1. **Gale Shapley:**
   Given a set of men and women design and implement Gale–Shapley algorithm to determine the stable set of marriages among them. Assumptions: Men propose first according to their preference list. Women can choose a better partner based on their preference.

Men's preference list

	1	2	3	4
A	W	X	Y	Z
B	X	W	Y	Z
C	Z	W	Y	X
D	Z	Y	X	W

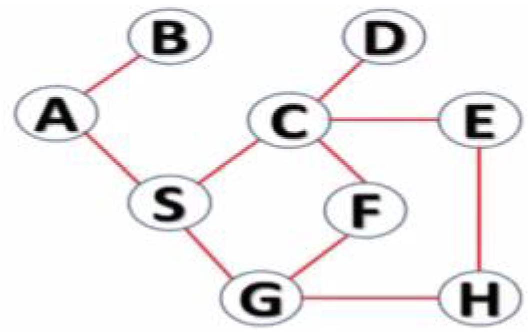
Women's preference list

	1	2	3	4
W	A	B	D	C
X	B	C	A	D
Y	C	D	A	B
Z	C	D	B	A

```
public class stablematching
{
   private int N, engagedCount;
   private String[][] menPref;
   private String[][] womenPref;
   private String[] men;
   private String[] women;
   private String[] womenPartner;
   private boolean[] menEngaged;

   public stablematching(String[] m, String[] w, String[][] mp, String[][] wp)
   {
      N = mp.length;
      engagedCount = 0;
      men = m;
      women = w;
      menPref = mp;
      womenPref = wp;
      menEngaged = new boolean[N];
      womenPartner = new String[N];
      calcMatches();
   }

   private void calcMatches()
   {
      while (engagedCount < N)
      {
         int free;
         for (free = 0; free < N; free++)
            if (!menEngaged[free])
```

```
                break;

            for (int i = 0; i < N && !menEngaged[free]; i++)
            {
                int index = womenIndexOf(menPref[free][i]);
                if (womenPartner[index] == null)
                {
                    womenPartner[index] = men[free];
                    menEngaged[free] = true;
                    engagedCount++;
                }
                else
                {
                    String currentPartner = womenPartner[index];
                    if (morePreference(currentPartner, men[free], index))
                    {
                        womenPartner[index] = men[free];
                        menEngaged[free] = true;
                        menEngaged[menIndexOf(currentPartner)] = false;
                    }
                }
            }
        }
        printCouples();
    }

    private boolean morePreference(String curPartner, String newPartner, int index)
    {
        for (int i = 0; i < N; i++)
        {
            if (womenPref[index][i].equals(newPartner))
                return true;
            if (womenPref[index][i].equals(curPartner))
                return false;
        }
        return false;
    }

    private int menIndexOf(String str)
    {
        for (int i = 0; i < N; i++)
            if (men[i].equals(str))
                return i;
        return -1;
    }

    private int womenIndexOf(String str)
    {
        for (int i = 0; i < N; i++)
            if (women[i].equals(str))
                return i;
        return -1;
    }

    public void printCouples()
    {
        System.out.println("Couples are : ");
        for (int i = 0; i < N; i++)
        {
            System.out.println(womenPartner[i] +" "+ women[i]);
        }
    }

    public static void main(String[] args)
    {
        System.out.println("Gale Shapley Marriage Algorithm\n");

            String[] m = {"A", "B", "C", "D"};
        String[] w = {"W", "X", "Y", "Z"};

            String[][] mp = {{"W","X","Y","Z"},
                          {"X","W","Y","Z"},
                          {"Z","W","Y","X"},
                          {"Z","Y","X","W"}};

        String[][] wp = {{"A","B","D","C"},
                          {"B","C","A","D"},
                          {"C","D","A","B"},
                          {"C","D","A","B"}};

        stablematching sm = new stablematching(m, w, mp, wp);
    }
}
```

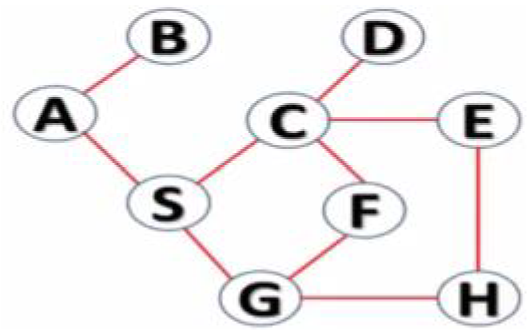
2. **BFS:**
   A GPS navigation system needs an approach to discover the reachable areas in a given geographical region from a given source area. Design and implement an algorithm to find which nodes can be reached from a given source node for the following graph.



```
import java.util.*;
class BFS{
        private int V;
        private LinkedList<Integer> adj[];
        private boolean visited[];
        private char[] vertex = {'A','B','C','D','E','F','G','H','S'};
        BFS(int V){
                this.V = V;
                adj = new LinkedList[V];
                visited = new boolean[V];
                for(int i=0;i<V;i++){
                        adj[i] = new LinkedList();
                }
        }

        void addEdge(int a,int b){
                adj[a].add(b);
                adj[b].add(a);
        }

        void BFS(int s){
                LinkedList<Integer> queue = new LinkedList();
                visited[s] = true;
                queue.add(s);
                while(queue.size() != 0){
                        s = queue.poll();
                        System.out.print(vertex[s]+" ");
                        Iterator<Integer> i = adj[s].listIterator();
                        while(i.hasNext()){
                                int n = i.next();
                                if(!visited[n]){
```

```
                    visited[n] = true;
                    queue.add(n);
                }
            }
        }
    }

    public static void main(String[] args) {
        BFS b = new BFS(9);
        b.addEdge(0,1);
        b.addEdge(0,8);
        b.addEdge(8,2);
        b.addEdge(8,6);
        b.addEdge(2,3);
        b.addEdge(2,4);
        b.addEdge(2,5);
        b.addEdge(6,7);
        b.addEdge(6,5);
        b.addEdge(4,7);

        b.BFS(0);
    }
}
```

3. **DFS:**
   A rat is searching for cheese in a maze that consists of nodes and edges as shown in the fig. It always goes deep into the maze, if it hits a wall then it returns back to the ancestor node and searches further. Suggest which algorithm can be chosen for it.



```
import java.util.*;
class DFS{
        private int V;
        private LinkedList<Integer> adj[];
        private boolean visited[];
        private char[] vertex = {'A','B','C','D','E','F','G','H','S'};
        DFS(int V){
                this.V = V;
                visited = new boolean[V];
                adj = new LinkedList[V];
                for(int i=0;i<V;i++){
                        adj[i] = new LinkedList();
                }
        }

        void addEdge(int a,int b){
                adj[a].add(b);
                adj[b].add(a);
        }

        void DFS(int n){
                visited[n] = true;
                System.out.print(vertex[n]+ " ");
                Iterator<Integer> i = adj[n].listIterator();
                while(i.hasNext()){
                        int s = i.next();
                        if(!visited[s]){
                                visited[s] = true;
                                DFS(s);
                        }
                }
```

```
	}

	public static void main(String[] args) {
		DFS b = new DFS(9);
		b.addEdge(0,1);
		b.addEdge(0,8);
		b.addEdge(8,2);
		b.addEdge(8,6);
		b.addEdge(2,3);
		b.addEdge(2,4);
		b.addEdge(2,5);
		b.addEdge(6,7);
		b.addEdge(6,5);
		b.addEdge(4,7);

		b.DFS(0);
	}
}
```

4. **Merge Sort:**
   Design and implement a merge sort algorithm that takes random number input and displays the execution time required.

```
import java.lang.Math;
class Mergesort{

        public void merge(int[] arr,int mid,int begin,int end){
                int n1 = mid-begin+1;
                int n2 = end - mid;

                int [] L = new int[n1];
                int [] R = new int[n2];

                for(int i=0;i<n1;i++)
                        L[i] = arr[begin+i];

                for(int i=0;i<n2;i++)
                        R[i] = arr[mid+1+i];

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

                while(i<n1 && j<n2){
                        if(L[i] <= R[j]){
                                arr[k] = L[i];
                                i++;
                        }
                        else{
                                arr[k] = R[j];
                                j++;
                        }
                        k++;
                }

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

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


        public void sort(int []arr,int begin,int end){
            if(begin < end){
                int mid = (begin+end)/2;
                sort(arr,begin,mid);
                sort(arr,mid+1,end);
                merge(arr,mid,begin,end);
            }
        }

        public static void main(String[] args){
            int n = 500;
            int arr[] = new int[n];
            for(int i=0;i<n;i++){
                arr[i] = (int)(Math.random()*n);
            }
            int begin = 0;
            int end = n-1;
            Mergesort ms = new Mergesort();
            Long start = System.currentTimeMillis();
            ms.sort(arr,begin,end);
            Long end1 = System.currentTimeMillis();
            for(int x: arr){
                System.out.print(x+" ");
            }
            System.out.println("\nExecution Time in nanoseconds: "+(end1 - start));
        }
    }
```

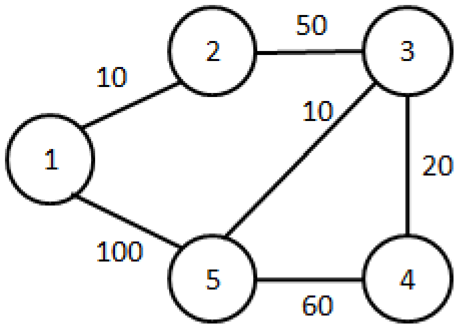
5. **Quick Sort:**
   In a database of numbers there is a table of unsorted numbers. The database admin now wants to sort these numbers using an approach wherein a pivot element is selected for sorting. At a certain point, the first half elements are less than the pivot and right half elements are greater than the pivot. Design and implement an algorithm to solve it using random numbers and also display the execution time.

```
class QuickSort{

        public void sort(int arr[],int begin,int end){
                if (begin < end){
                        int partitionIndex = partition(arr,begin,end);
                        sort(arr,begin,partitionIndex-1);
                        sort(arr,partitionIndex+1,end);
                }
        }

        private void swap(int arr[],int i,int j){
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
        }

        private int partition(int arr[],int begin,int end){
                int i = begin-1;
                int pivot = end;
                for(int j=begin;j<end;j++){
                        if(arr[j] <= arr[pivot]){
                                i++;
                                swap(arr,j,i);
                        }
                }
                swap(arr,i+1,end);
                return i+1;

        }

        public static void main(String[] args) {
                int arr[] = {2,3,10,1,8,6,7,5,9,0};
                int begin = 0;
                int end = arr.length - 1;

                QuickSort qs = new QuickSort();
                qs.sort(arr,begin,end);
```

```
		for(int x: arr){
			System.out.print(x+" ");
		}
	}
}
```

6. **Dijkstra's Algorithm**

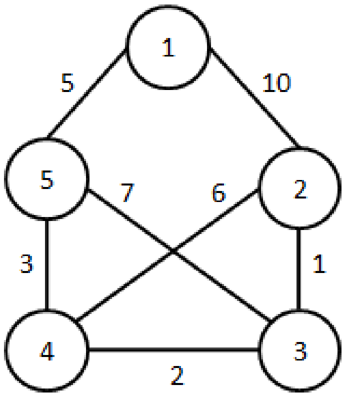
A truck driver is given a set of locations to be covered with their distances by a company. The company strictly orders that truck should be started from a particular location. Design and implement an algorithm that gives a greedy solution to the truck driver's problem and display the shortest path for a given source location to all other locations.



```
import java.util.*;
import java.lang.*;
import java.io.*;
class ShortestPath{

        static int V;

        int minDistance(int dist[], Boolean sptSet[])
        {
                int min = Integer.MAX_VALUE, min_index = -1;
                for (int v = 0; v < V; v++)
                        if (sptSet[v] == false && dist[v] <= min) {
                                min = dist[v];
                                min_index = v;
                        }
                return min_index;
        }

        void printSolution(int dist[], int n)
        {
                System.out.println("Vertex Distance from Source");
                for (int i = 0; i < V; i++)
                        System.out.println("Source to "+(i+1)+ " is " + dist[i]);
        }

        void dijkstra(int graph[][], int src)
        {
                int dist[] = new int[V];
                Boolean sptSet[] = new Boolean[V];
                for (int i = 0; i < V; i++) {
                        dist[i] = Integer.MAX_VALUE;
                        sptSet[i] = false;
                }

                dist[src] = 0;

                for (int count = 0; count < V - 1; count++) {
                        int u = minDistance(dist, sptSet);
                        sptSet[u] = true;
                        for (int v = 0; v < V; v++)
                                if (!sptSet[v] && graph[u][v] != 0 &&
                                                dist[u] != Integer.MAX_VALUE && dist[u] +
graph[u][v] < dist[v])
                                        dist[v] = dist[u] + graph[u][v];
                }

                printSolution(dist, V);
        }
        public static void main(String[] args)
        {
                V = 5;
                System.out.println("Enter the adjacency matric");
                int graph[][] = {   { 0, 10, 0, 0, 100 },
                                    { 10, 0, 50, 0, 0 },
                                    { 8, 50, 0, 20, 10},
                                    { 0, 5, 20, 0, 60 },
                                    { 100, 0, 10, 60, 0 } };
                ShortestPath t = new ShortestPath();
                long start = System.currentTimeMillis();
                t.dijkstra(graph, 0);
                long finish = System.currentTimeMillis();
                long timeElapsed = finish - start;
                System.out.println("TimeElapsed: "+ timeElapsed+"ms");
        }
}
```

7. **Prim's algorithm**

   A phone company wants to lay lines for communication in a city. Different amounts are charged for connecting between a pair of cities. Design and implement a greedy solution such that it forms a spanning tree with minimum cost.



```
class Prims{
        static int V;

        public int minkey(boolean[] mstset,int[] key){
                int min = Integer.MAX_VALUE;
                int minIndex = -1;

                for(int i=0;i<V;i++){
                        if(!mstset[i] && key[i] < min){
                                min = key[i];
                                minIndex = i;
                        }
                }

                return minIndex;
        }

        public void print(int [] parent,int [] key){
                for(int i=1;i<V;i++){
                        System.out.println((i+1)+" - " +(parent[i]+1)+" : "+key[i]);
                }
        }

        public void prims(int [][] graph){
                boolean [] mstset = new boolean[V];
                int [] key = new int[V];
                int [] parent = new int[V];
                for(int i=0;i<V;i++){
```

```
				key[i] = Integer.MAX_VALUE;
				mstset[i] = false;
			}

			key[0] = 0;

			for(int count = 0;count < V-1;count++){
				int u = minkey(mstset,key);
				mstset[u] = true;
				for(int v=0;v<V;v++){
					if(!mstset[v] && graph[u][v] != 0){
						key[v] = graph[u][v];
						parent[v] = u;
					}
				}
			}
			print(parent,key);
		}

		public static void main(String[] args) {
			V = 5;
			int graph[][] = {{ 0, 10, 0, 0, 5 },
				{ 10, 0, 1, 6, 0 },
				{ 8, 1, 0, 2, 7},
				{ 0, 6, 2, 0, 3 },
				{ 5, 0, 7, 3, 0 } };

			Prims t = new Prims();

			long start = System.currentTimeMillis();
			t.prims(graph);
			long finish = System.currentTimeMillis();
			long timeElapsed = finish - start;
			System.out.println("TimeElapsed: "+ timeElapsed+"ms");
		}
	}
```

8. **Interval scheduling**
A drama venue needs to be allocated for different drama school requests such that maximum profit is obtained for the company owning the drama venue. The requests are shown in the table with start –time, finish-time and the amount affordable by the drama school. Design and implement an algorithm such that maximum profit is obtained for the company owning the drama venue.

Drama School	Start-time	Finish-time	Value
1	1	2	100
2	2	5	200
3	3	6	300
4	4	8	400
5	5	9	500
6	6	10	100

```
import java.util.*;
class IntervalScheduling{
        private int [][] jobs;
        private int [] memo;
        private ArrayList<Integer> includedJobs = new ArrayList<Integer>();

        public void calcschedule(int [][] inputJobs){
                jobs = inputJobs;
                memo = new int[jobs.length];
                Arrays.sort(jobs,(a,b)-> Integer.compare(a[2],b[2]));

                memo[0] = 0;

                for(int i=1;i<jobs.length;i++){
                        memo[i] = Math.max(memo[i-1],jobs[i][3] + memo[leastCompatible(i)]);
                }

                System.out.println("Memoization array: "+ Arrays.toString(memo));
                System.out.println("Maximum profit from the optimal set of jobs: "+
memo[memo.length-1]);
                System.out.println("Jobs included in the optimal set:");
                findSolution(memo.length-1);
                for(int i = includedJobs.size()-1;i>=0;i--){
                        System.out.println(getInfo(includedJobs.get(i)));
                }
        }

        private int leastCompatible(int j){
                int low = 0,high = j-1;
                while(low <= high){
```

```
				int mid = (high+low)/2;
				if(jobs[mid][2] == jobs[j][1])
					return mid;
				else if(jobs[mid][2] < jobs[j][1])
					low = mid+1;
				else
					high = mid-1;
			}
			return 0;
		}

		private void findSolution(int j){
			if (j==0)
				return;
			else if(jobs[j][3] + memo[leastCompatible(j)] > memo[j-1]){
				includedJobs.add(j);
				findSolution(leastCompatible(j));
			}
			else
				findSolution(j-1);
		}

		private String getInfo(int i){
			return "Job "+jobs[i][0]+": Time ("+jobs[i][1]+" - "+jobs[i][2]+") Value = "+jobs[i][3] ;
		}

		public static void main(String[] args) {
			IntervalScheduling schedule = new IntervalScheduling();
			int [][] inputJobs = { {0,0,0,0},
									{1,1,2,100},
									{2,2,5,200},
									{3,3,6,300},
									{4,4,8,400},
									{5,5,9,500},
									{6,6,10,100}
								};

		schedule.calcschedule(inputJobs);
		}
}
```