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();
    }
}
```
OUTPUT
```
Enter the number of elements: 5
Enter the elements:
100
55
33
888
11
Original array: [100, 55, 33, 888, 11]

Sorted array: [11, 33, 55, 100, 888]

Time complexity: 0.00979 milliseconds
```

2) Quicksort - N numbers read from keyboard

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

public class QuickSort {

    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int partitionIndex = partition(arr, low, high);

            quickSort(arr, low, partitionIndex - 1);
            quickSort(arr, partitionIndex + 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++;

                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }

    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();

        quickSort(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();
    }
}
```

OUTPUT

```
Enter the number of elements: 5
Enter the elements:
55
33
77
22
88
Original array: [55, 33, 77, 22, 88]
Sorted array: [22, 33, 55, 77, 88]

Time complexity: 0.00641 milliseconds
```

5b) // Java program for Floyd Warshall All Pairs Shortest Path algorithm.

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

class AllPairShortestPath {
    final static int INF = 99999, V = 4;

    void floydWarshall(int dist[][])
    {

        int i, j, k;

        /* Add all vertices one by one
           to the set of intermediate
           vertices.
          ---> Before start of an iteration,
               we have shortest
               distances between all pairs
               of vertices such that
               the shortest distances consider
               only the vertices in
               set {0, 1, 2, .. k-1} as
               intermediate vertices.
          ----> After the end of an iteration,
                vertex no. k is added
                to the set of intermediate
                vertices and the set
                becomes {0, 1, 2, .. k} */
        for (k = 0; k < V; k++) {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++) {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++) {
                    // If vertex k is on the shortest path
                    // from i to j, then update the value of
                    // dist[i][j]
                    if (dist[i][k] + dist[k][j]
                        < dist[i][j])
                        dist[i][j]
                            = dist[i][k] + dist[k][j];
                }
            }
        }

        // Print the shortest distance matrix
        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();
        }
    }

    // Driver's code
    public static void main(String[] args)
    {
        /* Let us create the following weighted graph
           10
        (0)------->(3)
        |         /|\
        5 |          |
        |          | 1
        \|/         |
        (1)------->(2)
           3           */
```

```
        int graph[][] = { { 0, 5, INF, 10 },
                          { INF, 0, 3, INF },
                          { INF, INF, 0, 1 },
                          { INF, INF, INF, 0 } };
        AllPairShortestPath a = new AllPairShortestPath();

        // Function call
        a.floydWarshall(graph);
    }
}
```

```
The following matrix shows the shortest distances between every
pair of vertices
0   5   8   9
INF 0   3   4
INF INF 0   1
INF INF INF 0
```

6 B) Floyd's algorithm:

```
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++)
      {
            for(j=1;j<=n;j++)
            {
                  System.out.print(a[i][j]+" ");
            }
      System.out.println();
      }
sc.close();
}

}
```

Output:

```
enter the number of vertices
4
Enter the weighted matrix
0 99 3 99
2 0 99 99
99 7 0 1
6 99 99 0
The shortest path matrix is
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

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();
    }
}
```

OUTPUT

```
Enter the number of elements: 5
Enter the elements:
33
56
11
77
22
Original array: [33, 56, 11, 77, 22]
Sorted array: [11, 22, 33, 56, 77]
Time complexity: 0.01018 milliseconds
```

8. Java program that reads inputs from the keyboard, implements the Horspool string matching algorithm to search for a pattern in a given text, and displays the time complexity:

```
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();
    }
}
```

This program reads the text and the pattern to search for from the keyboard, implements the Horspool string matching algorithm, and displays whether the pattern is found along with the time complexity in milliseconds.

Remember to import the `java.util.Scanner` package at the beginning of your Java file to use the `Scanner` class for reading inputs.

OUTPUT

```
Enter the text: computer science
Enter the pattern to search for: put
Pattern found at index: 3
Time complexity: 0.0295 milliseconds
```

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();
    }
}
```

OUTPUT

```
Enter the number of items: 3
Enter the weights of the items:
1
2
3
Enter the values of the items:
```

```
10
12
14
Enter the knapsack capacity: 5
Dynamic Programming Matrix:
0 0 0 0 0 0
0 10 10 10 10 10
0 10 12 22 22 22
 0 10 12 22 24 26
Maximum value: 26
```

(10) **Prim's algorithm**. Implement the program in Java language.

```java
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();
}
}
```

Output:

```
Enter the number of vertices
6
Enter the weighted graph
0 3 99 99 6 5
3 0 1 99 99 4
99 1 0 6 99 4
99 99 6 0 8 5
6 99 99 8 0 2
5 4 4 5 2 0
Enter the source vertex
1
1->2=3
2->3=1
2->6=4
6->5=2
6->4=5
The cost of minimum spanning tree is15
```

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();
        }
}
```

Output:

```
Enter the number of vertices of the graph
6
Enter the wieghted matrix
0 3 99 99 6 5
3 0 1 99 99 4
99 1 0 6 99 4
99 99 6 0 8 5
6 99 99 8 0 2
5 4 4 5 2 0
(2,3)=1
(5,6)=2
(1,2)=3
(2,6)=4
(4,6)=5
The cost of minimum spanning tree = 15
```

12) From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Write the program in Java.

```
import java.util.Scanner;
public class Dijkstra {
/**
* @param args
*/
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();
}
}
```

Output:

```
enter the number of vertices
5
enter the weighted matrix
0 3 99 7 99
3 0 4 2 99
99 4 0 5 6
5 2 5 0 4
99 99 6 4 0
enter the source vertex
0
the shortest path between source0to remaining vertices are
0 ->1 =3
0 ->1 ->2 =7
0 ->1 ->3 =5
0 ->1 ->3 ->4 =9
```

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();
}
}
}
```

```java
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();
}
}
```

Output:

```
Enter no. of nodes:=>
4
Enter weight of 1 to 2:=>2
Enter weight of 1 to 3:=>5
Enter weight of 1 to 4:=>7
Enter weight of 2 to 1:=>2
Enter weight of 2 to 3:=>8
Enter weight of 2 to 4:=>3
Enter weight of 3 to 1:=>5
Enter weight of 3 to 2:=>8
Enter weight of 3 to 4:=>1
Enter weight of 4 to 1:=>7
Enter weight of 4 to 2:=>3
Enter weight of 4 to 3:=>1
Starting node assumed to be node 1.
The tour is 1-2-4-3-1
The final cost is 11
```

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();
    }
}
```

OUTPUT

```
Enter the value of N: 4
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
Time complexity: 13.266379 milliseconds


Enter the value of N: 8
```

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
Time complexity: 16.839419 milliseconds
```

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**

```java
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]);
                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();
    }
}
```

OUTPUT

```
Enter the number of elements: 4
Enter the elements:
1
2
3
4
Enter the target sum: 5
Subset sum exists: true
Subset contributing to the sum: [3, 2]
Time complexity: 0.1338 milliseconds
```