EXPERIMENT-1

Aim: To implement following algorithm using array as data structure and analyse its time complexity.

a) Bubble sort
b) Insertion sort
c) Selection sort

Pseudo Code:

Source Code:

```
import java.util.Scanner;

public class Program1 {
   static int n,comp;
   static int ar[];
   static void bubbleSort() {
      comp=0;
      for (int i=0;i<n-1;i++) {
         for (int j=0;j<n-1-i;j++) {
            if (ar[j]>ar[j+1]) {
               int temp=ar[j];     ar[j]=ar[j+1];       ar[j+1]=temp;   comp++;
            }
         }
      }
   }
   static void insertionSort() {
      comp=0;
      for (int i=0;i<n-1;i++) {
         for (int j=i+1;j>0;j--) {
            if (ar[j]<ar[j-1]) {
               int temp=ar[j];     ar[j]=ar[j-1];       ar[j-1]=temp;   comp++;
            }
            else {  break;}
         }
      }
   }
   static void selectionSort() {
      comp=0;
      for (int i=0;i<n-1;i++) {
         for (int j=i+1;j<n;j++) {
            if (ar[i]>ar[j]) {
               int temp=ar[j];     ar[j]=ar[i];       ar[i]=temp;   comp++;
            }
         }
      }
   }
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the number of elements in the array: ");
      n=s.nextInt();
      ar=new int[n];
      System.out.print("Enter the elements of the array: ");
      for (int i=0;i<n;i++) {
         ar[i]=s.nextInt();
      }
      System.out.println("Bubble sort(1), Insertion sort(2), Selection sort(3)");
      System.out.print("Enter your choice: ");
```

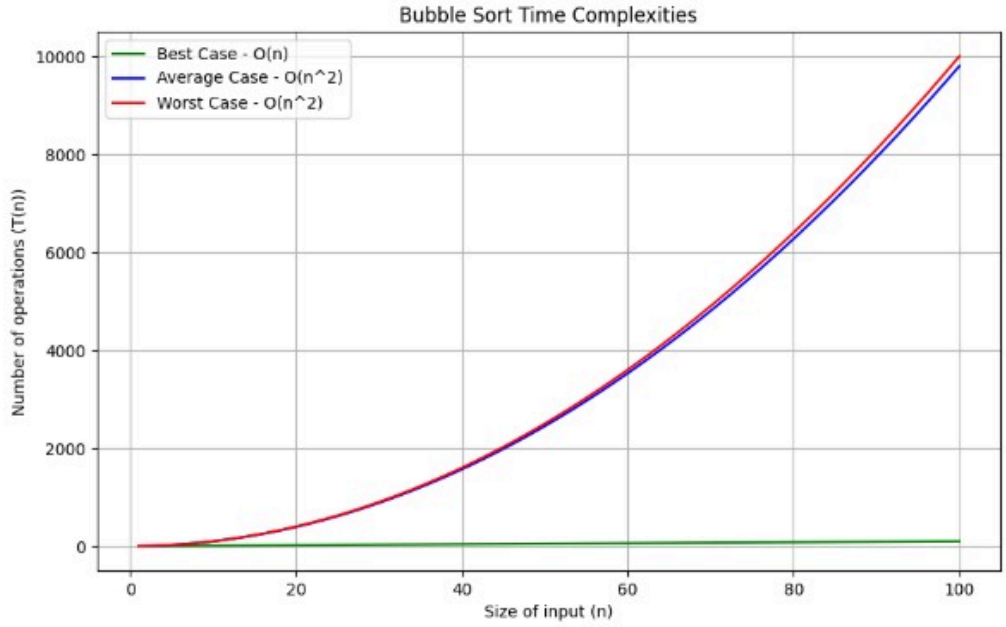
```
        int choice=s.nextInt();
        switch(choice) {
           case 1: bubbleSort();
              break;
           case 2: insertionSort();
              break;
           case 3: selectionSort();
              break;
        }
        System.out.print("The Sorted Array is: ");
        for (int i=0;i<n-1;i++) {
           System.out.print(ar[i]+ ",");
        }
        System.out.println(ar[n-1]);
        System.out.println("Number of comparisons are: "+ comp);
    }
}
```

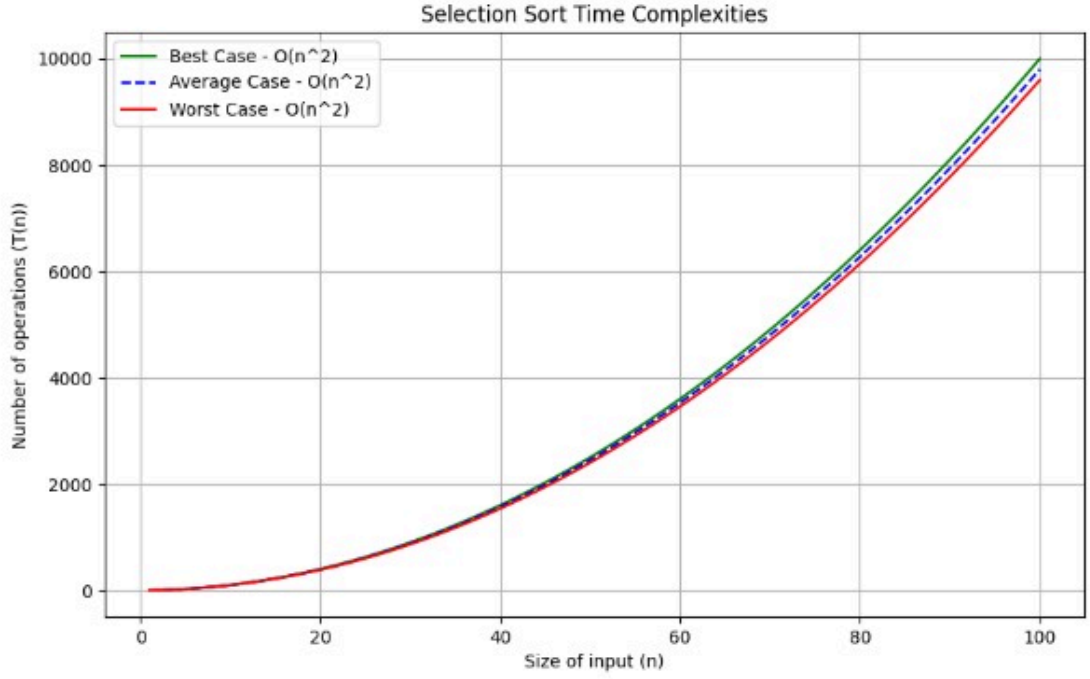
Output:

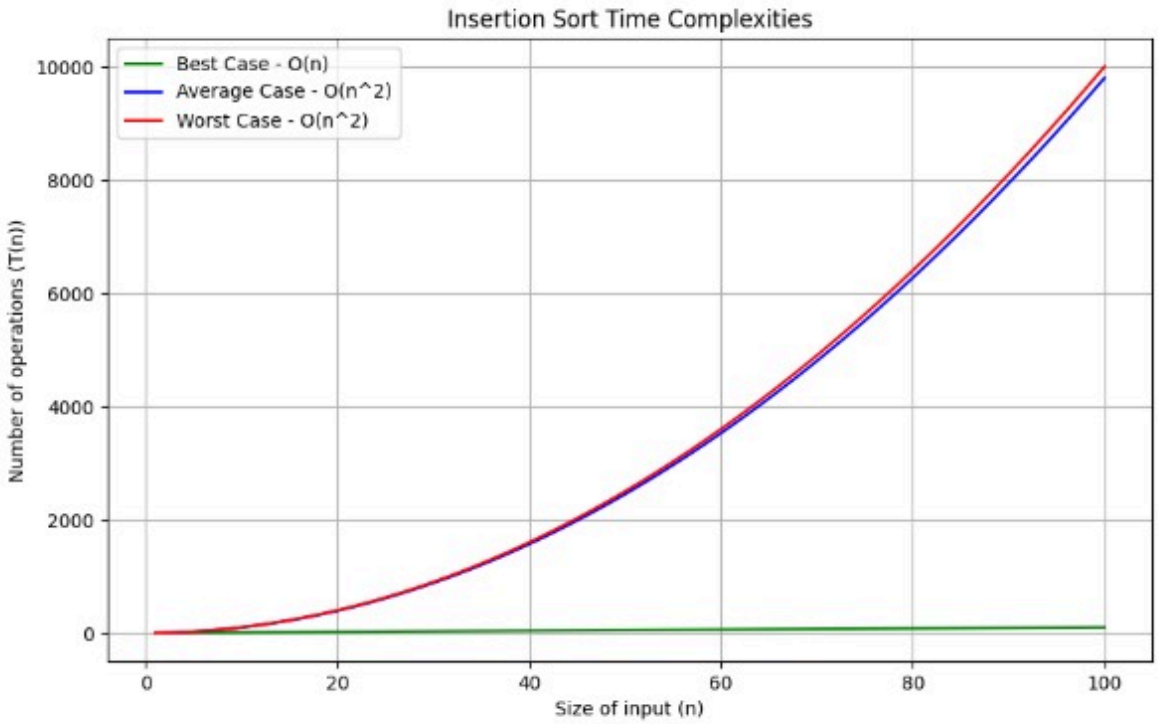
```
Enter the number of elements in the array: 8
Enter the elements of the array: 6 5 3 1 8 7 2 4
Bubble sort(1), Insertion sort(2), Selection sort(3)
Enter your choice: 1
The Sorted Array is: 1,2,3,4,5,6,7,8
Number of comparisons are: 16
```

```
Bubble sort(1), Insertion sort(2), Selection sort(3)
Enter your choice: 2
The Sorted Array is: 1,2,3,4,5,6,7,8
Number of comparisons are: 16
```

```
Bubble sort(1), Insertion sort(2), Selection sort(3)
Enter your choice: 3
The Sorted Array is: 1,2,3,4,5,6,7,8
Number of comparisons are: 16
```


Bubble Sort Time Complexities
Best Case - O(n)
Average Case - O(n^2)
Worst Case - O(n^2)
Number of operations (T(n))
Size of input (n)
0
2000
4000
6000
8000
10000
0
20
40
60
80
100


Selection Sort Time Complexities
Best Case - O(n^2)
Average Case - O(n^2)
Worst Case - O(n^2)
Number of operations (T(n))
Size of input (n)
0
2000
4000
6000
8000
10000
0
20
40
60
80
100


Insertion Sort Time Complexities
Best Case - O(n)
Average Case - O(n^2)
Worst Case - O(n^2)
Number of operations (T(n))
Size of input (n)
0
2000
4000
6000
8000
10000
0
20
40
60
80
100

EXPERIMENT-2

Aim: To implement Linear search and Binary search and analyse its time complexity.

Pseudo Code:

Source Code:

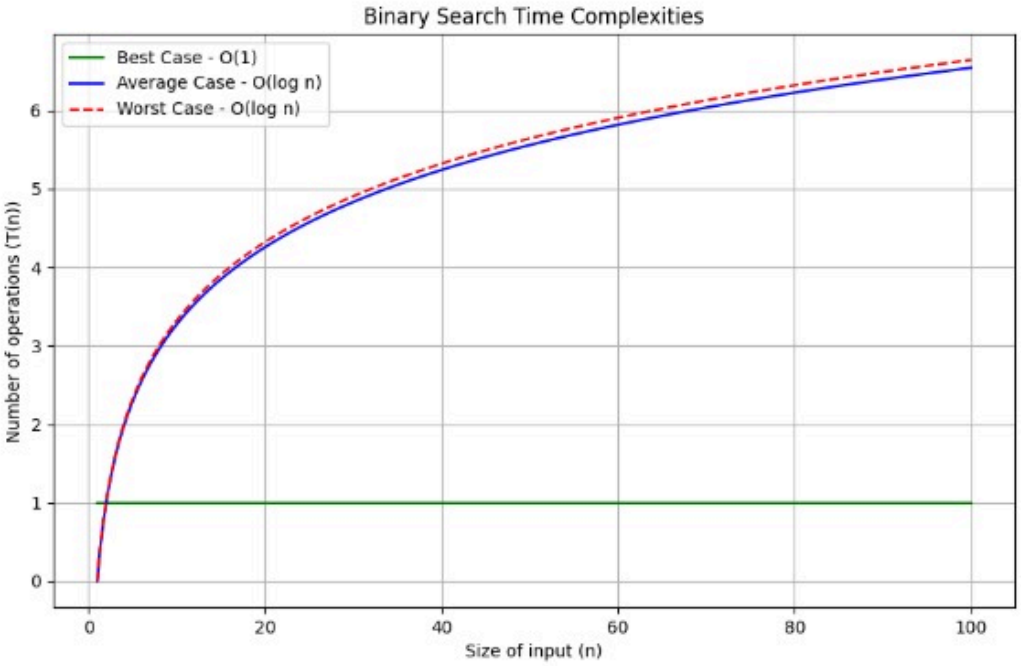
```
import java.util.*;

public class Program2 {
   static int n,item;
   static int ar[];
   static void linearSearch() {
      for (int i=0;i<n;i++) {
         if (item==ar[i]) {
            System.out.println("Element "+ item +" found at location "+ (i+1));
            return;
         }
      }
      System.out.println("Element "+ item +" is not present in the array");
   }
   static void binarySearch() {
      int beg=0,end=n-1;
      int mid=(beg+end)/2;
      while ((beg<=end) && (ar[mid]!=item)) {
         if (item<ar[mid]) {
            end=mid-1;
         }
         else {
            beg=mid+1;
         }
         mid=(beg+end)/2;
      }
      if (item==ar[mid]) {
         System.out.println("Element "+ item +" found at location "+ (mid+1));
      }
      else {
         System.out.println("Element "+ item +" is not present in the array");
      }
   }
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the number of elements in the array: ");
      n=s.nextInt();
      ar=new int[n];
      System.out.print("Enter the elements of the array: ");
      for (int i=0;i<n;i++) {
         ar[i]=s.nextInt();
      }
      System.out.print("Enter the element to search: ");
      item=s.nextInt();
      System.out.println("Linear search(1), Binary search(2)");
      System.out.print("Enter your choice: ");
      int choice=s.nextInt();
```

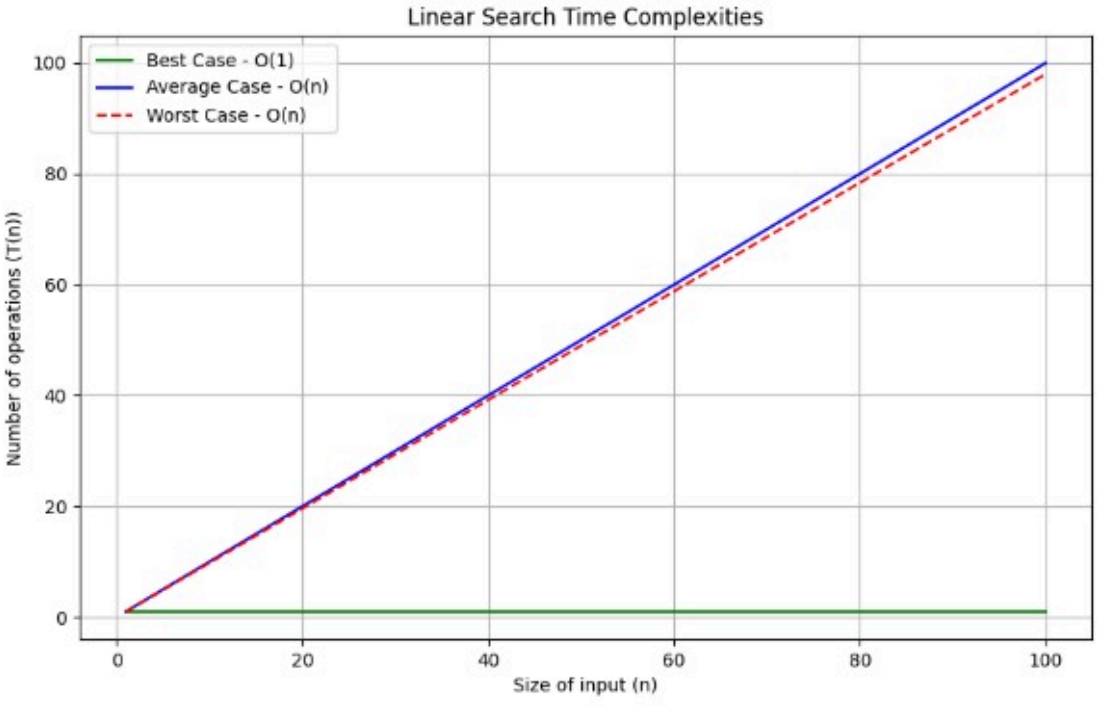
```
        switch (choice) {
            case 1: linearSearch();
                break;
            case 2: binarySearch();
                break;
        }
    }
}
```

Output:

```
Enter the number of elements in the array: 6
Enter the elements of the array: 6 5 3 1 8 7
Enter the element to search: 8
Linear search(1), Binary search(2)
Enter your choice: 1
Element 8 found at location 5
```

```
Enter the number of elements in the array: 6
Enter the elements of the array: 5 10 11 14 19 24
Enter the element to search: 24
Linear search(1), Binary search(2)
Enter your choice: 2
Element 24 found at location 6
```


Binary Search Time Complexities
Best Case - O(1)
Average Case - O(log n)
Worst Case - O(log n)
Number of operations (T(n))
Size of input (n)
0
1
2
3
4
5
6
0
20
40
60
80
100


Linear Search Time Complexities
Best Case - O(1)
Average Case - O(n)
Worst Case - O(n)
Number of operations (T(n))
Size of input (n)
0
20
40
60
80
100
0
20
40
60
80
100

EXPERIMENT-3

Aim: To implement following algorithm using array as data structure and analyse its time complexity.

a) Quick sort
b) Merge sort

Pseudo Code:

Source Code:

```
import java.util.Scanner;

public class Program3 {
   static void swap(int ar[], int i, int j) {
      int temp=ar[i];
      ar[i]=ar[j];
      ar[j]=temp;
   }
   static int partition(int ar[], int low, int high) {
      int pivot=ar[low];
      int j=high;
      for (int k=high;k>low;k--) {
         if (ar[k]>pivot)
            swap(ar,k,j--);
      }
      swap(ar,low,j);
      return j;
   }
   static void merge(int ar[], int low, int mid, int high) {
      int i=low;
      int j=mid+1;
      int k=low;
      int ar2[]=new int[high+1];
      while ((i<=mid) && (j<=high)) {
         if (ar[i]<=ar[j]) {
            ar2[k]=ar[i];
            i++;    k++;
         }
         else {
            ar2[k]=ar[j];
            j++;    k++;
         }
      }
      if (i>mid) {
         while (j<=high) {
            ar2[k]=ar[j];
            j++;    k++;
         }
      }
      else {
         while (i<=mid) {
            ar2[k]=ar[i];
            i++;    k++;
         }
      }
      for (k=low;k<=high;k++) {
         ar[k]=ar2[k];
```

```
        }
    }
    static void quickSort(int ar[], int low, int high) {
        if (low<high) {
            int loc=partition(ar,low,high);
            quickSort(ar,low,loc-1);
            quickSort(ar,loc+1,high);
        }
    }
    static void mergeSort(int ar[], int low, int high) {
        if (low<high) {
            int mid=(low+high)/2;
            mergeSort(ar,low,mid);
            mergeSort(ar,mid+1,high);
            merge(ar,low,mid,high);
        }
    }
    static void printArray(int ar[], int n, String str) {
        System.out.print("The Array "+ str +" sorting is: ");
        for (int i=0;i<n-1;i++) {
            System.out.print(ar[i] +",");
        }
        System.out.println(ar[n-1]);
    }
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n=s.nextInt();
        int ar[]=new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i=0;i<n;i++) {
            ar[i]=s.nextInt();
        }
        System.out.println("Quick sort(1), Merge sort(2)");
        System.out.print("Enter your choice: ");
        int choice=s.nextInt();
        printArray(ar,n,"before");
        switch(choice) {
            case 1: quickSort(ar,0,n-1);
                break;
            case 2: mergeSort(ar,0,n-1);
                break;
        }
        printArray(ar,n,"after");
    }
}
```

Output:

```
Enter the number of elements: 11
Enter the elements of the array: 44 33 11 55 77 90 40 60 99 22 88
Quick sort(1), Merge sort(2)
Enter your choice: 1
The Array before sorting is: 44,33,11,55,77,90,40,60,99,22,88
The Array after sorting is: 11,22,33,40,44,55,60,77,88,90,99
```

```
Enter the number of elements: 8
Enter the elements of the array: 12 31 25 8 32 17 40 42
Quick sort(1), Merge sort(2)
Enter your choice: 2
The Array before sorting is: 12,31,25,8,32,17,40,42
The Array after sorting is: 8,12,17,25,31,32,40,42
```

EXPERIMENT-4

Aim: Analyse the time complexity of Strassen Matrix Multiplication.

Pseudo Code:

Source Code:

```
import java.util.Scanner;

public class Program4 {
   public int[][] multiply(int[][] A, int[][] B) {
      int n = A.length;
      int[][] R = new int[n][n];
      if (n == 1)
         R[0][0] = A[0][0] * B[0][0];
      else {
         int[][] A11 = new int[n/2][n/2];
         int[][] A12 = new int[n/2][n/2];
         int[][] A21 = new int[n/2][n/2];
         int[][] A22 = new int[n/2][n/2];
         int[][] B11 = new int[n/2][n/2];
         int[][] B12 = new int[n/2][n/2];
         int[][] B21 = new int[n/2][n/2];
         int[][] B22 = new int[n/2][n/2];

         split(A, A11, 0 , 0);
         split(A, A12, 0 , n/2);
         split(A, A21, n/2, 0);
         split(A, A22, n/2, n/2);

         split(B, B11, 0 , 0);
         split(B, B12, 0 , n/2);
         split(B, B21, n/2, 0);
         split(B, B22, n/2, n/2);

         int [][] M1 = multiply(add(A11, A22), add(B11, B22));
         int [][] M2 = multiply(add(A21, A22), B11);
         int [][] M3 = multiply(A11, sub(B12, B22));
         int [][] M4 = multiply(A22, sub(B21, B11));
         int [][] M5 = multiply(add(A11, A12), B22);
         int [][] M6 = multiply(sub(A21, A11), add(B11, B12));
         int [][] M7 = multiply(sub(A12, A22), add(B21, B22));
         int [][] C11 = add(sub(add(M1, M4), M5), M7);
         int [][] C12 = add(M3, M5);
         int [][] C21 = add(M2, M4);
         int [][] C22 = add(sub(add(M1, M3), M2), M6);
         join(C11, R, 0 , 0);
         join(C12, R, 0 , n/2);
         join(C21, R, n/2, 0);
         join(C22, R, n/2, n/2);
      }
      return R;
   }
   public int[][] sub(int[][] A, int[][] B) {
```

```
    int n = A.length;
    int[][] C = new int[n][n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}
public int[][] add(int[][] A, int[][] B) {
    int n = A.length;
    int[][] C = new int[n][n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}
public void split(int[][] P, int[][] C, int iB, int jB) {
    for(int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
        for(int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
            C[i1][j1] = P[i2][j2];
}
public void join(int[][] C, int[][] P, int iB, int jB) {
    for(int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
        for(int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
            P[i2][j2] = C[i1][j1];
}
public static void main (String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the order of the matrix: ");
    int n=sc.nextInt();
    int A[][]=new int[n][n];
    int B[][]=new int[n][n];
    System.out.println("Enter the First matrix:");
    for (int i=0;i<n;i++) {
        for (int j=0;j<n;j++) {
            A[i][j]=sc.nextInt();
        }
    }
    System.out.println("Enter the Second matrix:");
    for (int i=0;i<n;i++) {
        for (int j=0;j<n;j++) {
            B[i][j]=sc.nextInt();
        }
    }
    Program4 p = new Program4();
    int[][] C = p.multiply(A, B);
    System.out.println("Product of the two matrices is:");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
```

```
            System.out.print(C[i][j] +" ");
        System.out.println();
    }
  }
}
```

Output:

```
Enter the order of the matrix: 4
Enter the First matrix:
1 2 1 4
5 2 9 6
4 3 2 0
3 5 7 6
Enter the Second matrix:
8 1 14 13
2 9 5 3
1 2 6 7
2 4 9 6
Product of the two matrices is:
21 37 66 50
65 65 188 170
40 35 83 75
53 86 163 139
```

EXPERIMENT-5

Aim: Implement the time complexity of Heap Sort.

Pseudo Code:

Source Code:

```
import java.util.Scanner;

public class Program5 {
   static int n;
   public void sort(int arr[]) {
      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);
      }
   }
   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);
      }
   }
   public static void main(String args[]) {
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the number of elements: ");
      n=s.nextInt();
      int arr[]=new int[n];
      System.out.print("Enter the array: ");
      for (int i=0;i<n;i++) {
         arr[i]=s.nextInt();
      }
      Program5 ob = new Program5();
      ob.sort(arr);
      System.out.print("Sorted array is: ");
      for (int e: arr) {
         System.out.print(e +" ");
      }
      System.out.println();
```

```
    }
}
```

Output:

```
Enter the number of elements: 6
Enter the array: 10 3 76 32 23 34
Sorted array is: 3 10 23 32 34 76
```

EXPERIMENT-6

Aim: To implement Huffman Coding and analyse its time complexity. To implement Minimum Spanning Tree and analyse its time complexity.

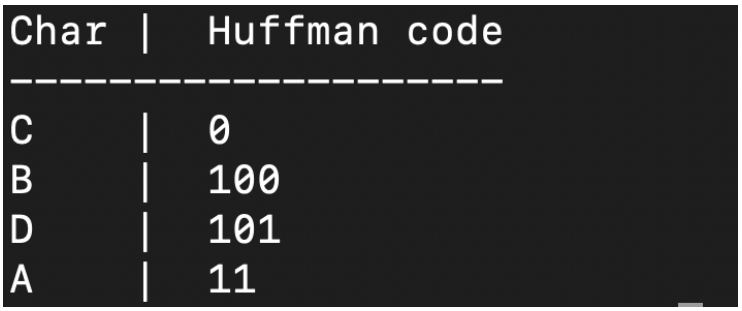
Pseudo Code:

Source Code (Huffman Coding):

```
import java.util.PriorityQueue;
import java.util.Comparator;

class HuffmanNode {
  int item;
  char c;
  HuffmanNode left;
  HuffmanNode right;
}

class ImplementComparator implements Comparator<HuffmanNode> {
  public int compare(HuffmanNode x, HuffmanNode y) {
    return x.item - y.item;
  }
}

public class Program6a {
  public static void printCode(HuffmanNode root, String s) {
    if (root.left == null && root.right == null && Character.isLetter(root.c)) {
      System.out.println(root.c + "    |  " + s);
      return;
    }
    printCode(root.left, s + "0");
    printCode(root.right, s + "1");
  }
  public static void main(String[] args) {
    int n = 4;
    char[] charArray = { 'A', 'B', 'C', 'D' };
    int[] charfreq = { 5, 1, 6, 3 };
    PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>(n, new
ImplementComparator());
    for (int i = 0; i < n; i++) {
      HuffmanNode hn = new HuffmanNode();
      hn.c = charArray[i];
      hn.item = charfreq[i];
      hn.left = null;
      hn.right = null;
      q.add(hn);
    }
    HuffmanNode root = null;
    while (q.size() > 1) {
      HuffmanNode x = q.peek();
      q.poll();
      HuffmanNode y = q.peek();
      q.poll();
      HuffmanNode f = new HuffmanNode();
      f.item = x.item + y.item;
```

```
      f.c = '-';
      f.left = x;
      f.right = y;
      root = f;
      q.add(f);
    }
    System.out.println("Char |  Huffman code ");
    System.out.println("--------------------");
    printCode(root, "");
  }
}
```

Output:



Source Code (Minimum Spanning Tree):

```
import java.util.*;

class Prim {
   Prim() {
      int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
            { 2, 0, 3, 8, 5 },
            { 0, 3, 0, 0, 7 },
            { 6, 8, 0, 0, 9 },
            { 0, 5, 7, 9, 0 } };
      primMST(graph);
   }
   private static final int V = 5;
   int minKey(int key[], Boolean mstSet[]) {
      int min = Integer.MAX_VALUE, min_index = -1;
      for (int v = 0; v < V; v++) {
         if (mstSet[v] == false && key[v] < min) {
            min = key[v];
            min_index = v;
         }
      }
      return min_index;
   }
   void printMST(int parent[], int graph[][]) {
      System.out.println("Edge   | Weight");
      System.out.println("----------------");
      for (int i = 1; i < V; i++)
         System.out.println(parent[i] + " - " + i + "  |  " + graph[i][parent[i]]);
   }
   void primMST(int graph[][]) {
      int parent[] = new int[V];
      int key[] = new int[V];
      Boolean mstSet[] = new Boolean[V];
      for (int i = 0; i < V; i++) {
         key[i] = Integer.MAX_VALUE;
         mstSet[i] = false;
      }
      key[0] = 0;
      parent[0] = -1;
      for (int count = 0; count < V - 1; count++) {
         int u = minKey(key, mstSet);
         mstSet[u] = true;
         for (int v = 0; v < V; v++) {
            if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
               parent[v] = u;
               key[v] = graph[u][v];
            }
         }
```

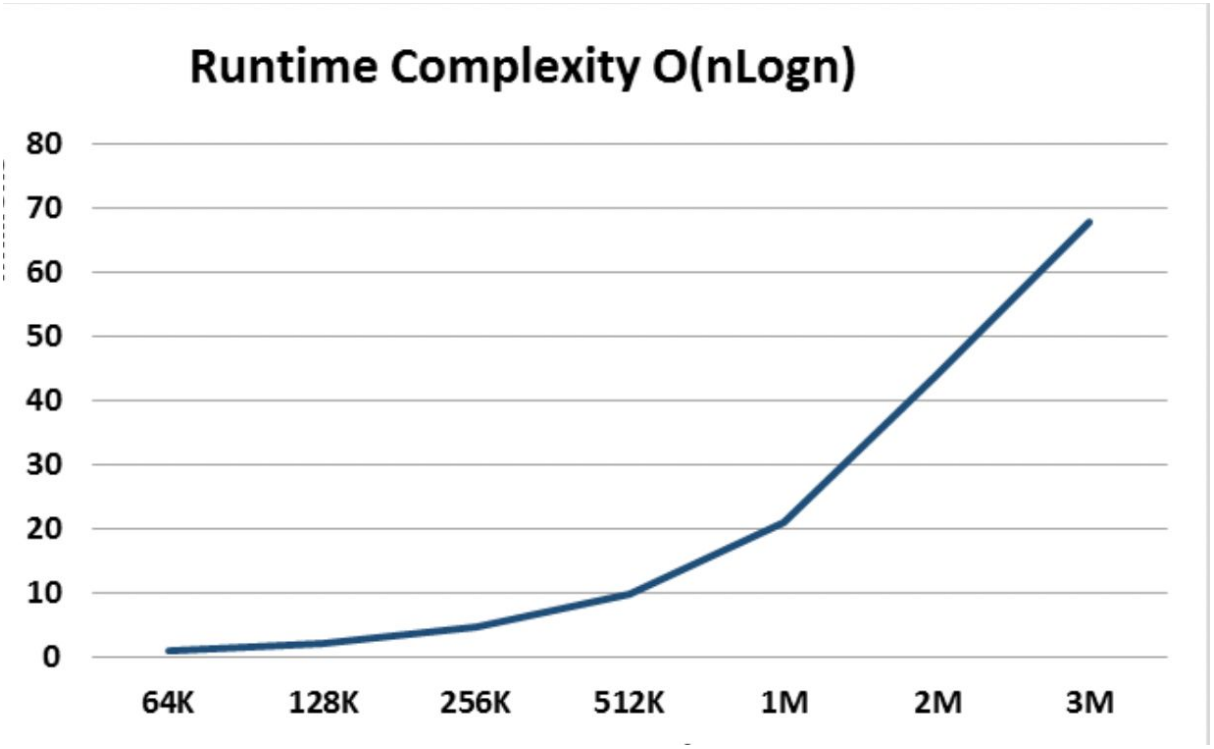
```
        }
        printMST(parent, graph);
    }
}

class Kruskal {
    Kruskal() {
        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);
    }
    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;
        }
    }
    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) {
```

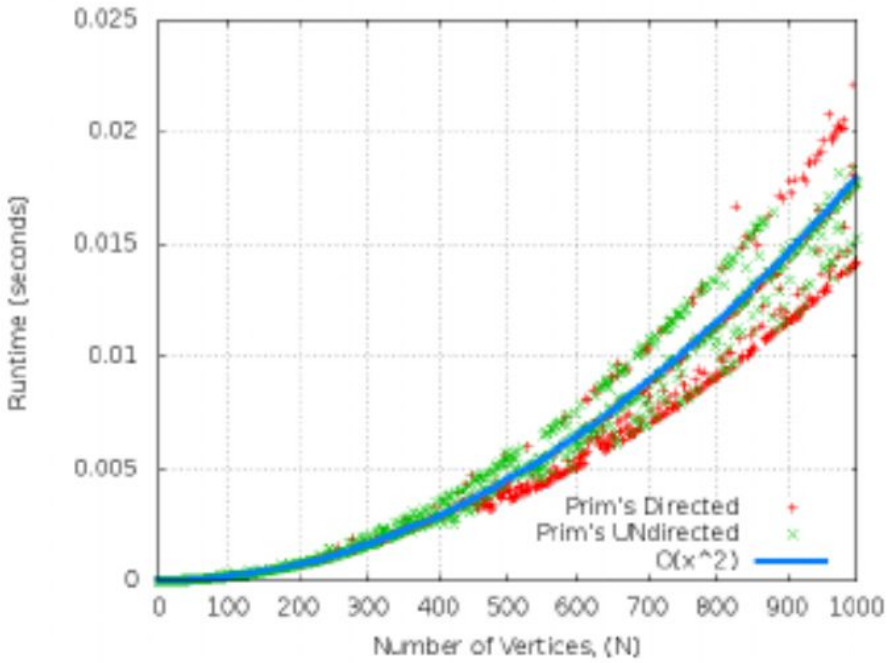
```
                results[noOfEdges] = nextEdge;
                union(subsets, x, y);
                noOfEdges++;
            }
            j++;
        }
        System.out.println("Following are the edges of the constructed MST:");
        System.out.println("--------------------------------------------");
        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;
    }
}
public class Program6b {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        boolean b=true;
        while (b) {
            System.out.println("\nPrim(1) Kruskal(2) Exit(0)");
            System.out.print("Enter your choice: ");
            int choice=s.nextInt();
            if (choice==1) {
                Prim t = new Prim();
            }
            else if (choice==2) {
                Kruskal k = new Kruskal();
            }
```

```
        else if (choice==0) {
            System.out.println("See you later!");
            b=false;
        }
        else {
            System.out.println("Invalid Input");
            b=false;
        }
    }
  }
}
```

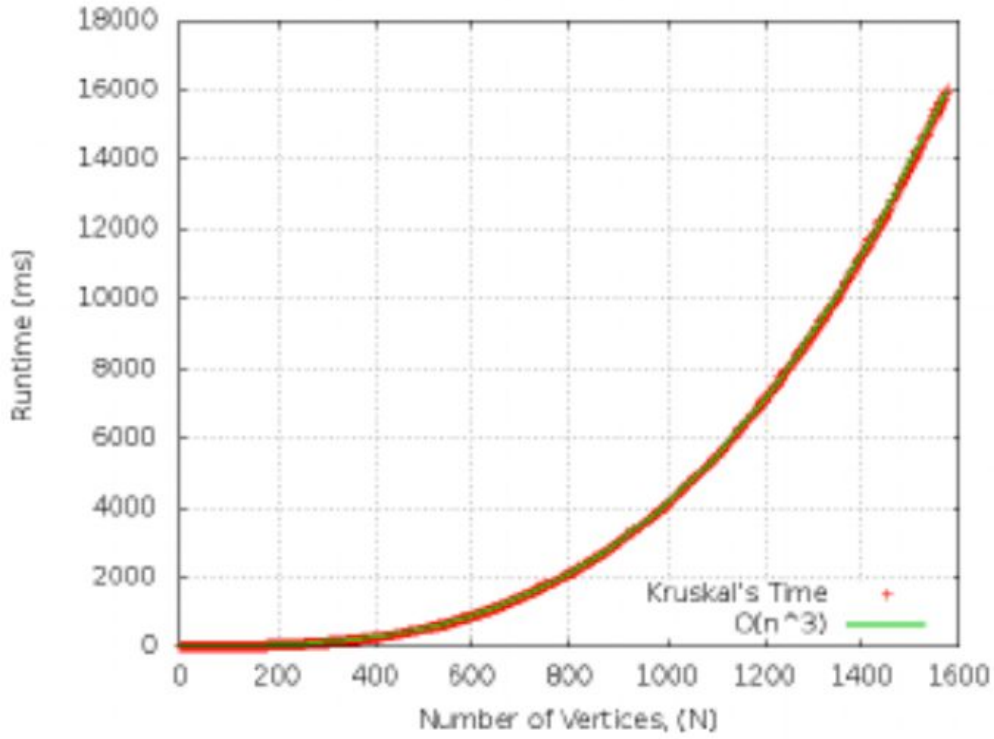
Output:

```
Prim(1) Kruskal(2) Exit(0)
Enter your choice: 1
Edge    |  Weight
---------------
0 - 1   |  2
1 - 2   |  3
0 - 3   |  6
1 - 4   |  5

Prim(1) Kruskal(2) Exit(0)
Enter your choice: 2
Following are the edges of the constructed MST:
-----------------------------------------------
2 - 3 --> 4
0 - 3 --> 5
0 - 1 --> 10
Total cost of MST: 19

Prim(1) Kruskal(2) Exit(0)
Enter your choice: 0
See you later!
```





Complexity of Prim's algorithm.



Complexity of Kruskal’s algorithm.

EXPERIMENT-7

Aim: To implement Dijkstra's algorithm and analyse its time complexity and to implement Bellman Ford algorithm and analyse its time complexity.

Pseudo Code:

Source Code:

```
import java.util.Scanner;

class Dijkstra {
   Dijkstra() {
      int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
      dijkstra(graph,0);
   }
   static final int V = 9;
   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[]) {
      System.out.println("Vertex | Distance from Source");
      System.out.println("----------------------------");
      for (int i = 0; i < V; i++)
         System.out.println(i + "      |     " + 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];
```

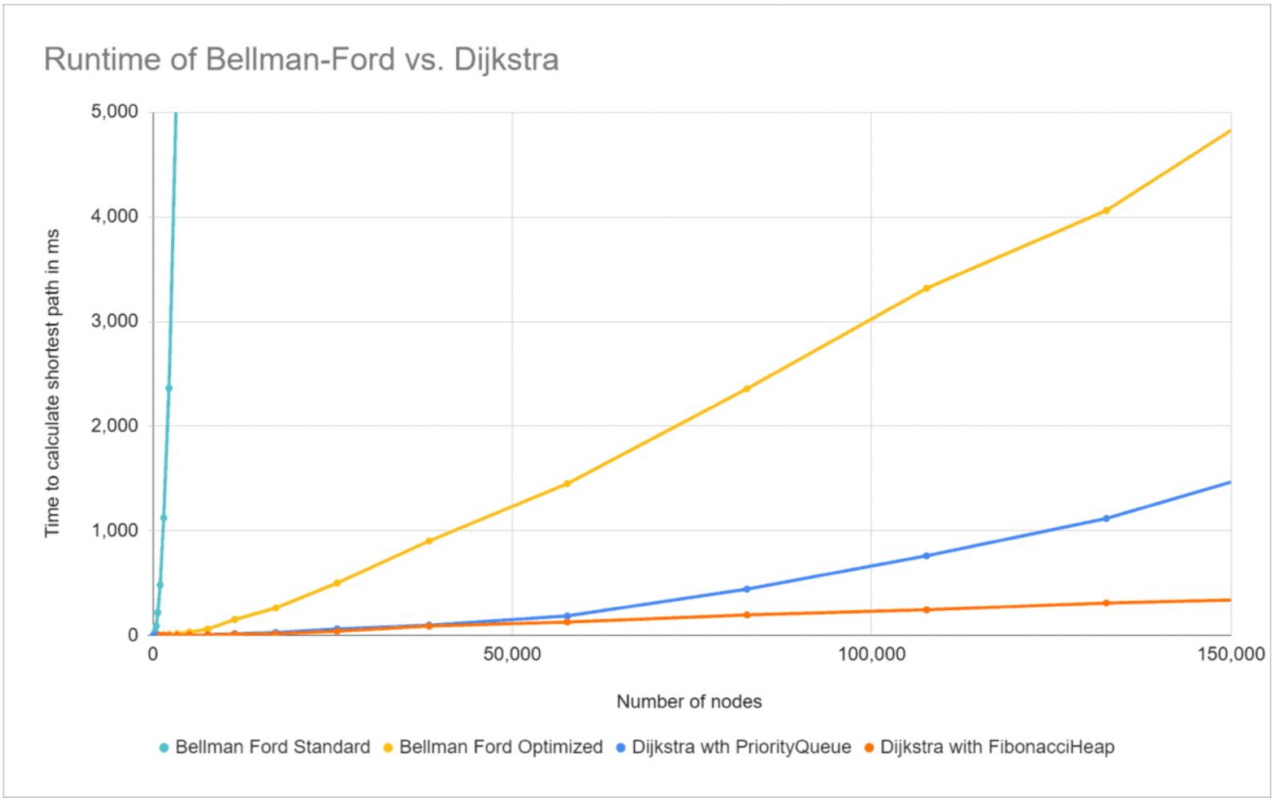
```
        }
        printSolution(dist);
    }
}
class Bellman {
    Bellman() {
        int V = 5;
        int E = 8;
        Bellman graph = new Bellman(V, E);
        graph.edge[0].src = 0;
        graph.edge[0].dest = 1;
        graph.edge[0].weight = -1;
        graph.edge[1].src = 0;
        graph.edge[1].dest = 2;
        graph.edge[1].weight = 4;
        graph.edge[2].src = 1;
        graph.edge[2].dest = 2;
        graph.edge[2].weight = 3;
        graph.edge[3].src = 1;
        graph.edge[3].dest = 3;
        graph.edge[3].weight = 2;
        graph.edge[4].src = 1;
        graph.edge[4].dest = 4;
        graph.edge[4].weight = 2;
        graph.edge[5].src = 3;
        graph.edge[5].dest = 2;
        graph.edge[5].weight = 5;
        graph.edge[6].src = 3;
        graph.edge[6].dest = 1;
        graph.edge[6].weight = 1;
        graph.edge[7].src = 4;
        graph.edge[7].dest = 3;
        graph.edge[7].weight = -3;
        graph.BellmanFord(graph, 0);
    }
    class Edge {
        int src, dest, weight;
        Edge() { src = dest = weight = 0; }
    };
    int V, E;
    Edge edge[];
    Bellman(int v, int e) {
        V = v;
        E = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge();
    }
```

```
    void BellmanFord(Bellman graph, int src) {
        int V = graph.V, E = graph.E;
        int dist[] = new int[V];
        for (int i = 0; i < V; ++i)
            dist[i] = Integer.MAX_VALUE;
        dist[src] = 0;
        for (int i = 1; i < V; ++i) {
            for (int j = 0; j < E; ++j) {
                int u = graph.edge[j].src;
                int v = graph.edge[j].dest;
                int weight = graph.edge[j].weight;
                if (dist[u] != Integer.MAX_VALUE
                    && dist[u] + weight < dist[v])
                    dist[v] = dist[u] + weight;
            }
        }
        for (int j = 0; j < E; ++j) {
            int u = graph.edge[j].src;
            int v = graph.edge[j].dest;
            int weight = graph.edge[j].weight;
            if (dist[u] != Integer.MAX_VALUE
                && dist[u] + weight < dist[v]) {
                System.out.println("Graph contains negative weight cycle");
                return;
            }
        }
        printArr(dist, V);
    }
    void printArr(int dist[], int V) {
        System.out.println("Vertex | Distance from Source");
        System.out.println("----------------------------");
        for (int i = 0; i < V; ++i)
            System.out.println(i + "      |      " + dist[i]);
    }
}

public class Program7 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        boolean c=true;
        while (c) {
            System.out.println("\nDijkstra(1) Bellman(2) Exit(0)");
            System.out.print("Enter your choice: ");
            int choice=s.nextInt();
            if (choice==1) {
                Dijkstra d = new Dijkstra();
            }
            else if (choice==2) {
```

```
            Bellman b = new Bellman();
        }
        else if (choice==0) {
            System.out.println("See you later!");
            c=false;
        }
        else {
            System.out.println("Invalid Input");
            c=false;
        }
    }
  }
}
```

Output:

```
Dijkstra(1) Bellman(2) Exit(0)
Enter your choice: 1
Vertex | Distance from Source
----------------------------
0      |     0
1      |     4
2      |     12
3      |     19
4      |     21
5      |     11
6      |     9
7      |     8
8      |     14

Dijkstra(1) Bellman(2) Exit(0)
Enter your choice: 2
Vertex | Distance from Source
----------------------------
0      |      0
1      |      -1
2      |      2
3      |      -2
4      |      1

Dijkstra(1) Bellman(2) Exit(0)
Enter your choice: 0
See you later!
```



Time complexity Bellman-Ford algorithm vs. Dijkstra algorithm

EXPERIMENT-8

Aim: Implement N Queen's problem using Back Tracking.

Pseudo Code:

Source Code:

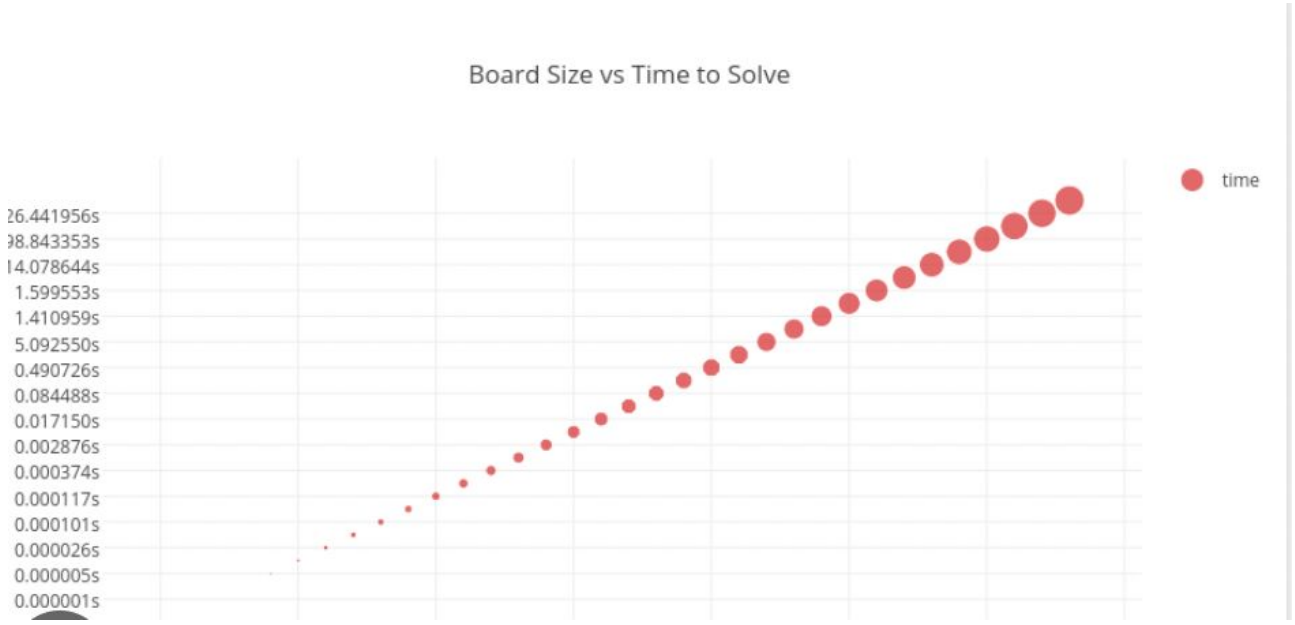
```
import java.util.Scanner;

public class Program8 {
   static int N;
   void printSolution(int board[][]) {
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++) {
            if (board[i][j] == 1)
               System.out.print("Q ");
            else
               System.out.print(". ");
         }
         System.out.println();
      }
   }
   boolean isSafe(int board[][], int row, int col) {
      int i, j;
      for (i = 0; i < col; i++)
         if (board[row][i] == 1)
            return false;
      for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
         if (board[i][j] == 1)
            return false;
      for (i = row, j = col; j >= 0 && i < N; i++, j--)
         if (board[i][j] == 1)
            return false;
      return true;
   }
   boolean solveNQUtil(int board[][], int col) {
      if (col >= N)
         return true;
      for (int i = 0; i < N; i++) {
         if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1) == true)
               return true;
            board[i][col] = 0;
         }
      }
      return false;
   }
   boolean solveNQ() {
      int board[][] = new int[N][N];
      for (int i=0;i<N;i++) {
         for (int j=0;j<N;j++) {
            board[i][j]=0;
         }
```

```
        }
        if (solveNQUtil(board, 0) == false) {
            System.out.print("Solution does not exist");
            return false;
        }
        printSolution(board);
        return true;
    }
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        System.out.print("Enter the number of queen's: ");
        N=s.nextInt();
        Program8 Queen = new Program8();
        Queen.solveNQ();
    }
}
```

Output:

```
Enter the number of queen's: 4
. . Q .
Q . . .
. . . Q
. Q . .
```

```
Enter the number of queen's: 8
Q . . . . . . .
. . . . . . Q .
. . . . Q . . .
. . . . . . . Q
. Q . . . . . .
. . . Q . . . .
. . . . . Q . .
. . Q . . . . .
```


Board Size vs Time to Solve
time
26.441956s
98.843353s
14.078644s
1.599553s
1.410959s
5.092550s
0.490726s
0.084488s
0.017150s
0.002876s
0.000374s
0.000117s
0.000101s
0.000026s
0.000005s
0.000001s

EXPERIMENT-9

Aim: To Implement Matrix Chain Multiplication and analyse its time complexity. To implement Longest Common Subsequence problem and analyse its time complexity.

Pseudo Code:

Source Code (Matrix Chain Multiplication):

```
import java.util.Scanner;

public class Program9a {
   static int MatrixChainOrder(int p[], int i, int j) {
      if (i == j)
         return 0;
      int min = Integer.MAX_VALUE;
      for (int k = i; k < j; k++) {
         int count = MatrixChainOrder(p, i, k)+ MatrixChainOrder(p, k + 1, j)+ p[i - 1] * p[k] * p[j];
         if (count < min)
            min = count;
      }
      return min;
   }
   public static void main(String args[]) {
      Scanner s = new Scanner(System.in);
      System.out.print("Enter the number of matrices: ");
      int N=s.nextInt();
      int arr[] = new int[N+1];
      System.out.print("Enter the order of matrices: ");
      for (int i=0;i<=N;i++) {
         arr[i]=s.nextInt();
      }
      System.out.print("The Order of the matrices are ");
      for (int i=0;i<N-1;i++) {
         System.out.print(arr[i] +"x"+ arr[i+1] +", ");
      }
      System.out.println(arr[N-1] +"x"+ arr[N]);
      System.out.println("Minimum number of multiplications are: "+ MatrixChainOrder(arr, 1, N));
   }
}
```

Output:

```
Enter the number of matrices: 4
Enter the order of matrices: 2 4 6 8 6
The Order of the matrices are 2x4, 4x6, 6x8, 8x6
Minimum number of multiplications are: 240
```

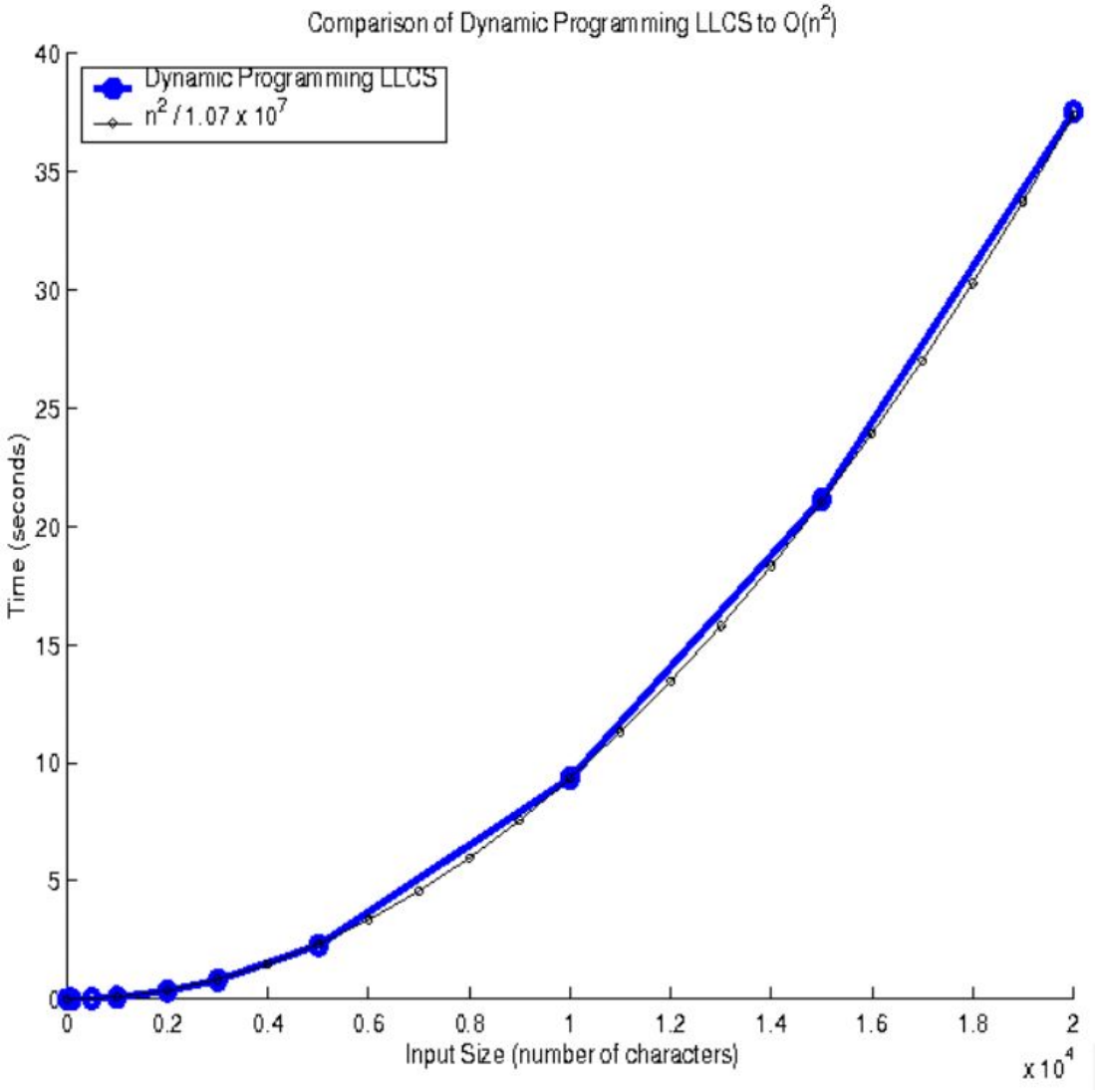
Source Code (Longest Common Subsequence):

```
import java.util.Scanner;

public class Program9b {
   static void lcs(String S1, String S2, int m, int n) {
      int[][] LCS_table = new int[m + 1][n + 1];
      for (int i = 0; i <= m; i++) {
         for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0)
               LCS_table[i][j] = 0;
            else if (S1.charAt(i - 1) == S2.charAt(j - 1))
               LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
            else
               LCS_table[i][j] = Math.max(LCS_table[i - 1][j], LCS_table[i][j - 1]);
         }
      }
      int index = LCS_table[m][n];
      int temp = index;
      char[] lcs = new char[index + 1];
      lcs[index] = '\0';
      int i = m, j = n;
      while (i > 0 && j > 0) {
         if (S1.charAt(i - 1) == S2.charAt(j - 1)) {
            lcs[index - 1] = S1.charAt(i - 1);
            i--;
            j--;
            index--;
         }
         else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
            i--;
         else
            j--;
      }
      System.out.print("First String: " + S1 + "\nSecond String: " + S2 + "\nLongest Common
Subsequence: ");
      for (int k = 0; k <= temp; k++)
         System.out.print(lcs[k]);
      System.out.println();
   }
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      System.out.print("Enter two Strings: ");
      String S1=s.next();
      String S2=s.next();
      int m = S1.length();
      int n = S2.length();
      lcs(S1, S2, m, n);
   }
```

}

Output:

```
Enter two Strings: AGGTAB GXTXAYB
First String: AGGTAB
Second String: GXTXAYB
Longest Common Subsequence: GTAB
```


Comparison of Dynamic Programming LLCS to O(n^2)
Dynamic Programming LLCS
n^2 / 1.07 x 10^7
Time (seconds)
Input Size (number of characters)
x 10^4
0
5
10
15
20
25
30
35
40
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

EXPERIMENT-10

Aim: To implement naive String Matching algorithm, Rabin Karp algorithm and Knuth Morris Pratt algorithm and analyse its time complexity.

Pseudo Code:

Source Code:

```
import java.util.Scanner;

class NSMA {
   static void search(String txt, String pat) {
      int l1 = txt.length();
      int l2 = pat.length();
      int i = 0, j = l2 - 1;
      for (i = 0, j = l2 - 1; j < l1;) {
         if (pat.equals(txt.substring(i, j + 1))) {
            System.out.println("Pattern found at index "+ i);
         }
         i++;
         j++;
      }
   }
}
class RKA {
   public final static int d = 256;
   static void search(String pat, String txt, int q) {
      int M = pat.length();
      int N = txt.length();
      int i, j;
      int p = 0;
      int t = 0;
      int h = 1;
      for (i = 0; i < M - 1; i++)
         h = (h * d) % q;
      for (i = 0; i < M; i++) {
         p = (d * p + pat.charAt(i)) % q;
         t = (d * t + txt.charAt(i)) % q;
      }
      for (i = 0; i <= N - M; i++) {
         if (p == t) {
            for (j = 0; j < M; j++) {
               if (txt.charAt(i + j) != pat.charAt(j))
                  break;
            }
            if (j == M)
               System.out.println("Pattern found at index "+ i);
         }
         if (i < N - M) {
            t = (d * (t - txt.charAt(i) * h)+ txt.charAt(i + M))% q;
            if (t < 0)
               t = (t + q);
         }
      }
   }
```

```
}
class KMPA {
   static void KMPSearch(String pat, String txt) {
      int M = pat.length();
      int N = txt.length();
      int lps[] = new int[M];
      int j = 0;
      computeLPSArray(pat, M, lps);
      int i = 0;
      while ((N - i) >= (M - j)) {
         if (pat.charAt(j) == txt.charAt(i)) {
            j++;
            i++;
         }
         if (j == M) {
            System.out.println("Pattern found at index "+ (i - j));
            j = lps[j - 1];
         }
         else if (i<N && pat.charAt(j) != txt.charAt(i)) {
            if (j != 0)
               j = lps[j - 1];
            else
               i = i + 1;
         }
      }
   }
   static void computeLPSArray(String pat, int M, int lps[]) {
      int len = 0;
      int i = 1;
      lps[0] = 0;
      while (i < M) {
         if (pat.charAt(i) == pat.charAt(len)) {
            len++;
            lps[i] = len;
            i++;
         }
         else {
            if (len != 0) {
               len = lps[len - 1];
            }
            else {
               lps[i] = len;
               i++;
            }
         }
      }
   }
}
```

```
public class Program10 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        boolean b=true;
        while (b) {
            System.out.print("\nEnter the text: ");
            String txt=s.next();
            System.out.print("Enter the pattern to match: ");
            String pat=s.next();
            System.out.println("Naive String Matching(1), Rabin Karp(2), Knuth Morris Pratt(3),
Exit(0)");
            System.out.print("Enter your choice: ");
            int choice=s.nextInt();
            if (choice==1) {
                NSMA.search(txt,pat);
            }
            else if (choice==2) {
                RKA.search(pat,txt,101);
            }
            else if (choice==3) {
                KMPA.KMPSearch(pat,txt);
            }
            else if (choice==0) {
                System.out.println("Have a good day!");
                b=false;
            }
            else {
                System.out.println("Invalid Input");
                b=false;
            }
        }
    }
}
```

Output:

```
Enter the text: AABAACAADAABAAABAA
Enter the pattern to match: AABA

Naive String Matching(1), Rabin Karp(2), Knuth Morris Pratt(3), Exit(0)
Enter your choice: 1
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13
Enter the text: ^Z
zsh: suspended  java Program10
kshitizsharma@MacBookAir % javac Program10.java
kshitizsharma@MacBookAir % java Program10

Enter the text: AABAACAADAABAAABAA
Enter the pattern to match: AABA
Naive String Matching(1), Rabin Karp(2), Knuth Morris Pratt(3), Exit(0)
Enter your choice: 1
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13

Enter the text: ABCCDDAEFG
Enter the pattern to match: CDD
Naive String Matching(1), Rabin Karp(2), Knuth Morris Pratt(3), Exit(0)
Enter your choice: 2
Pattern found at index 3

Enter the text: ABABDABACDABABCABAB
Enter the pattern to match: ABABCABAB
Naive String Matching(1), Rabin Karp(2), Knuth Morris Pratt(3), Exit(0)
Enter your choice: 3
Pattern found at index 10
```