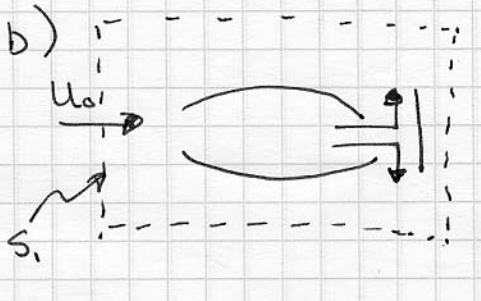
a) $T = \dot{m}(u_e - u_o) = 150(200-60) = \boxed{21\ kN}$

b)



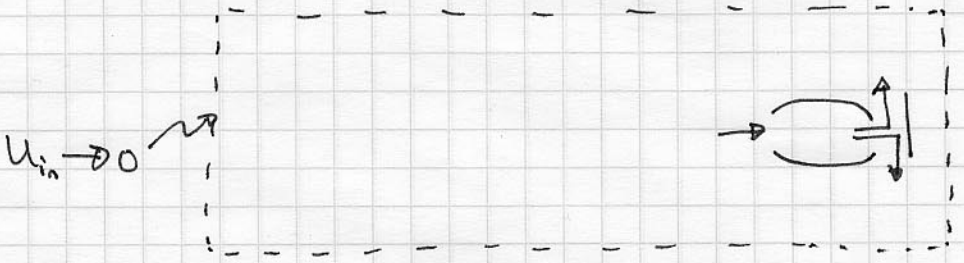
ONLY FLUX OF x-MOM IS ACROSS SURFACE S_1

$T = \dot{m}(0 - u_o)$

$= 150(-60) = \boxed{9\ kN}$

c) WHEN AIRPLANE COMES TO REST, THERE IS NO NET FORCE IN THE X-DIR. YOU CAN SEE THIS BY DRAWING THE CONTROL VOLUME LARGE RELATIVE TO THE ENGINE

$\boxed{T = 0\ kN}$

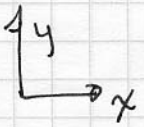


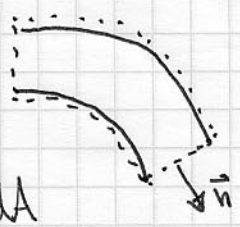
d) INTEGRAL MOMENTUM THEOREM IN FRAME ATTACHED TO VEHICLE:

$$\sum F_y - F_{y_o} = \int_V \frac{d}{dt}(\rho u_y)\, dV + \int_{S'} \rho u_y (\vec{u}\cdot\vec{n})\, dA$$

$F_{y_o} = 0$ assume no accel. of coord. frame

$\int_V \frac{d}{dt}(\rho u_y)\, dV = 0$ assume steady





$$\sum F_y = \int_{inlet} \rho u_y (\vec{u}\cdot\vec{n})\, dA + \int_{exit} \rho u_y (\vec{u}\cdot\vec{n})\, dA$$

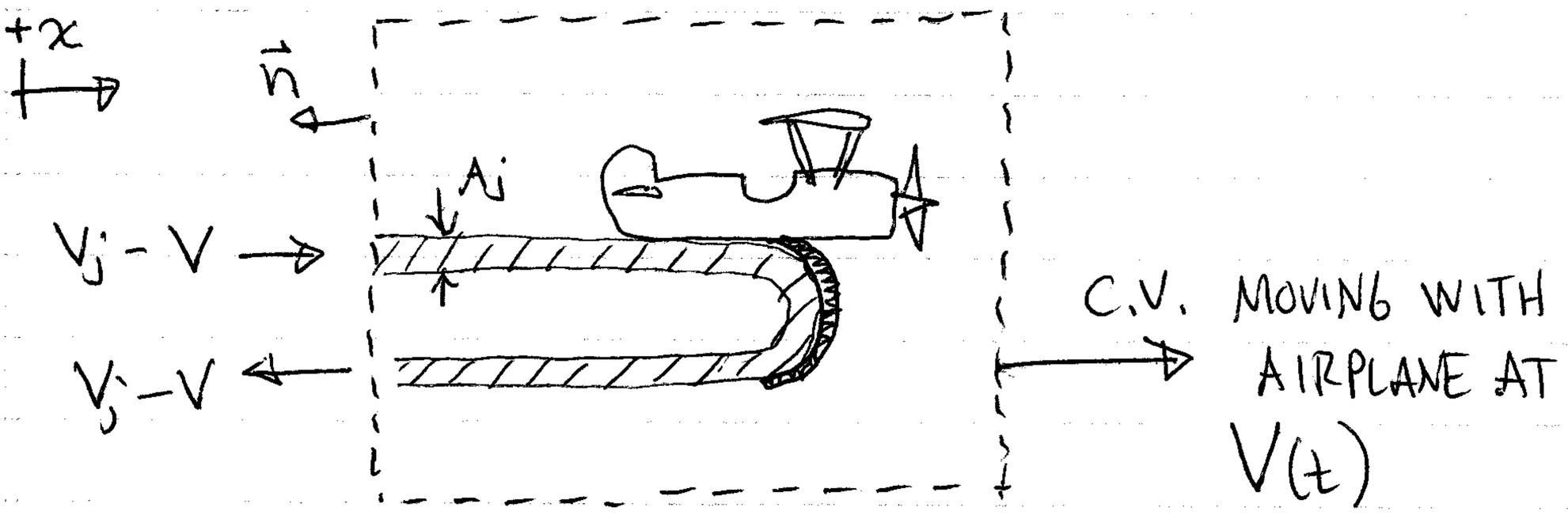
$$= \rho(-U_e \sin\alpha)\cdot U_e \cdot A_e$$

$$\boxed{F_y = -\rho A_e U_e^2 \sin\alpha}$$

UNIFIED PROPULSION PZ SOLUTIONS (WAITZ)

USE INTEGRAL MOMENTUM THEOREM IN VEHICLE REFERENCE FRAME.
VELOCITY OF REFERENCE FRAME = V (VELOCITY OF VEHICLE).
ALL OTHER VELOCITIES ARE RELATIVE TO V AND ARE LABELED $\tilde{u}$.

a) $$\underbrace{\sum F_x}_{\substack{\text{SUM OF EXTERNAL FORCES} = 0 \text{ SINCE PROBLEM SAYS TO NEGLECT DRAG}}} = \underbrace{\frac{dV}{dt}\int_{Vol}\rho\, dV_{ol}}_{\substack{\text{ACCELERATION RELATIVE TO INERTIAL REF. FRAME} \\ = M\frac{dV}{dt}}} + \underbrace{\int_{Vol}\frac{d(\rho u_x)}{dt}\, dV_{ol}}_{\substack{\text{CHANGE IN MOM. OF MASS WITHIN C.V. RELATIVE TO C.V. REF. FRAME} \\ = 0 \text{ SINCE MASS OF VEHICLE NOT CHANGING AND CAN NEGLECT ANY UNSTEADY CHANGES IN MOMENTUM OF WATER PER PROBLEM STATEMENT}}} + \underbrace{\int_S u_x(\rho\vec{u})\cdot\vec{n}\, dS}_{\substack{\text{NET FLUX OF MOM. ACROSS BOUNDARIES OF C.V.}}}$$



$$0 = M\frac{dV}{dt} + \int_S u_x(\rho\vec{u})\cdot\vec{n}\, dS$$

b) $$-M\frac{dV}{dt} = \overbrace{-A_j\rho_j(V_j-V)(V_j-V)}^{\text{flow in}} \overbrace{-A_j\rho_j(V_j-V)(V_j-V)}^{\text{flow out}}$$

OPPOSITE TO NORMAL

NEGATIVE X-DIRECTION

NOTE THE SIGNS ARE BOTH NEGATIVE BUT FOR DIFFERENT REASONS!

EQNS OF MOTION:

$$\frac{dV}{dt} = \frac{2A_j\rho_j}{M}(V_j-V)^2 \qquad V_j > V$$

$$\frac{dV}{dt} = 0 \qquad V_j < V$$

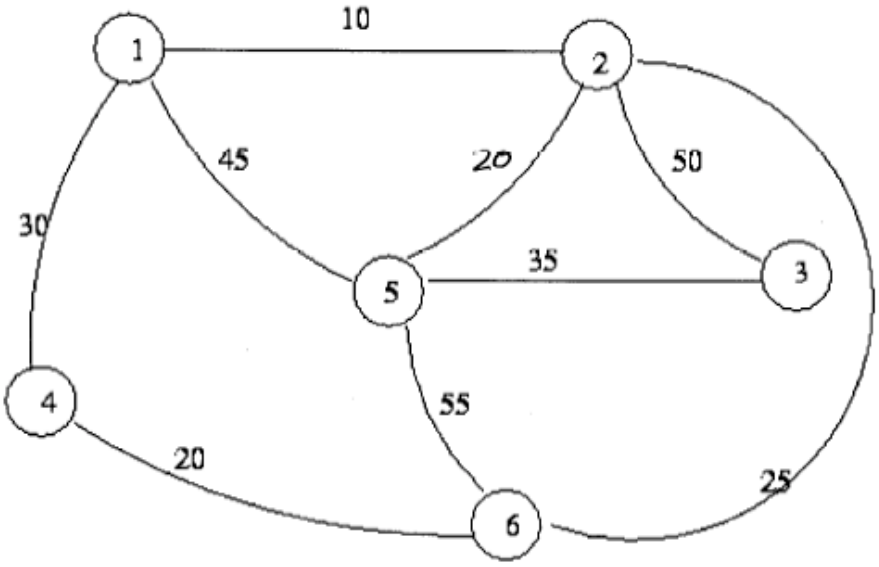
c) $$F = -2\rho_j A_j (V_j-V)^2$$

$$T = -F = 2\rho_j A_j (V_j-V)^2$$

Home Work 9

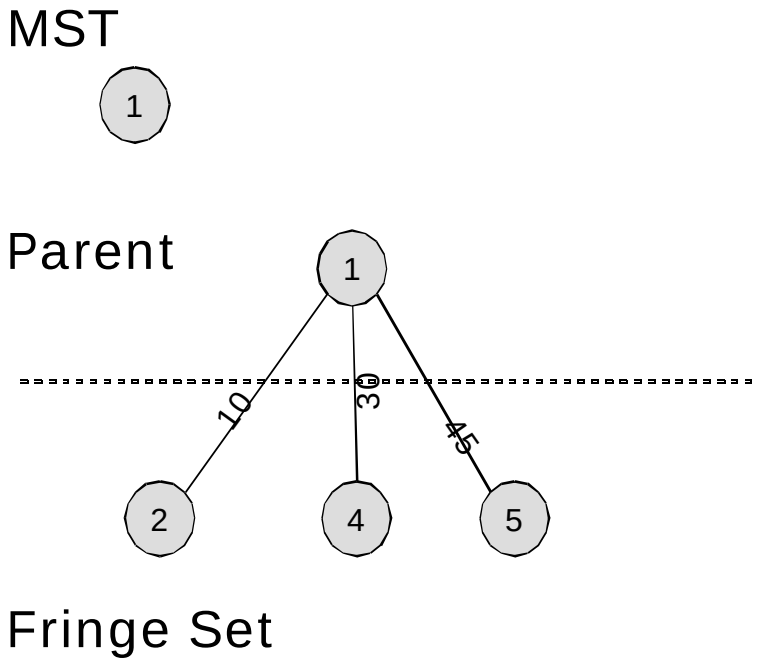
The problems in this problem set cover lectures C7, C8, C9 and C10

1. What is the Minimum Spanning Tree of the graph shown below using both Prim's and Kruskal's algorithm. Show all the steps in the computation of the MST (not just the final MST).

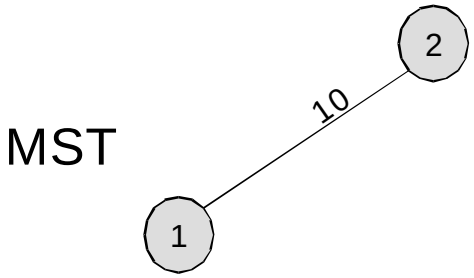


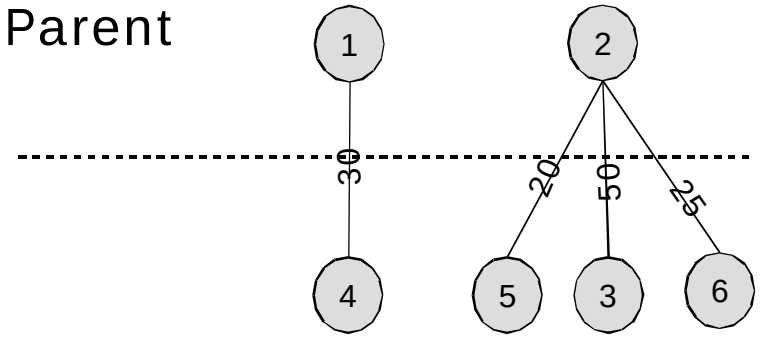
Prim's Algorithm

Step 1.



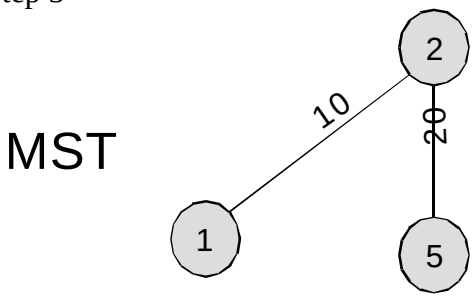
Step 2

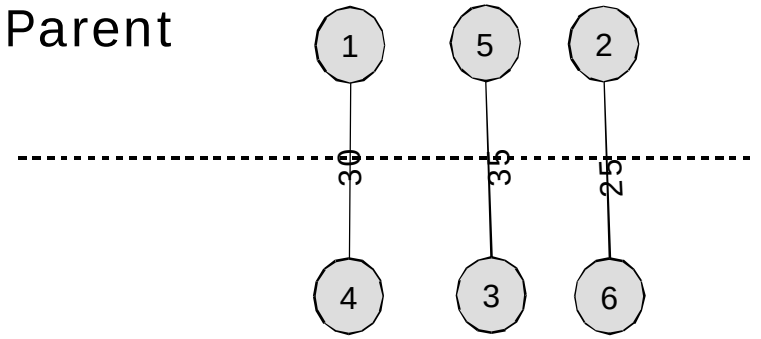




Fringe Set

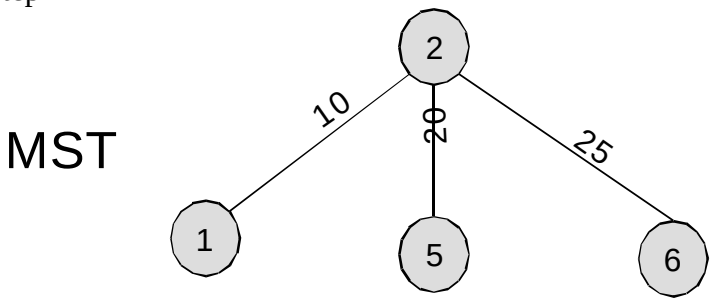
Step 3

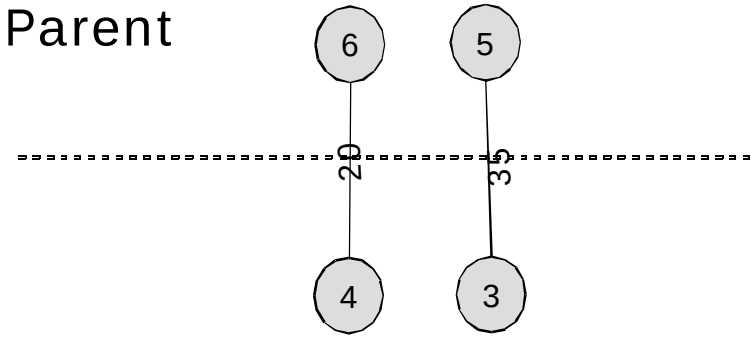




Fringe Set

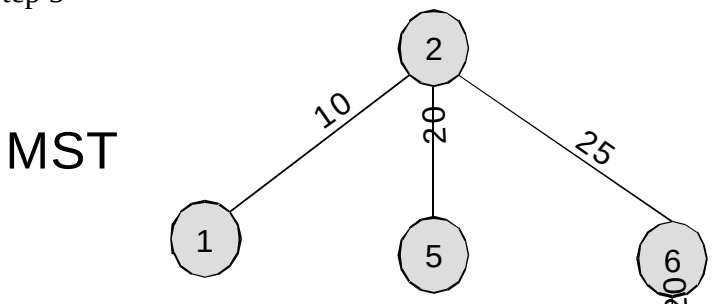
Step 4

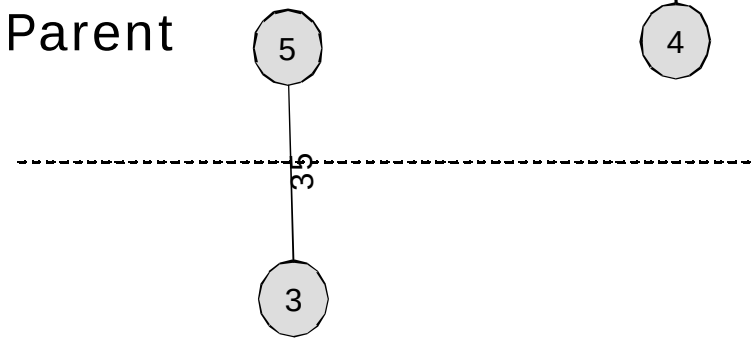




Fringe Set

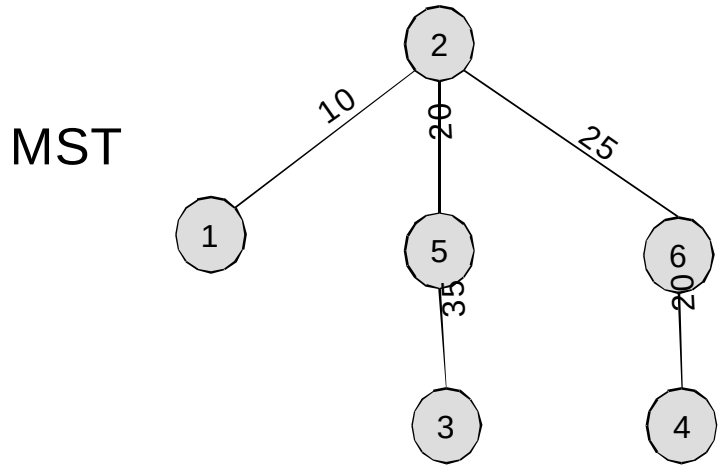
Step 5





Fringe Set

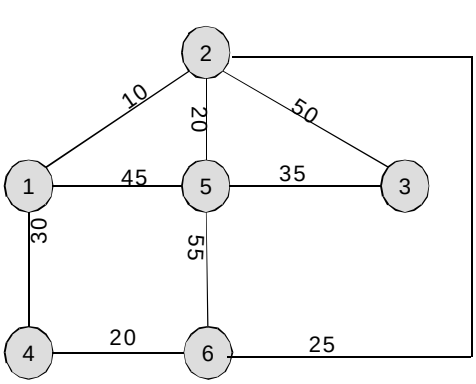
Minimum Spanning Tree

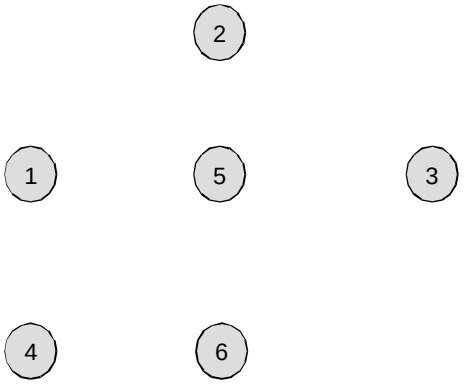


Weight of the MST = 10 + 20 + 25 + 20 + 35 = 110

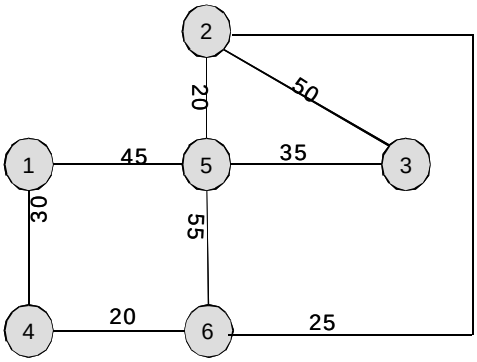
Kruskals Algorithm

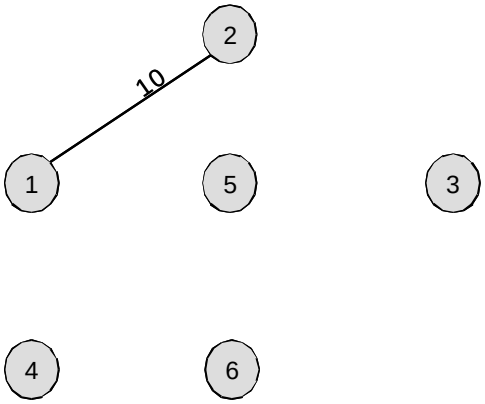
Initialization



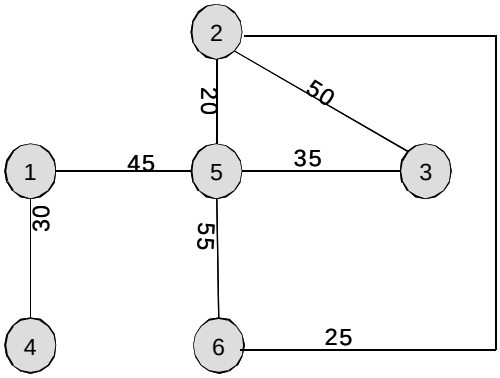


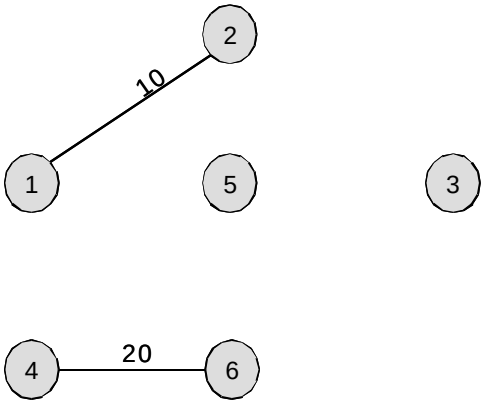
Step 1



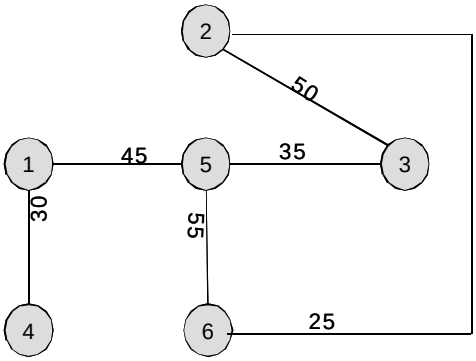


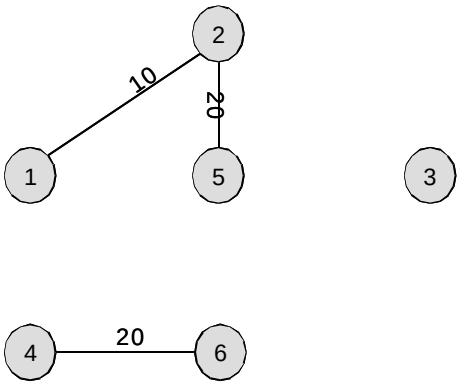
Step 2



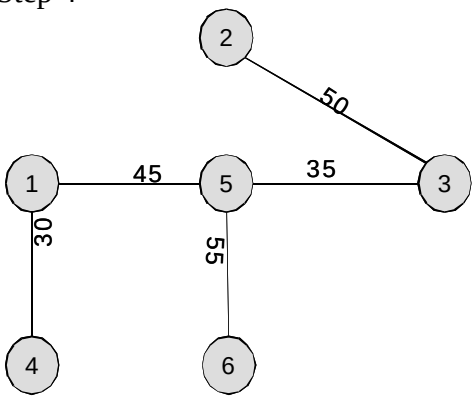


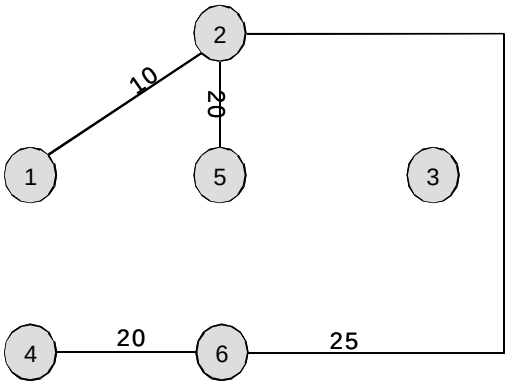
Step 3



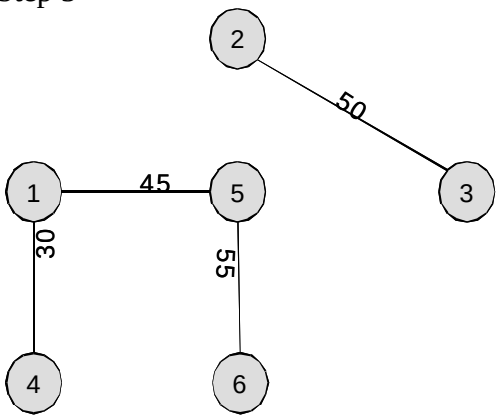


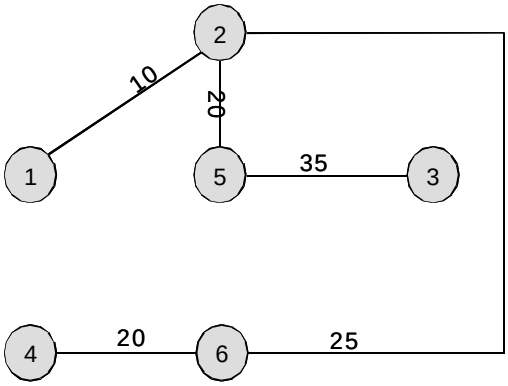
Step 4





Step 5

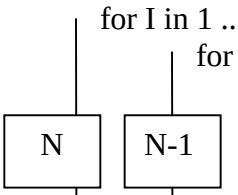




Weight of the MST = 10 + 20 + 20 + 25 + 35 = 110

2. Compute the computation complexity of the bubble sort algorithm. Show all the steps in the computation based on the algorithm.

Algorithm
Procedure Bubble_Sort(Input_Output_Array)




```
for I in 1 .. My_Array_Max loop
        for J in I+1 .. My_Array_Max loop
                if (Input_Output_Array(I) <= Input_Output_Array(J)) then
                        Temp := Input_Output_Array(I);
                        Input_Output_Array(I) := Input_Output_Array(J);
                        Input_Output__Array(J) := Temp;
                end if;
        end loop;
end loop;
```

end Bubble_Sort;

$O(N(N-1)) = O(N^2)$

3. What are the best case and worst case computation complexity of:

a. Inserting a node into an unsorted singly linked list

Inserting into an unsorted singly linked list is carried out using the add_to_front operation.
Both the best and worst case execution time is O(1).

b. Inserting a node into a sorted singly linked list

In the case of a sorted linked list, the list has to be traversed to find the right position. The list traversal takes O(n) in the worst case.

Best case execution time is O(1) if the element being inserted is the smallest element in the list (list in ascending order)

Worst case execution time is O(n) if the element being inserted is the largest element in the list (list in ascending order)

4.

a. Design an Ada95 Package to:

i. Read in N integers from an input file into an array. (N is user defined <=50)
ii. Sort the array in ascending order
iii. Perform binary search on the array.

The package is designed as follows:

Data Structures:

```
type my_array is array (1 .. 50) of integer;
```

Subprograms:

```
 -- procedure to create an array with <=50 elements
-- assumes that input can be found in input_file.txt
-- accepts the array
-- returns array with elements read from file and the number of elements read
procedure Create (
     Num_Array : in out My_Array;
     Size      :    out Integer   );

procedure Merge (
     Input_Array : in out My_Array;
     Lb_1        : in     Integer;
     Ub_1        : in     Integer;
     Lb_2        : in     Integer;
     Ub_2        : in     Integer   );
-- procedure to sort the array in ascending order
-- accepts the array and the lowerbound(lb) and upperbound(ub)
-- returns the sorted array
procedure Merge_Sort (
     Num_Array : in out My_Array;
     Lb        : in     Integer;
     Ub        : in     Integer   );

--function to perform binary search on the array
-- accepts the array, lb, ub and the element being searched for
-- returns the index of the element if it is found
-- returns -1 if the element is not found
function Binary_Search (
     Num_Array   : My_Array;
     Lb          : Integer;
     Ub          : Integer;
     Looking_For : Integer   )
  return Integer;
```

Algorithms:

Create
Pre-conditions: An array of 50 elements (num_array)
Post-Condition: Array with a maximum of 50 elements loaded from the file, size of the array.

Algorithm

1. Open the file input_file.txt

2. Initialize the counter to 0

3. While not End_Of_File (input_file)
    a. Increment the counter
    b. Read an element from the file into num_array(Counter)

4. Return num_array and Counter

Merge_Sort
Merge Sort is a *sort* algorithm that splits the items to be sorted into two groups, *recursively* sorts each group, and *merges* them into a final, sorted sequence.

Pre-Conditions: An array of 1 or more elements
Post-Condition : A sorted array

Algorithm

1. Check if LB < UB
    a. Call merge sort with
        1. Input Array
        2. LB = LB
        3. UB = (UB + LB)/2

    b. Call merge sort with
        1. Input Array
        2. LB = (UB + LB)/2 + 1
        3. UB = UB

    c. Call merge with
        1. Merge with parameters
            - Array to be sorted
            - Lower bound
            - (Lower bound + Upper bound) / 2
            - (Lower bound + Upper bound) / 2 +1
            - Upper bound

The merge sort procedure will recursively call itself until it has only single element arrays. These are inherently sorted. It will then merge the successively larger elements until the whole sorted array is produced.

Merge

Pre-Conditions: An array of 1 or more elements, Legal values of Lower_Bound_1, Upper_Bound_1, Lower_Bound_2 and Upper_Bound_2,

Post-Condition: A merged array that is sorted

Algorithm

1. Check if
    a. Lower_Bound_1 < Upper_Bound_1
    b. Lower_Bound_2 < Upper_Bound_2
    c. Lower_Bound_1 < Lower_Bound_2
    d. Upper_Bound_1 < Upper_Bound_2

2. If any of the above conditions are violated,
    a. Display Error
    b. Stop Executing the program

3. Set
    a. Index_1 := Lower_Bound_1;
    b. Index_2 := Lower_Bound_2;
    c. Index := Lower_Bound_1;
    d. Temp_Array := Input_Array

4. While (Index_1 <= Upper_Bound_1) and (Index_2 < =Upper_Bound_2)
    a. If (Input_Array(Index_1) < Input_Array (Index_2)
        i. Temp_Array(Index) := Input_Array(Index_1)
        ii. Index_1 := Index_1 + 1;
        iii. Index := Index + 1;
    b. Else
        i. Temp_Array(Index) := Input_Array(Index_2)
        ii. Index_2 := Index_2 + 1;
        iii. Index := Index + 1;

5. While (Index_1 <= Upper_Bound_1)
    a. Temp_Array(Index) := Input_Array(Index_1)
    b. Index_1 := Index_1 + 1;
    c. Index := Index + 1;

6. While (Index_2 <= Upper_Bound_2)
    a. Temp_Array(Index) := Input_Array(Index_2)
    b. Index_2 := Index_2 + 1;

c. Index := Index + 1;

7. Input_Array := Temp_Array

Binary Search

Pre-Conditions: Array to be searched
Item that you are searching for

Post-condition: Index location of the item you are searching for
Return –1 if the number is not found.

Algorithm

1. Set Return_Index to –1;

2. Set Current_Index to (Upper_Bound - Lower_Bound + 1) /2.

3. Loop
   i. if the lower_bound > upper_bound
      Exit;
   ii. if (Input_Array(Current_Index) = Number_to_Search) then
      Return_Index = Current_Index)
      Exit;

   iii. if (Input_Array(Current_Index) > Number_to_Search) then
      Lower_Bound = Current_Index +1
   else
      Upper_Bound = Current_Index – 1

4. Return Return_Index

b. Write a program to test your package that will
- Prompt the user for a number to search for.
- If the number is found using the binary search algorithm
  - Display the location (index)
  - Display the number
- If the number is not found using the binary search algorithm
  - Display “Number not in array to the user”

```
------------------------------------------------------------
-- program to test Home_Work_9 Package
-- Programmer: Jayakanth Srinivasan
-- Date Last Modified: April 06,2004
------------------------------------------------------------
with Ada.Text_Io;
with Ada.Integer_Text_IO;
with Home_Work_9;
use Home_Work_9;

procedure Test_Hw_9 is
```

```
   My_Test_Array : My_Array;
   Size          : Integer;
   Location      : Integer;
   Find : Integer;

begin
   -- create the array
   Create(My_Test_Array,Size);

   -- get number to search for from user
   Ada.Text_Io.Put("Please Enter the Number to Search For");
   Ada.Integer_Text_Io.Get(Find);

   -- display unsorted array to the user
   for I in 1..Size loop
      Ada.Text_Io.Put_Line(Integer'Image(My_Test_Array(I)));
   end loop;
   Ada.Text_Io.New_Line;

   -- sort the array using the merge sort algorithm
   Merge_Sort(My_Test_Array,1,Size);

   -- display the sorted algorithm
   for I in 1..Size loop
      Ada.Text_Io.Put_Line(Integer'Image(My_Test_Array(I)));
   end loop;

   -- perform a binary search on the array
   Location:= Binary_Search(My_Test_Array, 1, Size, Find);
   if Location /= -1 then
      Ada.Text_Io.Put("Found number at");
      Ada.Text_Io.Put_Line(Integer'Image(Location));
      Ada.Text_Io.Put_Line(Integer'Image(My_Test_Array(Location)));
   else
      Ada.Text_Io.Put_Line("Number Not Found in Array");
   end if;

end Test_Hw_9;
```

5. Implement the merge sort algorithm as an Ada95 program. Your program should
   - Read in N integers from an input file. (N is user defined <=50)
   - Sort using your merge sort implementation.
   - Display the sorted and unsorted inputs to the user

Solved in problem 4.

Unified Engineering II **Spring 2004**

Problem S8 Solution

1. The convolution is given by

$$y(t) = g(t) * u(t) = \int_{-\infty}^{\infty} g(t-\tau)u(\tau)\,d\tau \tag{1}$$

Note that $u(\tau)$ is nonzero only for $-3 \le \tau \le 0$, and $g(t-\tau)$ is nonzero only for $0 \le t-\tau \le 3$, that is, for $-3+t \le \tau \le t$. So there are four distinct regimes:

(a) $t < -3$

(b) $-3 \le t \le 0$

(c) $0 \le t \le 3$

(d) $t > 3$

For cases (a) and (d), there is no overlap between $g(t-\tau)$ and $u(\tau)$, so $y(t) = 0$. For case (b), the overlap is for $-3 \le \tau \le t$. So

$$\begin{aligned} y(t) &= \int_{-\infty}^{\infty} g(t-\tau)u(\tau)\,d\tau \\ &= \int_{-3}^{t} \sin(-2\pi(t-\tau))\sin(2\pi\tau)\,d\tau \end{aligned}$$

At this point, we have to do a little trig:

$$\begin{aligned} \sin(-2\pi(t-\tau))\sin(2\pi\tau) &= \sin(2\pi(\tau-t))\sin(2\pi\tau) \\ &= [\sin(2\pi\tau)\cos(2\pi t) - \cos(2\pi\tau)\sin(2\pi t)]\sin(2\pi\tau) \\ &= \cos(2\pi t)\sin^2(2\pi\tau) - \sin(2\pi t)\cos(2\pi\tau)\sin(2\pi\tau) \\ &= \cos(2\pi t)\frac{1-\cos(4\pi\tau)}{2} - \sin(2\pi t)\frac{\sin(4\pi\tau)}{2} \end{aligned}$$

So the integral is given by

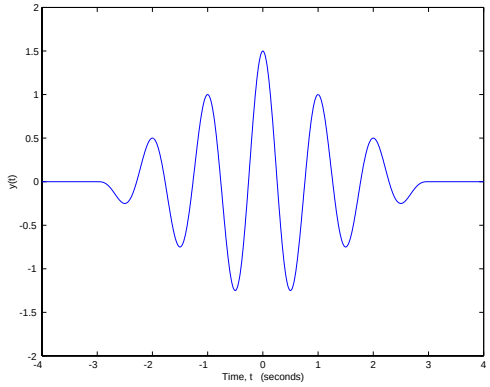
$$\begin{aligned} y(t) &= \int_{-3}^{t} \frac{\cos(2\pi t)}{2}\,d\tau - \int_{-3}^{t} \frac{\cos(2\pi t)}{2}\cos(4\pi\tau)\,d\tau - \int_{-3}^{t} \frac{\sin(2\pi t)}{2}\sin(4\pi\tau)\,d\tau \\ &= \frac{\cos(2\pi t)}{2}(t+3) - \frac{\cos(2\pi t)}{8\pi}\sin(4\pi\tau)\Big|_{\tau=-3}^{t} + \frac{\sin(2\pi t)}{8\pi}\cos(4\pi\tau)\Big|_{\tau=-3}^{t} \\ &= \frac{\cos(2\pi t)}{2}(t+3) - \frac{\cos(2\pi t)}{8\pi}\sin(4\pi t) + \frac{\sin(2\pi t)}{8\pi}[\cos(4\pi t) - 1] \end{aligned}$$

(As often happens with problems involving trig functions, there are other equivalent expressions.)

For case (c), the region of integration is $-3+t \le \tau \le 0$. So

$$\begin{aligned} y(t) &= \int_{-3+t}^{0} \frac{\cos(2\pi t)}{2}\, d\tau - \int_{-3+t}^{0} \frac{\cos(2\pi t)}{2} \cos(4\pi\tau)\, d\tau - \int_{-3+t}^{0} \frac{\sin(2\pi t)}{2} \sin(4\pi\tau)\, d\tau \\ &= \frac{\cos(2\pi t)}{2}(3-t) - \frac{\cos(2\pi t)}{8\pi} \sin(4\pi\tau)\Big|_{\tau=-3+t}^{0} + \frac{\sin(2\pi t)}{8\pi} \cos(4\pi\tau)\Big|_{\tau=-3+t}^{0} \\ &= \frac{\cos(2\pi t)}{2}(3-t) + \frac{\cos(2\pi t)}{8\pi} \sin(4\pi t) - \frac{\sin(2\pi t)}{8\pi} \left[\cos(4\pi t) - 1\right] \end{aligned}$$

2. $y(t)$ is plotted below.

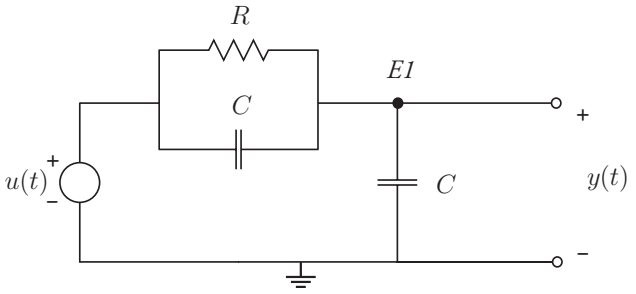


3. The maximum value of $y(t-T)$ occurs at time T. So I would use this center peak to identify the delay time T.

4. The adjacent peaks are nearly as tall as the center peak, so if noise were added to the signal, the tallest peak might not be the center peak, so we might use the wrong peak to determine the delay time.

5. The chirp signal of Problem S6 produces an ambiguity function with only one prominent peak. Therefore, the addition of noise should not make it difficult to accurately determine the delay time.

(a)



We can use impedance methods to solve for $Y(s)$ in terms of $U(s)$. Label ground and E_1 as shown. Then KCL at E_1 yields

$$Cs(E_1 - 0) + \left(Cs + \frac{1}{R}\right)(E_1 - U) = 0 \tag{1}$$

Simplifying, we have

$$\left(2Cs + \frac{1}{R}\right) E_1 = \left(Cs + \frac{1}{R}\right) U$$

Since we are finding the step response,

$$U(s) = \frac{1}{s}, \quad \text{Re}[s] > 0$$

Plugging in numbers, we have

$$(0.5s + 0.5)E_1(s) = (0.25s + 0.5)\,\frac{1}{s}$$

Solving for E_1, we have

$$E_1(s) = \frac{0.25s + 0.5}{(0.5s + 0.5)s} = \frac{0.5s + 1}{s(s + 1)}$$

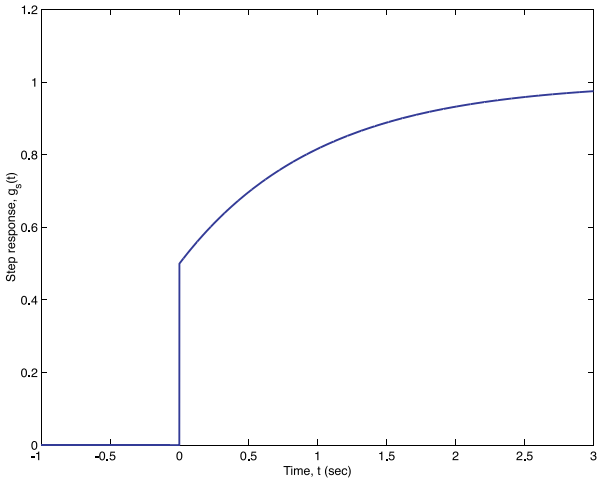
The region of convergence must be $\text{Re}[s] > 0$, since the step response is causal, and the pole at $s = 0$ is the rightmost pole. Using partial fraction expansions,

$$E_1(s) = \frac{1}{s} - \frac{0.5}{s + 1}$$

Therefore, $g_s(t) = y(t) = e_1(t)$ is the inverse transform of $E_1(t)$, so

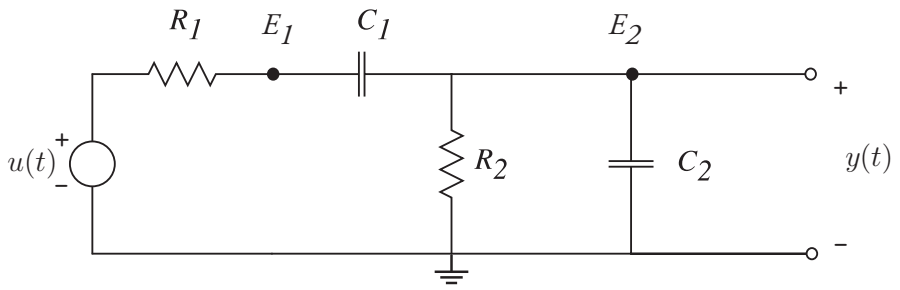
$$y(t) = \left(1 - 0.5e^{-t}\right)\sigma(t)$$

The step response is plotted below:



Normal differential equation methods are difficult to apply, because we cannot apply the normal initial condition that $e_1(0) = 0$. This is because the chain of capacitors running from the voltage source to ground causes there to be an impulse of current at time $t = 0$, and the voltages across the capacitors change instantaneously at $t = 0$. It is possible to use differential equation methods, we just have to be more careful about the initial conditions. However, Laplace methods are easier.

(b)



Again, use impedance methods, using the node labelling above. Then the node equations are

$$\begin{aligned} (C_1 s + G_1)E_1 \quad - \quad & C_1 s E_2 \quad = G_1 U \\ -C_1 s E_1 \quad + \quad & [(C_1 + C_2)s + G_2]E_2 \quad = 0 \end{aligned} \tag{2}$$

where $G = 1/R$. We can use Cramer's rule to solve for E_2:

$$\begin{aligned} E_2(s) &= \frac{\begin{vmatrix} C_1 s + G_1 & G_1 U(s) \\ -C_1 s & 0 \end{vmatrix}}{\begin{vmatrix} C_1 s + G_1 & -C_1 s \\ -C_1 s & (C_1 + C_2)s + G_2 \end{vmatrix}} \\ &= \frac{G_1 C_1 s}{C_1 C_2 s^2 + (G_1 C_1 + G_1 C_2 + G_2 C_1)s + G_1 G_2} U(s) \end{aligned} \tag{3}$$

Since we are finding the step response,

$$U(s) = \frac{1}{s}, \qquad \mathrm{Re}[s] > 0$$

Plugging in numbers, we have

$$Y(s) = E_2(s) = \frac{0.1s}{0.06s^2 + 0.35s + 0.25}\frac{1}{s} = \frac{5/3}{s^2 + 5.833\bar{3}s + 4.166\bar{6}} \tag{4}$$

In order to find $y(t)$, we must expand $Y(s)$ in a partial fraction expansion. To do so, we must factor the denominator, using either numerical techniques or the quadratic formula. The result is

$$s^2 + 5.833\bar{3}s + 4.166\bar{6} = (s + 5)(s + 0.833\bar{3}) \tag{5}$$

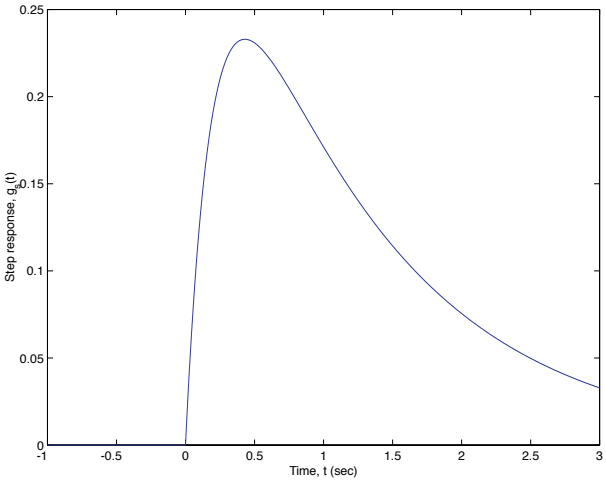
We can use the coverup method to factor $Y(s)$, so that

$$Y(s) = \frac{5/3}{(s + 5)(s + 0.833\bar{3})} = \frac{-0.4}{s + 5} + \frac{0.4}{s + 0.833\bar{3}} \tag{6}$$

The region of convergence must be $\mathrm{Re}[s] > -0.833\bar{3}$, since the step response is causal, and the r.o.c. is to the right of the right-most pole. Therefore, the step response is given by the inverse transform of $Y(s)$, so that

$$g_s(t) = \left(-0.4e^{-5t} + 0.4e^{-0.833\bar{3}t}\right)\sigma(t) \tag{7}$$

The step response is plotted below:


0.25
0.2
0.15
0.1
0.05
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
Step response, g (t)
Time, t (sec)