1)BST

*//import java.io.\*;*

class BST{

    class Node{

        int key;

        Node left;

        Node right;

        public Node(int data){

            key=data;

            left=null;

            right=null;

        }

    }

    Node root;

    BST(){

        root=null;

    }

    BST(int value){

        root=new Node(value);

    }

    void insert(int key){

        root = insert\_recur(root,key);

    }

    Node insert\_recur(Node root,int key){

        if(root==null){

            root=new Node(key);

            return root;

        }

        else if(key<root.key){

            root.left = insert\_recur(root.left,key);

        }

        else if(key>root.key){

            root.right = insert\_recur(root.right,key);

        }

        return root;

    }

    void inorder(){

        inorder\_rec(root);

    }

    void inorder\_rec(Node root){

        if (root!=null){

            inorder\_rec(root.left);

            System.out.println(root.key);

            inorder\_rec(root.right);

        }

    }

    void search(int key){

        search\_recur(root,key);

    }

    void search\_recur(Node root,int key){

        if(root==null){

            System.out.println("Key not found");

        }

        if(root.key==key){

            System.out.println("Key found");

            return;

        }

        else if(key<root.key){

            search\_recur(root.left,key);

        }

        else {

            search\_recur(root.right,key);

        }

    }

    void delete(int i){

        root = delete\_recur(root,i);

        }

    Node delete\_recur(Node root,int key){

        if (root==null){

            System.out.println("No such element exist");

            return root;

        }

        else if(key<root.key){

            root.left =  delete\_recur(root.left,key);

        }

        else if(key>root.key){

            root.right = delete\_recur(root.right,key);

        }

        else if(root.key==key){

            if(root.left==null && root.right==null){

                root=null;

                return root;

            }

            else if(root.left==null){

                Node temp=root.right;

                root=null;

                return temp;

            }

            else if(root.right==null){

                Node temp=root.left;

                root=null;

                return temp;

            }

            else{

                root.key = inordersec(root.right);

                root.right = delete\_recur(root.right,root.key);

            }

        }

        return root;

    }

    int inordersec(Node root){

        while(root.left!=null){

            root = root.left;

        }

        return root.key;

    }

    public static void main(String args[]){

        BST b1=new BST();

        b1.insert(2);

        b1.insert(4);

        b1.insert(1);

        b1.insert(5);

        b1.insert(3);

        b1.inorder();

*// b1.search(3);*

         b1.delete(2);

         b1.inorder();

    }

}

Output:

[Running] cd "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\" && javac BST.java && java BST

1

2

3

4

5

1

3

4

5

[Done] exited with code=0 in 1.17 seconds

2)MergeSort

import java.util.*\**;

class MergeSort{

    public void MS(int l,int h,int a[]){

        int mid=(l+h)/2;

        if(l<h){

            MS(l,mid,a);

            MS(mid+1,h,a);

            Merge(l,mid,h,a);

        }

    }

    static void Merge(int l,int mid,int h,int a[]){

        int k[]=new int[a.length];

        int i=l;

        int j=mid+1;

        int m=l;

        while(i<=mid && j<=h ){

            if(a[i]<=a[j]){

                k[m]=a[i];

                m+=1;

                i+=1;

            }

            else{

                k[m]=a[j];

                m+=1;

                j+=1;

            }

        }

        if(i>mid){

            while(j<=h){

                k[m]=a[j];

                j+=1;

                m+=1;

            }

        }

        else{

            while(i<=mid){

                k[m]=a[i];

                m+=1;

                i+=1;

            }

        }

        for(int q=l;q<=h;q+=1){

            a[q]=k[q];

        }

        }

    public static void main(String args[]){

        MergeSort m1=new MergeSort();

        int[] a={7,6,4,-125,1,3,2};

        int n=a.length-1;

        m1.MS(0,n,a);

        System.out.println(Arrays.toString(a));

    }

    }

Output:

[Running] cd "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\" && javac MergeSort.java && java MergeSort

[-125, 1, 2, 3, 4, 6, 7]

[Done] exited with code=0 in 1 seconds

3)Max-Min

public class MaxMin2 {

    static class Pair{

        int max;

        int min;

    }

    static Pair maxminit(int i,int j,int a[]){

        Pair minmax = new Pair();

        Pair mm1 = new Pair();

        Pair mm2 = new Pair();

        int mid;

        if(i==j){

            minmax.max=minmax.min=a[i];

            return minmax;

        }

        else if(j==i+1){

            if(a[i]<a[j]){

                minmax.max=a[j];

                minmax.min=a[i];

            }

            else{

                minmax.max=a[i];

                minmax.min=a[j];

            }

            return minmax;

        }

        else{

            mid=(i+j)/2;

            mm1 = maxminit(i,mid,a);

            mm2 = maxminit(mid+1,j,a);

            if(mm1.min<mm2.min){

                minmax.min = mm1.min;

            }

            else{

                minmax.min = mm2.min;

            }

            if(mm1.max<mm2.max){

                minmax.max = mm2.max;

            }

            else{

                minmax.max = mm1.max;

            }

            return minmax;

        }

    }

        public static void main(String args[]){

            int[] a={1000, 11, 445, 1, 330, 3000};

            int l=a.length-1;

            Pair minmax = maxminit(0,l,a);

            System.out.println("maximum element is"+minmax.max);

            System.out.println("minimum element is"+minmax.min);

    }

}

Output:

[Running] cd "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\" && javac MaxMin2.java && java MaxMin2

maximum element is3000

minimum element is1

[Done] exited with code=0 in 1.077 seconds

4) Job Sequencing with deadlines

import java.util.*\**;

class Job {

    char id;

    int deadline, profit;

*// Constructors*

    public Job() {}

    public Job(char id, int deadline, int profit)

    {

*this*.id = id;

*this*.deadline = deadline;

*this*.profit = profit;

    }

    void printJobScheduling(ArrayList<Job> arr, int t)

    {

        int n = arr.size();

        Collections.sort(arr,

                         (a, b) -> b.profit - a.profit);

        boolean result[] = new boolean[t];

        char job[] = new char[t];

        for (int i = 0; i < n; i++) {

            for (int j

                 = Math.min(t - 1, arr.get(i).deadline - 1);

                 j >= 0; j--) {

                if (result[j] == false) {

                    result[j] = true;

                    job[j] = arr.get(i).id;

                    break;

                }

            }

        }

         for (char jb : job)

            System.out.print(jb + " ");

        System.out.println();

    }

    public static void main(String args[])

    {

        ArrayList<Job> arr = new ArrayList<Job>();

        arr.add(new Job('a', 2, 100));

        arr.add(new Job('b', 1, 19));

        arr.add(new Job('c', 2, 27));

        arr.add(new Job('d', 1, 25));

        arr.add(new Job('e', 3, 15));

        System.out.println(

            "Following is maximum profit sequence of jobs");

        Job job = new Job();

        job.printJobScheduling(arr, 3);

    }

}

Output:

[Running] cd "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\" && javac JobSequencing.java && java JobSequencing

the sequence of jobs is: j3,j1,j4,

[Done] exited with code=0 in 1.155 seconds

5) Prims Algorithm

import sys

class Graph():

    def \_\_init\_\_(self, vertices):

*self*.V = vertices

*self*.graph = [[0 for column in range(vertices)]

                    for row in range(vertices)]

    def printMST(self, parent):

        print("Edge \tWeight")

        for i in range(1, *self*.V):

            print(parent[i], "-", i, "\t", *self*.graph[i][parent[i]])

    def minKey(self, key, mstSet):

*# Initialize min value*

        min = sys.maxsize

        for v in range(*self*.V):

            if key[v] < min and mstSet[v] == False:

                min = key[v]

                min\_index = v

        return min\_index

    def primMST(self):

*# Key values used to pick minimum weight edge in cut*

        key = [sys.maxsize] \* *self*.V

        parent = [None] \* *self*.V *# Array to store constructed MST*

        key[0] = 0

        mstSet = [False] \* *self*.V

        parent[0] = -1 *# First node is always the root of*

        for cout in range(*self*.V):

            u = *self*.minKey(key, mstSet)

            mstSet[u] = True

            for v in range(*self*.V):

                if *self*.graph[u][v] > 0 and mstSet[v] == False \

                and key[v] > *self*.graph[u][v]:

                    key[v] = *self*.graph[u][v]

                    parent[v] = u

*self*.printMST(parent)

*# Driver's code*

if \_\_name\_\_ == '\_\_main\_\_':

    g = Graph(5)

    g.graph = [[0, 2, 0, 6, 0],

            [2, 0, 3, 8, 5],

            [0, 3, 0, 0, 7],

            [6, 8, 0, 0, 9],

            [0, 5, 7, 9, 0]]

    g.primMST()

*# Contributed by Divyanshu Mehta*

Output:

[Running] python -u "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\prims.py"

Edge    Weight

0 - 1    2

1 - 2    3

0 - 3    6

1 - 4    5

[Done] exited with code=0 in 0.172 seconds

6) Kruskal’s Algorithm

class Graph:

    def \_\_init\_\_(self,vertices):

*self*.V=vertices

*self*.graph=[]

    def addEdge(self,u,v,w):

*self*.graph.append([u,v,w])

    def find(self,parent,i):

        if(parent[i]!=i):

            parent[i]=*self*.find(parent,parent[i])

        return parent[i]

    def union(self,parent,rank,x,y):

        if(rank[x]>rank[y]):

            parent[y]=x

        elif(rank[x]<rank[y]):

            parent[x]=y

        else:

            parent[y]=x

            rank[x]+=1

    def kruskals(self):

        i=e=0

        parent=[]

        rank=[]

        result=[]

*self*.graph = sorted(*self*.graph,key=lambda item:item[2])

        for node in range(*self*.V):

            parent.append(node)

            rank.append(0)

        while e<*self*.V-1:

            u,v,w=*self*.graph[i]

            i+=1

            x=*self*.find(parent,u)

            y=*self*.find(parent,v)

            if x!=y:

                e=e+1

                result.append([u,v,w])

*self*.union(parent,rank,x,y)

        mincost=0

        print("Edge in the constructed MST")

        for u,v,weight in result:

            mincost+=weight

            print("%d -- %d == %d"%(u,v,weight))

        print("Minimum Spanning Tree",mincost)

if \_\_name\_\_ == '\_\_main\_\_':

    g = Graph(4)

    g.addEdge(0,1,10)

    g.addEdge(0,2,6)

    g.addEdge(0,3,5)

    g.addEdge(1,3,15)

    g.addEdge(2,3,4)

*# Function call*

    g.kruskals()

output:

[Running] python -u "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\kruskals\_practise.py"

Edge in the constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Spanning Tree 19

[Done] exited with code=0 in 0.165 seconds

7) Dijkstra Algorithm

import sys

class Graph():

    def \_\_init\_\_(self, vertices):

*self*.V = vertices

*self*.graph = [[0 for column in range(vertices)]

                    for row in range(vertices)]

    def printSolution(self, dist):

        print("Vertex \tDistance from Source")

        for node in range(*self*.V):

            print(node, "\t", dist[node])

    def minDistance(self, dist, sptSet):

        min = sys.maxsize

        for u in range(*self*.V):

            if dist[u] < min and sptSet[u] == False:

                min = dist[u]

                min\_index = u

        return min\_index

    def dijkstra(self, src):

        dist = [sys.maxsize] \* *self*.V

        dist[src] = 0

        sptSet = [False] \* *self*.V

        for cout in range(*self*.V):

            x = *self*.minDistance(dist, sptSet)

            sptSet[x] = True

            for y in range(*self*.V):

                if *self*.graph[x][y] > 0 and sptSet[y] == False and \

                        dist[y] > dist[x] + *self*.graph[x][y]:

                    dist[y] = dist[x] + *self*.graph[x][y]

*self*.printSolution(dist)

if \_\_name\_\_ == "\_\_main\_\_":

    g = Graph(9)

    g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],

            [4, 0, 8, 0, 0, 0, 0, 11, 0],

            [0, 8, 0, 7, 0, 4, 0, 0, 2],

            [0, 0, 7, 0, 9, 14, 0, 0, 0],

            [0, 0, 0, 9, 0, 10, 0, 0, 0],

            [0, 0, 4, 14, 10, 0, 2, 0, 0],

            [0, 0, 0, 0, 0, 2, 0, 1, 6],

            [8, 11, 0, 0, 0, 0, 1, 0, 7],

            [0, 0, 2, 0, 0, 0, 6, 7, 0]

            ]

    g.dijkstra(0)

output:

[Running] python -u "c:\Users\Sai Ganesh\OneDrive\Desktop\Java\dijkstra.py"

Vertex  Distance from Source

0    0

1    4

2    12

3    19

4    21

5    11

6    9

7    8

8    14

[Done] exited with code=0 in 0.148 seconds