Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Synoptic

November/December 2018

Max. Marks: 60 | **Duration: 3 hrs**
Class: S.E. | **Semester: III**
Course Code: CE31/IT31 | **Branch: COMP/IT**
Name of the Course: Advanced Data Structures

Question No.	Question
Q1 (a)	**Marks distribution:** **1- create a sorted doubly linked list------03marks** **2- Merge the 2 sorted doubly linked lists by removing duplicates in it--------------03 marks** **2 marks for Merging logic** **1 mark for removing duplicates logic**

```
node *createsortedlist(node *head,int data)
{
    node *newnode;
    newnode=(node*)malloc(sizeof(node));
    newnode->data=data;
    newnode->prev=NULL;
  newnode->next=NULL;

    if(head==NULL)
    {
          head=newnode;
    }
    else
    {
          node *temp=head;
          while(temp->next!=NULL)
          {
                if(data<(temp->data))
                {
                      break;
                }
                temp=temp->next;
          }
          if(temp==head&&data<head->data)
          {
                newnode->next=head;
                head->prev=newnode;
                head=newnode;
          }
          else if(data<(temp->data))
          {
                newnode->prev=temp->prev;
                newnode->next=temp;
                temp->prev->next=newnode;
                temp->prev=newnode;


          }
          else
          {
```

```
                    temp->next=newnode;
                    newnode->prev=temp;
                }
        }
        return head;

}
```

2- Merge the 2 sorted doubly linked lists by removing duplicates in it----------03 marks

```
node *merge_removingdupli(node *head,node *head1,node *head2)
{
        node *temp1,*temp2;
        temp1=head;
        temp2=head1;
        while(temp1->next!=NULL)
        {
                head2=createsortedlist(head2,temp1->data);
                temp1=temp1->next;
        }
        head2=createsortedlist(head2,temp1->data);
        while(temp2->next!=NULL)
        {
                head2=createsortedlist(head2,temp2->data);
                temp2=temp2->next;
        }
        head2=createsortedlist(head2,temp2->data);
        node *temp3=head2;
        while(temp3->next!=NULL)
        {
                if(temp3->data==temp3->next->data)
                {
                        node *temp4=temp3->next;
                        temp3->next->next->prev=temp3;
                        temp3->next=temp3->next->next;
                        free(temp4);

                }
                temp3=temp3->next;
        }
        return head2;
}
```

Merging 2 sorted list logic 2 marks

Removing duplicates logic 1 marks

OR

Apply the concept of Linked list for the performing arithmetic operation on polynomial equations and write a program to perform the following operations

1- Create Linked representation of Polynomial equations--------02marks

```
node *insertnode(node *head,int degree,int coef)
{
        node *newnode;
        newnode=malloc(sizeof(node)); //dynamically allocating memory
        newnode->degree=degree;    //assigning data
        newnode->coef=coef;
```

```
        if(newnode==NULL)    //if memory not allocated
        {
              printf("Memory was not allocated\n");
        }
        else
        {
              if(head==NULL)    //linked list is empty
              {
                    head=newnode;
                    newnode->next=NULL;
              }
              else                  //adding to the last
              {
                    node *tp;
                    tp=head;        //initialising
                    while(tp->next!=NULL)     //traversing to the end
                    {
                          tp=tp->next;
                    }
                    tp->next=newnode;
                    newnode->next=NULL;    //last node points to Null
               }
        }
        return head;     //returning head
}
```

2- Perform addition of two polynomial equations. Consider sign of coefficient while performing operation---------- 03marks

```
node *add(node *head1,node *head2,node *head3)  //adding the two
equations
{
        while(head1!=NULL && head2!=NULL)  //till both are NULL
        {
              int n;
              n=head1->coef+head2->coef;       //adding coefficient of same
degree
              head3=insertnode(head3,head1->degree,n);    //calling
insertnode function
              head1=head1->next;  //updating pointers
              head2=head2->next;
        }
        return head3;     //returning head3
}
```

3- Display the resulting Polynomial equation--01marks

```
void display(node *head)
{
        node *temp;
        temp=head;
        if(head==NULL)  //linked list empty
        {
              printf("Linked List is empty\n");
        }
        else
        {
              while(temp->next!=NULL)   //traversing to second last node
```

```
                {
                printf("%dx^%d + ",temp->coef,temp->degree);
                    temp=temp->next;
                }
                printf("%dx^%d",temp->coef,temp->degree);     //printing last
element
        }
}
```

Q1 (b)

Marks Distribution:

2 marks---------BST creation

3 marks--------finding kth larges element

1 marks--------main() function

```
#include<iostream>
using namespace std;

struct Node
{
    int key;
    Node *left, *right;
};

// A utility function to create a new BST node
Node *newNode(int item)
{
    Node *temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}


// A function to find k'th largest element in a given tree.
void kthLargestUtil(Node *root, int k, int &c)
{
    // Base cases, the second condition is important to
    // avoid unnecessary recursive calls
    if (root == NULL || c >= k)
        return;

    // Follow reverse inorder traversal so that the
    // largest element is visited first
    kthLargestUtil(root->right, k, c);

    // Increment count of visited nodes
    c++;

    // If c becomes k now, then this is the k'th largest
    if (c == k)
    {
        cout << "K'th largest element is "
             << root->key << endl;
        return;
    }
```

3 Marks for Function finding Kth largest element

```
    // Recur for left subtree
    kthLargestUtil(root->left, k, c);
}

// Function to find k'th largest element
void kthLargest(Node *root, int k)
{
    // Initialize count of nodes visited as 0
    int c = 0;

    // Note that c is passed by reference
    kthLargestUtil(root, k, c);
}

/* A utility function to insert a new node with given key in BST */
Node* insert(Node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);

    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

    /* return the (unchanged) node pointer */
    return node;
}

// Driver Program to test above functions
int main()
{
    /* Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
       20   40  60   80 */
    Node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
    int c = 0;
    for (int k=1; k<=7; k++)
        kthLargest(root, k);

    return 0;
}
```

2 marks for Creation of BStree function

1 mark for main() and proper function calls

Q2 (a)

Marks Distribution:

Q.2.1.----------01 marks

Q.2.2----------05 marks

Distance matrix not shown---------- 1 mark deducted

Parent matrix not shown---------- 1 mark deducted

Partially correct queue status--------2 marks

Q.2.3----------01 marks

Q.2.4----------01 marks

1- Suggest a suitable graph traversal algorithm to complete a journey from city 5 to 8 with minimum stops

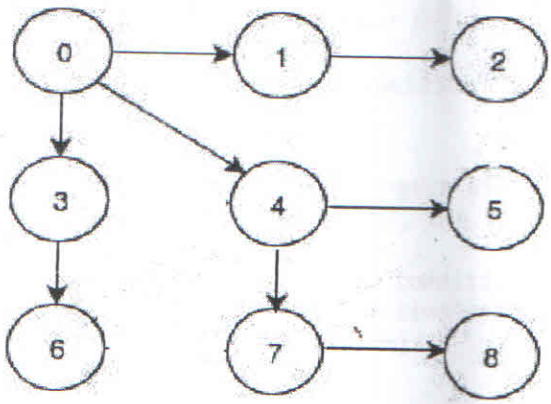
Ans: Breadth First Search graph traversal algorithm

2- Apply the suggested graph traversal algorithm on the given graph and also Show the status of the data structure used at every step.

Ans: Queue data structure status: **0 1 3 4 2 6 5 7 8**

Vertices	0	1	2	3	4	5	6	7	8
Distance	0	1	2	1	1	2	2	2	3
Parent	0	0	1	0	0	4	3	4	7

3- Draw the resulting minimum path tree staring from city '0'



4- state the path to fly from city '5' to '6'

5- >0->4->7->8 **OR**

Marks Distribution:

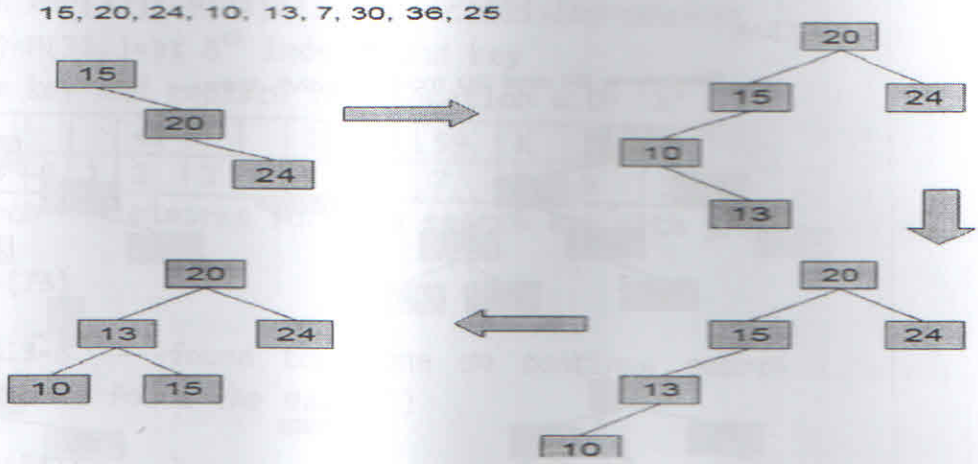
1- **Building AVL tree----------05 marks**

Total 4 Rotations identified correctly and drawn the updated AVL tree after each rotation----------04 marks(each 01 mark)

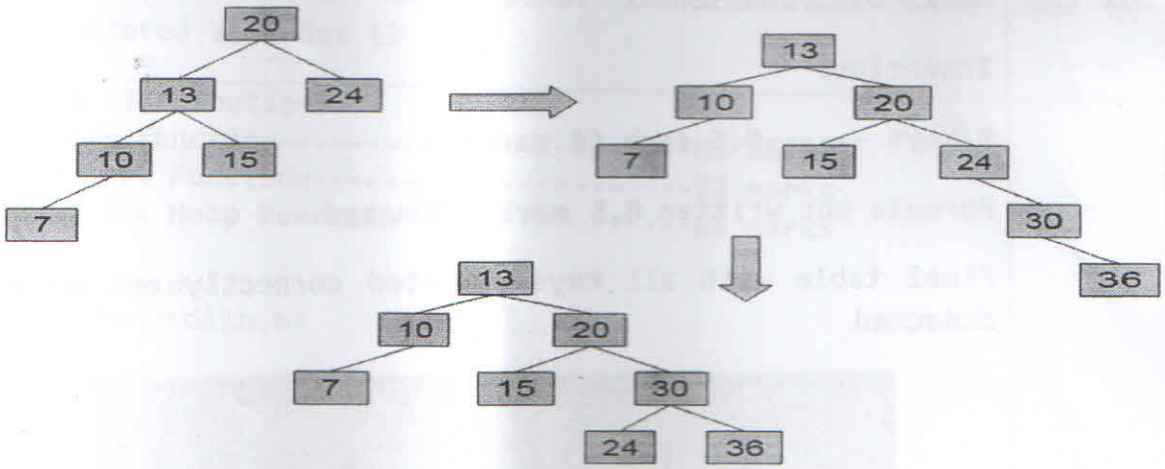
Final Correct tree ----------01 mark

2- **Each deletion 1.5 mark if correctly identified and performed rotation(03 marks)**

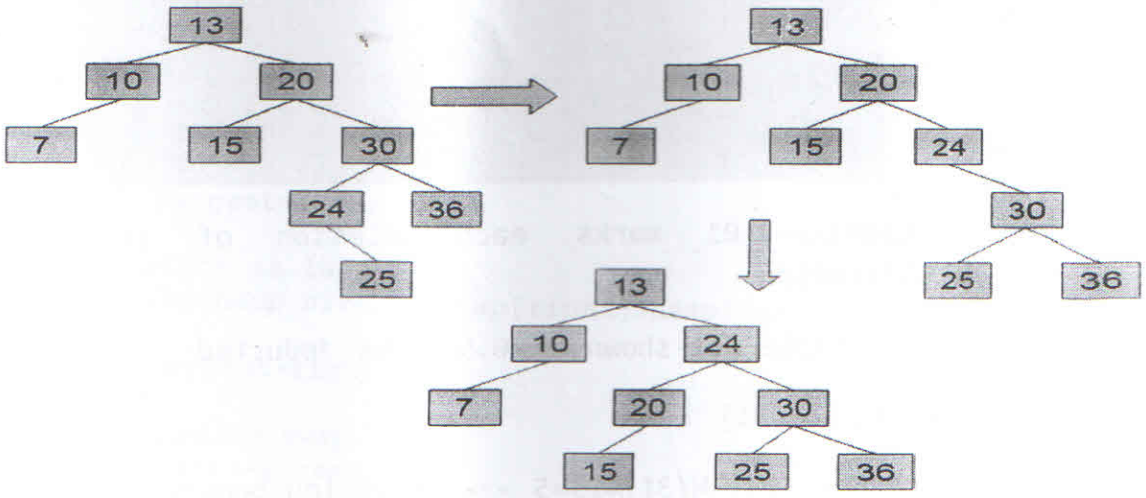
15, 20, 24, 10, 13, 7, 30, 36, 25

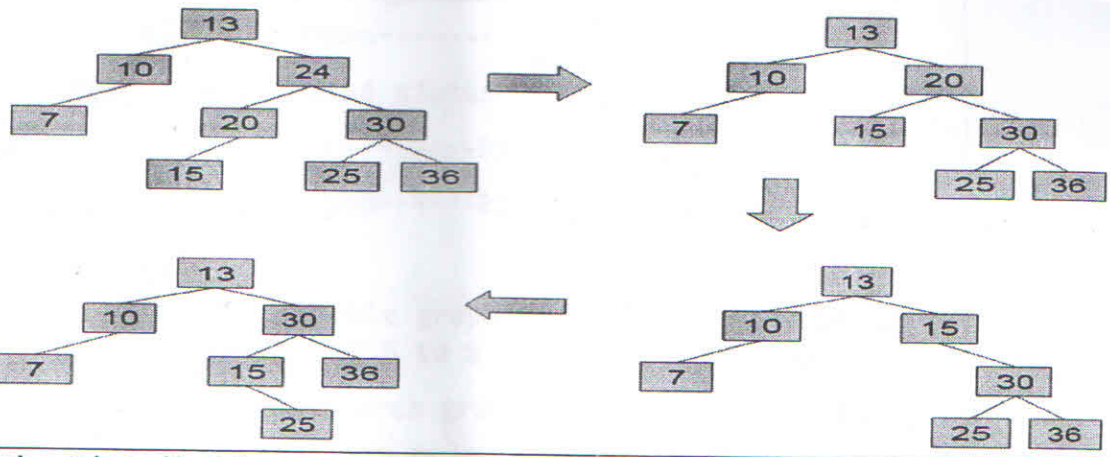
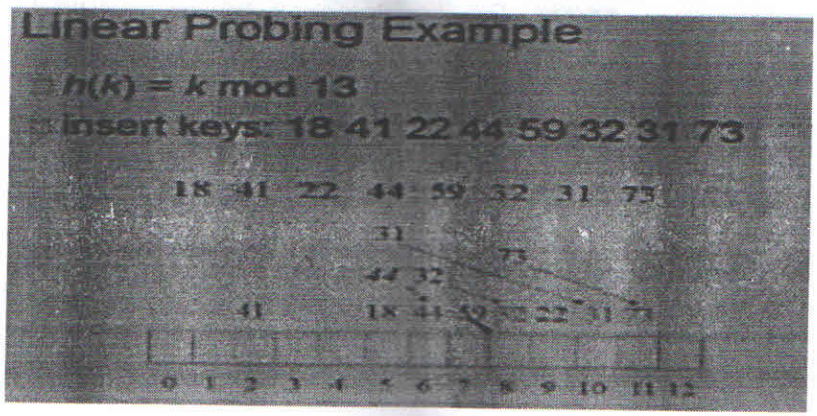


15, 20, 24, 10, 13, 7, 30, 36, 25



15, 20, 24, 10, 13, 7, 30, 36, 25



	Each deletion 1.5 mark if correctly identified and performed rotation Deletion Remove 24 and 20 from the AVL tree. 
Q2 (b)	**Marks Distribution:** **Insertion:** **8 keys ------0.5 each (4 marks)** **Formula not written 0.5 marks deducted** **Final table with all keys inserted correctly not shown 0.5 marks deducted**  **Deletion:**-----01 marks each deletion of keys with correct justification Updated table not shown ---0.5 marks deducted 1- Delete(31) Search key(31): H(31)%13=5 ----collision ocuured H(k,i)=H(31,5)=at 10^{th} index found key Delete key and replace that location with 'X'

2- Delete(32)

Search key(32): H(32)%13=6 ----collision ocuured

H(k,i)=H(31,)=at 8th index found key

Delete key and replace that location with 'X'

Keys			41			18	44	59	X	22	X	73	
Index	0	1	2	3	4	5	6	7	8	9	10	11	12

2-Search---- 01marks for each search key with proper justification

Search(73)

H(73)%13=8 ----found tombstone so continue search linearly so at index 11 we found the data(73)

Search(31):

H(31)%13=5 ----collision occurred. So search continues till a blank cell encountered or data found. In this case blank cell gets encountered at index 12

Q3 (a)

Marks Distribution:
Heapify function----------03 marks
Heap_sort Function----------02 marks
Build Max Heap Function----------01 marks

```
 #include<stdio.h>
#include<stdlib.h>

int max_size,heap_size;

//function to heapify the heap
void heapify(int *heap,int parent)
{
      int largest=parent;
      int left=2*parent;
      int right=2*parent + 1;

      //if left of parent is larger than parent
      if(left<heap_size && heap[left]>heap[parent])
      {
            largest=left;
      }
      //if right is larger
      if(right<heap_size && heap[right]>heap[largest])
      {
            largest=right;
      }
      //performing swap
      if(largest!=parent)
      {
            int temp=heap[parent];
            heap[parent]=heap[largest];
            heap[largest]=temp;
            heapify(heap,largest);//calling the same function
```

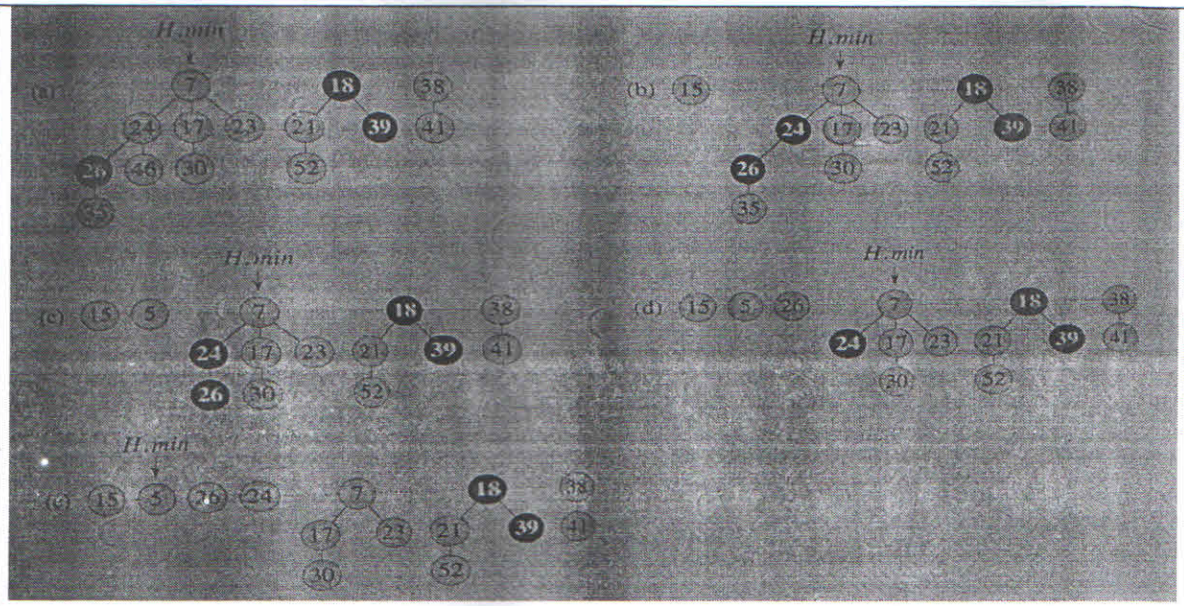
```
        }
}

//function to build heap
void buildheap(int *heap)
{
      int i;
      for(i=heap_size/2;i>=1;i--)
      {
            //calling heapify function
            heapify(heap,i);
      }
}


//function to add a value to the heap
void insert(int *heap,int val)
{
      heap_size++;
      max_size++;
      int i;
      heap=(int
*)realloc(heap,heap_size*sizeof(int));//reallocating the memory
      heap[heap_size-1]=val;
      for(i=heap_size-1;i>=2;i--)
      {
            if(heap[i]>heap[i/2])//if child has value greater than
parent the performing swap
            {
                  int temp=heap[i/2];
                  heap[i/2]=heap[i];
                  heap[i]=temp;
            }
      }
}

//function to delete the root
void delete(int *heap)
{
     printf("Deleting element.....\n");
     //swapping with last value in array
     int temp=heap[1];
     heap[1]=heap[heap_size-1];
     heap[heap_size-1]=temp;
     heap_size--;
     heap[heap_size]=temp;
     buildheap(heap);//calling buildheap
}

//function to print the entire array
void printfullheap(int *heap)
{
     int i;
     printf("\nThe full is heap is as follow:\n");
     //will give sorted array
     for(i=1;i<max_size;i++)
```

```
        {
            printf("%d\t",heap[i]);
        }
        printf("\n");
}

void main()
{
        int *heap;//creating dynamic array
        printf("Enter the size of the heap:");
        scanf("%d",&heap_size);
        heap=(int *)malloc(heap_size * sizeof(int));//dynamically
allocating memeory
        max_size=heap_size;

        int i;
        //take input
        for(i=1;i<max_size;i++)
        {
            printf("Enter the value to be inserted:\n");
            scanf("%d",&heap[i]);
        }

        //calling buildheap to make the heap
        buildheap(heap);

        int val;
        printf("\nEnter the value to be inserted:\n");
        scanf("%d",&val);
        insert(heap,val);//calling insert function
        printf("\nAfter Inserting.....");


        scanf("%d",&val);
        for(i=1;i<= max_size;i++)
        {
            delete(heap);//calling delete function multiple number
of times
        }

        printfullheap(heap);
}
```

OR

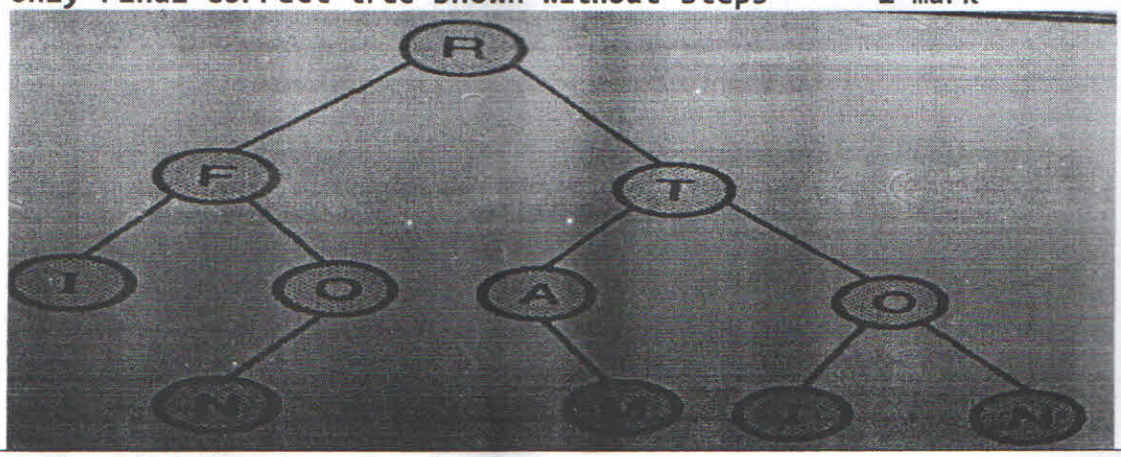
Marks Distribution:

Each decrease key operation with justification----------03 marks



Q3 (b)

Marks Distribution:
Correct tree updated at each steps----------4 marks
Only Final Correct tree shown without steps----- 1 mark



Q4

Marks Distribution: Total 13 steps
Each step 0.5 marks (6.5 marks) if shown correctly with updated pointer array and heap
01 mark--------final correct Fibonacci heap after extract Min
0.5 mark------Final correct H.min pointer

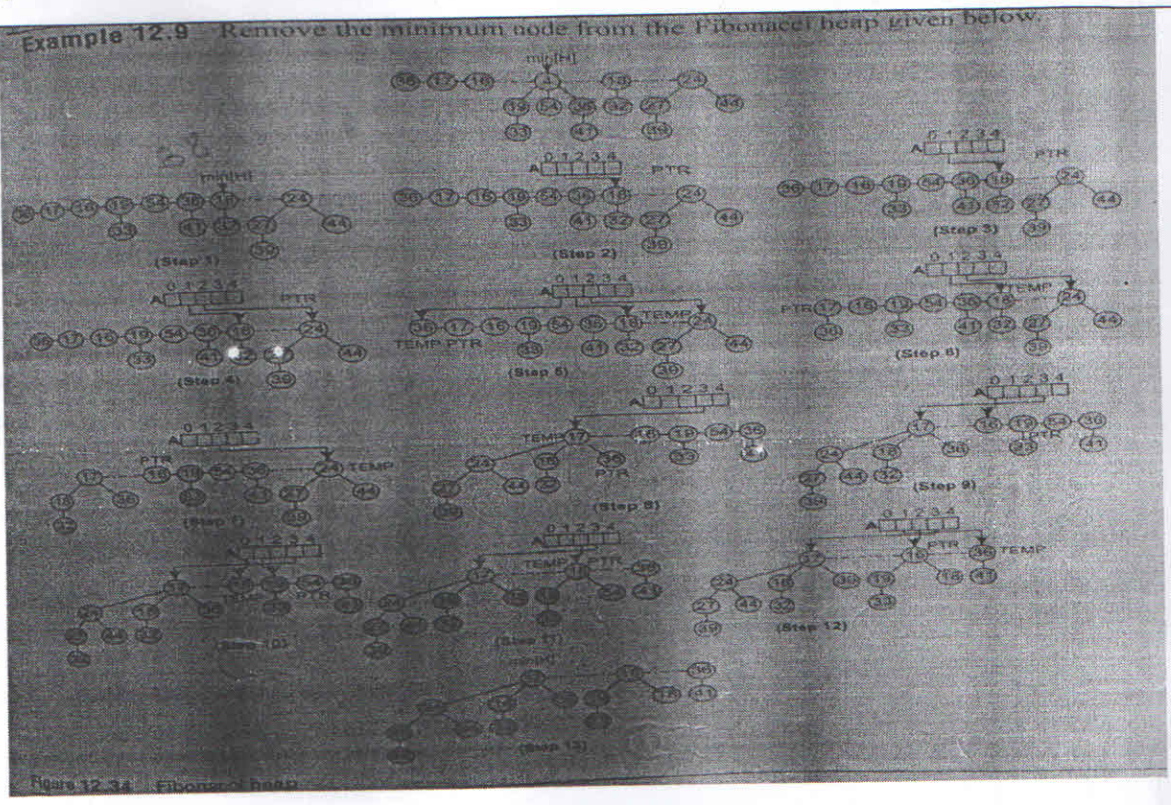


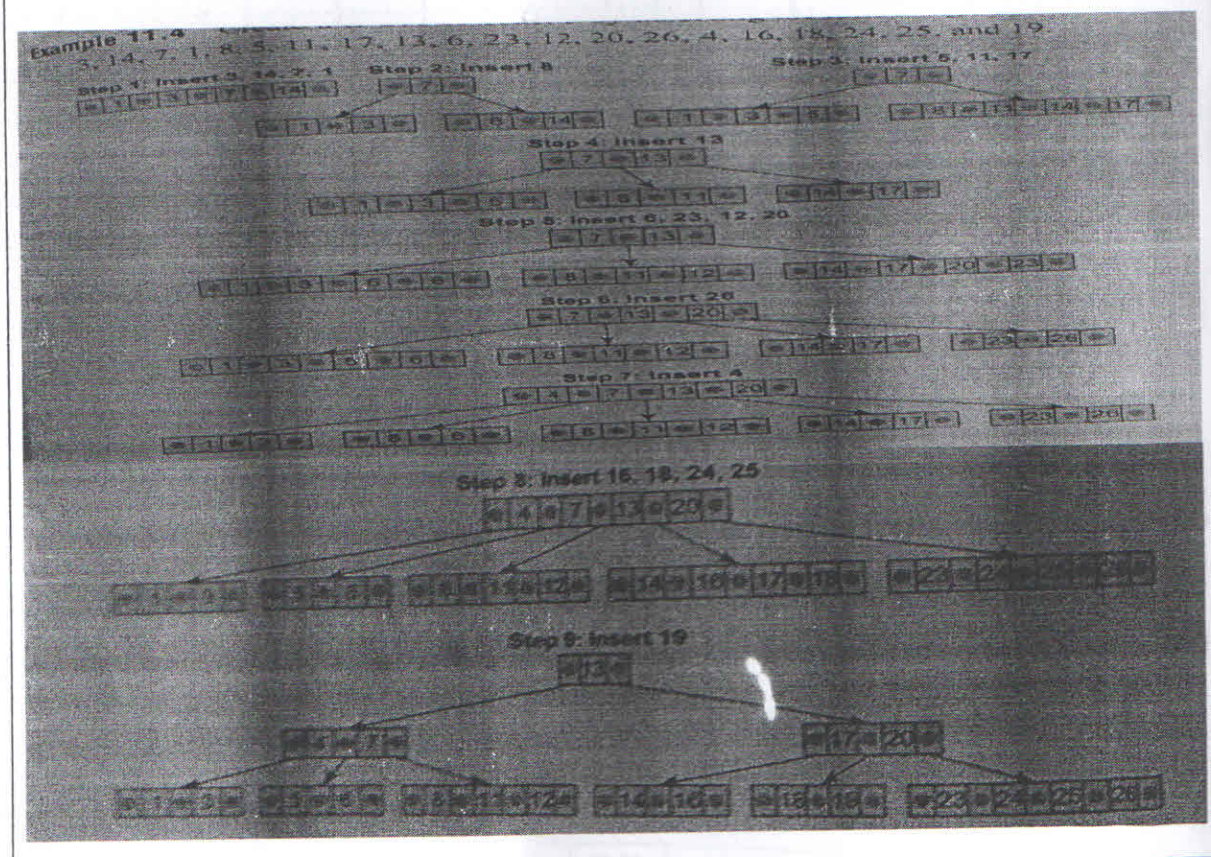
Figure 12.34 Fibonacci heap

Q5 (a)

Marks Distribution:

Total 8 steps each step ----------1 mark(1*8=8)

Only final B tree shown without steps-----1 mark



Q5 (b)

Marks Distribution:

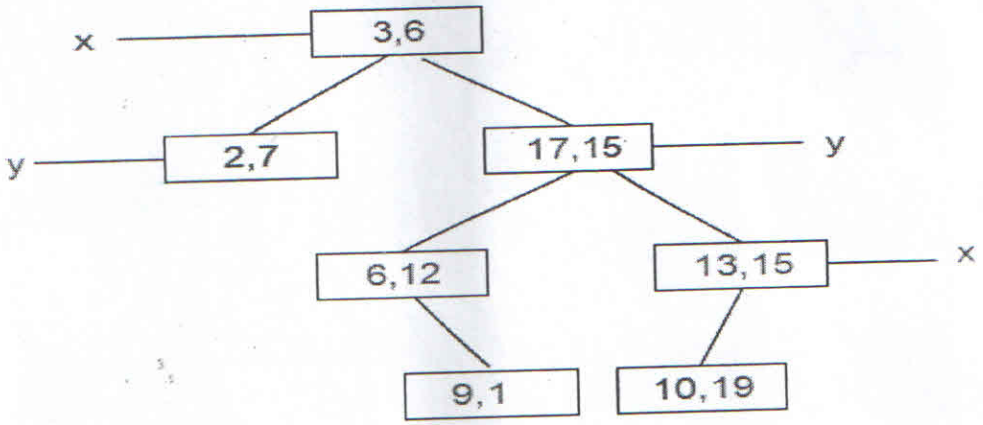
Insertion: 04 marks

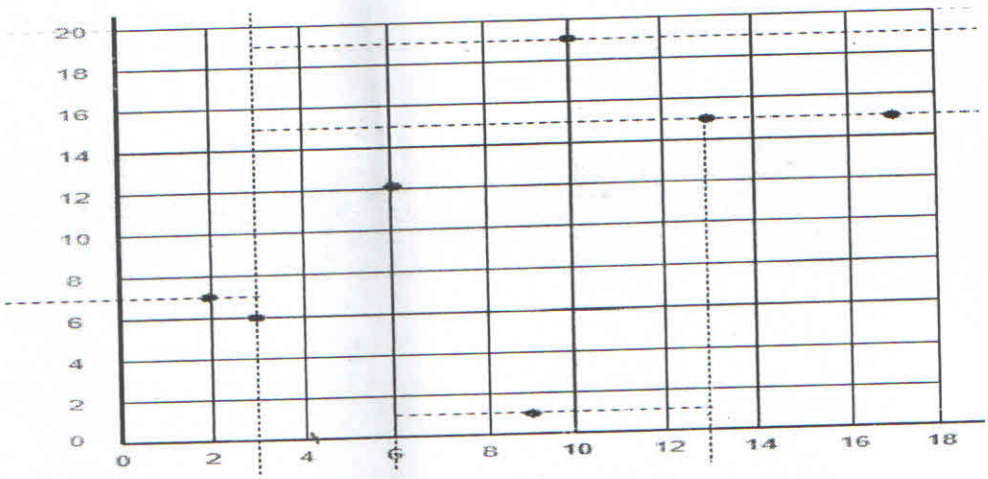
03 marks---------correctly inserting the points and showing updated KD tree after Insertion

01 mark ----------Drawing the partitioned state space

Deletion: 02 marks

01 mark---------- for each point deletion and showing updated KD tree after each deletion





Delete: (3, 6),

Delete (13, 15)