Name:
Class: MCA
Roll No:
Subject: Lab On CA 302 Design and Analysis of Algorithms
PRN No:

---

1) **Write a program for creating max./min. heap using INSERT**

```
//Write a program for creating max. heap using INSERT

import java.util.*;
public class InsertMaxHeap {
    int n;
    int[] a = new int[20];
    public void insert(int[] a, int n) {
        int i, j, item;
        j = n;
        i = n / 2;
        item = a[n];

        while (i > 0 && a[i] < item) {
            a[j] = a[i];
            j = i;
            i = i / 2;
        }
        a[j] = item;
    }
    public void get() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter how many elements to insert into the heap: ");
        n = scanner.nextInt();
        System.out.println("Enter heap elements: ");

        for (int i = 1; i <= n; i++) {
            a[i] = scanner.nextInt();
            insert(a, i);
        }
        scanner.close();
    }
    public void show() {
        System.out.println("Max heap using insert:");
        for (int i = 1; i <= n; i++) {
            System.out.print(a[i] + "\t");
```

```
        }
    }
    public static void main(String[] args) {
        InsertMaxHeap obj = new InsertMaxHeap();
        obj.get();
        obj.show();
    }
}
```

OUTPUT:

Enter how many elements to insert into the heap: 5
Enter heap elements:
9 1 4 6 7
Max heap using insert:
9 7 4 1 6

```java
//Write a program for creating min. heap using INSERT
import java.util.*;
public class InsertMinHeap {
    int n;
    int[] a = new int[20];
    public void insert(int a[], int n) {
        int i, j, item;
        j = n;
        i = n / 2;
        item = a[n];

        while (i > 0 && a[i] > item) {
            a[j] = a[i];
            j = i;
            i = i / 2;
        }
        a[j] = item;
    }
    public void get() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter how many elements to insert into heap: ");
        n = scanner.nextInt();
        System.out.println("Enter heap elements:");

        for (int i = 1; i <= n; i++) {
            a[i] = scanner.nextInt();
            insert(a, i);
        }
        scanner.close();
    }
    public void show() {
        System.out.println("Min heap using insert:");
        for (int i = 1; i <= n; i++) {
            System.out.print(a[i] + "\t");
        }
    }
    public static void main(String[] args) {
        InsertMinHeap obj = new InsertMinHeap();
        obj.get();
        obj.show();
    }
}
```

OUTPUT:

```
Enter how many elements to insert into heap: 5
Enter heap elements:
9 1 4 6 7
Min heap using insert:
1       6       4       9       7
```

2) **Write a program for creating max./min. heap using ADJUST/HEAPIFY**

```java
//write a program to create MAX heap using ADJUST/HEAPIFY
import java.util.*;
class AdjustHeapify {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] a = new int[100];
        int n;
        System.out.print("Enter value of n: ");
        n = sc.nextInt();
        System.out.print("Enter elements: ");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        heapify(a, n);
        System.out.print("Heap using heapify: ");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i] + "\t");
        }
    }
    public static void heapify(int[] a, int n) {
        // Build max heap
        for (int i = n / 2 - 1; i >= 0; i--) {
            adjust(a, i, n);
        }
    }
    public static void adjust(int[] a, int i, int n) {
        int item = a[i];
        int j = 2 * i + 1;
        while (j < n) {
            if (j + 1 < n && a[j] < a[j + 1]) {
                j++;
            }
            if (item >= a[j]) {
                break;
            }
            a[(j - 1) / 2] = a[j];
            j = 2 * j + 1;
        }
        a[(j - 1) / 2] = item;
    }
}
```

```java
//write a program to create MIN heap using ADJUST/HEAPIFY

import java.util.*;
class AdjustHeapify1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] a = new int[100];
        int n;
        System.out.print("Enter value of n: ");
        n = sc.nextInt();
        System.out.print("Enter elements: ");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        heapify(a, n);
        System.out.print("Heap using heapify: ");
        for (int i = 0; i < n; i++) {
            System.out.print(a[i] + "\t");
        }
    }
    public static void heapify(int[] a, int n) {
        // Build min heap
        for (int i = n / 2 - 1; i >= 0; i--) {
            adjust(a, i, n);
        }
    }
    public static void adjust(int[] a, int i, int n) {
        int item = a[i];
        int j = 2 * i + 1;
        while (j < n) {
            if (j + 1 < n && a[j + 1] < a[j]) {
                j++;
            }
            if (item <= a[j]) {
                break;
            }
            a[(j - 1) / 2] = a[j];
            j = 2 * j + 1;
        }
        a[(j - 1) / 2] = item;
    }
}
```

OUTPUT:

```
Enter value of n: 5
Enter elements: 9 1 4 3 6
Heap using heapify: 9   6       4       3       1
```

3) Write a program to implement union and find operation

```
//Write a program to implement union and find operation
import java.util.*;
public class UnionFind {
    private int n;
    private Node[] obj;

    private class Node {
        int info;
        int parent;

        Node(int info) {
            this.info = info;
            this.parent = 0;
        }
    }
    public void input() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter no of nodes:=");
        n = sc.nextInt();
        obj = new Node[n + 1];
        System.out.println("Enter the nodes:=");
        for (int i = 1; i <= n; i++) {
            int info = sc.nextInt();
            obj[i] = new Node(info);
        }
    }
    public void union(int a, int b) {
        int x;
        x = obj[a].parent + obj[b].parent;
        if (obj[a].parent > obj[b].parent) {
            obj[a].parent = b;
            obj[b].parent = x;
        } else {
            obj[b].parent = a;
            obj[a].parent = x;
        }
        System.out.println("\nParent of " + a + " is: " + obj[a].parent);
        System.out.println("Parent of " + b + " is: " + obj[b].parent);
    }
    public int find(int a) {
        int temp, back, store;
        store = back = a;
        while (obj[back].parent > 0) {
```

```
            back = obj[back].parent;
        }
        while (back != a) {
            temp = obj[a].parent;
            obj[a].parent = back;
            a = temp;
        }
        System.out.println("\nParent of " + store + " is: " + obj[store].parent);
        return obj[store].parent;
    }
    public void output() {
        System.out.println("Node\tParent");
        for (int i = 1; i <= n; i++) {
            System.out.println(obj[i].info + "\t" + obj[i].parent);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        UnionFind u = new UnionFind();
        int a, b, choice, flag;
        char ch;
        flag = 0;
        while (true) {
            System.out.println("Operations in this program. Please enter:");
            System.out.println("1 -> Input the node");
            System.out.println("2 -> Union the node");
            System.out.println("3 -> Find the parent of node");
            System.out.println("4 -> Show all nodes and their parents");
            System.out.println("0 -> Quit from the program");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice) {
                case 1:
                    u.input();
                    break;
                case 2:
                    System.out.print("Enter 2 nodes for union operation: ");
                    a = sc.nextInt();
                    b = sc.nextInt();
                    u.union(a, b);
                    break;
                case 3:
                    System.out.print("Enter node for find parent: ");
                    a = sc.nextInt();
```

```
                    u.find(a);
                    break;
                case 4:
                    u.output();
                    break;
                case 0:
                    System.out.print("Do you want to quit (y/n): ");
                    ch = sc.next().charAt(0);
                    if (ch == 'y' || ch == 'Y') {
                        flag = 1;
                    }
                    break;
                default:
                    System.out.println("You have entered an invalid choice.
Please enter a valid choice.");
            }
            if (flag == 1) {
                break;
            }
        }
        sc.close();
    }
}
```

Operations in this program. Please enter:
1 -> Input the node
2 -> Union the node
3 -> Find the parent of node
4 -> Show all nodes and their parents
0 -> Quit from the program
Enter your choice: 1
Enter no of nodes:= 3
Enter the nodes:=
1 2
3
Operations in this program. Please enter:
1 -> Input the node
2 -> Union the node
3 -> Find the parent of node
4 -> Show all nodes and their parents
0 -> Quit from the program
Enter your choice: 2
Enter 2 nodes for union operation: 1 2
Parent of 1 is: 0
Parent of 2 is: 1

Operations in this program. Please enter:
1 -> Input the node
2 -> Union the node
3 -> Find the parent of node
4 -> Show all nodes and their parents
0 -> Quit from the program
Enter your choice: 4
Node Parent
1 0
2 1
3 0
Operations in this program. Please enter:
1 -> Input the node
2 -> Union the node
3 -> Find the parent of node
4 -> Show all nodes and their parents
0 -> Quit from the program
Enter your choice: 4
Node Parent
1 0
2 1
3 0
Operations in this program. Please enter:
1 -> Input the node
2 -> Union the node
3 -> Find the parent of node
4 -> Show all nodes and their parents
0 -> Quit from the program
Enter your choice: 0
Do you want to quit (y/n)

4) **Write a program to find minimum and maximum form a given array**

```java
//Write a program to find minimum and maximum form a given array
import java.util.*;
public class MaxMinFind {
    public static class MaxMinPair {
        int max;
        int min;

        public MaxMinPair(int max, int min) {
            this.max = max;
            this.min = min;
        }
    }
    public static MaxMinPair findMaxMin(int[] arr, int start, int end) {
        if (start == end) {
            return new MaxMinPair(arr[start], arr[start]);
        }
        if (start + 1 == end) {
            return arr[start] < arr[end]
                ? new MaxMinPair(arr[end], arr[start])
                : new MaxMinPair(arr[start], arr[end]);
        }
        int mid = (start + end) / 2;
        MaxMinPair leftPair = findMaxMin(arr, start, mid);
        MaxMinPair rightPair = findMaxMin(arr, mid + 1, end);
        int globalMax = Math.max(leftPair.max, rightPair.max);
        int globalMin = Math.min(leftPair.min, rightPair.min);
        return new MaxMinPair(globalMax, globalMin);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random r = new Random();
        System.out.println("How many elements do you want to insert:\n");
        int n = sc.nextInt();
        int[] a = new int[n];
        System.out.println("System Generated Elements are:\n");

        for (int i = 0; i < n; i++) {
            a[i] = r.nextInt(1000);
        }
        for (int i = 0; i < n; i++) {
            System.out.print("\t" + a[i]);
        }
        MaxMinPair result = findMaxMin(a, 0, n - 1);
```

```
        System.out.println("\nMax value: " + result.max);
        System.out.println("Min value: " + result.min);
    }
}
```

OUTPUT:

How many Element You want Insert:
10
System Generated Element Is:
361 446 270 336 676 713 188 103 82 456
Max value: 713
Min value: 82
Press any key to continue . .

5) **Write a program forsearching element form given array using binary search for n=1000,2000,3000 find exact time of execution**

```
import java.util.*;
public class BinaryS {
    public static void main(String[] args) {
        Random r = new Random();
        Scanner sc = new Scanner(System.in);
        int[] a = new int[6000];
        System.out.println("How many elements do you want to insert:");
        int n = sc.nextInt();
        System.out.println("\nGenerated Elements:");
        for (int i = 1; i <= n; i++) {
            a[i] = r.nextInt(2000);
        }

        for (int i = 1; i <= n; i++) {
            System.out.print("\t" + a[i]);
        }
        for (int i = 1; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                if (a[i] > a[j]) {
                    int t = a[i];
                    a[i] = a[j];
                    a[j] = t;
                }
            }
        }
        System.out.println("\nAfter Sorting Generated Elements:");
        for (int i = 1; i <= n; i++) {
            System.out.print("\t" + a[i]);
        }
        System.out.println("\nEnter the element you want to search:");
        int x = sc.nextInt();
        long st = System.nanoTime();
        int g = binary(a, 0, n, x);
        long en = System.nanoTime();
        double sts = (en - st) / 1000000.0;
        if (g != -1) {
            System.out.println("Total Execution Time: " + sts + " ms");
            System.out.println("\nElement found at index " + g + ", Element is "
+ a[g]);
        } else {
            System.out.println("Total Execution Time: " + sts + " ms");
            System.out.println("\nElement not found.");
```

```
        }
        sc.close();
    }
    public static int binary(int a[], int beg, int end, int x) {
        int mid;
        if (end >= beg) {
            mid = (end + beg) / 2;
            if (a[mid] == x) {
                return mid;
            } else if (a[mid] < x) {
                return binary(a, mid + 1, end, x);
            } else {
                return binary(a, beg, mid - 1, x);
            }
        }
        return -1;
    }
}
```

OUTPUT:
How many Element You want to Insert:
10
Generated Elements:
1987 1798 1590 729 450 876 768 29 1113 1828
After Sorting Generated Elements:
29 450 729 768 876 1113 1590 1798 1828 1987
Enter the Element You want To Search:
450
Total Time Complexity is 0.015
Element Found Index is 2 Element is 450
Press any key to continue . .

6) **Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using HeapSort**

```
import java.util.*;
public class Heap {
    public 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 void heapify(int[] arr, int n, int rootIndex) {
        int largest = rootIndex;
        int leftChild = 2 * rootIndex + 1;
        int rightChild = 2 * rootIndex + 2;

        if (leftChild < n && arr[leftChild] > arr[largest]) {
            largest = leftChild;
        }
        if (rightChild < n && arr[rightChild] > arr[largest]) {
            largest = rightChild;
        }
        if (largest != rootIndex) {
            int temp = arr[rootIndex];
            arr[rootIndex] = arr[largest];
            arr[largest] = temp;
            heapify(arr, n, largest);
        }
    }
}
class MyHeap {
    public static void main(String[] args) {
        Heap h = new Heap();
        Scanner sc = new Scanner(System.in);

        System.out.println("How many elements do you want to insert:");
        int n = sc.nextInt();
        int[] a = new int[n];
```

```
        System.out.println("Enter the " + n + " elements:");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }

        System.out.println("Original array: " + Arrays.toString(a));
        long st = System.nanoTime();
        h.heapSort(a);
        long en = System.nanoTime();
        double s = (en - st) / 1000000.0;

        System.out.println("Sorted array: " + Arrays.toString(a));
        System.out.println("Exact Time Execution: " + s + " ms");
        System.out.println("Thank you:");

        sc.close();
    }
}
```

OUTPUT:
How many Elements You want to Insert:
10
Enter the 10 Elements:
98
63
45
25
78
36
12
52
63
10
Original array: [98, 63, 45, 25, 78, 36, 12, 52, 63, 10]
Sorted array: [10, 12, 25, 36, 45, 52, 63, 63, 78, 98]
Exact Time Execution is:0.031
Thank you:
Press any key to continue . .

7) **Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using MergeSort**

```
import java.util.*;
public class MergeSort {
    public void mergeSort(int[] array, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(array, left, mid);
            mergeSort(array, mid + 1, right);
            merge(array, left, mid, right);
        }
    }
    public void merge(int[] array, int left, int mid, int right) {
        int leftSize = mid - left + 1;
        int rightSize = right - mid;
        int[] leftArray = new int[leftSize];
        int[] rightArray = new int[rightSize];
        for (int i = 0; i < leftSize; i++) {
            leftArray[i] = array[left + i];
        }
        for (int i = 0; i < rightSize; i++) {
            rightArray[i] = array[mid + 1 + i];
        }
        int i = 0;
        int j = 0;
        int k = left;
        while (i < leftSize && j < rightSize) {
            if (leftArray[i] <= rightArray[j]) {
                array[k] = leftArray[i];
                i++;
            } else {
                array[k] = rightArray[j];
                j++;
            }
            k++;
        }
        while (i < leftSize) {
            array[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < rightSize) {
            array[k] = rightArray[j];
            j++;
```

```
            k++;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("How many elements do you want to insert:\n");
        int n = sc.nextInt();
        int[] a = new int[n];
        System.out.println("Enter the " + n + " elements:\n");

        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        MergeSort m = new MergeSort();
        System.out.println("Original Array: " + Arrays.toString(a));
        long startTime = System.nanoTime();
        m.mergeSort(a, 0, n - 1);
        long endTime = System.nanoTime();
        double executionTimeMs = (endTime - startTime) / 1000000.0;
        System.out.println("Sorted Array: " + Arrays.toString(a));
        System.out.println("Execution Time: " + executionTimeMs + " ms");
        System.out.println("Thank you:");
        sc.close();
    }
}
```

OUTPUT:
How many Element You want Insert:
10
Enter the 10 Elements:
46
88
50
58
36
98
87
68
90
59
Original Array: [46, 88, 50, 58, 36, 98, 87, 68, 90, 59]
Sorted Array: [36, 46, 50, 58, 59, 68, 87, 88, 90, 98]
Execution Time: 0.0386 ms
Thank you:
Press any key to continue . .

8) **Write a program for sorting given array in ascending/descending order with n=1000,2000,3000 find exact time of execution using QuickSort**

```
import java.util.*;
public class QuickSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("How many elements do you want to insert?");
        int n = sc.nextInt();
        int[] array = new int[n];

        System.out.println("Enter the elements:");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
        System.out.println("Before sorting:");
        printArray(array);

        long startTime = System.nanoTime();
        quickSort(array, 0, n - 1);
        long endTime = System.nanoTime();

        System.out.println("After sorting:");
        printArray(array);

        double executionTime = (endTime - startTime) / 1000000.0;
        System.out.println("Total execution time: " + executionTime + "
milliseconds");
    }
    public static void quickSort(int[] array, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(array, low, high);
            quickSort(array, low, pivotIndex - 1);
            quickSort(array, pivotIndex + 1, high);
        }
    }
    public static int partition(int[] array, int low, int high) {
        int pivot = array[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (array[j] < pivot) {
                i++;
                int temp = array[i];
                array[i] = array[j];
```

```
                array[j] = temp;
            }
        }
        int temp = array[i + 1];
        array[i + 1] = array[high];
        array[high] = temp;

        return i + 1;
    }
    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

OUTPUT:
How many Element You want Insert:
10
Enter the 10 Elements:
46
88
50
58
36
98
87
68
90
59
Original Array: [46, 88, 50, 58, 36, 98, 87, 68, 90, 59]
Sorted Array: [36, 46, 50, 58, 59, 68, 87, 88, 90, 98]
Execution Time: 0.0386 ms
Thank you:
Press any key to continue . .

9) **Write a program for matrix multiplication using Strassen's matrix multiplication**

```
import java.util.Scanner;
public class MatrixM {
    public static void main(String[] args) {
        int[][] a = new int[10][10];
        int[][] b = new int[10][10];
        int[][] c = new int[10][10];
        int P, Q, R, S, T, U, V;
        Scanner sc = new Scanner(System.in);
        System.out.println("\nEnter the First 2x2 Matrix:\n");
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                a[i][j] = sc.nextInt();
            }
        }
        System.out.println("\nMatrix is \n");
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                System.out.print("\t" + a[i][j]);
            }
            System.out.println("");
        }
        System.out.println("\nEnter the Second 2x2 Matrix:\n");
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                b[i][j] = sc.nextInt();
            }
        }
        System.out.println("\nMatrix is \n");
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                System.out.print("\t" + b[i][j]);
            }
            System.out.println("");
        }
        P = (a[1][1] + a[2][2]) * (b[1][1] + b[2][2]);
        Q = (a[2][1] + a[2][2]) * b[1][1];
        R = (a[1][1]) * (b[1][2] - b[2][2]);
        S = a[2][2] * (b[2][1] - b[1][1]);
        T = (a[1][1] + a[1][2]) * b[2][2];
        U = (a[2][1] - a[1][1]) * (b[1][1] + b[1][2]);
        V = (a[1][2] - a[2][2]) * (b[2][1] + b[2][2]);

        c[1][1] = P + S - T + V;
```

```
        c[1][2] = R + T;
        c[2][1] = Q + S;
        c[2][2] = P + R - Q + U;

        System.out.println("\nMatrix Multiplication is:\n");
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= 2; j++) {
                System.out.print("\t" + c[i][j]);
            }
            System.out.println("");
        }
        System.out.println("\nThank You!");
    }
}
```

OUTPUT:

Enter the First 2X2 Matrix:
1
2
3
4
Matrix is
1 2
3 4
Enter the Second 2X2 Matrix:
5
6
7
8
Matrix is
5 6
7 8
Matrix Multification is:
19 22
43 50

10) Write a program to find solution of Knapsack instant

```
import java.util.Scanner;
public class Knap {
    private float[] p, x, w;
    private float m;
    private int n;
    public void getData() {
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter the number of objects: ");
        n = sc.nextInt();
        System.out.print("\nEnter the capacity of knapsack: ");
        m = sc.nextFloat();
        p = new float[n + 1];
        x = new float[n + 1];
        w = new float[n + 1];
        System.out.println("\nEnter the profits:");
        for (int i = 1; i <= n; i++) {
            System.out.print("p[" + i + "] = ");
            p[i] = sc.nextFloat();
        }

        System.out.println("\nEnter the weights of knapsack:");
        for (int i = 1; i <= n; i++) {
            System.out.print("w[" + i + "] = ");
            w[i] = sc.nextFloat();
        }
    }
    public void knapSol() {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n - 1; j++) {
                if ((p[i] / w[j]) < (p[j + 1] / w[j + 1])) {
                    float tp = p[j];
                    float tw = w[j];
                    p[j] = p[j + 1];
                    w[j] = w[j + 1];
                    p[j + 1] = tp;
                    w[j + 1] = tw;
                }
            }
        }
        int i;
        float cu = m;
        for (i = 1; i <= n; i++) {
            x[i] = 0;
```

```
        }
        for (i = 1; i <= n; i++) {
            if (w[i] > cu) {
                break;
            }
            x[i] = 1;
            cu = cu - w[i];
        }
        if (i <= n) {
            x[i] = cu / w[i];
        }
    }
    public void show() {
        float s = 0;
        System.out.println("\n\nThe optimal solution is ==> x = {");
        for (int i = 1; i <= n; i++) {
            System.out.print(x[i] + ",");
            s = s + p[i] * x[i];
        }
        System.out.print("\t}\n\nOptimal value: " + s);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Knap k = new Knap();
        k.getData();
        k.knapSol();
        k.show();
        sc.close();
    }
}
```

OUTPUT:
Enter the number of objects: 3
Enter the capacity of knapsack: 20
Enter the profits:
p[1] = 24
p[2] = 15
p[3] = 25
Enter the weights of knapsack:
w[1] = 15
w[2] = 10
w[3] = 18
The optimal solution is ==> x = {1.0,0.6666667,0.0, }
Optimal value: 31.0Press any key to continue . .

11) Write a program to find shortest path using single source shortest path

```
import java.util.Scanner;
public class SPath {
    private int[][] cost;
    private int[] dist, s;
    private int n, v;
    private final int infinity = 10000;
    public void input() {
        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter the number of vertices of the graph: ");
        n = sc.nextInt();
        cost = new int[n + 1][n + 1];
        dist = new int[n + 1];
        s = new int[n + 1];
        System.out.println("\nEnter the cost matrix:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                cost[i][j] = sc.nextInt();
            }
        }
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (cost[i][j] == 0) {
                    if (i == j) {
                        cost[i][j] = 0;
                    } else {
                        cost[i][j] = infinity;
                    }
                }
            }
        }
        System.out.print("\nEnter the starting vertex: ");
        v = sc.nextInt();
        shortestPath(v);
        sc.close();
    }
    public void output() {
        System.out.println("\nDistances are:");
        for (int i = 1; i <= n; i++) {
            System.out.print(dist[i] + "\t");
        }
    }
    public void shortestPath(int v) {
        for (int i = 1; i <= n; i++) {
```

```
            s[i] = 0;
            dist[i] = cost[v][i];
        }
        s[v] = 1;
        dist[v] = 0;

        for (int num = 2; num <= n - 1; num++) {
            int temp = dist[1];
            int u = 1;
            for (int i = 1; i <= n; i++) {
                if ((temp > dist[i]) && (s[i] == 0)) {
                    temp = dist[i];
                    u = i;
                }
            }
            s[u] = 1;
            for (int j = 1; j <= n; j++) {
                if (s[j] == 0) {
                    dist[j] = min(dist[j], dist[u] + cost[u][j]);
                }
            }
        }
    }
    public int min(int x, int y) {
        return (x < y) ? x : y;
    }
    public static void main(String[] args) {
        SPath s = new SPath();
        s.input();
        s.output();
    }
}
```

OUTPUT:
Enter the number of vertices of the graph: 6
Enter the cost matrix:
0 50 45 10 25 0
0 0 10 0 0 0
0 0 0 0 30 0
20 0 0 0 15 0
0 20 25 0 0 0
0 0 0 0 3 0
Enter the starting vertex: 1
Distances are:0 50 45 10 25 10000
Press any key to continue . .

12) Write a program to find Minimum-Cost Spanning Trees (Prim's & Kruskal's Algorithm)

```
import java.util.Scanner;
public class SPrim {
    int n;
    int v;
    int cost[][];
    int i, j, s[];
    int e[];
    int near1[];
    int t[][];
    int m, minedge, k, l, mincost;
    int jindex;
    float dist[];

    public void get() {
        m = 1;
        minedge = 9999;
        Scanner sc = new Scanner(System.in);
        System.out.println("\nEnter the number of adjacency vertices: ");
        n = sc.nextInt();
        cost = new int[n + 1][n + 1];
        s = new int[n + 1];
        e = new int[n + 1];
        t = new int[n + 1][3];
        dist = new float[n + 1];
        near1 = new int[n + 1];

        System.out.println("\nEnter the adjacency matrix:");
        for (i = 1; i <= n; i++) {
            for (j = 1; j <= 1; j++) {
                cost[i][j] = sc.nextInt();
                if (cost[i][j] == -1) {
                    cost[i][j] = 9999;
                } else {
                    e[m] = cost[i][j];
                    if (e[m] < minedge) {
                        minedge = e[i];
                        k = i;
                        l = j;
                    }
                }
            }
        }
```

```
        sc.close();
    }

    public void prime() {
        t[1][1] = k;
        t[1][2] = l;
        mincost = cost[k][l];

        for (i = 1; i <= n; i++) {
            if (cost[i][1] < cost[i][k]) {
                near1[i] = l;
            } else {
                near1[i] = k;
            }
        }

        near1[k] = near1[l] = 0;
        int minj = 9999;

        for (i = 2; i <= n; i++) {
            minj = 9999;

            for (j = 1; j <= 1; j++) {
                if (near1[j] != 0) {
                    if (cost[j][near1[j]] < minj) {
                        minj = cost[j][near1[j]];
                        jindex = j;
                    }
                }
            }

            t[i][1] = jindex;
            t[i][2] = near1[jindex];
            mincost = mincost + cost[jindex][near1[jindex]];
            near1[jindex] = 0;

            for (int k1 = 1; k1 <= n; k1++) {
                if (near1[k1] != 0 && cost[k1][near1[k1]] > cost[k1][jindex]) {
                    near1[k1] = jindex;
                }
            }
        }

        System.out.println("Min cost: " + mincost);
    }
```

```
    public void display() {
        System.out.println("\nMinimum spanning tree path is as follows: ");
        System.out.print(t[1][1] + " -> " + t[1][2]);

        for (i = 2; i <= n; i++) {
            System.out.print(" -> " + t[i][j]);
        }
    }

    public static void main(String[] args) {
        SPrim d = new SPrim();
        d.get();
        d.prime();
        d.display();
    }
}
```

OUTPUT:
Enter the no.of vertices 7
Enter the adjacency matrix
0 28 -1 -1 -1 10 -1
28 -1 16 -1 -1 -1 14
-1 16 0 12 -1 -1 -1
-1 -1 12 0 22 -1 18
-1 -1 -1 22 0 25 24
10 -1 -1 -1 25 0 -1
-1 14 -1 18 25 -1 0
Mincost 99
Minimum spanning tree path as follow
1-->1-->6-->5-->4-->3-->--2 -->7

13) Write a program to find shortest path using all pair path

```
import java.util.Scanner;

public class SPath2 {
    private int[][] a;
    private int[][] s;
    private int n;

    public void read() {
        Scanner sc = new Scanner(System.in);
        System.out.print("\nHow many vertices in the graph: ");
        n = sc.nextInt();
        a = new int[n + 1][n + 1];
        s = new int[n + 1][n + 1];

        System.out.println("\nEnter the adjacency matrix:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                System.out.print("(" + i + "," + j + ") ");
                a[i][j] = sc.nextInt();
                if (a[i][j] == 0) {
                    s[i][j] = 9999;
                } else {
                    s[i][j] = a[i][j];
                }
            }
        }
        sc.close();
    }

    public int min(int m, int n) {
        return (m < n) ? m : n;
    }

    public void allPath() {
        for (int k = 1; k <= n; k++) {
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= n; j++) {
                    if (i == j) {
                        s[i][j] = 0;
                    }
                    s[i][j] = min(s[i][j], s[i][k] + s[k][j]);
                }
```

```
            }
        }

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (s[i][j] >= 9999) {
                    s[i][j] = 0;
                }
            }
        }
    }

    public void display() {
        System.out.println("\nThe shortest path matrix is:");
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                System.out.print("\t" + s[i][j]);
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        SPath2 s = new SPath2();
        s.read();
        s.allPath();
        s.display();
    }
}
```

OUTPUT:
how many vertices in graph 3
enter the adjancy matrix
(1,1)0
(1,2)4
(1,3)11
(2,1)6
(2,2)0
(2,3)2
(3,1)3
(3,2)0
(3,3)0
the shortest path matrix is:0 4 6
5 0 2
3 7 0

14) Write a program to find longest common subsequence

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

    public static String findLCS(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        int[][] dp = new int[m + 1][n + 1];

        // Fill the dp array using bottom-up approach
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        // Reconstruct the LCS
        StringBuilder lcs = new StringBuilder();
        int i = m, j = n;
        while (i > 0 && j > 0) {
            if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                lcs.insert(0, s1.charAt(i - 1));
                i--;
                j--;
            } else {
                if (dp[i - 1][j] > dp[i][j - 1]) {
                    i--;
                } else {
                    j--;
                }
            }
        }
        return lcs.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the first string: ");
        String s1 = sc.next();
        System.out.print("Enter the second string: ");
```

```
        String s2 = sc.next();
        String lcs = findLCS(s1, s2);
        System.out.println("Longest Common Subsequence: " + lcs);
        sc.close();
    }
}
```

OUTPUT:
Enter the first string: ABCBDAB
Enter the second string: BDCAB
Longest Common Subsequence: BCDAB

15) Write a program to implement breadth first and depth first search

```
import java.util.Scanner;

public class BFS1 {
    private int[][] a;
    private int[] v;
    private int n;
    private int[] q;
    private int f, b, r;

    public BFS1() {
        f = b = r = 0;
    }

    public void getdata() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        n = sc.nextInt();
        a = new int[n + 1][n + 1];
        v = new int[n + 1];
        q = new int[15];
        System.out.println("Enter the matrix:");

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

    public void bfs(int v1) {
        int w;
        v[v1] = 1;
        System.out.print(v1 + " ");

        while (true) {
            for (w = 1; w <= n; w++) {
                if (a[v1][w] == 1) {
                    if (v[w] == 0) {
                        if (f == 0 && r == 0) {
                            f = r = 1;
                        } else {
                            r++;
```

```
                        }
                        q[r] = w;
                        v[w] = 1;
                        System.out.print(w + " ");
                    }
                }
            }
            if (f == 0 && r == 0) {
                break;
            }
            v1 = q[f];
            if (f == r) {
                f = r = 0;
            } else {
                f++;
            }
        }
    }

    public void display() {
        System.out.print("Sequence of nodes in BFS is: ");
        bfs(1);
        System.out.println();

        for (int i = 1; i <= n; i++) {
            v[i] = 0;
        }
    }

    public static void main(String[] args) {
        BFS1 b = new BFS1();
        b.getdata();
        b.display();
    }
}
```

```
import java.util.Scanner;

public class DFS1 {
    private int[][] a;
    private int[] v;
    private int n;

    public void getdata() {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of vertices: ");
        n = sc.nextInt();
        a = new int[n + 1][n + 1];
        v = new int[n + 1];
        System.out.println("Enter the matrix:");

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

    public int dfs(int i) {
        int w;
        v[i] = 1;
        System.out.print(i + " ");

        for (w = 1; w <= n; w++) {
            if (a[i][w] == 1) {
                if (v[w] == 0) {
                    dfs(w);
                }
            }
        }
        return 0;
    }

    public void display() {
        System.out.print("Sequence of nodes in DFS is: ");
        dfs(1);
        System.out.println();

        for (int i = 1; i <= n; i++) {
            v[i] = 0;
        }
    }

    public static void main(String[] args) {
        DFS1 b = new DFS1();
        b.getdata();
        b.display();
    }
```

```
}
```

OUTPUT:

```
enter no of vertices= 5
0 1 1 0 0
0 0 0 1 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
sequence of node is = 1 2 3 4 5
```

16) Write a program to implement breadth first and depth first traversal

```java
import java.util.LinkedList;
public class Graph {
    private int V; // Number of vertices
    private LinkedList<Integer>[] adjList; // Adjacency list

    public Graph(int v) {
        V = v;
        adjList = new LinkedList[v];
        for (int i = 0; i < v; i++) {
            adjList[i] = new LinkedList<>();
        }
    }

    public void addEdge(int src, int dest) {
        adjList[src].add(dest);
        adjList[dest].add(src); // Uncomment this line for undirected graph
    }

    private void dfsUtil(int vertex, boolean[] visited) {
        visited[vertex] = true;
        System.out.print(vertex + " ");
        for (int neighbor : adjList[vertex]) {
            if (!visited[neighbor]) {
                dfsUtil(neighbor, visited);
            }
        }
    }

    public void dfs(int start) {
        boolean[] visited = new boolean[V];
        dfsUtil(start, visited);
    }

    public static void main(String[] args) {
        Graph graph = new Graph(6);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(1, 4);
        graph.addEdge(2, 4);
        graph.addEdge(3, 4);
        graph.addEdge(3, 5);
        graph.addEdge(4, 5);
```

```
        System.out.print("DFS Traversal: ");
        graph.dfs(0);
    }
}
```

OUTPUT;
DFS Traversal is 0 1 3 4 2 5

17) Write a program to find all solutions for 8-queen problem using backtracking

```
import java.util.*;

public class NQueens {
    private int n;
    private int[] x;
    private int c;

    public NQueens(int n) {
        this.n = n;
        x = new int[n + 1];
        c = 0;
    }

    private boolean place(int k) {
        for (int i = 1; i < k; i++) {
            if (x[i] == x[k] || Math.abs(x[i] - x[k]) == Math.abs(i - k)) {
                return false;
            }
        }
        return true;
    }

    private void nqueen(int n) {
        x[1] = 0;
        int k = 1;

        while (k > 0) {
            x[k] = x[k] + 1;

            while (x[k] <= n && !place(k)) {
                x[k] = x[k] + 1;
            }

            if (x[k] <= n) {
                if (k == n) {
                    c++;
                    System.out.print(c + "->");
                    for (int i = 1; i <= n; i++) {
                        System.out.print(x[i]);
                    }
                    System.out.println();
                } else {
                    k = k + 1;
```

```
                    x[k] = 0;
                }
            } else {
                k = k - 1;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of queens: ");
        int n = sc.nextInt();
        NQueens nQueens = new NQueens(n);
        System.out.println("The solution for " + n + " queen problem is...");
        nQueens.nqueen(n);
        System.out.println("The total solutions are: " + nQueens.c);
        sc.close();
    }
}
```

OUTPUT:

enter the no. of queen4
the solution for4queen problem is…
1->2413
2->3142
the total sol. Are 2