EXPERIMENT - 1

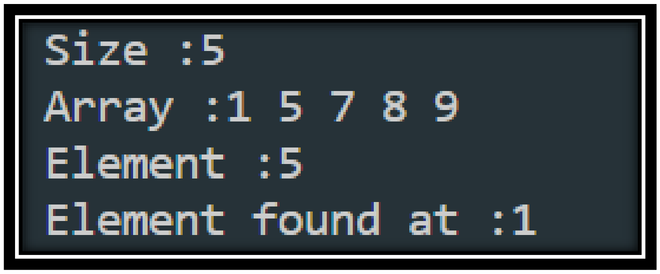
AIM : Program for Recursive Linear and Binary search.

PROGRAM :

#Linear Search

```
import java.util.* ;
import java.util.ArrayList ;
class daa{
   static int LinearSearch( ArrayList<Integer> arr, int a, int idx ){
      if ( idx < arr.size() ){
         if ( arr.get(idx) == a ){
            return idx ;
         }else{
            return LinerSearch( arr, a, idx+1) ;
         }
      }else
         return -1 ;
   }
   public static void main( String args[] ){
      Scanner sc = new Scanner(System.in);
      System.out.print("Size :") ;
      int n = sc.nextInt() ;
      System.out.print("Array :") ;
      ArrayList <Integer> arr = new ArrayList<Integer>() ;
      for( int i=0 ; i<n ; i++ ){
         arr.add(sc.nextInt()) ;
      }
      System.out.print("Element :") ;
      int ele = sc.nextInt() ;
      System.out.print("Element found at :") ;
      System.out.println(LinearSearch( arr, ele, 0)) ;
   }
}
```

OUTPUT :



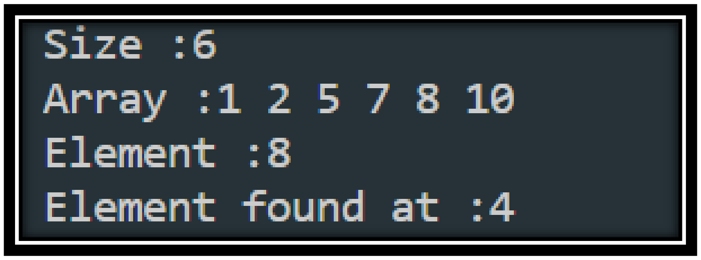
```
Size :5
Array :1 5 7 8 9
Element :5
Element found at :1
```

PROGRAM :

#Binary Search

```
import java.util.* ;
import java.util.ArrayList ;
class daa{
    static int BinarySearch( ArrayList<Integer> arr, int a, int l, int h){
        int mid = (l+h)/2 ;
        if (arr.get(mid) == a ){
            return mid ;
        }else if( arr.get(mid) > a){
            return BinarySearch( arr, a, l, mid-1 ) ;
        }else if( arr.get(mid) < a){
            return BinarySearch( arr, a, mid+1, h ) ;
        }
        return -1 ;
    }    public static void main( String args[] ){
        Scanner sc = new Scanner(System.in);
        System.out.print("Size :") ;
        int n = sc.nextInt() ;
        System.out.print("Array :") ;
        ArrayList <Integer> arr = new ArrayList<Integer>() ;
        for( int i=0 ; i<n ; i++ ){
            arr.add(sc.nextInt()) ;
        }
        System.out.print("Element :") ;
        int ele = sc.nextInt() ;
        System.out.print("Element found at :") ;
        System.out.println(BinarySearch( arr, ele, 0, n)) ;
    }
}
```

OUTPUT :



```
Size :6
Array :1 2 5 7 8 10
Element :8
Element found at :4
```

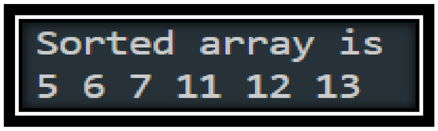
EXPERIMENT - 2

AIM : Program for Heap Sort.

PROGRAM :

```
public class daa {
        public static void heapsort(int arr[]){
                for (int i = arr.length / 2 - 1; i >= 0; i--)
                        heapify(arr, arr.length, i);
                for (int i = arr.length - 1; i >= 0; i--) {
                        int temp = arr[0];
                        arr[0] = arr[i];
                        arr[i] = temp;
                        heapify(arr, i, 0);
                }
        }
        static void heapify(int arr[], int n, int i){
                int largest = i;  int l = 2 * i + 1;  int r = 2 * i + 2;
                if (l < n && arr[l] > arr[largest])
                        largest = l;
                if (r < n && arr[r] > arr[largest])
                        largest = r;
                if (largest != i) {
                        int swap = arr[i];
                        arr[i] = arr[largest];
                        arr[largest] = swap;
                        heapify(arr, n, largest);
                }
        }
        static void printArray(int arr[]){
                for (int i = 0; i < arr.length; ++i)
                        System.out.print(arr[i] + " ");
                System.out.println();
        }
        public static void main(String args[]){
                int arr[] = { 12, 11, 13, 5, 6, 7 };
                heapsort( arr ) ;
                System.out.println("Sorted array is");
                printArray(arr);
        }
}
```

OUTPUT :



```
Sorted array is
5 6 7 11 12 13
```

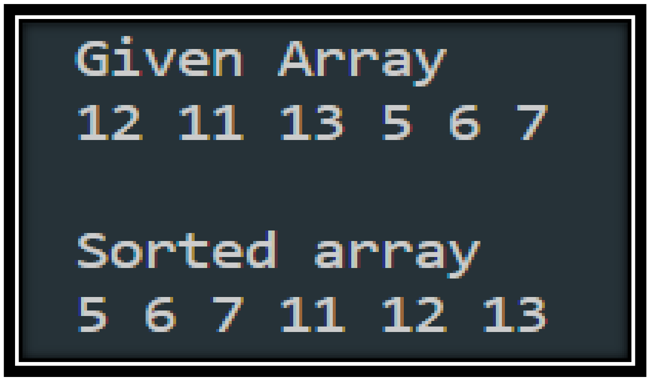
EXPERIMENT - 3

AIM : Program for Merge Sort.

PROGRAM :

```
class daa{
        static void merge(int arr[], int l, int m, int r){
                int n1 = m - l + 1;
                int n2 = r - m ;
                int L[] = new int[n1];
                int R[] = new int[n2];
                for (int i = 0; i < n1; ++i)
                        L[i] = arr[l + i];
                for (int j = 0; j < n2; ++j)
                        R[j] = arr[m + 1 + j];
                int i = 0, j = 0 ;
                int k = l;
                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++;
                }
        }

        static void sort(int arr[], int l, int r){
                if (l < r) {
                        int m = (l + r) / 2;
                        sort(arr, l, m);
                        sort(arr, m + 1, r);
                        merge(arr, l, m, r);
                }
        }
        static void printArray(int arr[]){
                for (int i = 0; i < arr.length; ++i)
                        System.out.print(arr[i] + " ");
                System.out.println();
        }
        public static void main(String args[]){
                int arr[] = { 12, 11, 13, 5, 6, 7 };
                System.out.println("Given Array");
                printArray(arr);
                sort(arr, 0, arr.length - 1);
                System.out.println("\nSorted array");
                printArray(arr);
        }
}
```

OUTPUT :



```
Given Array
12 11 13 5 6 7

Sorted array
5 6 7 11 12 13
```

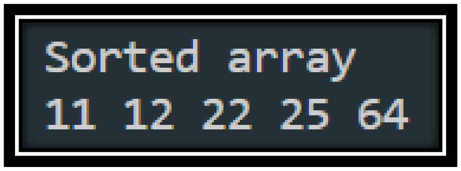
EXPERIMENT - 4

AIM : Program for Selection Sort.

PROGRAM :

```
import java.io.*;
public class daa{
    static void sort(int arr[]){
        int n = arr.length;
        for (int i = 0; i < n-1; i++){
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }
    static void printArray(int arr[]){
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
    public static void main(String args[]){
        int arr[] = {64,25,12,22,11};
        sort(arr);
        System.out.println("Sorted array");
        printArray(arr);
    }
}
```

OUTPUT :



```
Sorted array
11 12 22 25 64
```

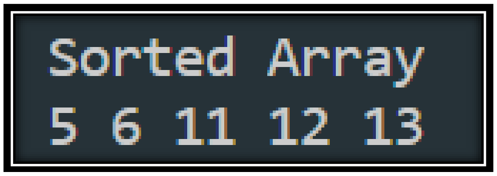
EXPERIMENT - 5

AIM : Program for Insertion Sort.

PROGRAM :

```
public class daa{
    static void sort(int arr[]){
        int n = arr.length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = key;
        }
    }
    static void printArray(int arr[]){
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
    public static void main(String args[]){
        int arr[] = { 12, 11, 13, 5, 6 };
   sort(arr) ;
   System.out.println("Sorted Array ") ;
        printArray(arr);
    }
}
```

OUTPUT :



```
Sorted Array
5 6 11 12 13
```

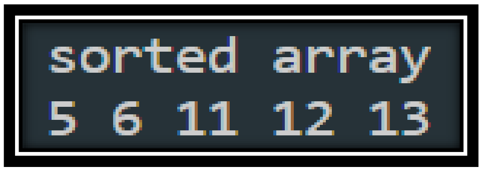
EXPERIMENT - 6

AIM : Program for Quick Sort.

PROGRAM :

```
class daa{
    static int partition(int arr[], int low, int high){
        int pivot = arr[high], i = (low-1);
        for (int j=low; j<high; j++){
            if (arr[j] <= pivot){
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;
        return i+1;
    }
    static void sort(int arr[], int low, int high){
        if (low < high){
            int pi = partition(arr, low, high);
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }
    static void printArray(int arr[]){
        for (int i=0; i<arr.length; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
    public static void main(String args[]){
        int arr[] = {12, 11, 13, 5, 6 };
        sort(arr, 0, arr.length-1);
        System.out.println("sorted array");
        printArray(arr);
    }
}
```

OUTPUT :



```
sorted array
5 6 11 12 13
```

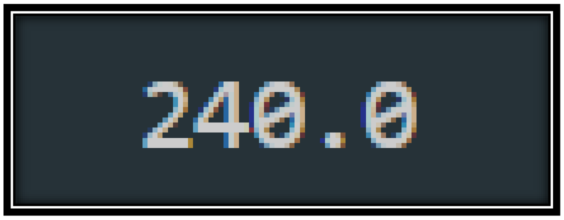
EXPERIMENT - 7

AIM : Knapsack Problem using Greedy Solution.

PROGRAM :

```
import java.lang.*;
import java.util.Arrays;
import java.util.Comparator;

public class daa{
    private static double getMaxValue(ItemValue[] arr, int capacity){
        Arrays.sort(arr, new Comparator<ItemValue>() {
            @Override
            public int compare(ItemValue item1, ItemValue item2){
                double cpr1 = new Double((double)item1.profit /
                                    (double)item1.weight);
                double cpr2 = new Double((double)item2.profit /
                                    (double)item2.weight);
                if (cpr1 < cpr2)
                    return 1;
                else
                    return -1;
            }
        });
        double totalValue = 0d;
        for (ItemValue i : arr) {
            int curWt = (int)i.weight;
            int curVal = (int)i.profit;
            if (capacity - curWt >= 0) {
                capacity = capacity - curWt;
                totalValue += curVal;
            }else {
                double fraction = ((double)capacity / (double)curWt);
                totalValue += (curVal * fraction);
                capacity = (int)(capacity - (curWt * fraction));
                break;
            }
        }
        return totalValue;
    }
    static class ItemValue {
        int profit, weight;
        public ItemValue(int val, int wt){
            this.weight = wt;
            this.profit = val;
        }
    }
    public static void main(String[] args){
        ItemValue[] arr = { new ItemValue(60, 10),
                                    new ItemValue(100, 20),
                                    new ItemValue(120, 30) };
        int capacity = 50;
        double maxValue = getMaxValue(arr, capacity);
        System.out.println(maxValue);
    }
}
```

OUTPUT :



```
240.0
```

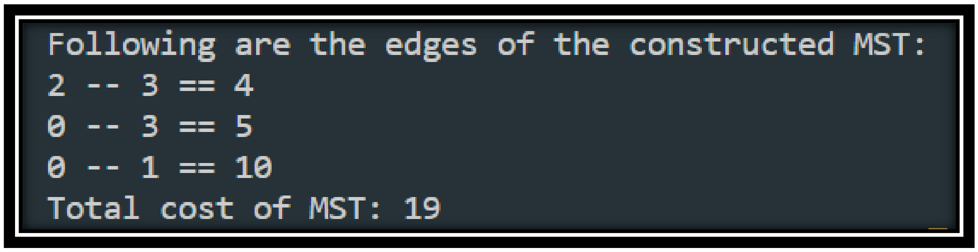
EXPERIMENT - 8

AIM : Minimum Spanning tree using Kruskal’s algorithm.

PROGRAM :

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
public class daa{
    static class Edge {
        int src, dest, weight;
        public Edge(int src, int dest, int weight){
            this.src = src;
            this.dest = dest;
            this.weight = weight;
        }
    }
    static class Subset {
        int parent, rank;
        public Subset(int parent, int rank){
            this.parent = parent;
            this.rank = rank;
        }
    }
    public static void main(String[] args) {
        int V = 4;
        List<Edge> graphEdges = new ArrayList<Edge>(
            List.of(new Edge(0, 1, 10), new Edge(0, 2, 6),
                    new Edge(0, 3, 5), new Edge(1, 3, 15),
                    new Edge(2, 3, 4)));
        graphEdges.sort(new Comparator<Edge>() {
            @Override public int compare(Edge o1, Edge o2){
                return o1.weight - o2.weight;
            }
        });
        kruskals(V, graphEdges);
    }
    private static void kruskals(int V, List<Edge> edges) {
        int j = 0;
        int noOfEdges = 0;
        Subset subsets[] = new Subset[V];
        Edge results[] = new Edge[V];
        for (int i = 0; i < V; i++) {
            subsets[i] = new Subset(i, 0);
        }
        while (noOfEdges < V - 1) {
            Edge nextEdge = edges.get(j);
            int x = findRoot(subsets, nextEdge.src);
            int y = findRoot(subsets, nextEdge.dest);
            if (x != y) {
                results[noOfEdges] = nextEdge;
                union(subsets, x, y);
                noOfEdges++;
            }
            j++;
        }
        System.out.println("Following are the edges of the constructed MST:");
        int minCost = 0;
        for (int i = 0; i < noOfEdges; i++) {
            System.out.println(results[i].src + " -- " + results[i].dest + " == " +
results[i].weight);
            minCost += results[i].weight;
        }
        System.out.println("Total cost of MST: " + minCost);
    }
    private static void union(Subset[] subsets, int x, int y){
        int rootX = findRoot(subsets, x);
        int rootY = findRoot(subsets, y);
        if (subsets[rootY].rank < subsets[rootX].rank) {
            subsets[rootY].parent = rootX;
        }
        else if (subsets[rootX].rank < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        }
        else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }
    private static int findRoot(Subset[] subsets, int i){
        if (subsets[i].parent == i)
            return subsets[i].parent;
        subsets[i].parent = findRoot(subsets, subsets[i].parent);
        return subsets[i].parent;
    }
}
```

OUTPUT :



```
Following are the edges of the constructed MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total cost of MST: 19
```

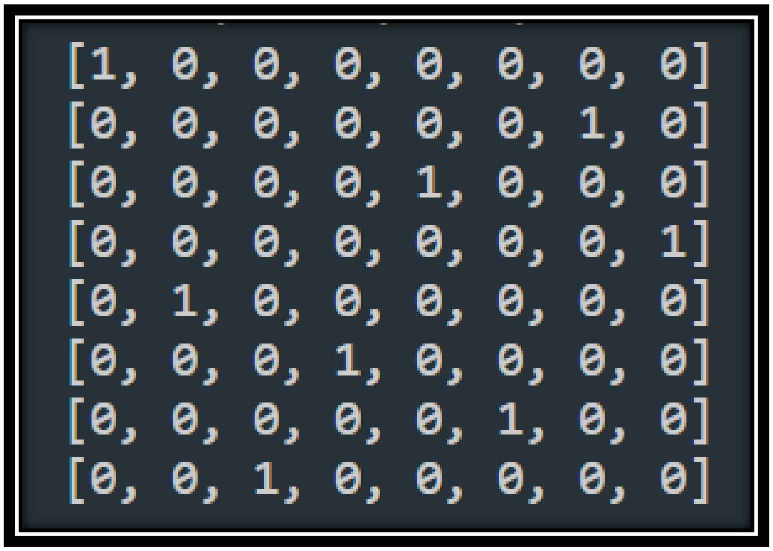
EXPERIMENT - 9

AIM : 8-Queens problem using Backtracking Programming.

PROGRAM :

```
import java.util.Arrays;

class daa{
    static final int N = 8;

    static boolean isSafe(int[][] board, int row, int col){
        for (int x = 0; x < col; x++)
            if (board[row][x] == 1)
                return false;
        for (int x = row, y = col; x >= 0 && y >= 0; x--, y--)
            if (board[x][y] == 1)
                return false;
        for (int x = row, y = col; x < N && y >= 0; x++, y--)
            if (board[x][y] == 1)
                return false;
        return true;
    }

    static boolean solveNQueens(int[][] board, int col){
        if (col == N) {
            for (int[] row : board)
                System.out.println(Arrays.toString(row));
            System.out.println();
            return true;
        }
        for (int i = 0; i < N; i++) {
            if (isSafe(board, i, col)) {
                board[i][col] = 1;
                if (solveNQueens(board, col + 1))
                    return true;
                board[i][col] = 0;
            }
        }
        return false;
    }

    public static void main(String[] args){
        int[][] board = new int[N][N];
        if (!solveNQueens(board, 0))
            System.out.println("No solution found");
    }
}
```

OUTPUT :



```
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
```

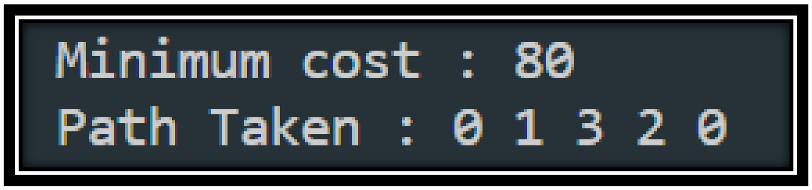
EXPERIMENT - 10

AIM : Traveling Salesman problem using branch-and-bound approach.

PROGRAM :

```
import java.util.*;
class daa{
    static int N = 4;
    static int final_path[] = new int[N + 1];
    static boolean visited[] = new boolean[N];
    static int final_res = Integer.MAX_VALUE;
    static void copyToFinal(int curr_path[]){
        for (int i = 0; i < N; i++)
            final_path[i] = curr_path[i];
        final_path[N] = curr_path[0];
    }
    static int firstMin(int adj[][], int i){
        int min = Integer.MAX_VALUE;
        for (int k = 0; k < N; k++)
            if (adj[i][k] < min && i != k)
                min = adj[i][k];
        return min;
    }
    static int secondMin(int adj[][], int i){
        int first = Integer.MAX_VALUE, second = Integer.MAX_VALUE;
        for (int j=0; j<N; j++){
            if (i == j)
                continue;
            if (adj[i][j] <= first){
                second = first;
                first = adj[i][j];
            }
            else if (adj[i][j] <= second &&
                    adj[i][j] != first)
                second = adj[i][j];
        }
        return second;
    }
    static void TSPRec(int adj[][], int curr_bound, int curr_weight, int level, int curr_path[]){
        if (level == N){
            if (adj[curr_path[level - 1]][curr_path[0]] != 0){
                int curr_res = curr_weight + adj[curr_path[level-1]][curr_path[0]];
                if (curr_res < final_res){
                    copyToFinal(curr_path);
                    final_res = curr_res;
                }
            }
            return;
        }
        for (int i = 0; i < N; i++){
            if (adj[curr_path[level-1]][i] != 0 && visited[i] == false){
                int temp = curr_bound;
                curr_weight += adj[curr_path[level - 1]][i];
                if (level==1)
                curr_bound -= ((firstMin(adj, curr_path[level - 1]) +
                                                firstMin(adj, i))/2);
                else
                curr_bound -= ((secondMin(adj, curr_path[level - 1]) +
                                                firstMin(adj, i))/2);
                if (curr_bound + curr_weight < final_res){
                    curr_path[level] = i;
                    visited[i] = true;
                    TSPRec(adj, curr_bound, curr_weight, level + 1,
                        curr_path);
                }
                curr_weight -= adj[curr_path[level-1]][i];
                curr_bound = temp;
                Arrays.fill(visited,false);
                for (int j = 0; j <= level - 1; j++)
                    visited[curr_path[j]] = true;
            }
        }
    }static void TSP(int adj[][]){
        int curr_path[] = new int[N + 1];
        int curr_bound = 0;
        Arrays.fill(curr_path, -1);
        Arrays.fill(visited, false);
        for (int i = 0; i < N; i++)
            curr_bound += (firstMin(adj, i) + secondMin(adj, i));
        curr_bound = (curr_bound==1)? curr_bound/2 + 1 : curr_bound/2;
        visited[0] = true;
        curr_path[0] = 0;
        TSPRec(adj, curr_bound, 0, 1, curr_path);
    }
    public static void main(String[] args) {
        int adj[][] = {{0, 10, 15, 20}, {10, 0, 35, 25},{15, 35, 0, 30}, {20, 25, 30, 0} };
        TSP(adj);
        System.out.printf("Minimum cost : %d\n", final_res);
        System.out.printf("Path Taken : ");
        for (int i = 0; i <= N; i++) {
            System.out.printf("%d ", final_path[i]);
        }
    }
}
```

OUTPUT :



```
Minimum cost : 80
Path Taken : 0 1 3 2 0
```