Assignment 2

Problem 4:
Consider this instance of the knapsack problem:
The Weights are : 12,8,16,4,20,28
The Prices are : 48,24,80,24,40,28
The Capacity = 28
Find the greedy the solution for this problem, showing the process step by step.

Solution :

Given the weights and prices, we need to calculate the price per weight (i.e price per 1 unit of weight)

Weights	12	8	16	4	20	28
Prices	48	24	80	24	40	28
Prices/Weight	4	3	5	6	2	1

Now, we have the price per weight. For every step we need to consider

1.Item to be considered with maximum price

2.Make sure you pick the item with weight such that total weight in the bag is <= Capacity

From the price per weight values, the maximum is 6 for item no.4

Picking item 4, Weight = 4, Price = 6

Bag Weight = 4 Updated Capacity = 28-4= 24

Picking item 3, Weight = 16, Price = 80

Bag Weight = 4+16=20 Updated Capacity = 24-16 = 8

Picking item 1, Weight = 12, Price = 48

We have the capacity reduced to 8, so we can't take all the weight. However, we have the option to take the partial weights, i.e we can take only 8 weights out of 12.

Bag Weight = 20+8 = 28 Updated Capacity = 8-8

Total Price of the Bag = 4*6 + 16*5 +8*4 = 24+80+32 =136

Problem 1:

a. Show step by step how the heapsort algorithm sorts the array 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7 starting from the min-heap you built in homework 1 (Problem 4a) for the same array.

Solution :

Heap sort works in this way

```
procedure HeapSort()
{
        Build Heap from the input array elements
        while(there are no elements left)
        {
                Delete root node from heap and store it in separate array one by one
                Adjust the Heap
        }
}
```

The Heap looks in this way for the given input.

5	7	13	27	8	56	18	67	35	62	48

We store the deleted elements in the separate array

Delete 5 and adjust heap :

7	8	13	27	48	56	18	67	35	62	

Adjusted heap

5										

Delete 7 and adjust heap:

8	27	13	35	48	56	18	67	62		

Adjusted heap

5	7									

Delete 8 and adjust heap:

13	27	18	35	48	56	62	67			

Adjusted heap

5	7	8								

Delete 13 and adjust heap:

18	27	56	35	48	67	62				

Adjusted heap

5	7	8	13							

Delete 18 and adjust heap:

27	35	56	62	48	67					

Adjusted heap

5	7	8	13	18						

Delete 27 and adjust heap:

35	48	56	62	67						

Adjusted heap

5	7	8	13	18	27					

Delete 35 and adjust heap:

48	62	56	67							

Adjusted heap

5	7	8	13	18	27	35				

Delete 48 and adjust heap:

56	62	67								

Adjusted heap

5	7	8	13	18	27	35	48			

Delete 56 and adjust heap:

62	67									

Adjusted heap

5	7	8	13	18	27	35	48	56		

Delete 62 and adjust heap:

67										

Adjusted heap

5	7	8	13	18	27	35	48	56	62	

Delete 67

5	7	8	13	18	27	35	48	56	62	67

Our final sorted heap (ascending order)

5	7	8	13	18	27	35	48	56	62	67

d. **Show step by step how the insertion sort algorithm sorts the same array. (This algorithm sorts by repeated insertion into a (initially empty) sorted array.)**

Solution
Initial sorted array is [] and input array is [**27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7**]
Step 1 : Insert 27 in to empty sorted array
Sorted array : [**27**]
Input array : [**13, 56, 35, 48, 8, 18, 67, 5, 62, 7**]

Step 2 : Insert 13 in the sorted array
Sorted array [27,13], since 13<27 we swap them
Sorted array [13,27]
Input array : [**56, 35, 48, 8, 18, 67, 5, 62, 7]**

Step 3 : Insert 56 in the sorted array
Sorted array : [13,27,56], since 56>27 and 27>13, they are in their correct positions
Input array : [**35, 48, 8, 18, 67, 5, 62, 7**]

Step 4 : Insert 35 in the sorted array
Sorted array : [13,27,56,35], since 35<56, we swap them
[13,27,35,56], since 56>35,35>27,27>13, they are in their correct positions
Input array : [**48, 8, 18, 67, 5, 62, 7]**

Step 5 : Insert 48 in the sorted array
Sorted array : [13,27,56,35,48], since 48>35, all the elements are in their correct positions.
Input array : [**8, 18, 67, 5, 62, 7**]

Step 6 : Insert 8 in the sorted array
Sorted array : [13,27,56,35,8], since 8<35, we swap them
[13,27,56,8,35], since 8<56,we swap them
[13,27,8,56,35], since 8<27, we swap them
[13,8,27,56,35], since 8<13, we swap them
[8,13,27,56,35],since 35<56, we swap them
[8,13,27,35,56]
Input array : [**18, 67, 5, 62, 7**]

Step 7 : Insert 18 in the sorted array
Sorted array : [8,13,27,35,56,18], since 18<56, we swap them
[8,13,27,35,18,56], since 18<35, we swap them
[8,13,27,18,35,56], since 18<37, we swap them
[8,13,18,27,35,56]
Input array : [**67, 5, 62, 7**]

Step 8 : Insert 67 in the sorted array
Sorted array : [8,13,18,27,35,56,67], since 67>56, we don’t swap them
Input array : [5,62,7]

Step 9 : Insert 5 into the array
Sorted array : [8,13,18,27,35,56,67,5], since 5<67, we swap them
[8,13,18,27,35,56,5,67], since 5<56, we swap them
[8,13,18,27,35,5,56,67], since 5<35, we swap them
[8,13,18,27,5,35,56,67], since 5<27, we swap them
[8,13,18,5,27,35,56,67], since 5<18, we swap them
[8,13,5,18,27,35,56,67], since 5<13, we swap them
[8,5,13,18,27,35,56,67], since 5<8, we swap them
[5,8,13,18,27,35,56,67]
Input array : [62,7]

Step 10: Insert 62 into the sorted array
Sorted array : [5,8,13,18,27,35,56,67,62], since 62<67, we swap them
[5,8,13,18,27,35,56,62,67], since 62>56, we don't swap them
Input array : [7]

Step 11 : Insert 7 into the sorted array
Sorted array : [5,8,13,18,27,35,56,62,67,7]. since 7<67, we swap them
[5,8,13,18,27,35,56,62,7,67], since 7<62, we swap them
[5,8,13,18,27,35,56,7,62,67], since 7<56, we swap them
[5,8,13,18,27,35,7,56,62,67], since 7<35, we swap them
[5,8,13,18,27,7,35,56,62,67], since 7<27, we swap them
[5,8,13,18,7,27,35,56,62,67], since 7<18, we swap them
[5,8,13,7,18,27,35,56,62,67], since 7<13, we swap them
[5,8,7,13,18,27,35,56,62,67], since 7<8, we swap them
[5,7,8,13,18,27,35,56,62,67], since 7>5, we don't swap them
Input array : []

Final sorted array : **[5,7,8,13,18,27,35,56,62,67]**

e. Show step by step how selection sort sorts the same array. (This finds the minimum first, then the 2nd min, then the 3rd min, and so on. It has nothing to do with quick-select.)

Solution:

Initial Input array :

27	13	56	35	48	8	18	67	5	62	7

Finding 1st minimum by scanning from index 2 (i.e from 13 element)

27	13	56	35	48	8	18	67	5	62	7

Since we found 5 less than 27, we swap them

5	13	56	35	48	8	18	67	27	62	7

Finding 2nd minimum by scanning each element starting from index 3 till last (i.e from 56 element) and comparing with element 13.

5	13	56	35	48	8	18	67	27	62	7

Since we found 7 which is less than 13, we swap them

5	7	56	35	48	8	18	67	27	62	13

Finding 3rd minimum by scanning each element starting from index 4 till last (i.e from 35 element) and comparing with element 56.

5	7	56	35	48	8	18	67	27	62	13

Since we found 8 which is less than 56, we swap them

5	7	8	35	48	56	18	67	27	62	13

Finding 4th minimum by scanning each element starting from index 5 till last(i.e from 48 element) and comparing with element 35

5	7	8	35	48	56	18	67	27	62	13

Since we found 8 which is less than 56, we swap them

5	7	8	13	48	56	18	67	27	62	35

Finding 5th minimum by scanning each element starting from index 6 till last(i.e from 56 element) and comparing with element 48

5	7	8	13	48	56	18	67	27	62	35

Since we found 18 which is less than 48, we swap them

5	7	8	13	18	56	48	67	27	62	35

Finding 5th minimum by scanning each element starting from index 7 till last(i.e from 48 element) and comparing with element 56

5	7	8	13	18	56	48	67	27	62	35

Since we found 27 which is less than 56, we swap them

5	7	8	13	18	27	48	67	56	62	35

Finding 6th minimum by scanning each element starting from index 7 till last(i.e from 67 element) and comparing with element 48

5	7	8	13	18	27	48	67	56	62	35

Since we found 35 which is less than 48, we swap them

5	7	8	13	18	27	35	67	56	62	48

Finding 7th minimum by scanning each element starting from index 8 till last(i.e from 56 element) and comparing with element 67

5	7	8	13	18	27	35	67	56	62	48

Since we found 48 which is less than 67, we swap them

5	7	8	13	18	27	35	48	56	62	67

Finding 8th minimum by scanning each element starting from index 9 till last(i.e from 62 element) and comparing with element 56

5	7	8	13	18	27	35	48	56	62	67

Since we found no element that is less than 56, we proceed with checking from next index

Finding 9th minimum by scanning each element starting from index 10 till last(i.e from 67 element) and comparing with element 62

5	7	8	13	18	27	35	48	56	62	67

Since we found no element that is less than 62, we proceed with checking from next index

Finding 9th minimum, since we have reached end of the array, it currently pointed element is in its correct position.

Our final array which is sorted is

5	7	8	13	18	27	35	48	56	62	67

c. Show step by step how the merge sort algorithm sorts the same array

Solution :

Given the input array of elements

27	13	56	35	48	8	18	67	5	62	7

length of the input array : 11
mid = 11/2 = ~5
Partition array into two halves at index as 5

27	13	56	35	48	8

Length = 6, mid = 3
Partition array into two at index 3

18	67	5	62	7

Length = 5, mid =2
Partition array into two at index 2

27	13	56

Length = 3, mid=1
Partitioning into two

35	48	8

Length = 3, mid=1
partitioning into two

18	67	5

Length = 3, mid =1
Partitioning into two

62	7

Length=2, mid=0
Partitioning into two

27 13	56	35 48	8	18 67	5	62	7		

Length=2,mid=1
Partitioning into two

Length=2,mid=1
Partitioning into two

Length=2,mid=1
Partitioning into two

27	13	56	35	48	8	18	67	5	62	7

Put them in asc order in a new array

13 27	56	35 48	8	18 67	5	7 62				

Put them in asc order in a new array

Put them in asc order in a new array

Put them in asc order in a new array

13 27 56	8 35 48	5 18 67	7 62

Put them in asc order in a new array

Put them in asc order in a new array

8 13 27 35 48 56	5 7 18 62 67

Put them in asc order in a new array

5	7	8	13	18	27	35	48	56	62	67

b. Show step by step how the quicksort algorithm sorts the same array 27, 13, 56, 35, 48, 8, 18, 67, 5, 62, 7. Indicate at each step what the partitioning element is.

Given the input array :

27	13	56	35	48	8	18	67	5	62	7

We need to partition the array into two halves about a pivot point, such that all the elements to the left of pivot are less that that element and all the elements to the right of the pivot are greater than that pivot element.

We repeatedly perform this partitioning till there are no more partitions are possible(i.e till one element).

We now follow partitioning about the middle index for the each array. i.e we put middle element as pivot for each subsequent partitioning.

The partitioning algorithm works as follows :

```
procedure Partition(A[start:end])
{
   int i=start, j = end;
   int pivot = A[mid] //mid is calculated as A(⌊(start+end)/2⌋)
   while(i<j)
      {
         while( A[i] < pivot && i<end){i++;}
         while(A[j]>pivot && j>start){j--;}
         if(i<j)
         {
            swap(A[i],A[j]);
            i++; j--;
         }
      }
   swap(A[start],a[j]);
   return j;
}
```

Partitioning about the pivot = 27
Length =11, i=2, j = 11, pivot A[1]=27
i=3,j=11, A[3]>27 & A[11] < 7, we swap them

27	13	56	35	48	8	18	67	5	62	7

i=3,j=11, A[1] > pivot , A[11]<pivot and i<j, we swap both and i++, j--

27	13	7	35	48	8	18	67	5	62	56

i=4, j=10, A[4]>27, A[10]>27 so j--
i=4,j=9, A[4]>27, A[9]<27, so we swap them and i++, j--

27	13	7	5	48	8	18	67	35	62	56

i=5, j=8, A[5] >27, A[8]>27 so j—
i=5, j=7, A[5]>27,A[7] <27 so we swap them and i++ and j--;

27	13	7	5	18	8	48	67	35	62	56

Now, the i and j crossed so we swap pivot element with A[j]

8	13	7	5	18	27	48	67	35	62	56

Now, we Perform partition on left of the array and right of the array.

8	13	7	5	18	27	48	67	35	62	56

For the left subarray
i=2,j=5, pivot = A[1]=8
For the right subarray
i=8,j=10, pivot = A[7]=48

Now iterating
Left sub array :
i=2, A[2]>8, j=5, A[j]>8, so j--
i=2, A[2]>8, j=4, A[4]<8, so we swap them

Right sub array :
i=8, A[8]>48, j=11, A[11]>48 so j--;
i=8, A[8]>48, j=10, A[10]>48 so j--;
i=8, A[8]>48, j=9, A[9]<48 so we swap them

8	5	7	13	18	27	48	35	67	62	56

Now i and j both crossed each other in both the subarrays, so we swap pivot element with A[j]

7	5	8	13	18	27	35	48	67	62	56

Now iterating through subarray

7	5

Pivot = 7, i=2,j=2;A[2],A[5]<7, so we swap A[i] & A[j], but since I and j point to same element, the array remains same and we i++ (but we have reached end) and j--;
Now i and j crossed each other, so we swap A[1] with A[j]

5	7

5	7	8	13	18	27	35	48	67	62	56

Now iterating through sub array

13	18

Pivot = 13, i=5,j=5; A[5]>13, A[5]>13 so j--;
Since i and j are crossed each other, so we swap A[4] (pivot) with A[4]

13	18

5	7	8	13	18	27	35	48	67	62	56

For the subarray

35

, we just return the index of the element.

5	7	8	13	18	27	35	48	67	62	56

For the subarray

67	62	56

Pivot = 67, i=10, j=11

67	62	56

I=10,j=11, A[i]<62 so i++, A[j]<56
I=11,j=11, A[i]<62 so i++, A[j]<56

67	62	56

Now I and j crossed each other, so we swap pivot with A[j]

56	62	67

So our final array looks like

5	7	8	13	18	27	35	48	56	62	67

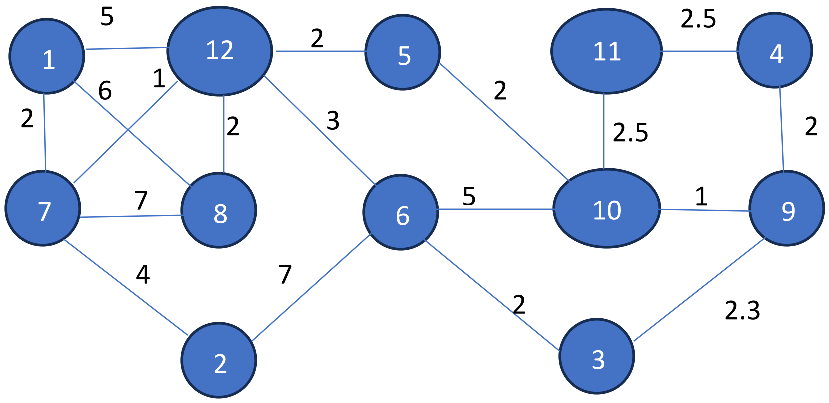
Problem 5:

Let G=(V,E) be the following weighted graph: V={1,2,3,4,5,6,7,8,9,10,11,12}
E={[(1,7) 2], [(1,8) 6], [(1,12) 5], [(7,12) 1], [(8,12) 2], [(12,6) 3], [(2,6) 7], [(2,7) 4], [(7,8) 7], [(12,5) 2], [(6,10) 5], [(5,10) 2], [(9,10) 1], [(9,4) 2], [(10,11) 2.5], [(11,4) 2.5], [(3,6) 2], [(3,9) 2.3]}, where [(*i*,*j*) *a*] means that (*i*,*j*) is an edge of weight *a*.
b. Using the greedy Dijkstra's single-source shortest-path algorithm, find the distances between node 1 and all the other nodes. Show the values of the DIST array at every step.
c. Show the actual shortest paths of part (b). Note that these paths together form a spanning tree of G. Is this tree a minimum spanning tree?

Solution :

a. Use the greedy method (Kruskal's algorithm) to find a minimum spanning tree of G. Show the tree after every step of the algorithm.

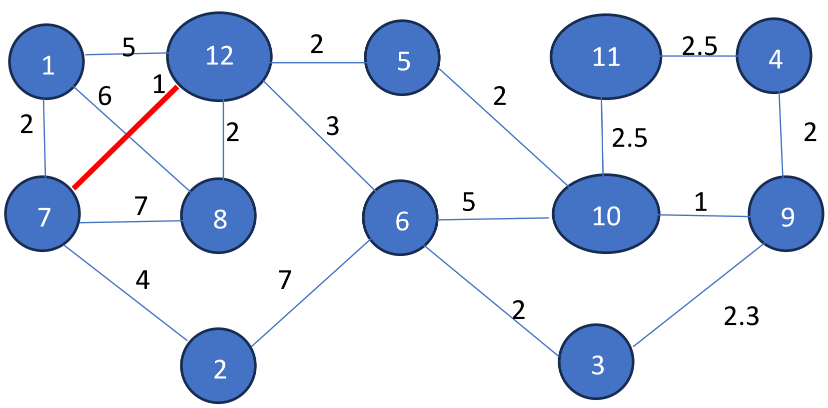
Given vertices set V=={1,2,3,4,5,6,7,8,9,10,11,12} and weighted edge set E=={[(1,7) 2], [(1,8) 6], [(1,12) 5], [(7,12) 1], [(8,12) 2], [(12,6) 3], [(2,6) 7], [(2,7) 4], [(7,8) 7], [(12,5) 2], [(6,10) 5], [(5,10) 2], [(9,10) 1], [(9,4) 2], [(10,11) 2.5], [(11,4) 2.5], [(3,6) 2], [(3,9) 2.3]}



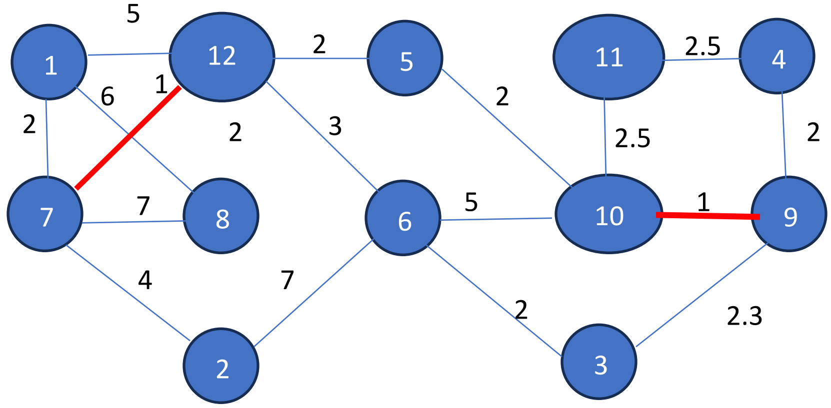
The tree looks like above for the given V and E sets.

According to kruskal's algorithm, we need to pick up the minimum weight edge each time
And add it to the tree if it doesn't form a cycle

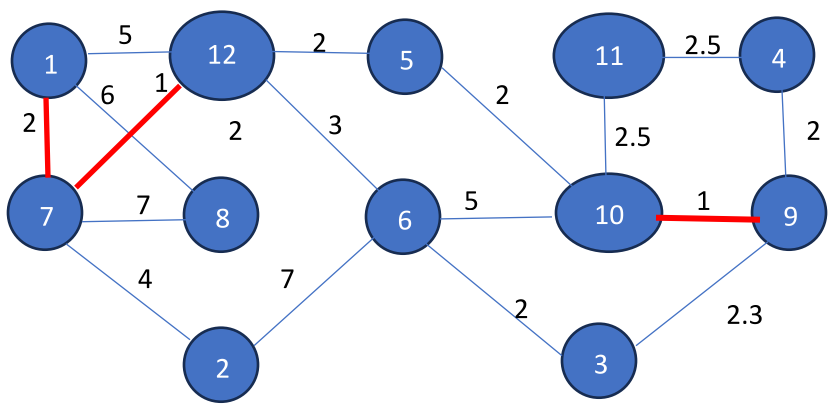
Picking edge (7,12) = No cycle detected – weight =1



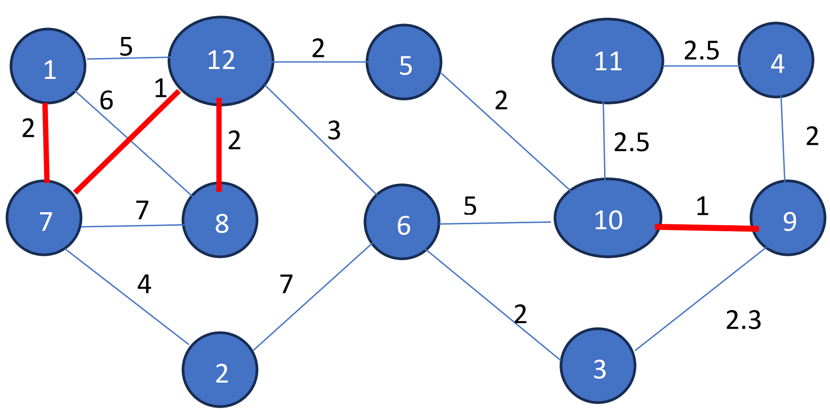
Pick edge (10,9) = No cycle detected – weight = 1



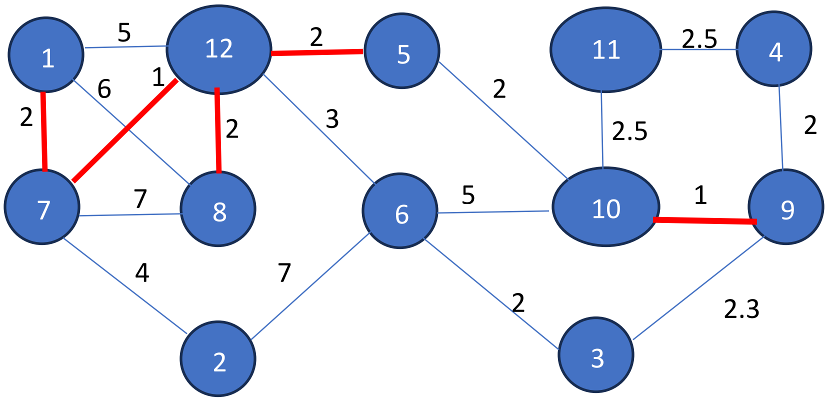
Pick edge (1,7) - No cycle detected. – weight =2



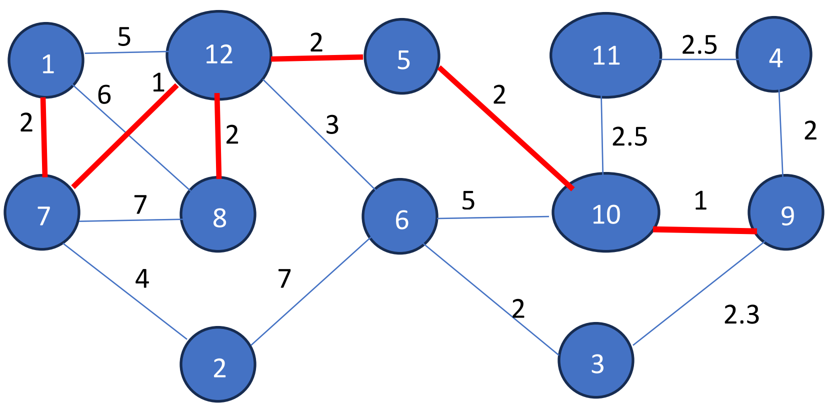
Pick edge (12,8) – No cycle detected - weight = 2



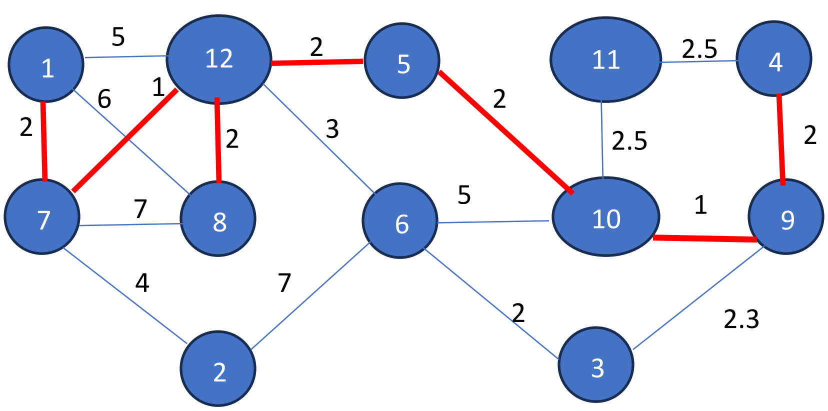
Pick (12,5) = No cycle detected - weight = 2



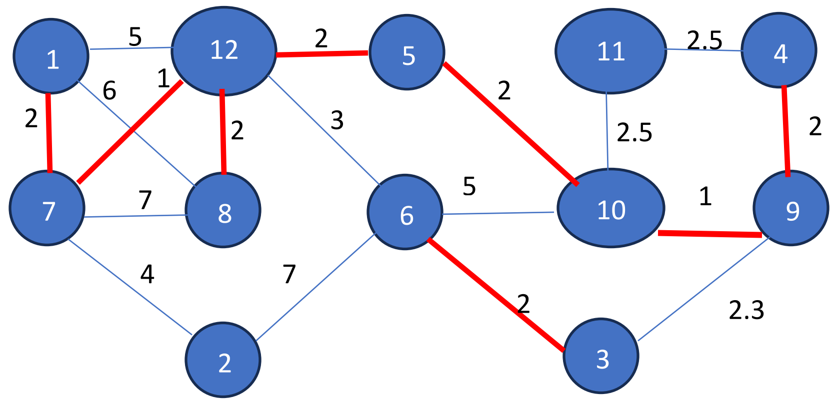
Pick (5,10) = No cycle detected – weight = 2



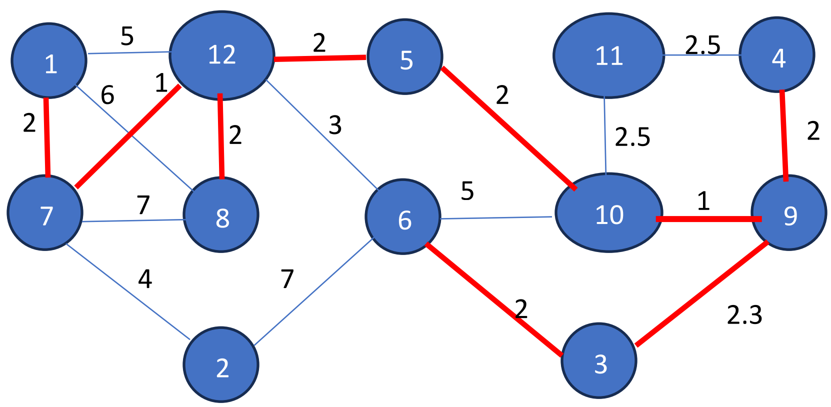
Pick(4,9) = No cycle detected - weight = 2



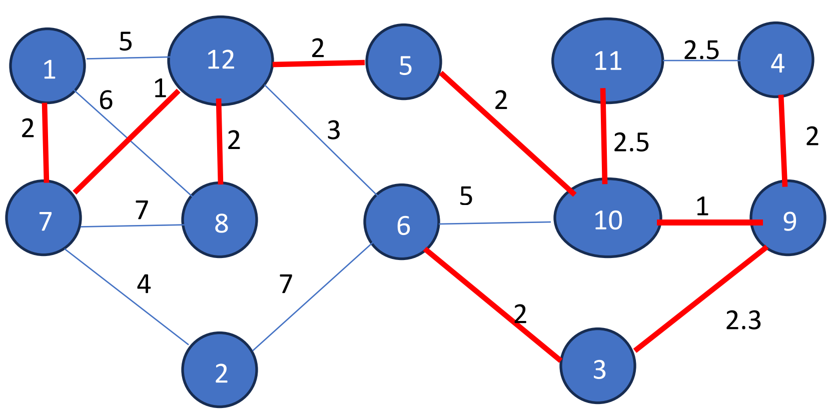
Pick (6,3) = No cycle detected - weight = 2



Pick(3,9) = No cycle detected – weight = 2.3

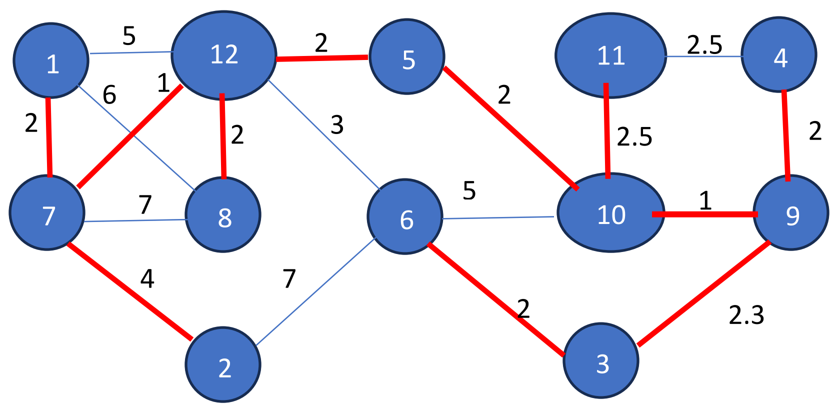


Pick edge (10,11)= No cycle detected – weight = 2.5

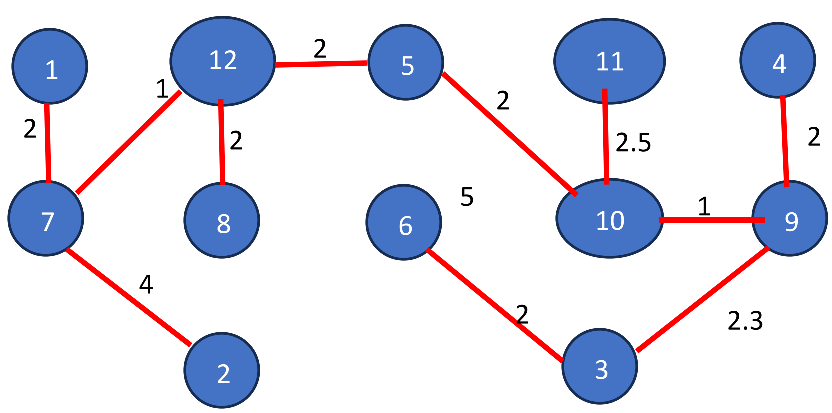


No we can't pick edge (11,4) as 11 and 4 are already nodes of same tree. Hence we discard it

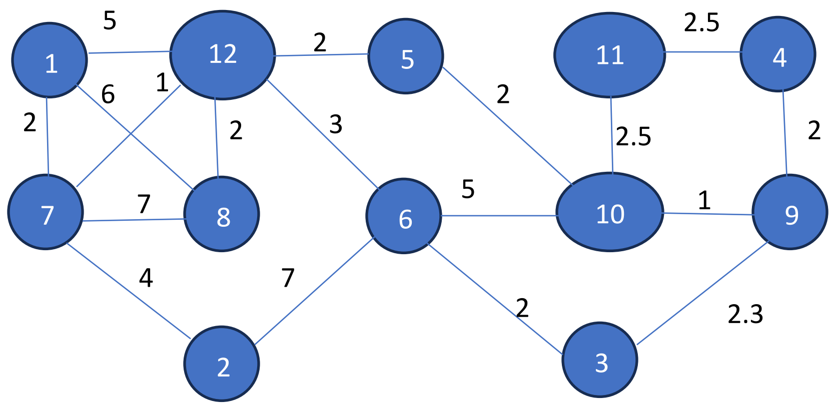
Next edge is (12,6) but this forms a cycle and both belongs to the same tree. Hence, we discard it
Next pick edge (7,2), both are from different trees so , we can proceed to connect them



Now, we have touched every nodes and there exists a path for every node from any give node with minimum weightage. Our final MST looks like



b. Using the greedy Dijkstra's single-source shortest-path algorithm, find the distances between node 1 and all the other nodes. Show the values of the DIST array at every step.



Given the graph, starting from node 1, we need to construct shortest path to each other node from source node 1.

Initial distance matrix is for source node 1

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞

And Y ={1}

From 1 now we reach nodes 7 , 12 and 8 with weights 2,5,6 respectively. Dist matrix is
Dist[7] = min(∞, W[1,7]) = min(∞, 2) = 2
Dist[12] = min(∞,min(Dist[1,7]+W[7,12], W[1,12]) = min(∞,min(5, 2+1)) = 3
Dist[8] = min(∞,min(Dist [1,12]+W[12,8], W[1,8], Dist[1,7]+W[7,8]) = min(∞,min(5+2, 6, 3+2) = 5

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	∞	∞	∞	∞	∞	2	5	∞	∞	∞	3

And Y = {1,7,8,12}

From 1 now we reach

From 1 now we reach 2
Dist[2] = = min(∞,2+W[7,2]) = = min(∞, 2+4) = 6

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	∞	∞	∞	∞	2	5	∞	∞	∞	3

And Y = {1,2,7,8,12}

From 1 now we reach both 5 and 6

Dist[5] = min(∞,5+W[12,5]) = min(∞,5) =5
Dist[6] = min(∞,5+W[12,6]) = min(∞,6) = 6

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	∞	∞	5	6	2	5	∞	∞	∞	3

And Y ={1,2,5,6,7,8,12}

From 1 now we reach 10. To reach 10 we can reach from E[1,5] and E[5,10] or we can reach from E[1,6] and E[6,10]. So we take minimum of both.
Dist[10] = min(∞,min(Dist[1,5]+W[5,10], Dist[1,6]+5)) = min(∞,min(7,11)) = 7

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	∞	∞	5	6	2	5	∞	7	∞	3

And Y= Y ={1,2,5,6,7,8,10,12}

From 1 now we reach 3.
Dist[3] = min(∞,Dist[1,6]+W[6,3]) = min(∞,6+2) = 8

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	8	∞	5	6	2	5	∞	7	∞	3

And Y = ={1,2,3,5,6,7,8,10,12}

From 1 now we reach 9. We can reach by E[1,10] + E[10,9] or E[1,3] +E[3,9]
Dist[9] = min(∞, min(Dist[1,10]+W[10,9] , Dist[1,3]+W[3,9])= min(∞,min(7+1 , 8+2.3))= 8

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	8	∞	5	6	2	5	8	7	∞	3

And Y = {1,2,3,5,6,7,8,9,10,11,12}

From 1 now we reach 4 and 11.
Dist[4] = min(∞,Dist[1,9]+W[9,4]) = min(∞,8+2) = 10
Dist[11] = min(∞, min(Dist[1,10]+W[10,11] , Dist[1,4]+W[4,11]) = min(∞,min(7+2.5, 10+2)) = 9.5

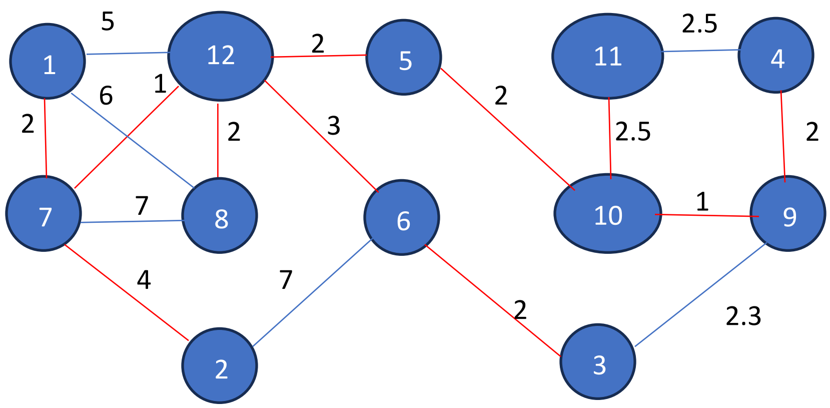
This is our final distance array from source node 1 to every other node in the given graph

i	1	2	3	4	5	6	7	8	9	10	11	12
Dist[i]	0	6	8	10	5	6	2	5	8	7	9.5	3

C. Show the actual shortest paths of part (b). Note that these paths together form a spanning tree of G. Is this tree a minimum spanning tree ?

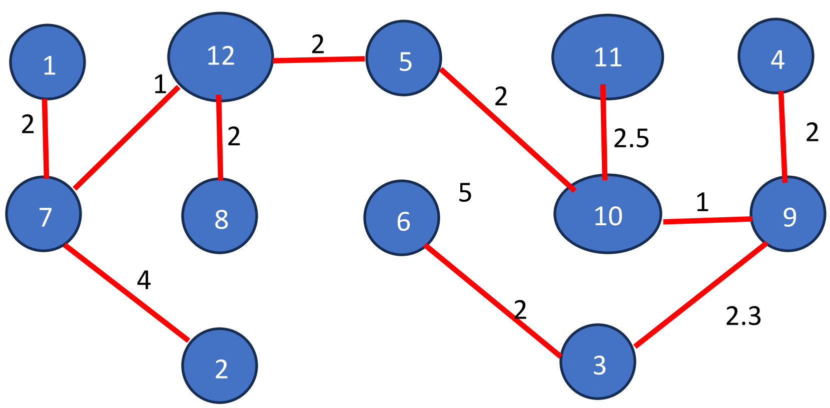
Solution :

The resultant tree for above array is as follows



The total weight of the MST is 2+4+1+2+3+2+2+2+2.5+1+2 = 23.5

But the actual shortest paths for this tree is



The total weight of the MST is 2+4+1+2+2+2+2.5+1+2+2.3+2 = **22.8**

No, the tree formed in 5b is not a minimum spanning tree.

Problem 2:

Let $AA[1: nn]$ be an array of real numbers. The *distance* between two elements $A[i]$ and $A[j]$ is the absolute value $|A[i] - A[j]|$. Two elements $A[i]$ and $A[j]$ of AA are called neighbors if $i = j - 1$ or $i = j + 1$. Two elements of A are called *the closest neighbors* if they are neighbors and the distance between them is the smallest distance between any two neighbors of A.

a. Write a divide-and-conquer algorithm that takes $A[1: n]$ and returns the following output: The minimum of A, the maximum of A, and the indices of the two closest neighbors in A. Note that there could be multiple closest neighbors; break the tie anyway you want.

Solution : **Procedure just to call our function**

```
procedure Caller()
 {
int leftMinNeighbour = MaximumIntegerValue, rightMinNeighbour = MaximumIntegerValue;
int[] arr = new int[n]; // our array input assuming it has some initial values
int globalMax = MinimumIntegerValue; //globalMax variable to find Max element of A
int globalMin = MaximumIntegerValue; //globalMin variable to find Max element of A


     CalcCloseNeighbour(0,n, leftMinNeighbour, rightMinNeighbour, globalMax,
globalMin,arr); //Call to our function
     if (leftMinNeighbour < rightMinNeighbour)
         Print(leftMinNeighbour);
     else
         Print(rightMinNeighbour);

     Print("Global Max" + globalMax);
     Print("Global Min" + globalMin);
 }
```

Start of our actual algorithm below:

```
func CalcCloseNeighbour(int start, int end int, inout leftmin, inout rightmin,
                        out globalmax, out globalmin, int[] arr)
{
    if (start == end || end<start || start>end) //terminating condition
        return;

    int leftDiff, rightDiff;
    int mid = (start + end) / 2; //calculate mid index value of the input array

    if (globalMin >= arr[mid]) //compare globalMin with mid value to find Min of A
        globalMin = arr[mid];
    if(globalMax <= arr[mid]) //compare globalMax with mid value to find Max of A
        globalMax = arr[mid];


    if (mid >= 1 && mid < arr.Length)
    {
        leftDiff = |arr[mid] - arr[mid - 1]| //store temp difference between A[mid]
and A[mid-1]
```

```
        rightDiff = |arr[mid + 1] - arr[mid]|//store temp difference between A[mid]
and A[mid+1]


        if (leftMinNeighbour > leftDiff) //if left temp value is less than globally
declared left minimum, mark globally declared left minimum to left temp value
            leftMinNeighbour = leftDiff;

        if (rightMinNeighbour > rightDiff) //if right temp value is less than globally
declared right minimum, mark globally declared right minimum to right temp value
            rightMinNeighbour = rightDiff;
    }


    CalcCloseNeighbour(start, mid - 1, leftDiff, rightDiff, globalMax,
globalMin,arr); //recursive call on left part of array input A[1:mid-1]
    CalcCloseNeighbour(mid + 1, end, leftDiff, rightDiff, globalMax, globalMin,arr
);
//recursive call on left part of array input A[mid+1:n]

    return;
}
```

b.Analyze the time complexity of your algorithm.
Note that for full credit, your algorithm should take less than $(n \log n)$.
Solution :
Lets say for example we have the input 5,71,18,9,2,10,35,4. Total length = 8

For the first iteration
Mid = 0+8/2 = 4 , A[4] = 9
We store left min = |A[4]-A[3]| = 9, We Store right min = |A[5]-A[4]| = 7

5	71	18	9	2	10	35	4

r

Now we recursively call function on A[1: (r-1)] and A[(r+1):n]

Time taken to compare and store minimum values = c
For the next recursive call we take $\frac{n}{2} - 1$ input
It means we calculate T(n) = T(r-1) + T(n-r) + cn (c*n for n comparisions)
So, From the lecture notes 4, it follows the same pattern as quick sort.
Hence the algorithm takes $\boldsymbol{O(nlogn)}$

We always take **r=always mid element** and not any arbitrary element. So the average, worst, best case could be $\boldsymbol{O(nlogn)}$