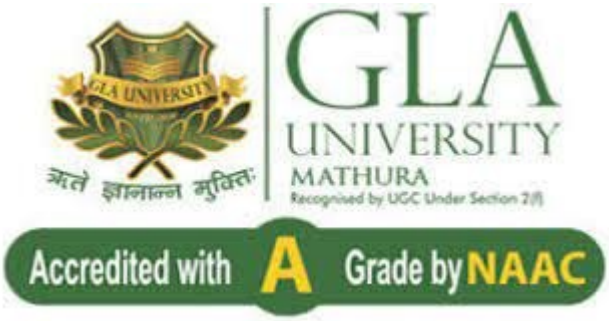
DEPARTMENT OF COMPUTER ENGINEERING & APPLICATIONS



PRACTICAL FILE

on

DESIGN & ANALYSIS OF ALGORITHMS

(BCSC0807)

SUBMITTED TO:
Puroo Kulshrestha
L (30)

SUBMITTED BY:
Mrs. Nishtha Parashar

LAB – 1

```
/*
        NAME - PUROO KULSHRESTHA
        ROLL NO - 201500535
        SEC / ROLLNO. - L/30
        SUB - DAA LAB
*/

import java.util.*;
import java.lang.*;

class linearsearch
{
    public static void main(String args[])
    {
        int array[]={4,5,8,2,7,3,6,9};
        int size=array.length;
        int value=7;
        for(int i=0;i<size-1;i++)
        {
            if(array[i]==value)
            {
                System.out.println("I Found Element :" +i);

            }
            else
            {
                System.out.println("Element not found");
            }
        }
    }
}
```

LAB – 2

```
/*
 * NAME - PUROO KULSHRESTHA
 * CLASS - L_2
 * ROLL NO - 201500535
 * CLASS ROLL NO -30
 */

import java.util.Scanner;

public class Insertion_sort {
    static void sort(int arr[]) {
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }

    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");

        System.out.println();
    }

    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < arr.length; i++) {
            arr[i] = s.nextInt();
        }
        sort(arr);
        printArray(arr);
        s.close();
    }
}
```

LAB – 3

```
/*
 * NAME - PUROO KULSHRESTHA
 * CLASS - L_2
 * ROLL NO - 201500535
 * CLASS ROLL NO -30
 */
#include <stdio.h>

void swap(int* xp, int* yp)
{
int temp = *xp;
	*xp = *yp;
	*yp = temp;
}

void bubbleSort(int arr[], int n)
{
	int i, j;
	for (i = 0; i < n - 1; i++)
		for (j = 0; j < n - i - 1; j++)
			if (arr[j] > arr[j + 1])
				swap(&arr[j], &arr[j + 1]);
}

void printArray(int arr[], int size)
{
	int i;
	for (i = 0; i < size; i++)
		printf("%d ", arr[i]);
	printf("\n");
}

int main()
{
	int arr[] = { 5, 1, 4, 2, 8 };
	int n = sizeof(arr) / sizeof(arr[0]);
	bubbleSort(arr, n);
	printf("Sorted array: \n");
	printArray(arr, n);
	return 0;
}
```

LAB – 4

Name - Puroo Kulshrestha
Sec - L2
Roll no - 30
University roll no - 201500535

Q1.

```
#include <stdio.h>

int main() {
        int n;
    scanf("%d", &n);

        int arr[n];
    for(int i=0;i<n;i++){
        scanf("%d", &arr[i]);
    }

    for(int i=0;i<n;i++){
        int min = i;
        for(int j=i+1;j<n;j++){
                if(arr[j]<arr[min])
                min=j;
         }
         int temp = arr[min];
         arr[min] = arr[i];
         arr[i] = temp;
    }

    for(int i=0;i<n;i++){
        printf("%d, \n", arr[i]);
    }
}
```

Q2.
time complexity - O(n^2logn)

Q3.

```
#include<stdio.h>

void main(){
        for(int i=0;i<=10;i++){
                for(int j=1;j<n;j<=j*2){
                        printf("DAA\n");
                }
        }
}
```

LAB – 5

Q.WAP of having time complexity O(n+m) , O(n^4), O(n^2 log n)

O(n^4)

```
#include<stdio.h>
void main(){
        for(int a=0;a<=50;a++){
                for(int b=0;b<=50;b++){
                        for(int c=0;c<=50;c++){
                                for(int d=0;d<=50;d++){
                                        printf("DAA");
                                }
                        }
                }
        }
}
```

O(n^2 logn)

```
#include<stdio.h>
void main(){
        int n=20;
        for(int i=n/2;i<=n;i++){
                for(int j=1;j<=n/2;j++){
                        for(int k=1;ki<n;k=K*2){
                                printf("DAA");
                        }
                }
        }
}
```

O(n+m)

```
#include<stdio.h>
void main(){
        int n=25,m=20;
        for(int i=0;i<n;i++)
                printf("DAA");
        for(int j=0;j<m;j++)
                printf("DAA");
}
```

```
public class HeapSort {

    public void sort(int arr[]) {
      int n = arr.length;


      for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
      }


      for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;


        heapify(arr, i, 0);
      }
    }

    void heapify(int arr[], int n, int i) {

      int largest = i;
      int l = 2 * i + 1;
      int r = 2 * i + 2;

      if (l < n && arr[l] > arr[largest])
        largest = l;

      if (r < n && arr[r] > arr[largest])
        largest = r;

      if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        heapify(arr, n, largest);
      }
    }


    static void printArray(int arr[]) {
      int n = arr.length;
      for (int i = 0; i < n; ++i)
        System.out.print(arr[i] + " ");
      System.out.println();
    }
    public static void main(String args[]) {
      int arr[] = { 1, 12, 9, 5, 6, 10 };

      HeapSort hs = new HeapSort();
      hs.sort(arr);

      System.out.println("Sorted array is");
      printArray(arr);
    }
  }
```

}

LAB – 6

Q. WAP of Merge SOrt.

```
class MergeSort
{
	void merge(int arr[], int l, int m, int r)
	{
		int n1 = m - l + 1;
		int n2 = r - m;

		int L[] = new int[n1];
		int R[] = new int[n2];

		for (int i = 0; i < n1; ++i)
			L[i] = arr[l + i];
		for (int j = 0; j < n2; ++j)
			R[j] = arr[m + 1 + j];

		int i = 0, j = 0;
		int k = l;

		while (i < n1 && j < n2) {
			if (L[i] <= R[j]) {
				arr[k] = L[i];
				i++;
			}
			else {
				arr[k] = R[j];
				j++;
			}
			k++;
		}
		while (i < n1) {
			arr[k] = L[i];
			i++;
			k++;
		}
		while (j < n2) {
			arr[k] = R[j];
			j++;
			k++;
		}
	}

	void sort(int arr[], int l, int r)
	{
		if (l < r) {
			int m =l+ (r-l)/2;
			sort(arr, l, m);
			sort(arr, m + 1, r);
			merge(arr, l, m, r);
		}
	}

	static void printArray(int arr[],int n)
	{
		for (int i = 0; i < n; ++i)
			System.out.println(arr[i] + " ");
```

```
	}

	public static void main(String args[])
	{
		int arr[] = { 12, 11, 13, 5, 6, 7 };

		MergeSort ob = new MergeSort();
		int n = arr.length-1;
		ob.sort(arr, 0, n);

		System.out.println("\nSorted array");
		printArray(arr,n);
	}
}
```

LAB – 7

```
/*
	NAME - PUROO KULSHRESTHA
	ROLL NO - 201500535
	SEC / ROLLNO. - L/30
	SUB - DAA LAB
*/


import java.util.*;
import java.lang.*;

class QUCIKSORT	{

	int partition (int a[], int start, int end)
	{
		int pivot = a[end];
		int i = (start - 1);

		for (int j = start; j <= end - 1; j++)
		{

			if (a[j] < pivot)
			{
				i++;
				int t = a[i];
				a[i] = a[j];
				a[j] = t;
			}
		}
		int t = a[i+1];
		a[i+1] = a[end];
		a[end] = t;
		return (i + 1);
	}
```

```
	void quick(int a[], int start, int end)
	{
		if (start < end)
		{
			int p = partition(a, start, end);
			quick(a, start, p - 1);
			quick(a, p + 1, end);
		}
	}


	void printArr(int a[], int n)
	{
		int i;
		for (i = 0; i < n; i++)
			System.out.print(a[i] + " ");
	}

	public static void main(String[] args) {
		int a[] = { 13, 18, 27, 2, 19, 25 };
		int n = a.length;
		System.out.println("\nBefore sorting array elements are - ");
		Quick q1 = new Quick();
		q1.printArr(a, n);
		q1.quick(a, 0, n - 1);
		System.out.println("\nAfter sorting array elements are - ");
		q1.printArr(a, n);
		System.out.println();
	}

}
```

LAB – 8

```
/*
        NAME - PUROO KULSHRESTHA
        ROLL NO - 201500535
        SEC / ROLLNO. - L/30
        SUB - DAA LAB
*/

import java.util.*;
import java.lang.*;

class BinaryTree {

    Node root;

    BinaryTree()
    {
        root = null;
    }

    void printInorder(Node node)
    {
        if (node == null)
            return;

        printInorder(node.left);

        System.out.print(node.key + " ");

        printInorder(node.right);
    }

    void printInorder() {
       printInorder(root);
    }

    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        System.out.println("\nInorder traversal of binary tree is ");
        tree.printInorder();
    }
}
```

LAB – 9

```
/*
        NAME - PUROO KULSHRESTHA
        ROLL NO - 201500535
        SEC / ROLLNO. - L/30
        SUB - DAA LAB
*/

import java.util.*;
import java.lang.*;

public class BFSTraversal
{
    private int vertex;
    private LinkedList<Integer> adj[];
    private Queue<Integer> que;
    BFSTraversal(int v)
    {
        vertex = v;
        adj = new LinkedList[vertex];
        for (int i=0; i<v; i++)
        {
            adj[i] = new LinkedList<>();
        }
        que = new LinkedList<Integer>();
    }

    void insertEdge(int v,int w)
    {
        adj[v].add(w);
    }

    void BFS(int n)
    {
        boolean nodes[] = new boolean[vertex];
       int a = 0;
        nodes[n]=true;
        que.add(n);
        while (que.size() != 0)
        {
            n = que.poll();
            System.out.print(n+" ");
            for (int i = 0; i < adj[n].size(); i++)
            {
                a = adj[n].get(i);
                if (!nodes[a])
                {
                    nodes[a] = true;
                    que.add(a);
                }
            }
        }
    }
```

```
    public static void main(String args[])
    {
        BFSTraversal graph = new BFSTraversal(10);
        graph.insertEdge(0, 1);
        graph.insertEdge(0, 2);
        graph.insertEdge(0, 3);
        graph.insertEdge(1, 3);
        graph.insertEdge(2, 4);
        graph.insertEdge(3, 5);
        graph.insertEdge(3, 6);
        graph.insertEdge(4, 7);
        graph.insertEdge(4, 5);
        graph.insertEdge(5, 2);
        graph.insertEdge(6, 5);
        graph.insertEdge(7, 5);
        graph.insertEdge(7, 8);
        System.out.println("Breadth First Traversal for the graph is:");
        graph.BFS(2);
    }
}
```

LAB – 9

```
import java.util.*;
import java.io.*;
import java.lang.*;

public class DijkstraExample
{
        static final int totalVertex = 9;
        int minimumDistance(int distance[], Boolean spSet[])
        {
                int m = Integer.MAX_VALUE, m_index = -1;

                for (int vx = 0; vx < totalVertex; vx++)
                {
                        if (spSet[vx] == false && distance[vx] <= m)
                        {
                                m = distance[vx];
                                m_index = vx;
                        }
                }

                return m_index;
        }

        void printSolution(int distance[], int n)
        {
                System.out.println("The shortest Distance from source 0th
node to all other nodes are: ");
                for (int j = 0; j < n; j++)
                System.out.println("To " + j + " the shortest distance is: "
+ distance[j]);
        }
```

```
        void dijkstra(int graph[][], int s)
        {
                int distance[] = new int[totalVertex];
                Boolean spSet[] = new Boolean[totalVertex];

                for (int j = 0; j < totalVertex; j++)
                {
                        distance[j] = Integer.MAX_VALUE;
                        spSet[j] = false;
                }

                distance[s] = 0;

                for (int cnt = 0; cnt < totalVertex - 1; cnt++)
                {
                        int ux = minimumDistance(distance, spSet);

                        spSet[ux] = true;
                        for (int vx = 0; vx < totalVertex; vx++)
                                if (!spSet[vx] && graph[ux][vx] != -1 &&
distance[ux] != Integer.MAX_VALUE && distance[ux] + graph[ux][vx] <
distance[vx]){
                                        distance[vx] = distance[ux] +
graph[ux][vx];
                                }
                }

                printSolution(distance, totalVertex);
        }

        public static void main(String argvs[])
        {
                int grph[][] = new int[][] {  { -1, 3, -1, -1, -1, -1, -1, 7,
-1 },
                                              { 3, -1, 7, -1, -1, -1, -1, 10,
4 },
                                              { -1, 7, -1, 6, -1, 2, -1, -1, 1
},
                                              { -1, -1, 6, -1, 8, 13, -1, -1,
3 },
                                              { -1, -1, -1, 8, -1, 9, -1, -1,
-1 },
                                              { -1, -1, 2, 13, 9, -1, 4, -1, 5
},
                                              { -1, -1, -1, -1, -1, 4, -1, 2,
5 },
                                              { 7, 10, -1, -1, -1, -1, 2, -1,
6 },
                                              { -1, 4, 1, 3, -1, 5, 5, 6, -1 }
};

                DijkstraExample obj = new DijkstraExample();
                obj.dijkstra(grph, 0);
        }
}
```

LAB – 10

```
/*
* Name - Puroo Kulshrestha
* Roll no - 201500535
* Sec/Rollno. - L/30
* Bellman ford Algortihm
*/


import java.util.ArrayList;
import java.util.List;

class Graph
{
	private int V;
	private List<Edge> edges;
	public Graph(int v)
	{
		V = v;
		edges = new ArrayList<Edge>();
	}

	public int getV()
	{
		return V;
	}

	public void setV(int v)
	{
		V = v;
	}


	public List<Edge> getEdges()
	{
		return edges;
	}

	public void setEdges(List<Edge> edges)
	{
		this.edges = edges;
	}

	public void addEdge(int u, int v, int w)
	{
		Edge e = new Edge(u, v, w);
		edges.add(e);
	}
}

class Edge
{
	private int u;
	private int v;
	private int w;

	public int getU()
	{
		return u;
```

```
        }

        public void setU(int u)
        {
                this.u = u;
        }

        public int getV()
        {
                return v;
        }

        public void setV(int v)
        {
                this.v = v;
        }

        public int getW()
        {
                return w;
        }

        public void setW(int w)
        {
                this.w = w;
        }

        public Edge(int u, int v, int w)
        {
                this.u = u;
                this.v = v;
                this.w = w;
        }
}

public class BellmanFordImplementation
{
        public static void main(String args[])
        {
                Graph g = createGraph();
                int distance[] = new int[g.getV()];
                boolean hasNegativeCycle = getShortestPaths(g, 1, distance);
                if(!hasNegativeCycle)
                {
                        System.out.println("Vertex \t: Distance");
                        for(int i = 1; i < distance.length; i++)
                                System.out.println("\t"+i + " " +
"\t\t"+(distance[i] == Integer.MAX_VALUE ? "-" : distance[i]));
                }
                else
                {
                        System.out.println("Negative cycle exists in the
graph, no solution found!!!");
                }
        }

        private static Graph createGraph()
        {
                int v = 7;
                Graph g = new Graph(v);
                g.addEdge(1, 2, 4);
                g.addEdge(1, 4, 9);
                g.addEdge(2, 3, -1);
```

```
                g.addEdge(3, 6, 3);
                g.addEdge(4, 3, 2);
                g.addEdge(4, 5, -5);
                g.addEdge(5, 6, 0);
                return g;
        }

        public static boolean getShortestPaths(Graph g, int source, int[]
distance)
        {
                int V = g.getV();
                for(int i = 1; i < V; i++)
                {
                        distance[i] = Integer.MAX_VALUE;
                }
                distance[source] = 0;
                for(int i = 1; i < V; i++)
                {
                        for(Edge e: g.getEdges())
                        {
                                int u = e.getU(), v = e.getV(), w = e.getW();
                                if(distance[u] != Integer.MAX_VALUE &&
distance[v] > distance[u] + w)
                                {
                                        distance[v] = distance[u] + w;
                                }
                        }
                }

                for(Edge e: g.getEdges())
                {
                        int u = e.getU(), v = e.getV(), w = e.getW();
                        if(distance[u] != Integer.MAX_VALUE && distance[v] >
distance[u] + w)
                        {
                                return true;
                        }
                }

        return false;

        }
}
```

LAB – 11

```
/*
* Name - Puroo Kulshrestha
* Roll no - 201500535
* Sec/Rollno. - L/30
* Q. Explain how Bellman ford algorithm is different from Dijkstra's
algorithm. Justify your answer with an example.
*/

76
```

Bellman-Ford algorithm is a single-source shortest path algorithm, so when you have negative edge weight then it can detect negative cycles in a graph.The only difference between the two is that Bellman-Ford is also capable of handling negative weights whereas Dijkstra Algorithm can only handle positives.

Dijkstra is however generally considered better in the absence of negative weight edges, as a typical binary heap priority queue implementation has O((|E|+|V|)log|V|) time complexity [A Fibonacci heap priority queue gives O(|V|log|V| + |E|)], while the Bellman-Ford algorithm has O(|V||E|) complexity.The only difference is that Dijkstra's algorithm cannot handle negative edge weights which Bellman-ford handles. And bellman-ford also tells us whether the graph contains negative cycle. If graph doesn't contain negative edges then Dijkstra's is always better.
Bellman Ford’s Algorithm has less scalability than Dijkstra’s Algorithm.
Dynamic Programming approach is taken to implement the Bellman Ford algorithm.
reedy approach is taken to implement the Dijkstra’s algorithm.

LAB – 12

```
/*
        NAME - Puroo Kulshrestha
        SEC - L
        ROLL NO. - 30
        University roll no. - 201500535
*/



import java.util.*;

public class Prims
{
    private boolean unsettled[];
    private boolean settled[];
    private int numberofvertices;
    private int adjacencyMatrix[][];
    private int key[];
    public static final int INFINITE = 999;
    private int parent[];

    public Prims(int numberofvertices)
    {
        this.numberofvertices = numberofvertices;
        unsettled = new boolean[numberofvertices + 1];
        settled = new boolean[numberofvertices + 1];
        adjacencyMatrix = new int[numberofvertices + 1][numberofvertices + 1];
        key = new int[numberofvertices + 1];
        parent = new int[numberofvertices + 1];
    }

    public int getUnsettledCount(boolean unsettled[])
    {
        int count = 0;
        for (int index = 0; index < unsettled.length; index++)
        {
            if (unsettled[index])
            {
                count++;
            }
        }
        return count;
    }

    public void primsAlgorithm(int adjacencyMatrix[][])
    {
        int evaluationVertex;
        for (int source = 1; source <= numberofvertices; source++)
        {
            for (int destination = 1; destination <= numberofvertices;
destination++)
            {
                this.adjacencyMatrix[source][destination] =
adjacencyMatrix[source][destination];
            }
        }

        for (int index = 1; index <= numberofvertices; index++)
        {
            key[index] = INFINITE;
        }
```

```
        key[1] = 0;
        unsettled[1] = true;
        parent[1] = 1;

        while (getUnsettledCount(unsettled) != 0)
        {
            evaluationVertex = getMimumKeyVertexFromUnsettled(unsettled);
            unsettled[evaluationVertex] = false;
            settled[evaluationVertex] = true;
            evaluateNeighbours(evaluationVertex);
        }
    }

    private int getMimumKeyVertexFromUnsettled(boolean[] unsettled2)
    {
        int min = Integer.MAX_VALUE;
        int node = 0;
        for (int vertex = 1; vertex <= numberofvertices; vertex++)
        {
            if (unsettled[vertex] == true && key[vertex] < min)
            {
                node = vertex;
                min = key[vertex];
            }
        }
        return node;
    }

    public void evaluateNeighbours(int evaluationVertex)
    {

        for (int destinationvertex = 1; destinationvertex <= numberofvertices;
destinationvertex++)
        {
            if (settled[destinationvertex] == false)
            {
                if (adjacencyMatrix[evaluationVertex][destinationvertex] !=
INFINITE)
                {
                    if (adjacencyMatrix[evaluationVertex][destinationvertex] <
key[destinationvertex])
                    {
                        key[destinationvertex] =
adjacencyMatrix[evaluationVertex][destinationvertex];
                        parent[destinationvertex] = evaluationVertex;
                    }
                    unsettled[destinationvertex] = true;
                }
            }
        }
    }

    public void printMST()
    {
        System.out.println("SOURCE  : DESTINATION = WEIGHT");
        for (int vertex = 2; vertex <= numberofvertices; vertex++)
        {
            System.out.println(parent[vertex] + "\t:\t" + vertex +"\t=\t"+
adjacencyMatrix[parent[vertex]][vertex]);
        }
    }

    public static void main(String... arg)
    {
        int adjacency_matrix[][];
        int number_of_vertices;
```

```
        Scanner scan = new Scanner(System.in);

        try
        {
            System.out.println("Enter the number of vertices");
            number_of_vertices = scan.nextInt();
            adjacency_matrix = new int[number_of_vertices +
1][number_of_vertices + 1];

            System.out.println("Enter the Weighted Matrix for the graph");
            for (int i = 1; i <= number_of_vertices; i++)
            {
                for (int j = 1; j <= number_of_vertices; j++)
                {
                    adjacency_matrix[i][j] = scan.nextInt();
                    if (i == j)
                    {
                        adjacency_matrix[i][j] = 0;
                        continue;
                    }
                    if (adjacency_matrix[i][j] == 0)
                    {
                        adjacency_matrix[i][j] = INFINITE;
                    }
                }
            }

            Prims prims = new Prims(number_of_vertices);
            prims.primsAlgorithm(adjacency_matrix);
            prims.printMST();

        } catch (InputMismatchException inputMismatch)
        {
            System.out.println("Wrong Input Format");
        }
        scan.close();
    }
}
```

LAB – 13

```
/*
        NAME - Puroo Kulshrestha
        SEC - L-2
        ROLL NO. - 30
        University roll no. - 201500535
*/


import java.util.*;

public class kruskals {

    public static class edge implements Comparable<edge> {
        int u;
        int v;
        int weight;

        public edge(int u, int v, int weight) {
            this.u = u;
            this.v = v;
            this.weight = weight;
        }

        public String toString() {
            return this.u + " " + this.v + " " + this.weight;
        }

        @Override
        public int compareTo(edge o) {
            return this.weight - o.weight;
        }
    }

    public static void main(String[] args) {

        Scanner scn = new Scanner(System.in);
        int nodes = scn.nextInt();

        int[][] graph = new int[nodes + 1][nodes + 1];
        int numEdges = scn.nextInt();
        edge[] edges = new edge[numEdges];
        for (int edge = 0; edge < numEdges; edge++) {
            int u = scn.nextInt(), v = scn.nextInt(), w = scn.nextInt();

            graph[u][v] = graph[v][u] = w;

            edges[edge] = new edge(u, v, w);
        }

        kruskalsAlgo(nodes, numEdges, edges, graph);

    }

    public static void kruskalsAlgo(int numVertices, int numEdges, edge[]
edges, int[][] graph) {

        Arrays.sort(edges);

        int[] parents = new int[numVertices + 1];
        int[] size = new int[numVertices + 1];
        for (int vertex = 1; vertex < graph.length; vertex++) {
```

```
            parents[vertex] = vertex;
            size[vertex] = 1;
        }

        int edgeCounter = 0;
        int edgedTaken = 1;

        while (edgedTaken <= numVertices - 1) {
            edge e = edges[edgeCounter];
            edgeCounter++;

            if (isCyclic(e.u, e.v, parents))
                continue;

            union(findParent(e.u, parents), findParent(e.v, parents), parents,
size);
            mst[e.u][e.v] = e.weight;
            edgedTaken++;

        }

        for (int u = 1; u < mst.length; u++) {
            for (int v = 0; v < mst.length; v++) {
                if (mst[u][v] != 0) {
                    System.out.println(u + " " + v + " " + mst[u][v]);
                }
            }
        }

    }

    public static boolean isCyclic(int u, int v, int[] parents) {
        return findParent(u, parents) == findParent(v, parents);
    }

    public static void union(int u, int v, int[] parents, int[] size) {
         u = findParent(u, parents);
         v = findParent(v, parents);
        if (size[u] > size[v]) {
            parents[v] = u;
            size[u] += size[v];
        } else {
            parents[u] = v;
            size[v] += size[u];
        }
    }

    public static int findParent(int u, int[] parents) {

        if (parents[u] == u) {
            return u;
        } else {

            parents[u] = findParent(parents[u], parents);
            return parents[u];
        }
    }

}
```