1. **Design and implement a Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

```
import java.util.Random;
import java.util.Scanner;
public class SelectionSort
{
	public static void main(String args[])
	 {
		int n, i, j,min;
		int arr[] = new int[6000];
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter the Array Size :");
		n = scan.nextInt();
		Random generator=new Random();
		for( i=0;i<n;i++)
		arr[i]=generator.nextInt(100);
		System.out.println("Array before sorting is: ");
		for( i=0;i<n;i++)
		System.out.print(arr[i]+"  ");
		long startTime=System.nanoTime();
		for(i=0; i<n-1; i++)
		{
			min=i;
			for(j=i+1; j<n; j++)
			{
				if(arr[i] > arr[j])
				{
					min = arr[i];
					arr[i] = arr[j];
					arr[j] = min;
```

```
                        }
                    }
                }
                long stopTime=System.nanoTime();
                System.out.println("\nArray after Sorting is :");
                for(i=0; i<n; i++)
                {
                    System.out.print(arr[i]+ "  ");
                }
                long elapseTime=stopTime-startTime;
                System.out.println("\n Time taken to sort array is : "+elapseTime+
                "nanoseconds");
        }
}
```

Results:

```
Enter the Array Size :
10
Array before sorting is:
76  15  15  58  89  70  67  74  56  30
Array after Sorting is :
15  15  30  56  58  67  70  74  76  89
Time taken to sort array is : 4226nanoseconds
```

OBSERVATION TABLE / LOOKUP TABLE / TRUTH

TABLE: Observation table: Run the algorithm for following

Input Size:

Elements	1000	2000	3000	4000	5000	6000
Time Taken	4935613	12557726	8048330	13494378	19323177	26878580

2. Design and implement a Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
import java.util.Random;
import  java.util.Scanner;
public class merge
{
        static int max=30000;
        void merge( int[] array,int low, int mid,int high)
        {
                int i=low;
                int j=mid+1;
                int k=low;
                int[]resarray;
                resarray=new int[max];
                while(i<=mid&&j<=high)
                {
                        if(array[i]<array[j])
                        {
                                resarray[k]=array[i];
                                i++;
                                k++;
                        }
                        else
                        {
                                resarray[k]=array[j];
                                j++;
                                k++;
                        }
                }
                while(i<=mid)
```

```
            resarray[k++]=array[i++];
            while(j<=high)
            resarray[k++]=array[j++];
            for(int m=low;m<=high;m++)
            array[m]=resarray[m];
    }
    void sort( int[] array,int low,int high)
    {
            if(low<high)
            {
                    int mid=(low+high)/2;
                    sort(array,low,mid);
                    sort(array,mid+1,high);
                    merge(array,low,mid,high);
            }
    }
    public static void main(String[] args)
    {
            int[] array;
            int i;
            System.out.println("Enter the array size:");
            Scanner sc =new Scanner(System.in);
            int n=sc.nextInt(); array= new int[max];
            Random generator=new Random();
            for( i=0;i<n;i++) array[i]=generator.nextInt(20);
          System.out.println("Array before sorting is:");
            for( i=0;i<n;i++)
            System.out.print(array[i]+"  ");
            long startTime=System.nanoTime();
            merge m=new merge();
            m.sort(array,0,n-1);
            long stopTime=System.nanoTime();
            System.out.println("\nArray after sorting is:");
            for(i=0;i<n;i++)
            System.out.print(array[i]+"  ");
```

```
                long elapseTime=(stopTime-startTime);
                System.out.println("\nTime  taken to sort array is:"+elapseTime+
               "nanoseconds");

        }
}
```

Results:

Enter the array size:

10

Array before sorting is:

14 4 5 8 10 12 13 9 17 2

Array after sorting is:

2 4 5 8 9 10 12 13 14 17

Time taken to sort array is : 232427nanoseconds

OBSERVATION TABLE / LOOKUP TABLE / TRUTH

TABLE: Observation table: Run the algorithm for following

Input Size:

Elements	5000	10000	15000	20000	25000	30000
Time Taken	50332628	91008346	506232830	671926169	830230427	976503994

3. Design and implement a Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

```
import java.util.Scanner;
public class knapsackgreedy
{
	public static void main(String[] args)
	{
		int i,j=0,max_qty,m,n;
		float sum=0,max;
		Scanner sc = new Scanner(System.in);
		int array[][]=new int[2][20];
		System.out.println("Enter no of items");
		n=sc.nextInt();
		System.out.println("Enter the weights of each items");
		for(i=0;i<n;i++)
		array[0][i]=sc.nextInt();
		System.out.println("Enter the profit of each items");
		for(i=0;i<n;i++)
		array[1][i]=sc.nextInt();
		System.out.println("Enter maximum volume of knapsack :");
		max_qty=sc.nextInt();
		m=max_qty; while(m>=0)
		{
			max=0; for(i=0;i<n;i++)
			{
				if(((float)array[1][i])/((float)array[0][i])>max)
				{
					max=((float)array[1][i])/((float)array[0][i]);
					j=i;
				}
			}
			if(array[0][j]>m)
			{
			System.out.println("Quantity of item number: "+(j+1)+"added is" +m);
```

```
                    sum+=m*max; m=-1;
                    }
                    else
                    {
                        System.out.println("Quantity of item number:"+(j+1) + "added
                        is " + array[0][j]);
                        m-=array[0][j]; sum+=(float)array[1][j];
                        array[1][j]=0;
                    }
                }
                System.out.println("The total profit is " + sum);
                sc.close();
        }
}
```

Results:

```
Enter no of items
4
Enter the weights of each items
2
1
3
2
Enter the values of each items
12
10
20
15
Enter maximum volume of knapsack :
5
Quantity of item number: 2 added is 1
Quantity of item number: 4 added is 2
Quantity of item number: 3 added is 2
The total profit is 38.333332
```

4. **Design an application for a thermal power station and electrical lines that are connected among various power stations. The costs of electrification involved appear as weights on the edges. Obtain the minimum possible connection among the thermal stations so that any two thermal stations can be linked with the minimum cost involved**.

```
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 thermal power stations");
		Scanner sc=new Scanner(System.in);
		n=sc.nextInt();
		for(i=1;i<=n;i++)
		sol[i]=0;
		System.out.println("Enter the cost of electrification for each connection");
		for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		w[i][j]=sc.nextInt();
		System.out.println("Enter the source thermal station");
		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 connections among the thermal station(spanning
        tree)");
        else
        System.out.println("The cost of minimum connections among the thermal
        station(spanning tree) is "+sum);
        sc.close();
    }
}
```

Results:

```
Enter the number of thermal power stations
6
Enter the cost of electrification for each connection
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 thermal station
1
```

1->2=3

2->3=1

2->6=4

6->5=2

6->4=5

The cost of minimum connections among the thermal station(spanning tree) is 15

5. Develop an optimal route for a scenario where a person wants to buy a ticket to a baseball game. Along the way from house to reaching the destination, the person using it is a toll road, and has to pay a certain amount of money.

```
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 cities");
            Scanner sc = new Scanner(System.in);
            n=sc.nextInt();
            System.out.println("Enter the toll amount to be paid through all roads");
            for(i=0;i<n;i++)
```

```
        for(j=0;j<n;j++)
        a[i][j]=sc.nextInt();
        System.out.println("Enter the source city");
        s=sc.nextInt();
        dijkstra tr=new dijkstra();
        tr.dijk(a,s,n);
        System.out.println("The shortest path between source city "+s+" to remaining cities
        are");
        tr.display(s,n);
        sc.close();
    }
}
```

Results

```
Enter the number of cities
5
Enter the toll amount to be paid through all roads
0 3 99 7 99
3 0 4 2 99
99 4 0 5 6
7 2 5 0 4
99 99 6 4 0
Enter the source city
0
The shortest path between source city 0 to remaining cities are

0 ->1 =3
0 ->1 ->2 =7
0 ->1 ->3 =5
0 ->1 ->3 ->4 =9
```

6a. Design and implement a Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
import java.util.Scanner;
public class Floyd
{
	void floyd(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.floyd(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();
        }
}
```

Results:

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

6 b. Design and implement a Program to find the transitive closure using Warshal's algorithm.

```
import java.util.Scanner;
public class Warshall
{
	void warshall(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++)
		if(w[i][k]==1&&w[k][j]==1)
		{
			w[i][j]=1;
		 }
		else
		{
			w[i][j]=w[i][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 adjacency matrix of a digraph");
		for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		a[i][j]=sc.nextInt();
		Warshall f=new Warshall();
		f.warshall(a, n);
		System.out.println("The transitive closure of the digraph is");
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
```

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

Results:

Enter the number of vertices

4

Enter the adjacency matrix of a digraph

0 1 0 0

0 0 0 1

0 0 0 0

1 0 1 0

The transitive closure of the digraph is

1 1 1 1

1 1 1 1

0 0 0 0

1 1 1 1

7. The owner of a gourmet coffee shop wishes to mix a 10-pound bag of coffee using various types of coffee beans in such a way to produce the coffee blend at the maximum cost. The weights of the objects in the problem correspond to the quantity in pounds available of each type of coffee bean. The value of each quantity of coffee beans is the total cost of that quantity in rupees. Apply the Knapsack algorithm to maximize the profit.

```
import java.util.Scanner;
public class knapsackdp
{
    public void solve(int[] wt, int[] val, int W, int N)
    {
        int i,j;
        int sol[][] = new int[N + 1][W + 1];
        for ( i = 0; i <= N; i++)
        {
            for ( j = 0; j <= W; j++)
            {
                if(i==0||j==0)
                sol[i][j]=0;
                else if(wt[i]>j)
                sol[i][j]=sol[i-1][j];
                else
                sol[i][j]=Math.max((sol[i-1][j]), (sol[i - 1][j - wt[i]] + val[i]));
            }
        }
        System.out.println("The optimal solution is: "+sol[N][W]);
        int[] selected = new int[N + 1];
        for(i=0;i<N+1;i++)
        selected[i]=0;
        i=N;
        j=W;
        while (i>0&&j>0)
        {
            if (sol[i][j] !=sol[i-1][j])
            {
                selected[i] = 1; j = j - wt[i];
```

```
                    }
                    i--;
                }
                System.out.println("\nItems selected : ");
                for ( i = 1; i < N + 1; i++)
                if(selected[i] == 1)
                System.out.print(i +" ");
                System.out.println();
        }
        public static void main(String[] args)
        {
                Scanner scan = new Scanner(System.in);
                knapsackdp ks = new knapsackdp();
                System.out.println("Enter number of coffee beans ");
                int n = scan.nextInt();
                int[] wt = new int[n + 1];
                int[] val = new int[n + 1];
                System.out.println("\nEnter the quantity in pounds available for "+ n +" type of
                coffee bean");
                for (int i = 1; i <= n; i++)
                wt[i] = scan.nextInt();
                System.out.println("\nEnter value of "+ n +" coffee bean");
                for(int i = 1; i <= n; i++)
                val[i] = scan.nextInt();
                System.out.println("\nEnter size of the bag ");
                int W = scan.nextInt();
                ks.solve(wt, val, W, n);
        }
  }
```

Results:

```
Enter number of coffee beans
4
Enter the quantity in pounds available for 4 type of coffee bean
2
1
```

3

2

Enter value of 4 coffee bean

12

10

20

15

Enter size of the bag

5

The optimal solution is: 37

Items selected :

1 2 4

8. Design an application for drilling an optimal printed circuit board. To drill two holes of different diameters consecutively, the head of the machine has to move to a toolbox and change the drilling equipment. This is quite time consuming. Thus, it is clear that one has to choose some diameter, drill all holes of the same diameter, change the drill, drill the holes of the next diameter, etc. Thus, this drilling problem has to minimize the travel time for the machine head. Find the optimal time to drill the circuit board.

```
import java.util.Scanner;
public class TSP
{
    public static void main(String[] args)
    {
        int c[][]=new int[10][10], tour[]=new int[10];
        Scanner in = new Scanner(System.in);
        int i, j,cost;
        System.out.println("**** TSP DYNAMIC PROGRAMMING*****");
        System.out.println("Enter the number of cities: ");
        int n = in.nextInt();
        if(n==1)
        {
            System.out.println("Path is not possible");
            System.exit(0);
        }
        System.out.println("Enter the cost matrix");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        c[i][j] = in.nextInt();
        System.out.println("The entered cost matrix is");
        for(i=1;i<=n;i++)
        {
```

```
                for(j=1;j<=n;j++)
                {
                        System.out.print(c[i][j]+"\t");
                }
                System.out.println();
        }
        for(i=1;i<=n;i++)
        tour[i]=i;
        cost = tspdp(c, tour, 1, n);
        System.out.println("The accurate path is");
        for(i=1;i<=n;i++)
        System.out.print(tour[i]+"->");
        System.out.println("1");
        System.out.println("The accurate mincost is "+cost);
        System.out.println("******* ************ ***************");
}
static int tspdp(int c[][], int tour[], int start, int n)
{
        int mintour[]=new int[10], temp[]=new int[10], mincost=999, ccost, i, j, k;
        if(start == n-1)
        {
                return (c[tour[n-1]][tour[n]] + c[tour[n]][1]);
        }
        for(i=start+1; i<=n; i++)
        {
                for(j=1; j<=n; j++)
                temp[j] = tour[j];
```

```
                temp[start+1] = tour[i];
                temp[i] = tour[start+1];
                if((c[tour[start]][tour[i]]+(ccost=tspdp(c,temp,start+1,n)))<mincost)
                {
                        mincost = c[tour[start]][tour[i]] + ccost;
                        for(k=1; k<=n; k++)
                        mintour[k] = temp[k];
                }
        }
        for(i=1; i<=n; i++)
        tour[i] = mintour[i];
        return mincost;
    }
}
```

Results:

**** TSP DYNAMIC PROGRAMMING*****

Enter the number of cities:

5

Enter the cost matrix

0 8 99 99 5

12 0 99 16 99

99 7 0 8 20

1 6 19 0 3

9 99 12 18 0

The entered cost matrix is

0	8	99	99	5
12	0	99	16	99
99	7	0	8	20
1	6	19	0	3
9	99	12	18	0

The accurate path is

1->5->3->2->4->1

The accurate mincost is 41

******* ********** **********

9. Design and implement for a given chess board having N×N cells, place *N* queens on the board in such a way that no queen attacks any other queen. If it is possible to place all the *N* queens in such a way that no queen attacks another queen, then print *N* lines having *N* Queens. If there is more than one solution of placing the queens, print all of them. If it is not possible to place all *N* queens in the desired way, then print "Not possible".

```
import java.util.Scanner;
import java.io.*;
class operation
{
    int x[]=new int[20];
    int count=0;
    public boolean place(int row,int column)
    {
        int i;
        for(i=1;i<=row-1;i++)
        {
            if(x[i] == column)
            return false;
            else
            if (Math.abs(x[i]-column)== Math.abs(i-row))
            return false;
        }
        return true;
    }
    public void Queen(int row,int n)
    {
        int column;
        for(column=1;column<=n;column++)
        {
            if(place(row,column))
            {
                x[row] = column;
                if(row==n)
```

```
                                {
                                        print_board(n);
                                }
                                else
                                Queen(row+1,n);
                        }
                }
        }

        public void print_board(int n)
        {
                int i;
                System.out.println("\n\nSolution :"+(++count));
                for(i=1;i<=n;i++)
                {
                        System.out.print(" " +i);
                }
                for(i=1;i<=n;i++)
                {
                        System.out.print("\n\n"+i);
                        for(int j=1;j<=n;j++)
                        {
                                if(x[i]==j)
                                System.out.print(" Q");
                                else
                                System.out.print(" -");
                        }
                }
        }
}
class NQueens
{
        public static void main (String args[] )throws IOException
        {
                Scanner scanner = new Scanner(System.in);
```

```
                System.out.print("Enter the size of the chessboard (N): ");
                int n = scanner.nextInt();
                if(n==2 || n==3)
            {
                    System.out.print("Not Possible");
            }
            scanner.close();
            operation op=new operation();
            op.Queen(1,n);
        }
}
```

Results:

Enter the size of the chessboard (N):

4

Solution :1

1 2 3 4

1 – Q – –

2 – – – Q

3 Q – – –

4 – – Q –

Solution :2

1 2 3 4

1 – – Q –

2 Q – – –

3 – – – Q

4 – Q – –