):
    # circular array A is of size n
    n = len(A)
    max_cell = max([A[0], A[⌊n/2⌋], A[⌊n/2⌋+1], A[n-1]])
    if max_cell is a peak:
        return max_cell
    else:
        if max_cell == A[0] or max_cell == A[⌊n/2⌋]:
            return find_peak(A[:⌊n/2⌋+1])
        else:
            return find_peak(A[⌊n/2⌋+1:])

Part B

Problem 1-4. [40 points] 6006LE

(a) Submitted on alg.csail.mit.edu
(b) Submitted on alg.csail.mit.edu
(c) Submitted on alg.csail.mit.edu
(d) [2 points] Runtime analysis:

Solution: Part (b) \(\Theta(nm + n + k)\), Part (c) \(\Theta(nm + nq + n + q + k)\)

- Part (b):
  \(\Theta(nm)\) creating dictionaries of tf, df, tf_idf values and calculating distances
  \(\Theta(n)\) sorting a list of length n in the final step
  \(\Theta(k)\) slicing the sorted list in the final step
  \(\Rightarrow \Theta(nm) + \Theta(n) + \Theta(k) = \Theta(nm + n + k)\)

- Part (c):
  \(\Theta(nm)\) creating dictionaries of tf, df, tf_idf
  \(\Theta(q)\) getting the frequency dictionary of the query
  \(\Theta(nq)\) calculating the relevancy between the query and each of the documents
  \(\Theta(n)\) sorting a list of length n in the final step
  \(\Theta(k)\) slicing the sorted list in the final step
  \(\Rightarrow \Theta(nm) + \Theta(nq) + \Theta(n) + \Theta(q) + \Theta(k) = \Theta(nm + nq + n + q + k)\)
(e) [8 points] Conclusion:

1. Solution:

Example of when get_relevant_articles_doc_dist is more intuitive:
Sample Database = {'science_doc': ['many', 'many', 'many', 'scientists', 'believe', 'water', 'is', 'present', 'in', 'mars'], 'food_doc': ['we', 'eat', 'carbohydrates', 'scientists', 'recommend', 'healthy', 'food', 'and', 'water'], 'book_doc': ['many', 'people', 'read', 'fiction'], 'question_doc': ['is', 'it', 'true', 'that', 'many', 'scientists', 'believe', 'water', 'is', 'present', 'in', 'mars']}

- get_relevant_articles_doc_dist('question_doc', 1) returns 'science_doc' as the best match because many terms in the document match with those in 'science_doc' with a high frequency.
- get_relevant_articles_tf_idf('question_doc', 1) returns 'food_doc' as the best match because the function believes 'many' is less important (∵ df('many') is large) while the word 'many' is actually important to us in this case.
⇒ get_relevant_articles_doc_dist did better.

Example of when get_relevant_articles_tf_idf is more intuitive:
Sample Database = {'science_doc': ['many', 'many', 'many', 'scientists', 'believe', 'water', 'is', 'present', 'in', 'mars'], 'food_doc': ['we', 'eat', 'carbohydrates', 'scientists', 'recommend', 'healthy', 'food', 'and', 'water'], 'book_doc': ['many', 'people', 'read', 'fiction'], 'question_doc': ['what', 'do', 'many', 'people', 'like', 'to', 'eat', 'i', 'want', 'to', 'eat', 'i', 'like', 'to', 'eat']}

- get_relevant_articles_doc_dist('question_doc', 1) returns 'science_doc' as the best match because $tf('many', science_doc)$ is large.
- get_relevant_articles_tf_idf('question_doc', 1) returns 'food_doc' as the best match because many is considered less important (∵ $df('many')$ is large, $df('eat')$ is small). In fact, we also thought 'eat' was important.
⇒ get_relevant_articles_tf_idf did better.

Reason why one method returns a more intuitive result over another:
- The get_relevant_articles_doc_dist function produces a more intuitive result when more percentage of the words in the document are significant keywords. Moreover, large $df(t)$ does not scale down the relevancy scores.
- On the other hand, the get_relevant_articles_tf_idf function produces more intuitive results when many of the words in the document are insignificant (i.e., most of the words in it are common words). In this case, we can see less of articles with high relevancy.
**Modification to TF-IDF calculation:**

For `get_relevant_articles_tf_idf` function, $df(t)$ scales down the relevancy scores. In some cases, however, we do not want this to happen. If $df(t)$ is exceptionally high, then the term is more likely to be insignificant. But if $df(t)$ is moderately high, the term may be an importance source of a valuable relevancy. We can modify the TF-IDF calculation so that it only uses $df(t)$ to scale down the relevancy score when $df(t)$ is above a certain cutoff value.

2. **Solution:**

- Same term frequency of $tf(t, d_i) = 5$ and $tf(t, d_j) = 5$ can have very different significance if $len(d_i) = 10$ and $len(d_i) = 10000$. We want to make sure that the term frequency relative to the document size is properly reflected in TF-IDF calculation.

- Define $tfidf(t, d_i) = rtf(t, d_i) \cdot idf(t)$, where $rtf(t, d_i)$ is the relative term frequency of a term $t$ in $d_i$. $rtf(t, d_i)$ is defined as $rtf(t, d_i) = tf(t, d_i) / len(d_i)$

3. **Solution:**

`get_relevant_articles_doc_dist` returns:

- Apple Inc (score 0.860271)
- Apple (score 1.113709)
- Macintosh (score 1.256221)

`get_relevant_articles_tf_idf` returns:

- Apple Inc (score 451.990982)
- Macintosh (score 181.212974)
- Steve Jobs (score 142.679181)

I think `get_relevant_articles_doc_dist` performs better. It returned the desired article “Apple” as the second best match. But `get_relevant_articles_tf_idf` only returned documents related to ‘Apple Inc.’, not the fruit, ‘apple’. To address this problem of `get_relevant_articles_tf_idf`, we should give more weight to ‘apple’ over ‘Apple’ (case sensitive).

For instance, we can create two separate combinations of tf, df, tf_idf dictionaries with keys of casesensitive words and non-case-sensitive words.

(i.e., Define $Tf(t, d_i)$, $Df(t)$, and $Tf_idf(t, d_i)$ in addition to regular functions which ignore cases entirely.)

Say $t$ = ‘Apple’ and $t$.lower() = ‘apple’.

Define $tf_idf(t, d_i) = m \cdot Tf(t, d_i) \cdot idf(t) + n \cdot tf(t$.lower(), $d_i) \cdot idf(t$.lower())$, where $m < n$ for some postive $m$ and $n$. 
