Solution of DSA-28 (CSEDU)

- Sativa

1. Consider the input string: "The quick brown mouse jumps over the lazy cat."

a) Build a Huffman Tree from the input text. [7]

b) Encode the text "bob jumps over the river" using your tree. [2]

c) You have to encode the text "bob runs over the river". However, the character "u" is not in your Huffman tree. How will you modify the tree to assign a code to the letter "u"? [5]

a-2 b-1 c-2 d-0 e-4 f-0 g-0 h-2 i-1 j-1 k-1 l-1 m-2 n-1

o-3 p-1 q-1 r-2 s-2 t-2 u-3 v-1 w-1 x-0 y-1 z-1 T-1 Space-8 .-1

0 d f g x 4

1 b i j k l n p q v w y z T . 14

2 a c h m r s t 7

3 o u 2

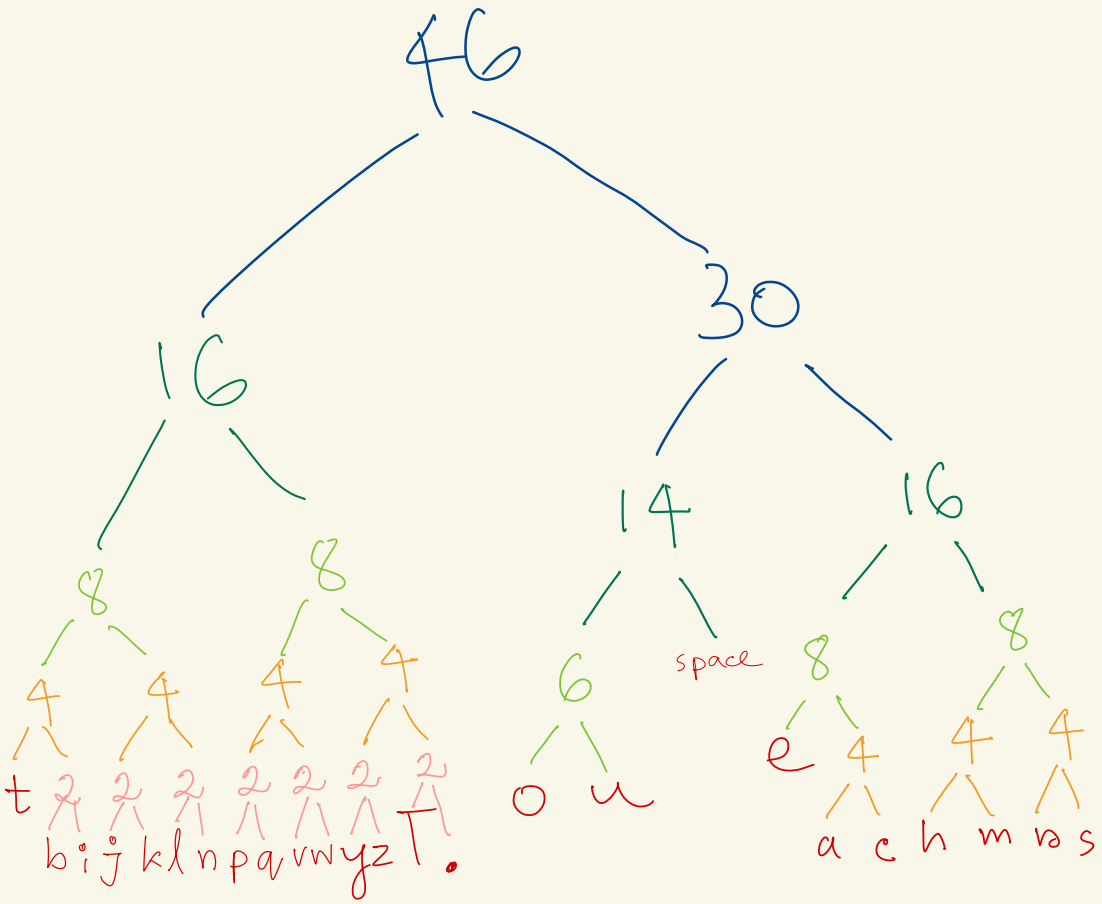
4 e 1

8 space 1

2+3 : 1/29

0.4=0
1.14=14
2.7=14
3.2=6
4.1=4
8.1=8

Length=46

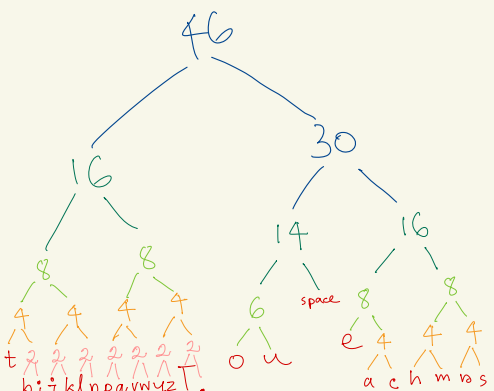


1(b)

bob jumps over the river

b	0 0 0 1 0
o	1 0 0 0
b	0 0 0 1 0
space	1 0 1
j	0 0 1 0 0
u	1 0 0 1
m	1 1 1 0 1
p	0 1 0 0 0
s	1 1 1 1 1
space	1 0 1
o	1 0 0 0
v	0 1 0 1 0
e	1 1 0 0
r	1 1 1 1 0
space	1 0 1
t	0 0 0 0
h	1 1 1 0 0
e	1 1 0 0
space	1 0 1

r	1 1 1 1 0
i	0 0 0 1 1
v	0 1 0 1 0
e	1 1 0 0
r	1 1 1 1 0



Here in Union(a,b), b is parent

2. Consider a disjoint set data structure considering the elements 1-10 (inclusive).

a) Perform the following operations without path compression. You must draw the trees in each step. [4]

a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

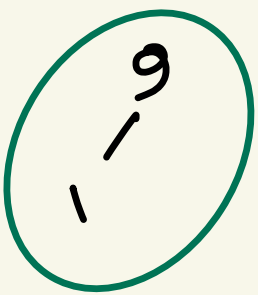
e. union(6,7) f. union(6,8) g. union(9,6)

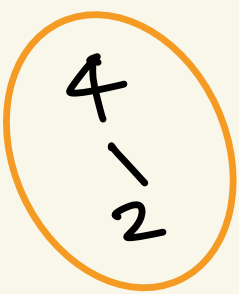
b) Perform the following operations with path compression. You must draw the trees in each step. [5]

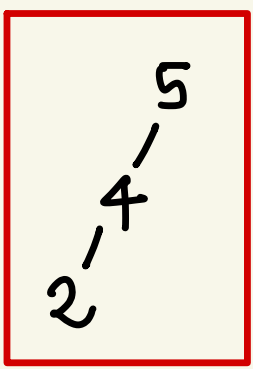
a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

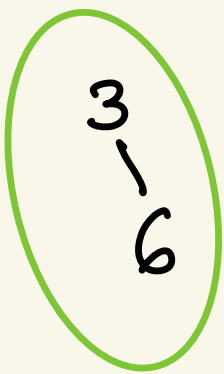
e. union(6,7) f. union(6,8) g. union(9,6)

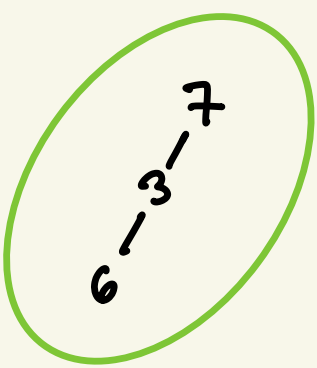
c) Show an example of two trees where union by size and union by rank will result in two different trees. [5]

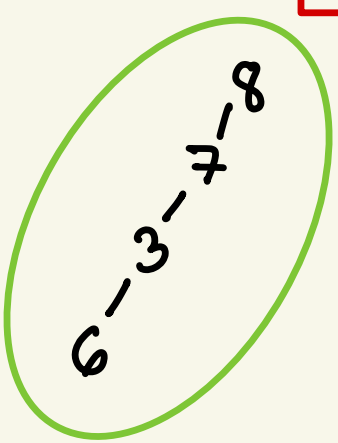


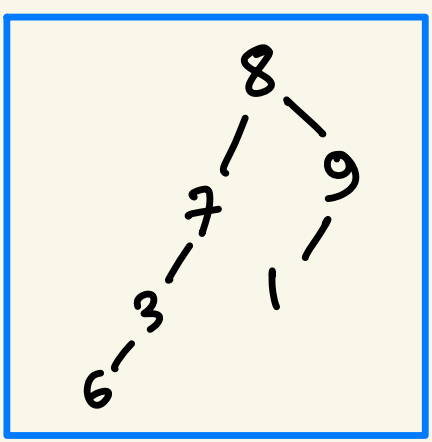













```
//with out path compression
int find_set(int v){
    // by recursion
    if (v == parent[v]) return v;
    return find_set(parent[v]);
}

/*
//with  path compression
int find_set(int v){
    // by recursion
    if (v == parent[v]) return v;
    return parent[v]=find_set(parent[v]);
}
```



```
void union_naive(int a, int b){
    a = find_set(a);
    b = find_set(b);
    if(a==b) return;
    parent[a] = b;
}
```

Here in Union(a,b), b is parent

2. Consider a disjoint set data structure considering the elements 1-10 (inclusive).

a) Perform the following operations without path compression. You must draw the trees in each step. [4]

a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

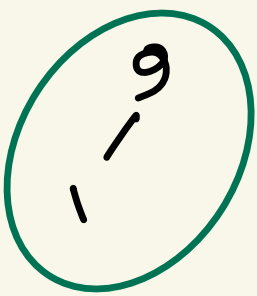
e. union(6,7) f. union(6,8) g. union(9,6)

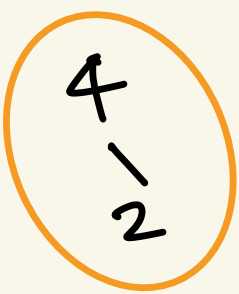
b) Perform the following operations with path compression. You must draw the trees in each step. [5]

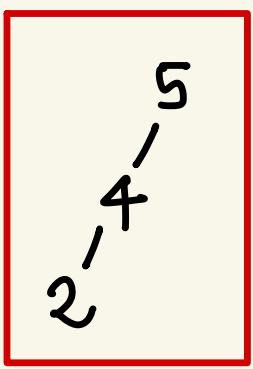
a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

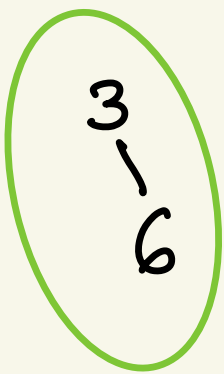
e. union(6,7) f. union(6,8) g. union(9,6)

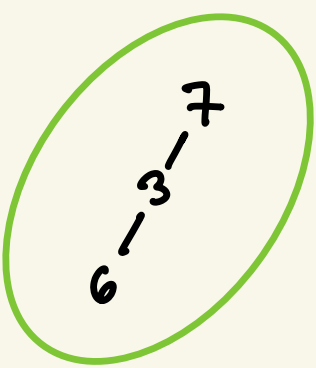
c) Show an example of two trees where union by size and union by rank will result in two different trees. [5]

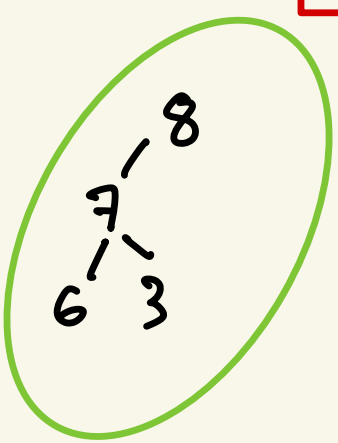


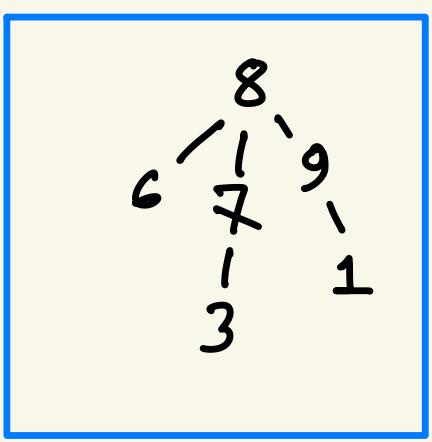













```
//with out path compression
int find_set(int v){
    // by recursion
    if (v == parent[v]) return v;
    return find_set(parent[v]);
}

/*
//with  path compression
int find_set(int v){
    // by recursion
    if (v == parent[v]) return v;
    return parent[v]=find_set(parent[v]);
}
```



```
void union_naive(int a, int b){
    a = find_set(a);
    b = find_set(b);
    if(a==b) return;
    parent[a] = b;
}
```

2. Consider a disjoint set data structure considering the elements 1-10 (inclusive).

a) Perform the following operations without path compression. You must draw the trees in each step. [4]

a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

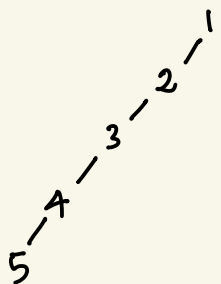
e. union(6,7) f. union(6,8) g. union(9,6)

b) Perform the following operations with path compression. You must draw the trees in each step. [5]

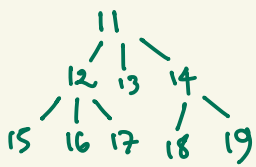
a. union(1,9) b. union(2,4) c. union(4,5) d. union(6,3)

e. union(6,7) f. union(6,8) g. union(9,6)

c) Show an example of two trees where union by size and union by rank will result in two different trees. [5]



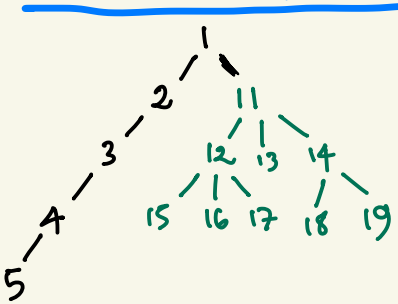
rop = 4
size = 5



rop = 2
size = 9

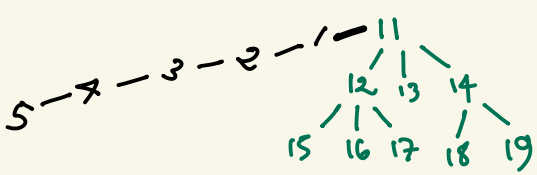
rop ↓ rank of parents

Union by rank



rop = 4
size = 14

Union by size



rop = 5
size = 14

3. a) Design a linked list with the following properties: [6]
   - Insertion of an item in the first or the last position has the complexity of *O(1)*
   - Deletion of an item from the middle position has the Complexity *O(1)*

   b) Write down the pseudocode to find whether a linked list is circular or not. [4]

   c) Design an approach to sort the nodes of a linked list containing integer values without using any external array. [4]

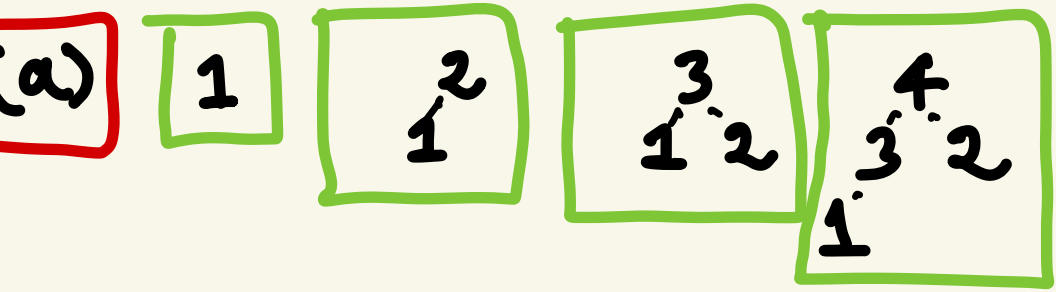
```
#include<iostream>
using namespace std;
#define ll long long int

struct doubly_linked_list{

    ll size=0;
    struct node{
        node *next;
        node *prev;
        ll val;
    };
    node *head=NULL;
    node *tail=NULL;
    node *mid=NULL;

    void insert_first(ll x){
        size++;
        node *a=(node*)malloc(sizeof(node));
        a->val=x;
        a->next=NULL;
        a->prev=NULL;
        if(head==NULL){
            head=a;
            tail=a;
            mid=a;
        }
        else{
            a->next=head;
            head->prev=a;
            head=a;
            if(size%2==0) mid=mid->prev;
        }
    }

    void insert_last(ll x){
        size++;
        node *a=(node*)malloc(sizeof(node));
        a->val=x;
        a->next=NULL;
        a->prev=NULL;

        if(tail==NULL){
            head=a;
            tail=a;
            mid=a;
        }
        else{
            a->prev=tail;
            tail->next=a;
            tail=a;
            if(size%2==1) mid=mid->next;
        }
    }

    void delete_mid(){
        if(mid==NULL) return;
        size--;
        if(mid->prev) mid->prev->next= mid->next;
        if(mid->next) mid->next->prev= mid->prev;
        if(size%2==0) mid=mid->prev;
        if(size%2==1) mid=mid->next;
    }

    bool isCircular(){
        node *a=(node*)malloc(sizeof(node));
        node *b=(node*)malloc(sizeof(node));
        a=head;
        b=head;
        while(b!=NULL && b->next!=NULL){
            a=a->next;
            b=b->next->next;
            if(a==b){
                return true;
            }
        }
        return false;
    }

    void sort(){
        node *a=(node*)malloc(sizeof(node));
        node *b=(node*)malloc(sizeof(node));
        a=head;
        b=NULL;
        int x;
        if(head==NULL) return;
        else{
            while(a!=NULL){
                b=a->next;
                while(b!=NULL){
                    if(a->val>b->val){
                        x=a->val;
                        a->val=b->val;
                        b->val=x;
                    }
                    b=b->next;
                }
                a = a->next;
            }
        }
    }

    void print(){
        node *a=head;
        ll x=size,y=size;
        while(y--){
            printf("%d",a->val);
            if(--x) cout<<" ";
            a=a->next;
        }
        cout<<endl;
    }

    void mp(){
        cout<<mid->val<<endl;
    }

    ll print_size(){
        return size;
    }

};

int main(){
    doubly_linked_list s;
    cout<<s.print_size()<<endl;
    s.print();

    s.insert_first(510);
    s.mp();
    cout<<s.print_size()<<endl;
    s.print();

    s.insert_last(100);
    s.mp();
    cout<<s.print_size()<<endl;
    s.print();

    cout<<"Is circular?\n"<<s.isCircular()<<endl;

    s.sort();

    s.delete_mid();
    cout<<s.print_size()<<endl;
    s.print();

    return 0;
}
```

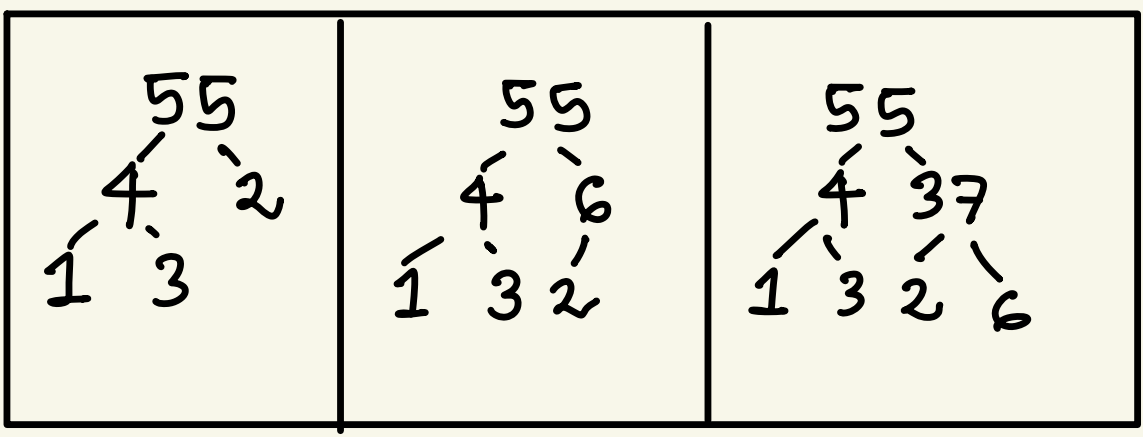
6. a) Starting from an empty Max Heap tree, insert the following values into it according to the given order. Draw the tree after inserting each value. [7]

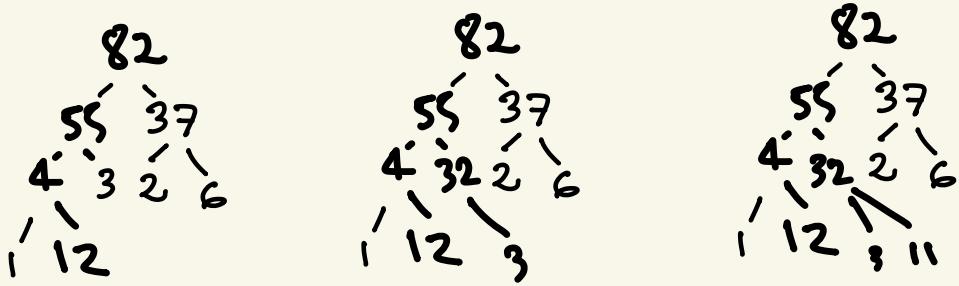
1	2	3	4	55	6	37	12	82	32	11	43	19	17

b) Starting from the Max Heap tree created in 6.a), and extract the maximum value from it till it is nonempty. Draw the tree after deleting each value. [7]

6(a)



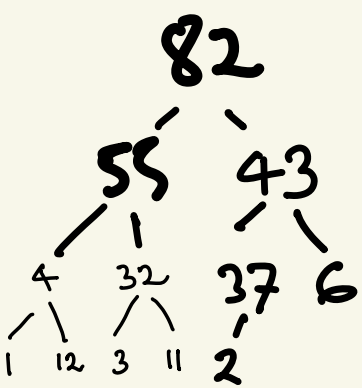


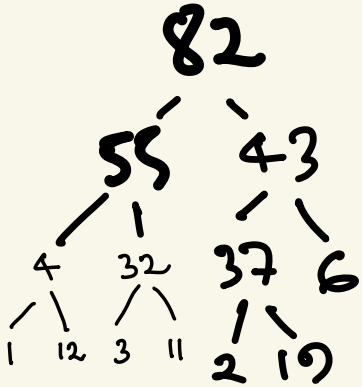


6. a) Starting from an empty Max Heap tree, insert the following values into it according to the given order. Draw the tree after inserting each value. [7]

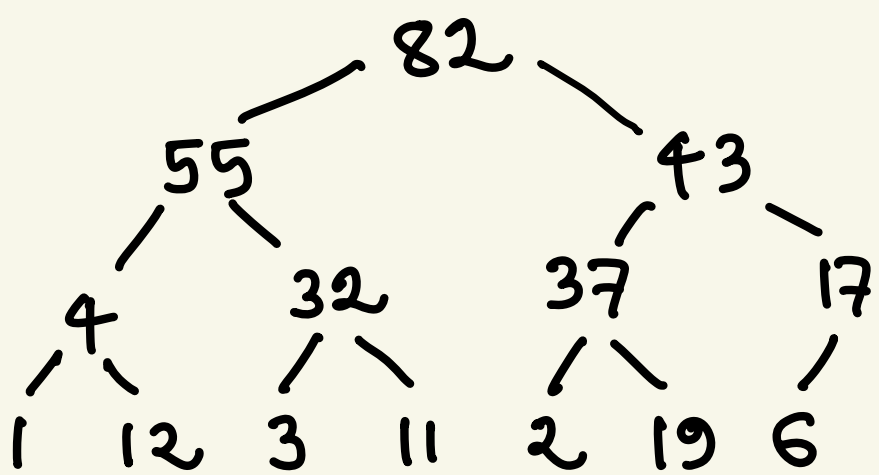
1	2	3	4	55	6	37	12	82	32	11	43	19	17

b) Starting from the Max Heap tree created in **6.a)**, and extract the maximum value from it till it is nonempty. Draw the tree after deleting each value. [7]





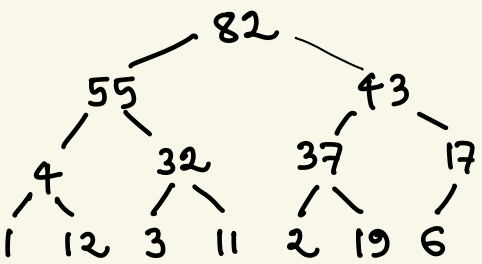
Answer



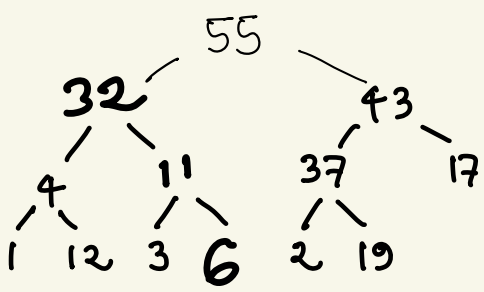
6. a) Starting from an empty Max Heap tree, insert the following values into it according to the given order. Draw the tree after inserting each value. [7]

1	2	3	4	55	6	37	12	82	32	11	43	19	17

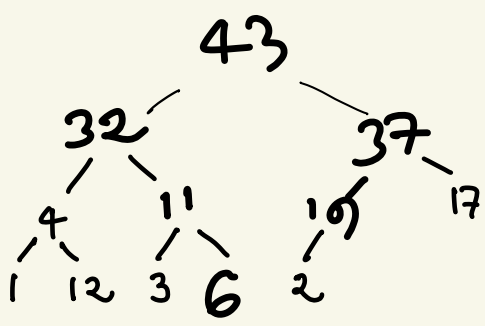
b) Starting from the Max Heap tree created in 6.a), and extract the maximum value from it till it is nonempty. Draw the tree after deleting each value. [7]



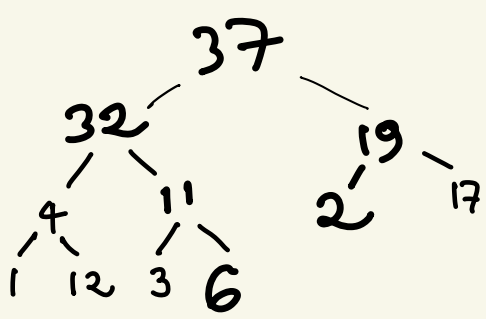
82



55



43



[Cont. till empty]

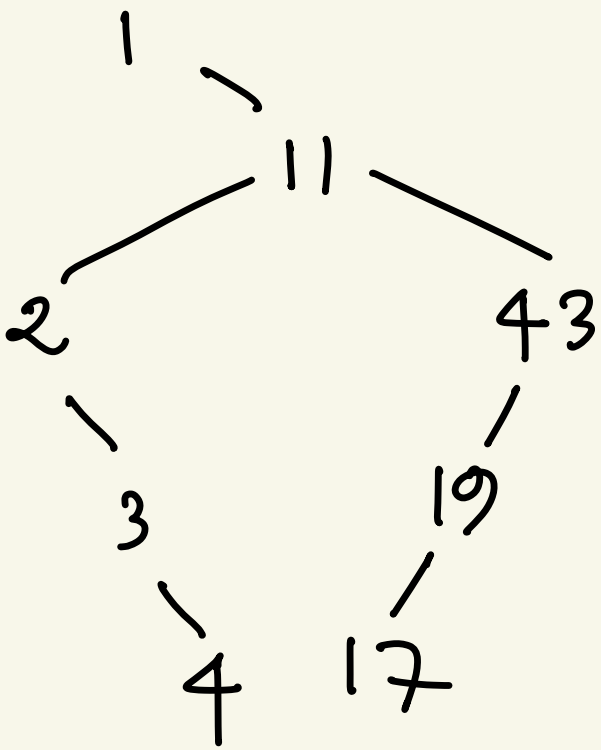
7. a) Given two binary search trees, each node containing an integer, having no element in common, write a pseudocode of an efficient algorithm that merges the trees into a combined binary search tree containing elements from both trees. [7]

b) Starting from an empty binary search tree, insert the following values into it according to the given order. Draw the tree after inserting each value. [4]

1	11	43	19	17	2	3	4

c) Starting from the tree created in 7.b), delete the following values from it according to the given order. Draw the tree after deleting each value. [3]

11	19	3



7. a) Given two binary search trees, each node containing an integer, having no element in common, write a pseudocode of an efficient algorithm that merges the trees into a combined binary search tree containing elements from both trees. [7]

b) Starting from an empty binary search tree, insert the following values into it according to the given order. Draw the tree after inserting each value. [4]

1	11	43	19	17	2	3	4

c) Starting from the tree created in 7.b), delete the following values from it according to the given order. Draw the tree after deleting each value. [3]

11	19	3