Final DAA Programs

1. Mergesort for N numbers from keyboard using Java and also display the time complexity

```
import java.util.Arrays;

 import java.util.Scanner;

public class MergeSort {

public static void mergeSort(int[] arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid); mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right); }

}

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

int n1 = mid - left + 1;
int n2 = right - mid;

int[] leftArray = new int[n1]; int[] rightArray = new int[n2];

for (int i = 0; i < n1; i++) { leftArray[i] = arr[left + i];

}
for (int j = 0; j < n2; j++) {

rightArray[j] = arr[mid + 1 + j]; }

int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
if (leftArray[i] <= rightArray[j]) {

arr[k++] = leftArray[i++]; } else {

arr[k++] = rightArray[j++]; }

}

while (i < n1) {
arr[k++] = leftArray[i++];

}

while (j < n2) {
arr[k++] = rightArray[j++];

}

}

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements: "); int n = scanner.nextInt();
int[] arr = new int[n];

System.out.println("Enter the elements:"); for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt(); }

System.out.println("Original array: " + Arrays.toString(arr));

long startTime = System.nanoTime();

mergeSort(arr, 0, arr.length - 1);

long endTime = System.nanoTime();

System.out.println("Sorted array: " + Arrays.toString(arr));

double timeElapsed = (endTime - startTime) / 1e6; // Convert nanoseconds to milliseconds

System.out.println("Time complexity: " + timeElapsed + " milliseconds");

scanner.close(); }
```

```
}
```

(or)

```
import java.util.Scanner;
import java.util.Random;
public class MergeSortExp {
        public static void mergeSort(int[] a,int low,int high)
        {
                int N=high-low;
                if(N<=1)
                        return;
                int mid=low+(N/2);
                mergeSort(a,low,mid);
                mergeSort(a,mid,high);
                int[] temp=new int[N];
                int i=low, j=mid;
                for(int k=0;k<N;k++)
                {
                        if(i==mid)
                                temp[k]=a[j++];
                        else if(j==high)
                                temp[k]=a[i++];
                        else if(a[j]<a[i])
                                temp[k]=a[j++];
                        else
                                temp[k]=a[i++];
                }
                for(int k=0;k<N;k++)
                        a[low++]=temp[k];
        }
        public static void main(String[] args) {
                Scanner scan = new Scanner(System.in);
                int i;
                Random r = new Random();
                System.out.println("Merge Sort\nEnter the Number of times the algorithm should
Run");
```

```
		int times = scan.nextInt();
		double totaldur=0;
		for(int j=0;j<times;j++)
		{
			System.out.println("Random Number Generated are at POS  "+j+"  as 
follows : ");
			int[] a = new int[10];
			for(i=0;i<10;i++)
			{
				a[i]=r.nextInt(1000);
				System.out.print(a[i]+" ");
			}
			System.out.println("");
			long StartTime = System.nanoTime();
			mergeSort(a,0,10);
			double EndTime = System.nanoTime();
			double duration = (EndTime - StartTime);
			System.out.println("Elements after Sorting are");
			for(i=0;i<10;i++)
				System.out.print(a[i]+" ");
			System.out.println("");
			totaldur=totaldur+duration;
		}
		System.out.println("\nTotal time taken to Sort is :"+totaldur+" Nano Seconds");
		double miliseconds = (totaldur / 1000000);
		System.out.println("\nTotal time taken to Sort is :"+miliseconds+" Mili 
Seconds");
		double avg=totaldur/times;
		System.out.println("The Average time Spend by the System is : "+avg+" Nano 
Second");
		double miliavg=(avg/1000000);
		System.out.println("The Avergae time Spend by the System is : "+miliavg+" Mili 
Seconds");
		scan.close();
	}
}
```

2. Quicksort - N numbers read from keyboard

```
import java.util.Random;
import java.util.Scanner;
public class quicksort {
static int max=2000;
int partition (int[] a, int low,int high)
{
int p,i,j,temp;
p=a[low];
i=low+1;
j=high;
while(low<high)
{
while(a[i]<=p&&i<high)
i++;
while(a[j]>p)
j--;
if(i<j)
{
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
else
{
temp=a[low];
a[low]=a[j];
a[j]=temp;
return j;
}
}
return j;
}
void sort(int[] a,int low,int high)
{
if(low<high)
{
int s=partition(a,low,high);
sort(a,low,s-1);
sort(a,s+1,high);
}
}
public static void main(String[] args) {
// TODO Auto-generated method stub
int[] a;
int i;
System.out.println("Enter the array size");
Scanner sc =new Scanner(System.in);
int n=sc.nextInt();
a= new int[max];
Random generator=new Random();
for( i=0;i<n;i++)
a[i]=generator.nextInt(20);
System.out.println("Array before sorting");
for( i=0;i<n;i++)
System.out.println(a[i]+" ");
long startTime=System.nanoTime();
quicksort m=new quicksort();
m.sort(a,0,n-1);
long stopTime=System.nanoTime();
long elapseTime=(stopTime-startTime);
System.out.println("Time taken to sort array is:"+elapseTime+"nano

seconds");

System.out.println("Sorted array is");
for(i=0;i<n;i++)
System.out.println(a[i]);

}
}
```

3. Perform insert and delete operations in Binary Search Tree.

```
class BinarySearchTree { static class Node {

int key;
Node left, right;

public Node(int item) { key = item;
left = right = null;
```

```
} }

// Root of the BST Node root;

BinarySearchTree() { root = null;

}

// Insert a key into the BST void insert(int key) {

root = insertRec(root, key); }

Node insertRec(Node root, int key) { if (root == null) {

root = new Node(key);

return root; }

if (key < root.key) {
root.left = insertRec(root.left, key);

} else if (key > root.key) {
root.right = insertRec(root.right, key);

}

return root; }

// Delete a key from the BST void delete(int key) {

root = deleteRec(root, key); }

Node deleteRec(Node root, int key) { if (root == null) {

return root; }

if (key < root.key) {
root.left = deleteRec(root.left, key);

} else if (key > root.key) {
root.right = deleteRec(root.right, key);

} else {
// Node with only one child or no child if (root.left == null) {
```

```
return root.right;
} else if (root.right == null) {

return root.left; }

// Node with two children root.key = minValue(root.right);

// Delete the in-order successor

root.right = deleteRec(root.right, root.key); }

return root; }

int minValue(Node root) { int minValue = root.key; while (root.left != null) {

minValue = root.left.key;

root = root.left; }

return minValue; }

// Print the inorder traversal of the tree void inorder() {

inorderRec(root);

}

void inorderRec(Node root) { if (root != null) {

inorderRec(root.left); System.out.print(root.key + " "); inorderRec(root.right);

} }

// Main method for testing the BST operations public static void main(String[] args) {

BinarySearchTree tree = new BinarySearchTree();

// Insert elements tree.insert(50); tree.insert(30); tree.insert(20); tree.insert(40); tree.insert(70);
tree.insert(60); tree.insert(80);

System.out.println("Inorder traversal:"); tree.inorder();
System.out.println();

// Delete elements System.out.println("Delete 20:"); tree.delete(20);
tree.inorder(); System.out.println();
```

```
System.out.println("Delete 30:"); tree.delete(30);
tree.inorder(); System.out.println();

} }
```

4. Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
import java.util.LinkedList; import java.util.Queue; import java.util.Scanner;

public class BFSDirectedGraph {

public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: "); int vertices = scanner.nextInt();

int[][] adjacencyMatrix = new int[vertices][vertices];

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

for (int i = 0; i < vertices; i++) { for (int j = 0; j < vertices; j++) {

adjacencyMatrix[i][j] = scanner.nextInt(); }

}

System.out.print("Enter the starting vertex: "); int startVertex = scanner.nextInt();

System.out.println("BFS traversal starting from vertex " + startVertex + ":");
bfs(adjacencyMatrix, startVertex, vertices);

scanner.close(); }

private static void bfs(int[][] adjacencyMatrix, int startVertex, int vertices) { boolean[] visited =
new boolean[vertices];
Queue<Integer> queue = new LinkedList<>();

visited[startVertex] = true; queue.add(startVertex);

while (!queue.isEmpty()) {
int currentVertex = queue.poll(); System.out.print(currentVertex + " ");
for (int i = 0; i < vertices; i++) {
if (!visited[i]) {

visited[i] = true;

queue.add(i); }

} }

} }
```

5. a) Obtain Topological ordering of the vertices in a given digraph.

```
import java.util.*;

public class TopologicalSort { private int V; // Number of vertices private List<Integer>
adjList[];

public TopologicalSort(int v) { V = v;

adjList = new LinkedList[v]; for (int i = 0; i < v; ++i)

adjList[i] = new LinkedList<>(); }

// Function to add an edge to the graph private void addEdge(int v, int w) {

adjList[v].add(w); }

// Recursive function to perform topological sort
private void topologicalSortUtil(int v, boolean visited[], Stack<Integer> stack) {

visited[v] = true;

for (Integer neighbor : adjList[v]) { if (!visited[neighbor])

topologicalSortUtil(neighbor, visited, stack); }

stack.push(v); }

// Function to perform topological sort private void topologicalSort() {

Stack<Integer> stack = new Stack<>();
boolean visited[] = new boolean[V]; Arrays.fill(visited, false);

for (int i = 0; i < V; ++i) { if (!visited[i])

topologicalSortUtil(i, visited, stack); }

// Print the topological order System.out.println("Topological Sort:"); while (!stack.isEmpty())

System.out.print(stack.pop() + " "); }

public static void main(String args[]) {
Scanner scanner = new Scanner(System.in);

// Taking input for the number of vertices System.out.print("Enter the number of vertices: "); int
V = scanner.nextInt();

TopologicalSort g = new TopologicalSort(V);

// Taking input for the adjacency matrix System.out.println("Enter the adjacency matrix:"); for
(int i = 0; i < V; i++) {

for (int j = 0; j < V; j++) {
if (scanner.nextInt() == 1) {

g.addEdge(i, j); }

} }

g.topologicalSort();

scanner.close(); }

}
```

b). Compute the Transitive closure of a given directed graph using Warshall's Algorithm.

```
import java.util.Scanner;

public class WarshallsAlgorithm {
public static void main(String[] args) {
```

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

// Get the number of vertices System.out.print("Enter the number of vertices: "); int vertices =
scanner.nextInt();

// Initialize the adjacency matrix
int[][] graph = new int[vertices][vertices];

// Get the adjacency matrix from the user
System.out.println("Enter the adjacency matrix (0 for no edge, 1 for edge):"); for (int i = 0; i <
vertices; i++) {

for (int j = 0; j < vertices; j++) { graph[i][j] = scanner.nextInt();

} }

// Find the transitive closure using Warshall's Algorithm for (int k = 0; k < vertices; k++) {

for (int i = 0; i < vertices; i++) { for (int j = 0; j < vertices; j++) {

graph[i][j] = graph[i][j] | (graph[i][k] & graph[k][j]); }

} }

// Display the transitive closure System.out.println("Transitive Closure:"); for (int i = 0; i <
vertices; i++) {

for (int j = 0; j < vertices; j++) { System.out.print(graph[i][j] + " ");

}

System.out.println(); }

scanner.close(); }

}
```

6. a) Check whether the given graph is connected or not using DFS method.

```
import java.util.Scanner;
```

```
public class GraphDFS {
private int vertices;
private int[][] adjacencyMatrix;

public GraphDFS(int v) {
vertices = v;
adjacencyMatrix = new int[v][v];

}

public void addEdge(int start, int end) { adjacencyMatrix[start][end] = 1;
adjacencyMatrix[end][start] = 1;

}

public void dfs(int startVertex, boolean[] visited) { System.out.print(startVertex + " ");
visited[startVertex] = true;

for (int i = 0; i < vertices; i++) {
if (adjacencyMatrix[startVertex][i] == 1 && !visited[i]) {

dfs(i, visited); }

} }

public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: "); int v = scanner.nextInt();

GraphDFS graph = new GraphDFS(v);

System.out.println("Enter the adjacency matrix:"); for (int i = 0; i < v; i++) {

for (int j = 0; j < v; j++) { graph.adjacencyMatrix[i][j] = scanner.nextInt();

} }

System.out.print("Enter the starting vertex for DFS: "); int startVertex = scanner.nextInt();

boolean[] visited = new boolean[v];
System.out.print("DFS traversal starting from vertex " + startVertex + ": ");
graph.dfs(startVertex, visited);

scanner.close(); }
}
```

b) Implement the Floyd's algorithm for all pairs Shortest path problem.

```
import java.util.Scanner;

public class floyd {
void flyd(int[][] w,int n)

{

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

for(i=1;i<=n;i++)
for(j=1;j<=n;j++) w[i][j]=Math.min(w[i][j], w[i][k]+w[k][j]);

}
public static void main(String[] args) {

int a[][]=new int[10][10]; int n,i,j;

System.out.println("enter the number of vertices"); Scanner sc=new Scanner(System.in);
n=sc.nextInt();
System.out.println("Enter the weighted matrix"); for(i=1;i<=n;i++)

for(j=1;j<=n;j++) a[i][j]=sc.nextInt();

floyd f=new floyd(); f.flyd(a, n);

System.out.println("The shortest path matrix is"); for(i=1;i<=n;i++)

{
{

} System.out.println();

    }
sc.close();

}
}
```

7. HeapSort

```
import java.util.Arrays; import java.util.Scanner;

public class HeapSort {

public static void heapSort(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); }

}

public static void heapify(int[] arr, int n, int i) { int largest = i;
int left = 2 * i + 1;
int right = 2 * i + 2;

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

}

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

}

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

heapify(arr, n, largest); }

}

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);
System.out.print("Enter the number of elements: "); int n = scanner.nextInt();
int[] arr = new int[n];

System.out.println("Enter the elements:"); for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt(); }

System.out.println("Original array: " + Arrays.toString(arr)); long startTime =
System.nanoTime();
heapSort(arr);
long endTime = System.nanoTime(); System.out.println("Sorted array: " + Arrays.toString(arr));

double timeElapsed = (endTime - startTime) / 1e6; // Convert nanoseconds to milliseconds

System.out.println("Time complexity: " + timeElapsed + " milliseconds");

scanner.close(); }

}
```

8. Horspool string matching algorithm

```
import java.util.Scanner;
public class HorspoolStringMatching {

public static int[] shiftTable(String pattern) { int[] table = new int[256];
int m = pattern.length();

for (int i = 0; i < 256; i++) { table[i] = m;

}

for (int i = 0; i < m - 1; i++) { table[pattern.charAt(i)] = m - 1 - i;

}

return table; }

public static int horspoolSearch(String text, String pattern) { int n = text.length();

int m = pattern.length();
int[] table = shiftTable(pattern);
int i = m - 1; while (i < n) {

int k = 0;
while (k < m && pattern.charAt(m - 1 - k) == text.charAt(i - k)) {

k++; }

if (k == m) {
return i - m + 1; // Pattern found at index i - m + 1

} else {
i += table[text.charAt(i)];

} }

return -1; // Pattern not found }

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

System.out.print("Enter the text: "); String text = scanner.nextLine();

System.out.print("Enter the pattern to search for: "); String pattern = scanner.nextLine();

long startTime = System.nanoTime();
int index = horspoolSearch(text, pattern); long endTime = System.nanoTime();

if (index != -1) {
System.out.println("Pattern found at index: " + index);

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

}

double timeElapsed = (endTime - startTime) / 1e6; // Convert nanoseconds to milliseconds

System.out.println("Time complexity: " + timeElapsed + " milliseconds");

scanner.close(); }

}
```

9. 0/1 Knapsack Problem using Dynamic Programming

```
import java.util.Scanner; public class KnapsackDP {

public static int knapsack(int[] weights, int[] values, int capacity) { int n = weights.length;
int[][] dp = new int[n + 1][capacity + 1];

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

if (i == 0 || w == 0) { dp[i][w] = 0;

} else if (weights[i - 1] <= w) {
dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

} else {
dp[i][w] = dp[i - 1][w];

} }

} displayMatrix(dp);

return dp[n][capacity]; }

public static void displayMatrix(int[][] matrix) { for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) { System.out.print(matrix[i][j] + " ");

}

System.out.println(); }

}

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of items: "); int n = scanner.nextInt();
int[] weights = new int[n];
int[] values = new int[n];

System.out.println("Enter the weights of the items:"); for (int i = 0; i < n; i++) {

weights[i] = scanner.nextInt(); }
System.out.println("Enter the values of the items:"); for (int i = 0; i < n; i++) {

values[i] = scanner.nextInt(); }

System.out.print("Enter the knapsack capacity: "); int capacity = scanner.nextInt();

System.out.println("Dynamic Programming Matrix:"); int maxValue = knapsack(weights, values, capacity);

System.out.println("Maximum value: " + maxValue);

scanner.close(); }

}
```

10. Prim's algorithm

```
import java.util.Scanner;

public class prims {

public static void main(String[] args) { int w[][]=new int[10][10];
int n,i,j,s,k=0;
int min;

int sum=0;
int u=0,v=0;
int flag=0;
int sol[]=new int[10];
System.out.println("Enter the number of vertices"); Scanner sc=new Scanner(System.in);

n=sc.nextInt();
for(i=1;i<=n;i++)
sol[i]=0;
System.out.println("Enter the weighted graph"); for(i=1;i<=n;i++)

for(j=1;j<=n;j++)
w[i][j]=sc.nextInt(); System.out.println("Enter the source vertex"); s=sc.nextInt();
sol[s]=1;
k=1;
while (k<=n-1)
{
min=99; for(i=1;i<=n;i++)

for(j=1;j<=n;j++) if(sol[i]==1&&sol[j]==0)

if(i!=j&&min>w[i][j]) {

                                min=w[i][j];
                                 u=i;
                                 v=j;


}

sol[v]=1;
sum=sum+min;
k++; System.out.println(u+"->"+v+"="+min); }

for(i=1;i<=n;i++) if(sol[i]==0) flag=1;

if(flag==1)
System.out.println("No spanning tree");

else

System.out.println("The cost of minimum spanning tree is"+sum); sc.close();

} }
```

11. Kruskal's algorithm

```
import java.util.Scanner;

public class kruskal { int parent[]=new int[10]; int find(int m)
{

int p=m; while(parent[p]!=0) p=parent[p]; return p;

}
void union(int i,int j) {

if(i<j) parent[i]=j; else parent[j]=i;
}
void krkl(int[][]a, int n) {

int u=0,v=0,min,k=0,i,j,sum=0; while(k<n-1)
{

min=99; for(i=1;i<=n;i++) for(j=1;j<=n;j++) if(a[i][j]<min&&i!=j) {

                    min=a[i][j];
                     u=i;
                     v=j;


} i=find(u); j=find(v); if(i!=j) {

union(i,j); System.out.println("("+u+","+v+")"+"="+a[u][v]); sum=sum+a[u][v];
k++;

               }
                a[u][v]=a[v][u]=99;
                }


System.out.println("The cost of minimum spanning tree = "+sum); }

public static void main(String[] args) {
int a[][]=new int[10][10];
int i,j;
System.out.println("Enter the number of vertices of the graph"); Scanner sc=new
Scanner(System.in);

int n;
n=sc.nextInt();
System.out.println("Enter the wieghted matrix"); for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
a[i][j]=sc.nextInt();
kruskal k=new kruskal();
k.krkl(a,n);
sc.close();
}

}
```

12. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
import java.util.Scanner;
public class Dijkstra {

int d[]=new int[10];
int p[]=new int[10];
int visited[]=new int[10];
public void dijk(int[][]a, int s, int n)
{
int u=-1,v,i,j,min;
for(v=0;v<n;v++)
{
d[v]=99;
p[v]=-1;
} d[s]=0;
for(i=0;i<n;i++){
min=99;
for(j=0;j<n;j++){
if(d[j]<min&& visited[j]==0)
{
min=d[j];
u=j;
}}
visited[u]=1;
for(v=0;v<n;v++){ if((d[u]+a[u][v]<d[v])&&(u!=v)&&visited[v]==0) {
d[v]=d[u]+a[u][v];
p[v]=u;
}
}
}
}
void path(int v,int s)
{
if(p[v]!=-1)
path(p[v],s);
if(v!=s)
System.out.print("->"+v+" ");
}
void display(int s,int n){
int i;
for(i=0;i<n;i++)
{

if(i!=s){
System.out.print(s+" ");
path(i,s);

}
if(i!=s)
System.out.print("="+d[i]+" "); System.out.println();

}
}
public static void main(String[] args) { int a[][]=new int[10][10];

int i,j,n,s;
System.out.println("enter the number of vertices");
Scanner sc = new Scanner(System.in);
n=sc.nextInt();
System.out.println("enter the weighted matrix");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
a[i][j]=sc.nextInt();
System.out.println("enter the source vertex");
s=sc.nextInt();
Dijkstra tr=new Dijkstra();
tr.dijk(a,s,n);
System.out.println("the shortest path between source"+s+"to remaining vertices are");
tr.display(s,n);
sc.close();
}
}
```

13. **Travelling Sales Person problem** using Dynamic programming:

```
import java.util.Scanner;
class TSPExp {
int weight[][],n,tour[],finalCost;

final int INF=1000;
TSPExp()
{
Scanner s=new Scanner(System.in); System.out.println("Enter no. of nodes:=>"); n=s.nextInt();

weight=new int[n][n]; tour=new int[n-1]; for(int i=0;i<n;i++) {

for(int j=0;j<n;j++)
{
if(i!=j)
{
System.out.print("Enter weight of "+(i+1)+" to "+(j+1)+":=>"); weight[i][j]=s.nextInt();

} } }

System.out.println();
System.out.println("Starting node assumed to be node 1."); eval();
}
public int COST(int currentNode,int inputSet[],int setSize)
{
if(setSize==0)
return weight[currentNode][0];
int min=INF;
int setToBePassedOnToNextCallOfCOST[]=new int[n-1];
for(int i=0;i<setSize;i++)
{
int k=0;//initialise new set
for(int j=0;j<setSize;j++)
{
if(inputSet[i]!=inputSet[j]) setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];
}
int temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);
if((weight[currentNode][inputSet[i]]+temp) <
min)
{
min=weight[currentNode][inputSet[i]]+temp;
}
}
return min;
}
public int MIN(int currentNode,int inputSet[],int setSize)
{
if(setSize==0)
return weight[currentNode][0];
int min=INF,minindex=0;
int setToBePassedOnToNextCallOfCOST[]=new int[n-1];
for(int i=0;i<setSize;i++)//considers each node of inputSet
{
int k=0;
for(int j=0;j<setSize;j++)
{
if(inputSet[i]!=inputSet[j]) setToBePassedOnToNextCallOfCOST[k++]=inputSet[j];
}
int temp=COST(inputSet[i],setToBePassedOnToNextCallOfCOST,setSize-1);
if((weight[currentNode][inputSet[i]]+temp) < min)
{
min=weight[currentNode][inputSet[i]]+temp;
minindex=inputSet[i];
}
}
return minindex;
}
public void eval()
{
int dummySet[]=new int[n-1];
for(int i=1;i<n;i++)
dummySet[i-1]=i;
finalCost=COST(0,dummySet,n-1);
constructTour();
}
public void constructTour()

{
int previousSet[]=new int[n-1];
int nextSet[]=new int[n-2]; for(int i=1;i<n;i++) previousSet[i-1]=i;
int setSize=n-1; tour[0]=MIN(0,previousSet,setSize);
for(int i=1;i<n-1;i++)
{
int k=0;
for(int j=0;j<setSize;j++)
{
if(tour[i-1]!=previousSet[j]) nextSet[k++]=previousSet[j];
}
--setSize;
tour[i]=MIN(tour[i-1],nextSet,setSize);
for(int j=0;j<setSize;j++) previousSet[j]=nextSet[j];
}
display();
}
public void display()
{
System.out.println();
System.out.print("The tour is 1-");
for(int i=0;i<n-1;i++) System.out.print((tour[i]+1)+"-"); System.out.print("1");
System.out.println();
System.out.println("The final cost is "+finalCost); }
}
class TSP
{
public static void main(String args[])
{
TSPExp obj=new TSPExp();
}
}
```

14. N Queens Problem, Reads N value from keyboard

```
import java.util.Scanner;
public class NQueensBacktracking {

private static boolean isSafe(int[][] board, int row, int col, int N) { for (int i = 0; i < col; i++) {

if (board[row][i] == 1) { return false;
} }

for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) { if (board[i][j] == 1) {

return false; }

}
for (int i = row, j = col; i < N && j >= 0; i++, j--) {

if (board[i][j] == 1) { return false;

} }

return true; }

private static boolean solveNQueensUtil(int[][] board, int col, int N) { if (col >= N) {

return true; }

for (int i = 0; i < N; i++) {
if (isSafe(board, i, col, N)) {

board[i][col] = 1;

if (solveNQueensUtil(board, col + 1, N)) { return true;

}

board[i][col] = 0; // Backtrack }

}

return false; }

public static boolean solveNQueens(int N) { int[][] board = new int[N][N];

if (!solveNQueensUtil(board, 0, N)) { return false;

}

displayBoard(board, N);

return true; }

public static void displayBoard(int[][] board, int N) { for (int i = 0; i < N; i++) {
```

```
for (int j = 0; j < N; j++) { System.out.print(board[i][j] + " ");

}

System.out.println(); }

}

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

System.out.print("Enter the value of N: "); int N = scanner.nextInt();

long startTime = System.nanoTime(); boolean solutionExists = solveNQueens(N); long endTime
= System.nanoTime();

if (!solutionExists) {
System.out.println("No solution exists for N = " + N);

}

double timeElapsed = (endTime - startTime) / 1e6; // Convert nanoseconds to milliseconds

System.out.println("\nTime complexity: " + timeElapsed + " milliseconds");

scanner.close(); }

}
```

15. Java program that reads inputs from the keyboard to solve the Subset Sum problem using dynamic programming and displays the time complexity, including finding and displaying the actual subsets that contribute to the sum

```
import java.util.ArrayList; import java.util.Scanner;

public class SubsetSumDP {

public static boolean subsetSum(int[] arr, int sum, ArrayList<Integer> subset) { int n =
arr.length;
boolean[][] dp = new boolean[n + 1][sum + 1];
```

```
for (int i = 0; i <= n; i++) { dp[i][0] = true;

}

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

if (j >= arr[i - 1]) {
dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];

} else {
dp[i][j] = dp[i - 1][j];

} }

}

if (!dp[n][sum]) { return false;

}

int i = n, j = sum;
while (i > 0 && j > 0) {

if (dp[i][j] != dp[i - 1][j]) { subset.add(arr[i - 1]);
```

Java program that reads inputs from the keyboard to solve the Subset Sum problem

using dynamic programming and displays the time complexity, including finding and

displaying the actual subsets that contribute to the sum

```
j -= arr[i - 1]; }

i--; }

return true; }

public static void main(String[] args) { Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements: "); int n = scanner.nextInt();
int[] arr = new int[n];

System.out.println("Enter the elements:"); for (int i = 0; i < n; i++) {
```

```
arr[i] = scanner.nextInt(); }

System.out.print("Enter the target sum: "); int sum = scanner.nextInt();

ArrayList<Integer> subset = new ArrayList<>();
long startTime = System.nanoTime();
boolean hasSubsetSum = subsetSum(arr, sum, subset); long endTime = System.nanoTime();

System.out.println("Subset sum exists: " + hasSubsetSum); if (hasSubsetSum) {

System.out.println("Subset contributing to the sum: " + subset); }

double timeElapsed = (endTime - startTime) / 1e6; // Convert nanoseconds to milliseconds

System.out.println("Time complexity: " + timeElapsed + " milliseconds");

scanner.close(); }

}
```