**1a. Implement Sequential Search Algorithm and Analyze its Time Complexity**

public class BruteForSearch

{

public static int search(int[]arr,int target)

{

for(int i=0;i<arr.length;i++)

{

if(arr[i]==target) return i;

}

return-1;

}

public static void main(String[]args)

{

int[]arr={4,2,7,1,3};

System.out.println(search(arr,7));

}

}

Output: 2

**1 b. Implement finding Factorial of a given number using recursive me Complexity**

public class FactorialRecursive

{

public static long factorial(int n) {

if(n==0) {

return 1;

}

return n\*factorial(n-1);

}

public static void main(String[] args){

int num=5;

System.out.println("Factorial of"+num+"is"+factorial(num));

}

}

Output : Factorial of 5is 120

**2. Brute force Technique**

**2a. Implement selection sort algorithm and Analyze its Time Complexity**

public class selectionsort {

public static void selectionSort(int[] arr)

{

int n=arr.length;

for(int i=0;i<n-1;i++)

{

int minIndex=i;

for(int j=i+1;j<n;j++)

{

if(arr[j]<arr[minIndex])

{

minIndex =j;

}

}

swap(arr,i,minIndex);

}

}

public static void swap(int[] arr,int i,int j)

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

public static void printArray(int[] arr)

{

for(int num :arr)

{

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

}

System.out.println();

}

public static void main(String[] args) {

int[] arr={64 ,25 ,12 ,22 ,11 };

System.out.println("Unsorted Array:");

printArray(arr);

selectionSort(arr);

System.out.println("Sorted Array:");

printArray(arr);

}

}

Output: Unsorted Array

64, 25, 12, 22, 11,

Sorted Array

11, 12, 22, 25, 64

**2b. Implement Euclid’s algorithm and Analyze its Time Complexity**

public class EuclidGCD {

public static int euclidGCD(int a, int b) {

if (b == 0) {

return a;

}

return euclidGCD(b, a % b);

}

public static void main(String[] args) {

int a = 56;

int b = 98;

int gcd = euclidGCD(a, b);

System.out.println("The GCD of " + a + " and " + b + " is: " + gcd);

}

}

Output: The GCD of 56 and 98 is: 14

**3.Decrease-and-Conquer Method**

**3a. Implement Binary search algorithm and Analyze its Time Complexity**

public class BinarySearch {

public static int search(int[] arr, int target, int low, int high) {

if (low > high) return -1;

int mid = (low + high) / 2;

if (arr[mid] == target) return mid;

else if (arr[mid] < target) return search(arr, target, mid + 1, high);

else return search(arr, target, low, mid - 1);

}

public static void main(String[] args) {

int[] arr = {1, 3, 5, 7, 9};

System.out.println(search(arr, 5, 0, arr.length - 1));

}

}

Output:2

**4. Divide-and-Conquer Technique**

**4a. Implement Merge sort algorithm and Analyze its Time Complexity**

import java.util.Arrays;

public class MergeSort {

public static void mergeSort(int[] arr) {

if (arr.length <= 1) return;

int mid = arr.length / 2;

int[] left = Arrays.copyOfRange(arr, 0, mid);

int[] right = Arrays.copyOfRange(arr, mid, arr.length);

mergeSort(left);

mergeSort(right);

merge(arr, left, right);

}

private static void merge(int[] arr, int[] left, int[] right) {

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

while (i < left.length && j < right.length) {

if (left[i] < right[j]) arr[k++] = left[i++];

else arr[k++] = right[j++];

}

while (i < left.length) arr[k++] = left[i++];

while (j < right.length) arr[k++] = right[j++];

}

public static void main(String[] args) {

int[] arr = {6, 3, 8, 5, 2};

mergeSort(arr);

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

}

}

Output: [2, 3, 5, 6, 8]

**4b. Implement Quick sort algorithm and Analyze its Time Complexity**

public class QuickSort {

public static void quickSort(int[] arr,int low,int high)

{

if(low<high)

{

int pivotIndex=partition(arr,low,high);

quickSort(arr,low,pivotIndex-1);

quickSort(arr,pivotIndex+1,high);

}

}

public static int partition(int[] arr,int low,int high)

{

int pivot=arr[high];

int i=low-1;

for(int j=low;j<high;j++)

{

if(arr[j]<=pivot)

{

i++;

swap(arr,i,j);

}

}

swap(arr,i+1,high);

return i+1;

}

public static void swap(int[] arr,int i,int j) {

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

public static void main(String[] args) {

int[] arr={10,7,8,9,1,5};

int n=arr.length;

System.out.println("Unsorted Array:");

printArray(arr);

quickSort(arr,0,n-1);

System.out.println("Sorted Array:");

printArray(arr);

}

public static void printArray(int[] arr) {

for(int num : arr)

{

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

}

System.out.println();

}

}

Output:

Unsorted Array:

10 7 8 9 1 5

Sorted Array:

1 5 7 8 9 10

**5.Greedy Method**

**5a. Implement prim’s algorithm and Analyze its Time Complexity**

import java.util.Scanner;

public class PrimsAlgorithm1 {

static final int INF = 9999;

static final int MAX = 20;

static int[][] G = new int[MAX][MAX];

static int[][] spanning = new int[MAX][MAX];

static int n;

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: ");

n = scanner.nextInt();

System.out.println("\nEnter the adjacency matrix:");

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

G[i][j] = scanner.nextInt();

}

}

int totalCost = prims();

System.out.println("\nSpanning tree matrix:");

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

System.out.print(spanning[i][j] + "\t");

}

System.out.println();

}

System.out.println("\nTotal cost of the spanning tree = " + totalCost);

}

static int prims() {

int[][] cost = new int[MAX][MAX];

int[] distance = new int[MAX];

int[] from = new int[MAX];

int[] visited = new int[MAX];

int minCost = 0;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

if (G[i][j] == 0) {

cost[i][j] = INF;

} else {

cost[i][j] = G[i][j];

}

spanning[i][j] = 0;

}

}

distance[0] = 0;

visited[0] = 1;

for (int i = 1; i < n; i++) {

distance[i] = cost[0][i];

from[i] = 0;

visited[i] = 0;

}

int noOfEdges = n - 1;

while (noOfEdges > 0) {

int minDistance = INF, v = -1;

for (int i = 1; i < n; i++) {

if (visited[i] == 0 && distance[i] < minDistance) {

v = i;

minDistance = distance[i];

}

}

int u = from[v];

spanning[u][v] = distance[v];

spanning[v][u] = distance[v];

noOfEdges--;

visited[v] = 1;

for (int i = 1; i < n; i++) {

if (visited[i] == 0 && cost[i][v] < distance[i]) {

distance[i] = cost[i][v];

from[i] = v;

}

}

minCost += cost[u][v];

}

return minCost;

}

}

Output: Enter the number of vertices: 3

Enter the adjacency matrix:

1 1 0

1 0 1

0 1 0

Spanning tree matrix:

0 1 0

1 0 1

0 1 0

**5b. Dijkstra’s algorithm and Analyze its Time Complexity**

**Dijkstra’s Algorithm**

public class DijkstraAlgorithm {

public void dijkstraAlgorithm(int[][]graph,int source) {

int nodes=graph.length;

boolean[] visited\_vertex=new boolean[nodes];

int[] dist=new int[nodes];

for(int i=0;i<nodes;i++) {

visited\_vertex[i]=false;

dist[i]=Integer.MAX\_VALUE;

}

dist[source]=0;

for(int i=0;i<nodes;i++) {

int u=find\_min\_distance(dist,visited\_vertex);

visited\_vertex[u]=true;

for(int v=0;v<nodes;v++) {

if(!visited\_vertex[v]&&graph[u][v]!=0&&(dist[u]+graph[u][v]<dist[v])) {

dist[v]=dist[u]+graph[u][v];

}

}

}

for(int i=0;i<dist.length;i++) {

System.out.println(String.format("Distance from Vertex %s to Vertex %s is %s",source,i,dist[i]));

}

}

private static int find\_min\_distance(int[] dist,boolean[] visited\_vertex) {

int minimum\_distance=Integer.MAX\_VALUE;

int minimum\_distance\_vertex=-1;

for(int i=0;i<dist.length;i++) {

if(!visited\_vertex[i]&&dist[i]<minimum\_distance) {

minimum\_distance\_vertex=i;

minimum\_distance\_vertex=i;

}

}

return minimum\_distance\_vertex;

}

public static void main(String[] args) {

int graph[][]=new int[] [] {

{0,1,1,2,0,0,0},

{0,0,2,0,0,3,0},

{1,2,0,1,3,0,0},

{2,0,1,1,3,0,0},

{0,0,3,0,0,2,0},

{0,3,0,0,2,0,1},

{0,2,0,1,0,1,0}

};

DijkstraAlgorithm Test=new DijkstraAlgorithm();

Test.dijkstraAlgorithm(graph,0);

}

}

Output:

Distance from Vertex 0 to Vertex 0 is 0

Distance from Vertex 0 to Vertex 1 is 1

Distance from Vertex 0 to Vertex 2 is 1

Distance from Vertex 0 to Vertex 3 is 2

Distance from Vertex 0 to Vertex 4 is 5

Distance from Vertex 0 to Vertex 5 is 7

Distance from Vertex 0 to Vertex 6 is 8

**Pgm6 a &b : Implement Warshall and Floyd’s algorithm**

public class AllPairShortestPath {

final static int INF=99999,V=4;

void floydWarshall(int dist[][])

{

int i,j,k;

for(k=0;k<V;k++) {

for(i=0;i<V;i++) {

for(j=0;j<V;j++) {

if(dist[i][k]+dist[k][j]<dist[i][j])

dist[i][j]=dist[i][k]+dist[k][j];

}

}

}

printSolution(dist);

}

void printSolution(int dist[][])

{

System.out.println("THe following matrix shows the shortest"+"distances between every pair of vertices");

for(int i=0;i<V;i++) {

for(int j=0;j<V;j++) {

if(dist[i][j]==INF)

System.out.print("INF");

else

System.out.print(dist[i][j]+" ");

}

System.out.println();

}

}

public static void main(String[]args)

{

int graph[][]={{0,5,INF,10},

{INF,0,3,INF},

{INF,INF,0,1},

{INF,INF,INF,0}};

AllPairShortestPath a=new AllPairShortestPath();

a.floydWarshall(graph);

}

}

Output:

THe following matrix shows the shortestdistances between every pair of vertices

0 5 8 9

INF0 3 4

INFINF0 1

INFINFINF0

**Backtracking**

**7a. Implement Hamiltonian cycles algorithm**

public class HamiltonianCycle {

final int V = 5;

int path[];

boolean isSafe(int v, int graph[][], int path[], int pos) {

if (graph[path[pos - 1]][v] == 0) {

return false;

}

for (int i = 0; i < pos; i++) {

if (path[i] == v) {

return false;

}

}

return true;

}

boolean hamCycleUtil(int graph[][], int path[], int pos) {

if (pos == V) {

if(graph[path[pos - 1]][path[0]] == 1) {

return true;

} else {

return false;

}

}

for (int v = 1; v < V; v++) {

if (isSafe(v, graph, path, pos)) {

path[pos] = v;

if (hamCycleUtil(graph, path, pos + 1) == true) {

return true;

}

path[pos] = -1;

}

}

return false;

}

int hamCycle(int graph[][]) {

path = new int[V];

for (int i = 0; i < V; i++) {

path[i] = -1;

}

path[0] = 0;

if (hamCycleUtil(graph, path, 1) == false) {

System.out.println("\nSolution does not exist");

return 0;

}

printSolution(path);

return 1;

}

void printSolution(int path[]) {

System.out.println("Solution Exists: Following is one Hamiltonian Cycle");

for (int i = 0; i < V; i++) {

System.out.print(" " + path[i] + " ");

}

System.out.println(" " + path[0] + " ");

}

public static void main(String args[]) {

HamiltonianCycle hamiltonian = new HamiltonianCycle();

int graph1[][] = {{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 1},

{0, 1, 1, 1, 0}};

hamiltonian.hamCycle(graph1);

System.out.println("For Graph 2:");

int graph2[][] = {{0, 1, 0, 1, 0},

{1, 0, 1, 1, 1},

{0, 1, 0, 0, 1},

{1, 1, 0, 0, 0},

{0, 1, 1, 0, 0}};

hamiltonian.hamCycle(graph2);

}

}

OUTPUT

Solution Exists: Following is one Hamiltonian Cycle

 0  1  2  4  3  0

For Graph 2:

Solution does not exist

**Pgm 8:Implement LCM algorithm**

**Transform and Conquer Approach**

public class LCMCalculator {

private static int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

private static int lcm(int a, int b) {

return (a \* b) / gcd(a, b);

}

public static int lcmArray(int[] arr) {

int result = arr[0]; // Start with the first element

for (int i = 1; i < arr.length; i++) {

result = lcm(result, arr[i]);

}

return result;

}

public static void main(String[] args) {

int[] numbers = {12, 15, 20, 25};

int result = lcmArray(numbers);

System.out.println("LCM of the array is: " + result);

}

}

**OUTPUT:** LCM of the array is: 300