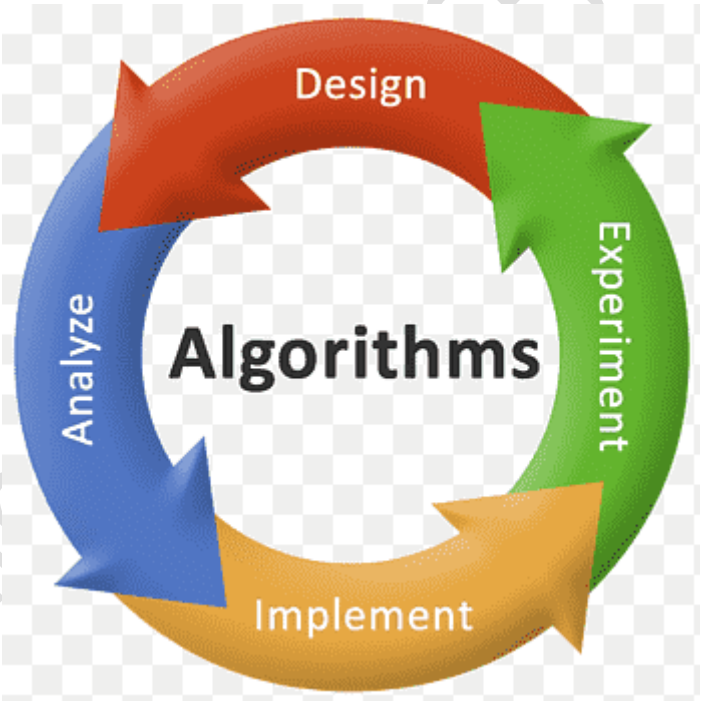
VIDYAVARDHAKA COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Mysuru, Karnataka-570002

"DESIGN AND ANALYSIS OF ALGORITHMS LAB MANUAL"

(20CS47)

As per Autonomous Syllabus



PREPARED BY:

Prof. NITHIN KUMAR

Asst. Professor, Department of Computer Science & Engineering
VVCE, Mysuru
Email: nithingowda021@vvce.ac.in

1. Sort a given set of elements using the quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the 1st to be sorted and 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 Quicksort
{
	static final int MAX = 10005;
	static int[] a = new int[MAX];

	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter Max array size: ");
		int n = input.nextInt();
		Random random = new Random();
		System.out.println("Enter the array elements: ");
		for (int i = 0; i < n; i++)
			a[i] = random.nextInt(1000);
		for (int i = 0; i < n; i++)
			System.out.print(a[i] + " ");
		long startTime = System.nanoTime();
		QuickSortAlgorithm(0, n - 1);
		long stopTime = System.nanoTime();
		long elapsedTime = stopTime - startTime;
		System.out.println("Time Complexity (ms) for n = " + n + " is : " +
		(double)elapsedTime / 1000000);
		System.out.println("Sorted Array (Quick Sort):");
		for (int i = 0; i < n; i++)
			System.out.print(a[i] + " ");
		input.close();
	}

	public static void QuickSortAlgorithm(int p, int r)
	{
			int i, j, temp, pivot;
			if (p < r)
			{
				i = p;
				j = r;
				pivot = a[p];
				while(true)
				{
					i++;
					while (a[i] < pivot && i<r)
					{
```

```
                    i++;
                }
                while (a[j] > pivot)
                {
                    j--;
                }
                if (i < j)
                {
                    temp = a[i];
                    a[i] = a[j];
                    a[j] = temp;
                }
                else
                {
                    break;
                }
            }
            a[p] = a[j];
            a[j] = pivot;
            QuickSortAlgorithm(p, j - 1);
            QuickSortAlgorithm(j + 1, r);
        }
    }
}
```

2. Implement a merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and 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 Mergesort
{
	static final int MAX = 10005;
	static int[] a = new int[MAX];

	public static void main(String[] args)
	{
		Scanner input = new Scanner(System.in);
		System.out.print("Enter Max array size: ");
		int n = input.nextInt();
		Random random = new Random();
		System.out.println("Enter the array elements: ");
		for (int i = 0; i < n; i++)
			a[i] = random.nextInt(1000);
		for (int i = 0; i < n; i++)
			System.out.print(a[i] + " ");
		long startTime = System.nanoTime();
		MergeSortAlgo(0, n - 1);
		long stopTime = System.nanoTime();
		long elapsedTime = stopTime - startTime;
		System.out.println("Time Complexity (ms) for n = " + n + " is : " +
		(double)elapsedTime / 1000000);
		System.out.println("Sorted Array (Merge Sort):");
		for (int i = 0; i < n; i++)
			System.out.print(a[i] + " ");
		input.close();
	}
	public static void MergeSortAlgo(int low, int high)
		{
			int mid;
			if (low < high)
			{
				mid = (low + high) / 2;
				MergeSortAlgo(low, mid);
				MergeSortAlgo(mid + 1, high);
				Merge(low, mid, high);
			}
		}

		public static void Merge(int low, int mid, int high)
		{
```

```
                int[] b = new int[MAX];
                int i, h, j, k;
                h = i = low;
                j = mid + 1;
                while ((h <= mid) && (j <= high))
                        if (a[h] < a[j])
                                b[i++] = a[h++];
                        else
                                b[i++] = a[j++];

                if (h > mid)
                        for (k = j; k <= high; k++)
                                b[i++] = a[k];
                else
                        for (k = h; k <= mid; k++)
                                b[i++] = a[k];

                for (k = low; k <= high; k++)
                        a[k] = b[k];
        }
}
```

3. Implement transitive closure using Warshalls algorithm for the given directed graph.

```
import java.util.Scanner;
public class Warshal {
	static int a[][];
	static int n;

	public static void main(String args[])
	{
		System.out.println("Enter the number of vertices\n");
		Scanner scanner = new Scanner(System.in);
		n = scanner.nextInt();
		a = new int[n][n];
		System.out.println("Enter the Cost Matrix (0's and 1's) \n");
		for (int i = 0; i < n; i++)
		{
			for (int j = 0; j < n; j++)
			{
				a[i][j] = scanner.nextInt();
			}
		}
		getClosure();
		PrintMatrix();
		scanner.close();
	}

	public static void getClosure()
	{
		for (int k = 0; k < n; k++)
		{
			for (int i = 0; i < n; i++)
				for (int j = 0; j < n; j++)
					if(a[i][j]==1 || (a[i][k]==1 && a[k][j]==1))
						a[i][j]=1;
		}
	}

	public static void PrintMatrix()
	{
		System.out.println("Transitive Closure:\n");
		for(int i=0; i<n; i++)
		{
			for(int j=0; j<n; j++)
				System.out.print(a[i][j] +" ");
			System.out.println();
		}
	}
}
```

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

```
import java.util.Scanner;
public class Dijakstras {
        static int a[][];
        static int n;
        public static void main(String args[])
        {
         Scanner in = new Scanner(System.in);
         System.out.println("Enter the number of vertices:");
         n = in.nextInt();
         System.out.println("Enter the cost adjacency matrix");
         a = new int[n][n];
         for (int i=0;i<n;i++)
         {
          for (int j=0;j<n;j++)
          {
           a[i][j] = in.nextInt();
          }
         }
         System.out.println("\nEnter the source vertex");
         int s=in.nextInt();
         Dijkstra(s);
         in.close();
        }

        public static void Dijkstra(int s)
        {
         int visited[] = new int[n];
         int d[] = new int[n];
         int i,u,v;
         for(i=0;i<n;i++)
         {
          visited[i]=0;
          d[i] = a[s][i];
         }
         visited[s]=1;
         d[s]=0;
         i=1;
         while(i<=n-1)
         {
          u = Extract_Min(visited,d);
          visited[u]=1;
          i++;
          for(v=0;v<n;v++)
          {
           if((d[u]+a[u][v]<d[v]) && visited[v]==0)
```

```
                d[v]= d[u]+a[u][v];
              }
             }
             for(i=0;i<n;i++)
             {
               if(i!=s)
                System.out.println(s+"->"+i+":"+d[i]);
             }
            }

            public static int Extract_Min(int visited[],int d[])
            {
             int i,j=0,min=999;
             for(i=0;i<n;i++)
              if(d[i]<min && visited[i]==0)
              {
               min = d[i];
               j=i;
              }
              return j;
            }
}
```

5. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
import java.util.Scanner;
public class Kruskals {
static int n, parent[], a[][];

        public static int find(int p)
        {
                while(parent[p]!=0)
                {
                        p=parent[p];
                }
                return p;
        }

   public static void union(int i, int j)
        {
                if(i<j)
                        parent[i]=j;
                else
                        parent[j]=i;
        }

   public static void kruskal()
        {
        int u=0,v=0,min,k=0,i,j,sum=0;
        while(k<n-1)
        {
                min=999;
                for(i=0;i<n;i++)
                        for(j=0;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]=999;
        }
```

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

    public static void main(String[] args)
        {
            int i,j;
            System.out.println("Enter the number of vertices of the graph");
            Scanner in=new Scanner(System.in);
            n=in.nextInt();
            a=new int[n][n];
            parent=new int[n];
            System.out.println("Enter the weighted matrix");
            for(i=0;i<n;i++)
            {
                for(j=0;j<n;j++)
                {
                    a[i][j]=in.nextInt();
                }
            }
            kruskal();
            in.close();
        }
}
```

6. Implement Fractional Knapsack problem using Greedy Technique.

```
import java.util.Scanner;
public class Gknapsack {
        static int n;
        static float capacity, item[], weight[], profit[], ratio[];
        public static void main(String args[])
         {
          int i, j;
          float temp;
          Scanner in = new Scanner(System.in);
          System.out.println("\n Enter the number of objects ");
          n = in.nextInt();
          item=new float[n];
          weight=new float[n];
          profit=new float[n];
          ratio=new float[n];
          System.out.println("\n Enter items, weights and profit ");
          for(i=0; i<n; i++)
          {
           item[i]=in.nextFloat();
           weight[i]=in.nextFloat();
           profit[i]=in.nextFloat();
          }
          System.out.println("\n Enter the capacity of knapsack ");
          capacity = in.nextFloat();
          for(i=0; i<n; i++)
          {
           ratio[i]= profit[i]/weight[i];
          }
          for(i=0; i<n; i++)
           for(j=i+1; j<n; j++)
            if(ratio[i]<ratio[j])
            {
              temp = ratio[j];
              ratio[j] = ratio[i];
              ratio[i] = temp;

              temp = profit[j];
              profit[j] = profit[i];
              profit[i] = temp;

              temp = weight[j];
              weight[j] = weight[i];
              weight[i] = temp;

              temp = item[j];
              item[j] = item[i];
              item[i] = temp;
```

```
    }
    knapsack();
   in.close();
}
public static void knapsack()
{
   float tp = 0;
   float u = capacity;
   int i;
   float x[]= new float[n];
   for(i=0; i<n; i++)
   {
    x[i]=0;
   }
   for(i=0;i<n;i++)
   {
    if(weight[i]>u)
     break;
    else
    {
       x[i]=1;
       tp = tp + profit[i];
       u = (u-weight[i]);
    }
   }
   if(i<n)
   {
    x[i]=u/weight[i];
    tp = tp +(x[i] * profit[i]);
   }
   System.out.println("\n The resultant vector is");
   for(i=0; i<n; i++)
   {
    System.out.println("item"+(int)item[i]+":"+x[i]);
   }
   System.out.println("Maximum profit is "+tp);
  }
}
```

7. Implement 0/1 Knapsack problem using Dynamic Programming.

```
import java.util.Scanner;
public class Dknapsack {
        static int n, m, w[], v[][],value[];
  public static int knap(int i,int j)
  {
    if(i==0||j==0)
    {
      v[i][j] = 0;
    }
    else if(j<w[i])
    {
      v[i][j] = knap(i-1,j);
    }
    else
    {
      v[i][j] = Math.max(knap(i-1,j),value[i]+knap(i-1,j-w[i]));
    }
    return v[i][j];
  }

  public static void optimal(int i,int j)
  {
    if(i>=1 || j>=1) {
      if(v[i][j]!=v[i-1][j])
      {
        System.out.println("Item : "+i);
        j = j-w[i];
        optimal(i-1,j);
      }
      else
      {
        optimal(i-1,j);
      }
    }
  }
  public static void main(String[] args) {
    int profit,i;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the number of items:");
    n = in.nextInt();
    System.out.println("Enter the capacity of the knapsack:");
    m = in.nextInt();
    w=new int[n+1];
    value=new int[n+1];
    v=new int[n+1][m+1];
    System.out.println("\nEnter weights:");
```

```
            for(i=1; i<=n; i++)
            {
              w[i]=in.nextInt();
            }
            System.out.println("\nEnter profits:");
            for(i=1; i<=n; i++)
            {
              value[i]=in.nextInt();
            }
        profit = knap(n,m);
        System.out.println("Profit: "+profit);
        System.out.println("Items to be added for Optimal Solution:");
        optimal(n,m);
        in.close();
    }
}
```

8. Implement a subset concept for a given set S = {sl, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

```
import java.util.Scanner;
public class SumofSubset {
	static int n, S[], soln[], d;
	public static void main(String args[])
	{
		int sum = 0;
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter number of elements: ");
		n = scanner.nextInt();
		S = new int[n+1];
		soln = new int[n+1];
		System.out.println("Enter the set in increasing order: ");
		for (int i = 1; i <= n; i++)
		{
			S[i] = scanner.nextInt();
		}
		System.out.println("Enter the max. subset value(d): ");
		d = scanner.nextInt();
		for (int i = 1; i <= n; i++)
		{
			sum = sum + S[i];
		}
		if (sum < d || S[1] > d)
		{
			System.out.println("No Subset possible");
		}
		else
		{
			SumofSub(0, 0, sum);
		}
		scanner.close();
	}

	public static void SumofSub(int i, int weight, int total)
	{
		if (promising(i, weight, total) == true)
			if (weight == d)
			{
				for (int j = 1; j <= n; j++)
				{
					if (soln[j] == 1)
						System.out.print(S[j] + "  ");
				}
				System.out.println();
```

```
                }
                else
                {
                        soln[i+1] = 1;
                        SumofSub(i+1, weight + S[i+1], total - S[i+1]);
                        soln[i+1] = 0;
                        SumofSub(i+1, weight, total - S[i+1]);
                }
        }

        public static boolean promising(int i, int weight, int total)
        {
        return ((weight + total >= d) && (weight == d || weight + S[i+1] <= d));
        }
}
```

9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
import java.util.Arrays;
import java.util.Scanner;
public class Prims {
        static int a[][];
        static int V;
        public static void main(String args[])
        {
                System.out.println("Enter the number of vertices\n");
                Scanner scanner = new Scanner(System.in);
                V = scanner.nextInt();
                a = new int[V][V];
                System.out.println("Enter the Cost Matrix \n");
                for (int i = 0; i < V; i++)
                {
                        for (int j = 0; j < V; j++)
                        {
                                a[i][j] = scanner.nextInt();
                        }
                }
                Prim();
                scanner.close();
        }
        public static void Prim()
        {
           int no_edge=0,sum=0;
           boolean[] selected = new boolean[V];
           Arrays.fill(selected, false);
           selected[0] = true;
           System.out.println("Edge : Weight");
           while (no_edge < V - 1)
           {
            int x=0,y=0,min = 999;
            for (int i = 0; i < V; i++)
            {
             if (selected[i] == true)
             {
              for (int j = 0; j < V; j++)
              {
               if (!selected[j] && a[i][j] != 0)
               {
                if (min > a[i][j])
                  min = a[i][j];
                  x=i;
                  y=j;
               }
              }
```

```
                }
               }
               System.out.println(x + " - " + y + " :  " + a[x][y]);
               sum=sum+a[x][y];
               selected[y] = true;
               no_edge++;
              }
              System.out.println("Cost of Tree: "+sum);
             }
}
```

10. Implement All-Pairs Shortest Paths Problem using Floyds algorithm.

```
import java.util.Scanner;
public class Floyds
{
	static int a[][];
	static int n;

	public static void main(String args[])
	{
		System.out.println("Enter the number of vertices\n");
		Scanner scanner = new Scanner(System.in);
		n = scanner.nextInt();
		a = new int[n][n];
		System.out.println("Enter the Cost Matrix (999 for infinity) \n");
		for (int i = 0; i < n; i++)
		{
			for (int j = 0; j < n; j++)
			{
				a[i][j] = scanner.nextInt();
			}
		}
		getPath();
		PrintMatrix();
		scanner.close();
	}

	public static void getPath()
	{
		for (int k = 0; k < n; k++)
		{
			for (int i = 0; i < n; i++)
				for (int j = 0; j < n; j++)
					if ((a[i][k] + a[k][j]) < a[i][j])
						a[i][j] = a[i][k] + a[k][j];
		}
	}

	public static void PrintMatrix()
	{
		System.out.println("The All Pair Shortest Path Matrix is:\n");
		for(int i=0; i<n; i++)
		{
			for(int j=0; j<n; j++)
				System.out.print(a[i][j] +" ");
			System.out.println();
		}
	}
}
```