Name : Omkar Dixit
GWID : G42396299

Home Work II

1 a.

Min Heap (As constructed in HWI)

Input array :

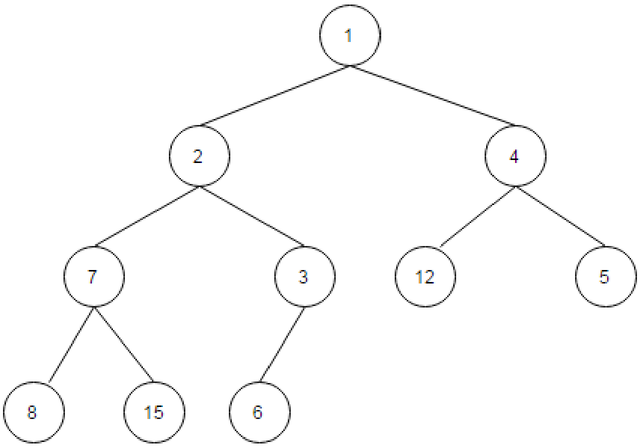
1	2	4	7	3	12	5	8	15	6

Output array :

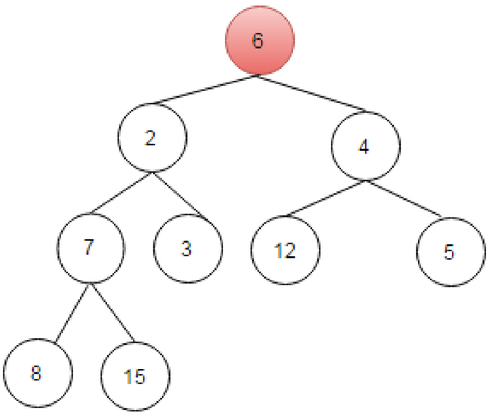
Null	Null	Null	Null	Null	Null	Null	Null	Null	Null

Assumption :

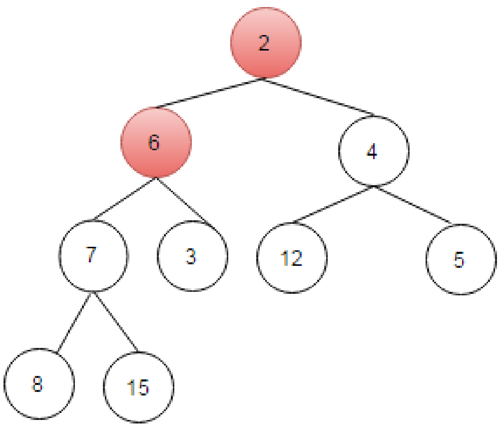
While replacing root node, last element in array will take it's place as last element in array guarantees to be leaf node and it's easily accessible.



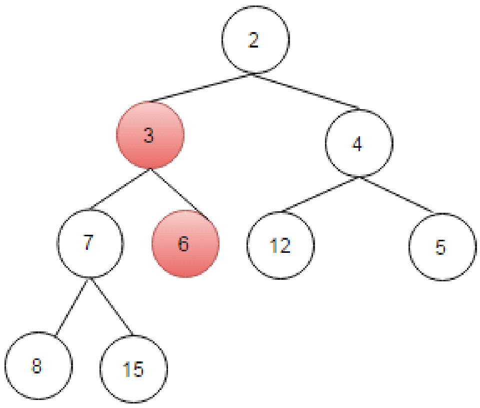
Step I : Remove minimum (Root off node) from input heap and add it to output array.



Input	6	2	4	7	3	12	5	8	15	-
Output	1	-	-	-	-	-	-	-		-

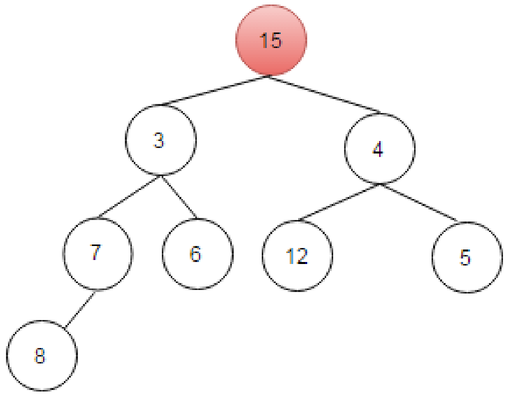


Input	2	6	4	7	3	12	5	8	15	-
Output	1	-	-	-	-	-	-	-	-	-

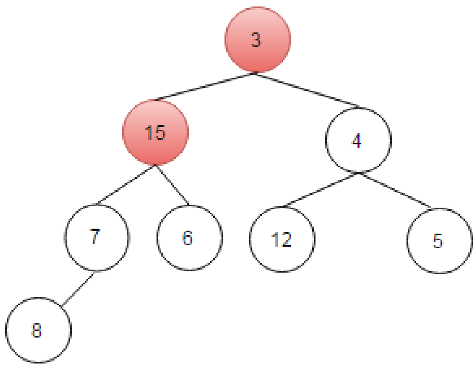


Input	2	3	4	7	6	12	5	8	15	-
Output	1	-	-	-	-	-	-	-	-	-

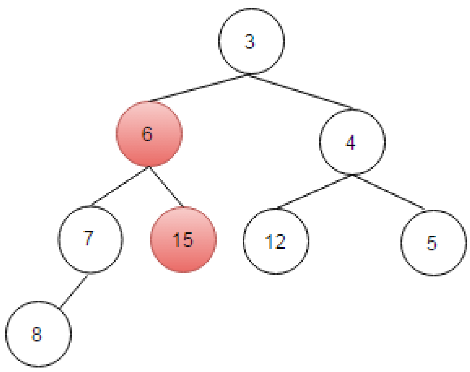
Step II : Remove minimum (Root off node) from input heap and add it to output array.



Input	15	3	4	7	6	12	5	8	-	
Output	1	2	-	-	-	-	-	-	-	-

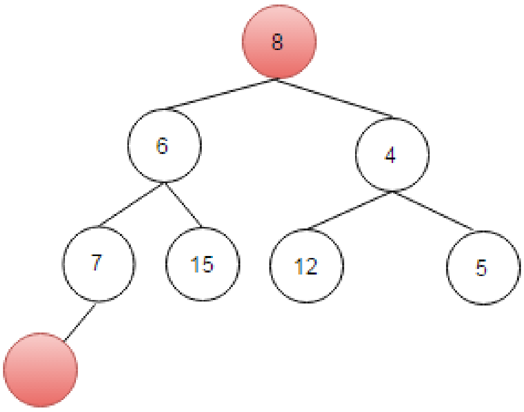


Input	3	15	4	7	6	12	5	8	-	-
Output	1	2	-	-	-	-	-	-	-	-

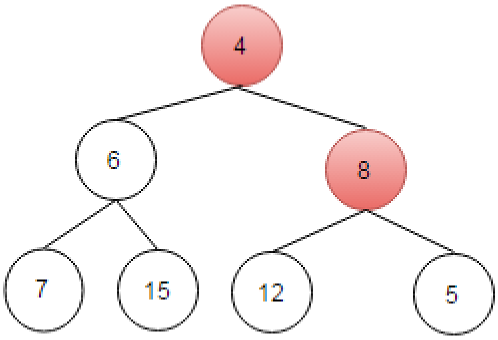


Input	3	6	4	7	15	12	5	8	-	-
Output	1	2	-	-	-	-	-	-	-	-

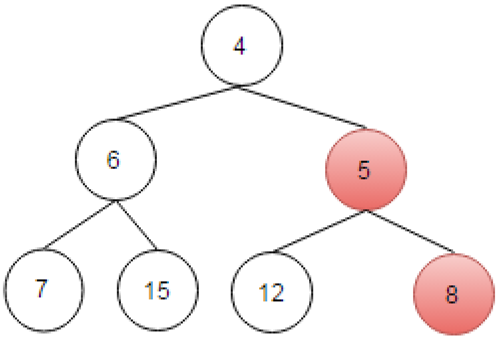
Step III: remove 3



Input	8	6	4	7	15	12	5	-	-	-
op	1	2	3	-	-	-	-	-	-	-

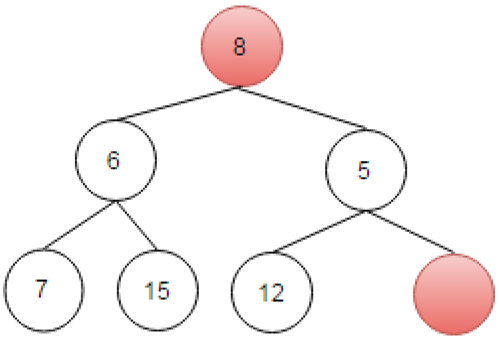


Input	4	6	8	7	15	12	5	-	-	-
Output	1	2	3	-	-	-	-	-	-	-

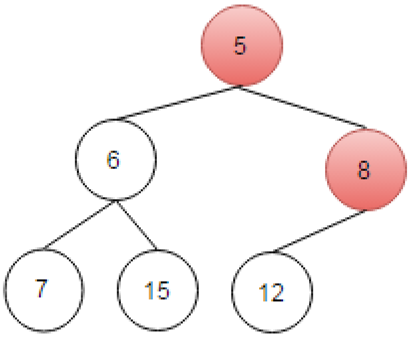


Input	4	6	5	7	15	12	8	-	-	-
op	1	2	3	-	-	-	-	-	-	-

Step IV : Remove 4

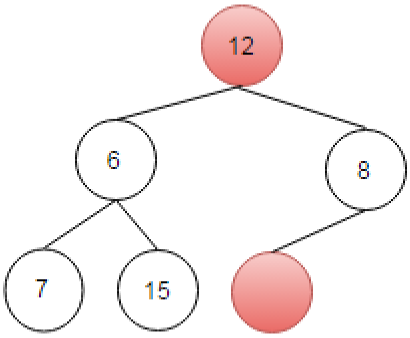


Input	8	6	5	7	15	12	-	-	-	-
op	1	2	3	4	-	-	-	-	-	-

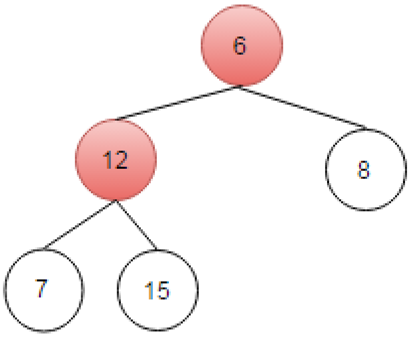


Input	5	6	8	7	15	12	-	-	-	-
op	1	2	3	4	-	-	-	-	-	-

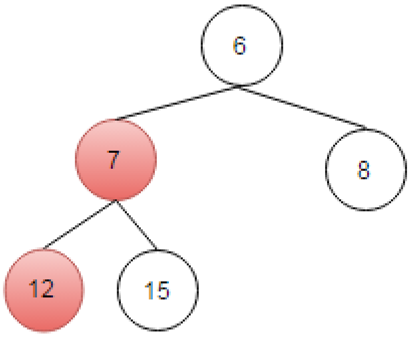
Step V : Remove 5.



Input	12	6	8	7	15	-	-	-	-	-
op	1	2	3	4	5	-	-	-	-	-

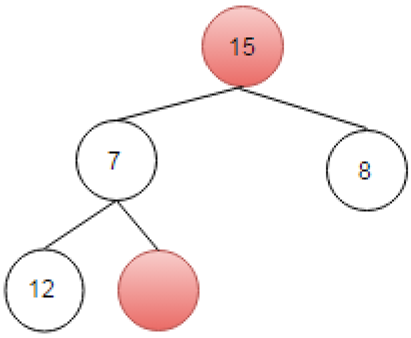


Input	6	12	8	7	15	-	-	-	-	-
op	1	2	3	4	5	-	-	-	-	-

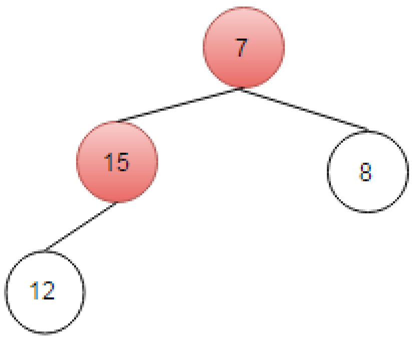


Input	6	7	8	12	15	-	-	-	-	-
op	1	2	3	4	5	-	-	-	-	-

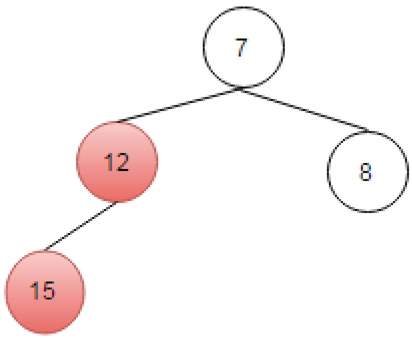
Step VI : Remove 6



Input	15	7	8	12	-	-	-	-	-	-
op	1	2	3	4	5	6	-	-	-	-

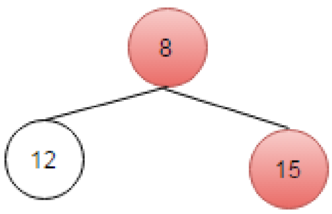


Input	7	15	8	12	-	-	-	-	-	-
op	1	2	3	4	5	6	-	-	-	-

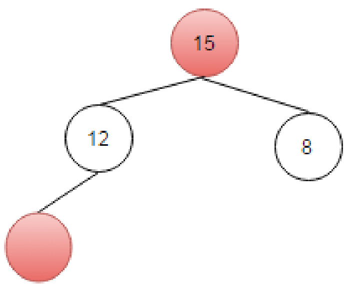


Input	7	12	8	15	-	-	-	-	-	-
op	1	2	3	4	5	6	-	-	-	-

Step VII : Remove 7

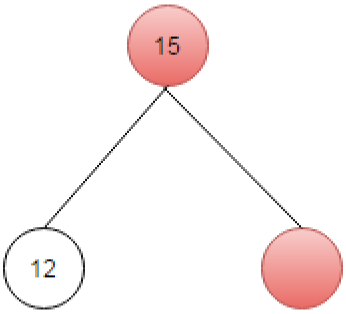


Input	15	12	8	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	-	-	-

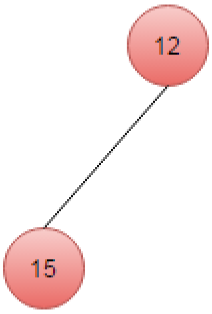


Input	8	12	15	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	-	-	-

Step VIII : Remove 8

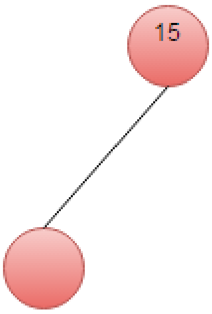


Input	15	12	-	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	8	-	-



Input	12	15	-	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	8	-	-

Step IX : Remove 12



Input	15	-	-	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	8	12	-

Step X : Remove 15

Empty heap

Input	-	-	-	-	-	-	-	-	-	-
op	1	2	3	4	5	6	7	8	12	15

1b :

Step I:

$\boxed{4}$ 6 3 7 1 12 5 8 15 2`

Swap

$\boxed{4}$ 2 3 7 1 12 5 8 15 6

$\boxed{4}$ 2 3 7 1 12 5 8 15 6

Swap

$\boxed{4}$ 2 3 1 7 12 5 8 15 6

$\boxed{4}$ 2 3 1 7 12 5 8 15 6

Swap

1 2 3 $\boxed{4}$ 7 12 5 8 15 6

After Step I :

Output	-	-	-	4	-	-	-	-	-	-
Left	1	2	3							
Right	7	12	5	8	15	6				

Step II : Solve left

$\boxed{1}$ 2 3

After Step II :

Output	1	-	-	4	-	-	-	-	-	-
Left										
Right	2	3								

Step III: Solve right of step II :

$\boxed{2}$ 3

After Step III :

Output	1	2	-	4	-	-	-	-	-	-
Left										
Right	3									

Step IV :

3

After Step IV :

Output	1	2	3	4	-	-	-	-	-	-
Left										
Right										

Step V: Solve right of output of step 1

$\boxed{7}$ 12 5 8 15 6

Swap

$\boxed{7}$ 6 5 8 15 12

$\boxed{7}$ 6 5 8 15 12

Swap

5 6 $\boxed{7}$ 8 15 12

After Step V :

Output	1	2	3	4	-	-	7	-	-	-	
Left	5	6									
Right	8	15	12								

Step VI : Solve left of V,

$\boxed{5}$ 6

After Step VI :

Output	1	2	3	4	5	-	7	-	-	-	
Left											
Right	6										

Step VII : Solve right of VI,

$\boxed{6}$

After Step VI :

Output	1	2	3	4	5	6	7	-	-	-	
Left											
Right											

Step VIII : Solve right of V,

$\boxed{8}$ 15 12

After Step VI :

Output	1	2	3	4	5	6	7	8	-	-	
Left											
Right	15	12									

Step IX : Solve right of VIII,

$\boxed{15}$ 12

Swap,

12 $\boxed{15}$

After Step VI :

Output	1	2	3	4	5	6	7	8	-	15
Left	12									
Right										

Step X: Solve left of IX,

$\boxed{12}$

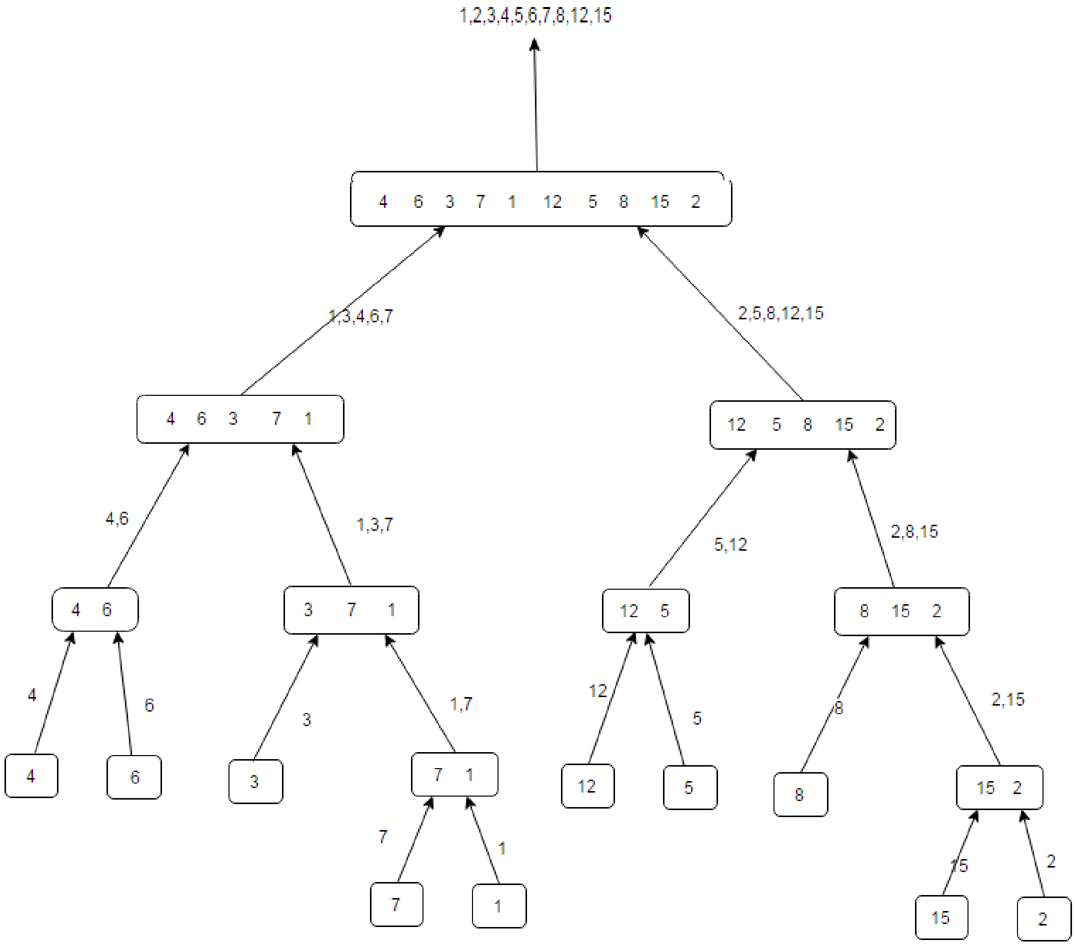
After Step VI :

Output	1	2	3	4	5	6	7	8	12	15
Left										
Right										

Output of array after quick sort :

1	2	3	4	5	6	7	8	12	15

1c. Output = 1,2,3,4,5,6,7,8,12,15



Assumption: If input array is of odd length, left array = (length of array -1) /2
Right array= (length of array +1) /2

1d. Using insert sort. Output array after each step (inserting each element) has been shown :

Input	4	6	3	7	1	12	5	8	15	2
Step 1	4	6	3	7	1	12	5	8	15	2
Step 2	4	6	3	7	1	12	5	8	15	2
Step 3.1	4	6	3	7	1	12	5	8	15	2
Step 3.1	4	3	6	7	1	12	5	8	15	2
Step 3	3	4	6	7	1	12	5	8	15	2
Step 4	3	4	6	7	1	12	5	8	15	2
Step 5	3	4	6	7	1	12	5	8	15	2
Step 5.1	3	4	6	1	7	12	5	8	15	2
Step 5.2	3	4	1	6	7	12	5	8	15	2
Step 5.3	3	1	4	6	7	12	5	8	15	2
Step 5	1	3	4	6	7	12	5	8	15	2
Step 6	1	3	4	6	7	12	5	8	15	2
Step 7.1	1	3	4	6	7	12	5	8	15	2
Step 7.2	1	3	4	6	7	5	12	8	15	2
Step 7.3	1	3	4	6	5	7	12	8	15	2
Step 7	1	3	4	5	6	7	12	8	15	2
Step 8.1	1	3	4	5	6	7	12	8	15	2
Step 8	1	3	4	5	6	7	8	12	15	2
Step 9	1	3	4	5	6	7	8	12	15	2
Step 10.1	1	3	4	5	6	7	8	12	15	2
Step 10.2	1	3	4	5	6	7	8	12	2	15
Step 10.3	1	3	4	5	6	7	8	2	12	15
Step 10.4	1	3	4	5	6	7	2	8	12	15
Step 10.5	1	3	4	5	6	2	7	8	12	15
Step 10.6	1	3	4	5	2	6	7	8	12	15
Step 10.7	1	3	4	2	5	6	7	8	12	15
Step 10.8	1	3	2	4	5	6	7	8	12	15
Step 10(OP)	1	2	3	4	5	6	7	8	12	15

1e. Input array and output array after each step has been shown :

Input	4	6	3	7	1	12	5	8	15	2
Step 1	4	6	3	7		12	5	8	15	2
Step 2	4	6	3	7		12	5	8	15	
Step 3	4	6		7		12	5	8	15	
Step 4		6		7		12	5	8	15	
Step 5		6		7		12		8	15	
Step 6				7		12		8	15	
Step 7						12		8	15	
Step 8						12			15	
Step 9									15	
ep 10										

Output										
Step 1	1									
Step 2	1	2								
Step 3	1	2	3							
Step 4	1	2	3	4						
Step 5	1	2	3	4	5					
Step 6	1	2	3	4	5	6				
Step 7	1	2	3	4	5	6	7			
Step 8	1	2	3	4	5	6	7	8		
Step 9	1	2	3	4	5	6	7	8	12	
Step 10	1	2	3	4	5	6	7	8	12	15

2a.

```
function initialize()
begin
        Integer k:=1;             //k is used as index to store output
        Datatype output[];        //output is array of data type containing one character and one integer
End initialize;

function datatype [] store_output (array1[],array2[])
begin

        Integer count:=0,i;
        Integer ch=array1[1];
        For i=1 to array1.length do        // traverse array1
                If array1[i] == ch then
                        count:= count + 1;
                else                        // we have encountered new char, so store output
                        output[k].char1=ch;
                        output[k].count=count;
                        ch=array1[i];   // reinitialize ch
                        k++;
                endif;
        endfor;
        For i=1 to array1.length do        // traverse array2
                If array1[i] == ch then
                        count:= count + 1;
                else                        // we have encountered new char, so store output
                        output[k].char1=ch;
                        output[k].count=count;
                        ch=array1[i];   // reinitialize ch
                        k++;
                endif;
                if (i = array1.length) then  //this is used to store output for last unique elements
                        output[k].char1=ch;
                        output[k].count=count;
                endif;
        endfor;
        return output;
end store_output;

function find_count(input array)
begin
        int p,q,r;
        p=1;
        q=array.length;
        if q=1 then                        //Base case
                output[k].count=count;
                output[k].char1=array[1];
                return 1;
        endif
        r=(p + q)/2
```

```
        arrayleft=array[1:r];                       //divide array into half
        arrayright=array[r+1:q];                    //divide array into half
        find_count(arrayleft);                      //recursive call for left
        find_count(arrayright);                     //recursive call for right
        output=only_count(arrayleft,arrayright)             //conquer and combine output.
        Return output;
end find_count;

function main()
begin
        initialize();
        character array1=a a a a c c c c c b b a d d d;
        find_count(array1);
endmain;
```

2b. Time complexity:

T(n) = Tinit + T(n/2) + T(n/2) +Tstore_op
Tinit is constant time thus, Tinit = O(1)
Tstore_op is traversing array once thus Tstore_output= c1n/2=cn.
Thus,

```
        T(n)            =       2T(n/2)         +       cn
        2T(n/2)         =       4T(n/4)         +       cn
        4T(n/4)         =       8T(n/8)         +       cn
                        .
                        .
                        .
                        .
                        .
2^k-1T(n/2&k-1)         =       2^kT(n/2^k)     +       cn
-----------------------------------------------------------------------------------------------Add and cancel
        T(n)            =       2^kT(n/2^k)     +       (k-1)cn

Let     2^k             =               n  =>   k=logn
        T(n)            =       nT(1)           +       (k-1)cn
        T(n)            =       kcn --------(approx, T(1) is nigligible)
        T(n)            =       cnlogn`
```

Thus,

T(n)	=	o(nlogn)

Note, this problem can be solved using linear search instead of divide and conquer to get time complexity of o(n).

3 a.

```
Procedure init()
begin
Dataype point contains x-coordinate as integer, y-coordinate as integer
Point input_points[1:n];  // Input array
Point output_points[1:n];  // output array
end

Procedure mergesort(input input_points[1:n],i,j,output output_points)
Begin
  points C[1:n];
  If i=j then B[i] = A[i]; Return; endif      //stores x and y coordinates of point)
  Mergesort (input_points,i,(i+j)/2;C); /* sorts the first half*/
  Mergesort(input_points,(i+j)/2 +1,j;C); /* sorts the second half*/
  Merge(C,i,j;B); /* merges the two sorted halves *
                    * into a single sorted list */
End

Procedure Merge(input: C i,j; output: B)
begin
        int k=(i+j)/2;
        int u,v,w;
        u=i;
        v=k+1;
        w=u;
        while  (u <= k and v <= j) do
                if C[u].x-cordinate <= C[v] .x-coordinate then
                        B[w]=C[u]; u++;w++;
                else
                        B[w]=C[v]; v++;w++;
                endif
        endwhile
        If u > k then
                Put C[v:j] in B[w:j];
        Elseif v>j
                Put C[u:k] in B[w:j];
        endif
end

function main() // main program starts here
begin
        init();
        mergesort(input_points,1,n,output_points);   // to sort according to x-cordinates
        initialize; /// using same function of 3a.
        Output=find_count(output_points);  // this is same function of 3a/It will store distinct x
                                           // coordinates and points having same coordinates.
end
```

3b.

```
// this function will return desired x coordinate.
Function integer find_vertical(input input_points)
Begin
        Megesort(input_points);  // sort according to x coordinates. Now output_points contain  .
                                  //sorted array wrt x
        If(n=odd)
        Then
                Return (n+1)/2   //middle element
        Else
                Return n/2     // middle element
End
```

3c.

Time complexity of problem 3a.

T(n)	=	T(sort)	+	T(find_count)
O(n) from sort	=	nlogn		

And as shown in problem 2.b,

O(n) from find_count = O(nlogn)

Thus,

Time complexity for 3a, O(n) = O(nlogn)

Time complexity of problem 3b.

T(n)	=	T(Sort)	+	T(if else)
O(n) for sort	=	o(nlogn)		
And O(if else)	=	O(1).		

Thus,

Time complexity for 3a, O(n) = O(nlogn)

Problem 4 :
Knapsack problem:

Name	A	B	C	D	E	F	G	H
Price	8	7	3	10	4	15	4	14
Weight	14	12	15	30	8	5	10	14
price/weight	0.571429	0.583333333	0.2	0.3333333	0.5	3	0.4	1
Rank	4	3	8	7	5	1	6	2

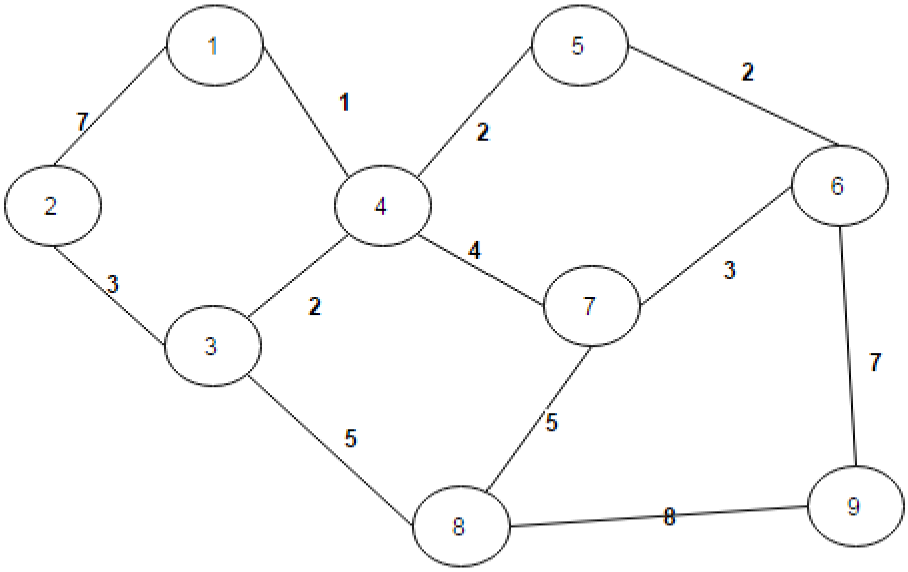
Best solution is defined as maximum price/weight.

Applying greedy algorithm,

	Weight	Weight remaining (42-weight)	Unit price of item taken	total price of item	Total price taken
Init	0	42	0	0	0
Take F and remove it from input	5	37	3	15	15
Take H and remove it from input	14	23	1	14	29
Take B and remove it from input	12	11	0.583333	7	36
Take A and remove it from input	11	0	0.571429	6.285719	**42.2857**

Output set will contain :
5 Kg of F, 14kgs of H, 12 kgs of B and 11 kgs of A and maximum price will be 42.2857

Problem 5 a: Graph mentioned can be represented as follows :

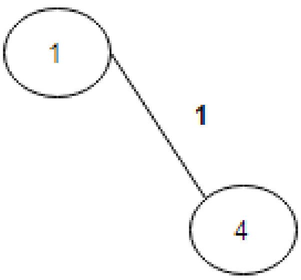


Minimum Spanning tree :

Init :

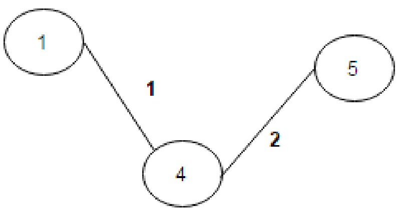
Number of Edges in spanning tree	0											
edges in spanning tree												
weights in spanning tree												
weights remaining in graph	1	2	2	2	3	3	4	5	5	7	7	8

Step I : select minimum weight (1) ,edge 1-4



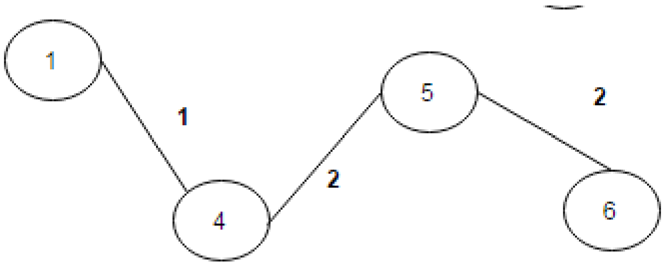
Number of Edges in spanning tree	1											
edges in spanning tree	1,4											
weights in spanning tree	1											
weights remaining in graph		2	2	2	3	3	4	5	5	7	7	8

Step 2 : select 4-5



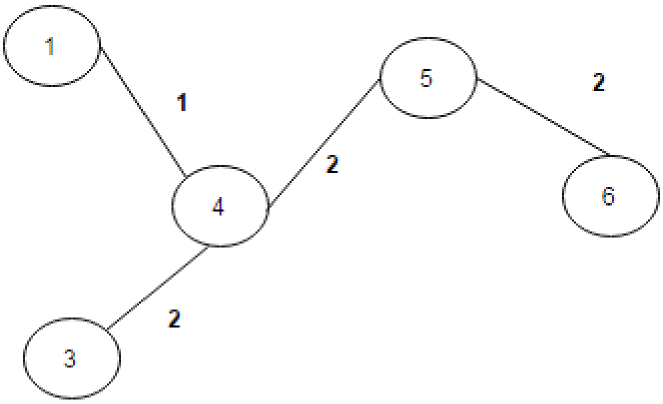
Number of Edges in spanning tree	2										
edges in spanning tree	1,4	2,5									
weights in spanning tree	1	2									
weights remaining in graph			2	2	3	3	4	5	5	7	7	8

Step 3: select 5-6



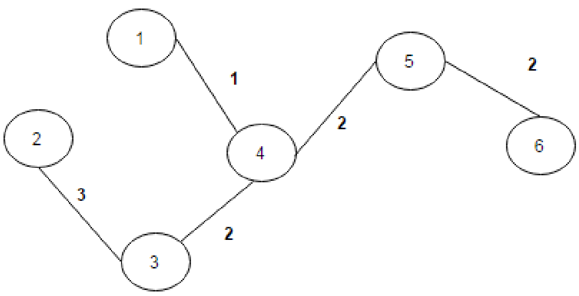
Number of Edges in spanning tree	3											
edges in spanning tree	1,4	2,5	5,6									
weights in spanning tree	1	2	2									
weights remaining in graph				2	3	3	4	5	5	7	7	8

Step 4 : select 3-4



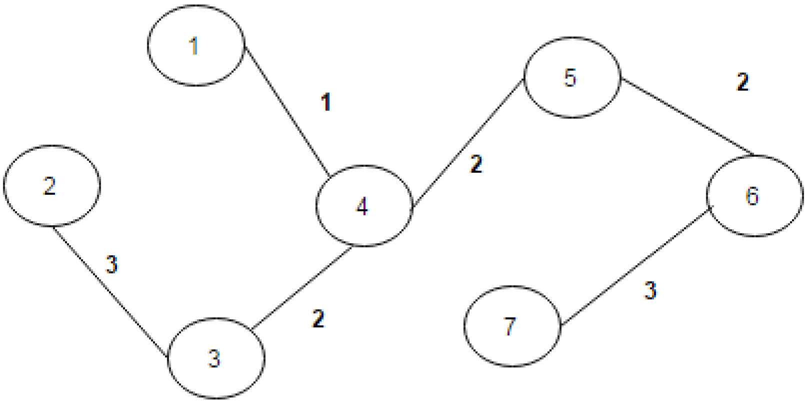
Number of Edges in spanning tree	4										
edges in spanning tree	1,4	2,5	5,6	3,4							
weights in spanning tree	1	2	2	2							
weights remaining in graph					3	3	4	5	5	7	7	8

Step 5 : select 2-3



Number of Edges in spanning tree	5											
edges in spanning tree	1,4	2,5	5,6	3,4	2,3							
weights in spanning tree	1	2	2	2	3							
weights remaining in graph						3	4	5	5	7	7	8

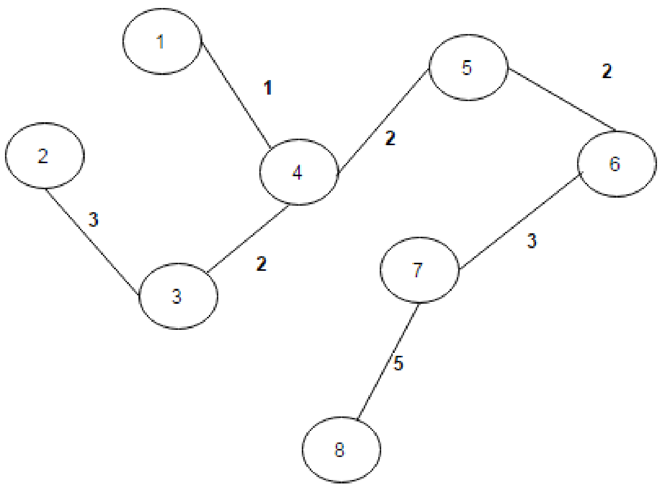
Step 6 : Select 7-6



Step 7 : select 4-7. But it is making cyclic graph thus it will be removed from input set but tree won't be changed.

Number of Edges in spanning tree	6											
edges in spanning tree	1,4	2,5	5,6	3,4	2,3	7,6						
weights in spanning tree	1	2	2	2	3	3						
weights remaining in graph								5	5	7	7	8

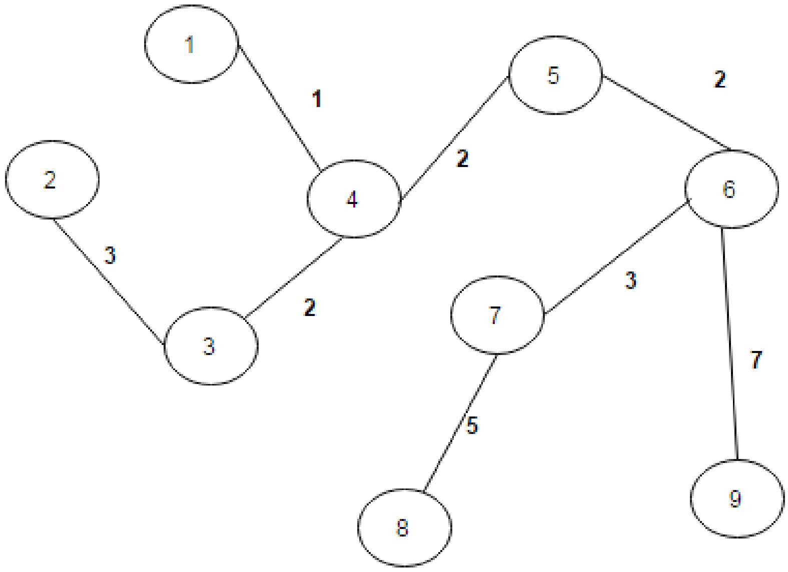
Step 8 : select 7-8



Number of Edges in spanning tree	7											
edges in spanning tree	1,4	2,5	5,6	3,4	2,3	7,6		7,8				
weights in spanning tree	1	2	2	2	3	3		5				
weights remaining in graph									5	7	7	8

Step 9 : select 3-8. But it is making tree cyclic thus it will be removed from input set but tree won't be changed.

Step 10 : select 6-9.

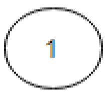


Number of Edges in spanning tree	8										
edges in spanning tree	1,4	2,5	5,6	3,4	2,3	7,6		7,8		6,9		
weights in spanning tree	1	2	2	2	3	3		5		7		
weights remaining in graph											7	8

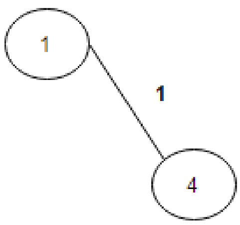
Number of edges = number of nodes -1 . Thus loop will stop and this is the minimum spanning tree.

Weight of spanning tree = 25.

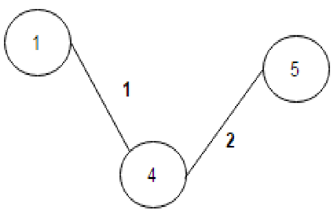
Problem 5b :



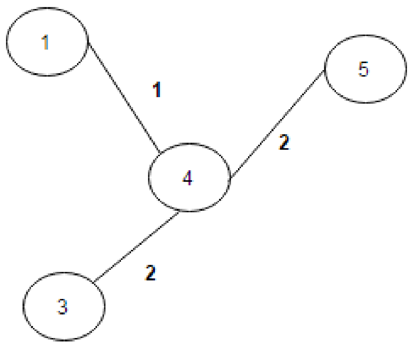
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	7	∞	1	∞	∞	∞	∞	∞



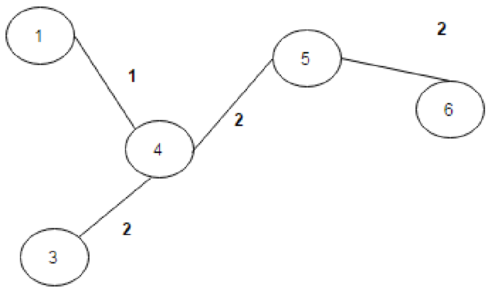
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	7	3	***1***	3	∞	5	∞	∞



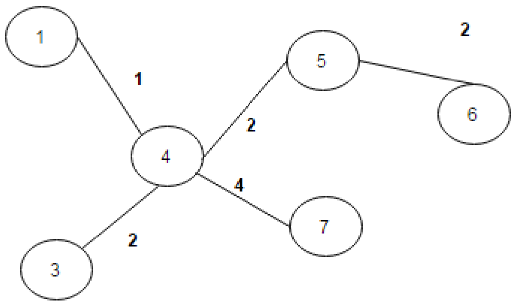
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	7	3	***1***	***3***	5	5	∞	∞



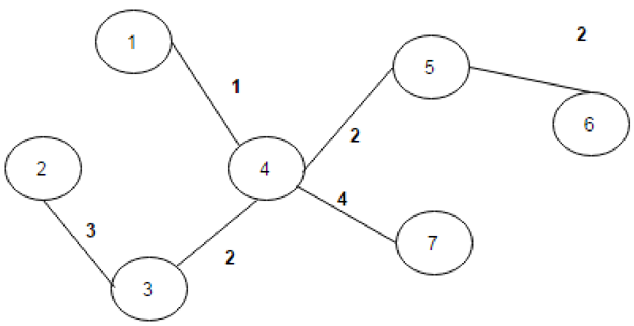
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	6	***3***	***1***	***3***	5	5	8	∞



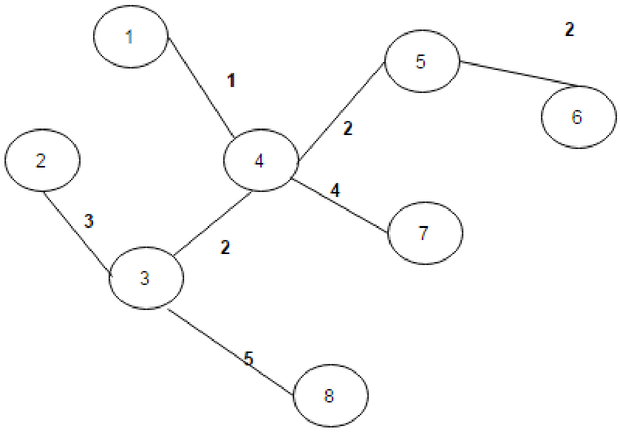
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	6	***3***	***1***	***3***	***5***	5	8	12



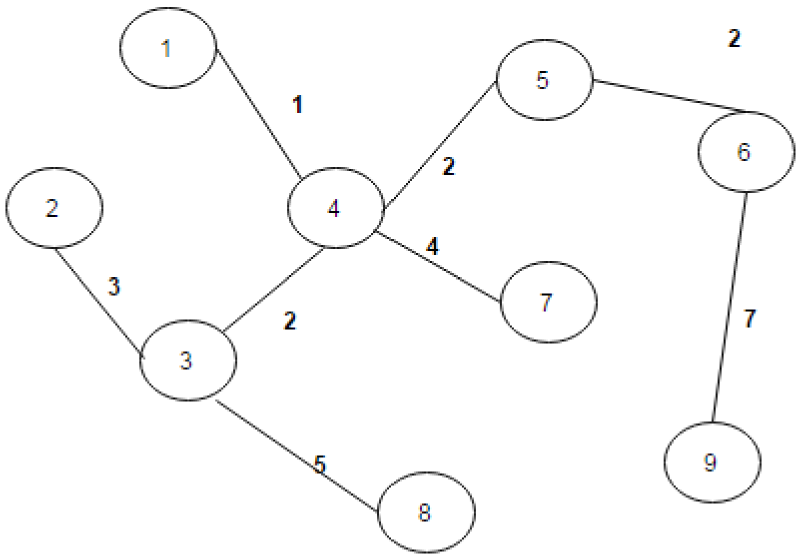
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	6	***3***	***1***	***3***	***5***	***5***	8	12



	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	***6***	***3***	***1***	***3***	***5***	***5***	8	12



	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	***6***	***3***	***1***	***3***	***5***	***5***	***8***	12



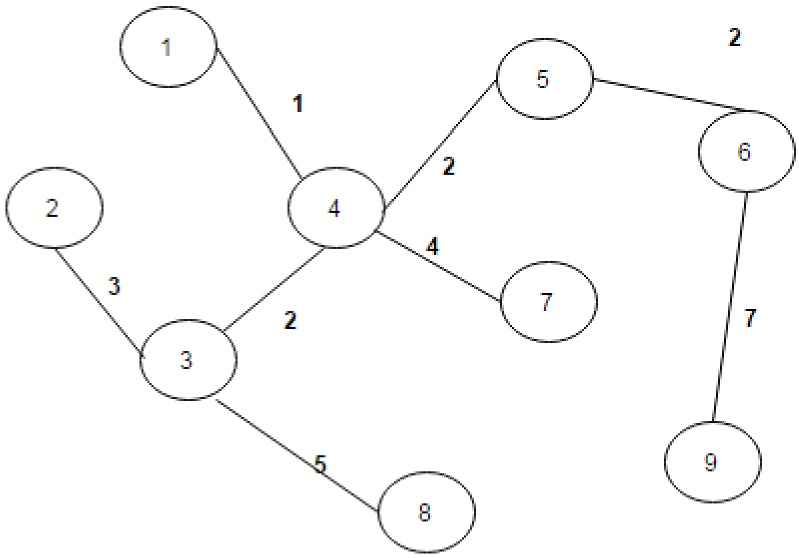
	1	2	3	4	5	6	7	8	9
Dist (1,n)	***0***	***6***	***3***	***1***	***3***	***5***	***5***	***8***	***12***

Finally all elements are traversed, thus Dis(1,n) = Distance (1,n).

Distance of (1.n) can be shown as below :

	1	2	3	4	5	6	7	8	9
Distance(1,n)	0	6	3	1	3	5	5	8	12

5c. After joining shortest path, graph becomes :



As this is acyclic and all nodes are connected, it is spanning tree.

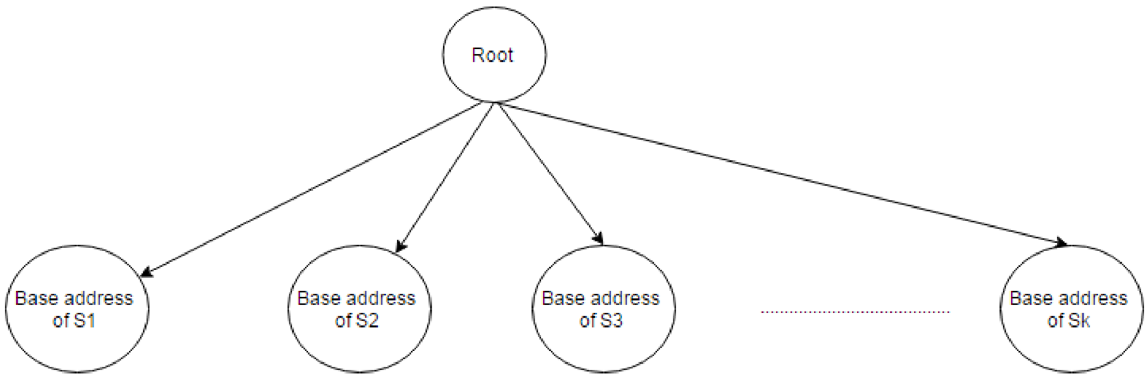
Weight of the spanning tree is 26.

But weight of minimum spanning tree as done in 5a (by Kruskals algorithm) is 25, spanning tree made by joining shortest paths is not necessarily a optimal minimum spanning tree.

6a.

```
Procedure find_maximum_weight(in:S,out:C)
begin
  initialize C as empty set.
  Initialize wsum to zero,
    While i= S1,S2,S3,.......,Sl
         Wi= choose maximum weight of Si.
         If Wi is greater than 0 then
                  Add wi to c
                  Wsum = wsum + we
         End if
         Remove Si from S.
    Endwhile
    Return wsum
end find_maximum_weight
```

Let S1,S2,S3,.....,Sk be simple arrays and S is a tree with root pointing to root base address of all other arrays. E.g :



Finding out best:

Max= $-\infty$

For j =1 to Si.length

Begin

```
If(max  <        Si[j])

Then

        Max    =        S[j]

End if

End for;

Return max;
```

To delete Si from S, we can just delete pointer address. But it will lead to memory leak. Thus, to delete, following procedure will be used.

```
For j=1 to Si.length

Begin

        Delete Si[j];

end for
```

6b.

To find maximum of j elements, it takes O(j).

To delete elements of Sj of length j takes O(j) time.

Thus, time complexity for k sets having n elements,

T(n) = time required to traverse while loop + time required to find maximum weight + time required to remove set heap from main set

= T(find maxS1) + T(delete all elements of S1) + T(find maxS2) + T(delete all elements of S2) +...........................+ T(find maxS1) + T(delete all elements of Sk)

For n elements, Worst case is when all sets have 1 element and one set has all n/k elements.

= k*O(n/k) + k*o(n/k) + O(k)

= O(n)

Time complexity of algorithm = O(n)

6c.

Proof : By induction.

Base case :

Suppose S contains only one sub set ,S1 of length 1 (only positive element)

As per algorithm, it will select the 1 element. As there is no other possible solution, our solution is the optimal solution.

Induction hypothesis :

Let C' is the optimal set of sub array.

If we prove our sub array C = C', our sub array will be optimal.

Suppose that C=C' till selecting items from set S1 till Si-1.

Thus, difference between C' and C can be as follows :

W(C') –W(C) = W(item selected from Si for C') - W(item selected from Si for C)

If we prove W(C') –W(C) <=0, W(C) will be optimal.

Let's assume W(C') – W(C) > 0.

Thus, W(item selected from Si for C') - W(item selected from Si for C) > 0.

Thus, W(item selected from Si for C') > W(item selected from Si for C)

But as per algorithm, we have selected maximum weight from Si.

Thus, W(item selected from Si for C') can not be greater than W(item selected from Si for C)

Thus, W(C') - W(C) < = 0 ie,

W(C') <= W(C).

Hence, C is optimal sub array.