<u>Lab – 1</u>

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

<u>Template :-</u>

```
import java.util.Random;
import java.util.Scanner;

class SortTimeComplexity{
    static void print(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    static void Sort(int arr[]) {
        // Change This function with the proper functions given below
    }

    static class BestCase {
        int[] arr;
        BestCase(int n) {
            arr = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = i;
            }
        }
    }

    static class WorstCase {
        int[] arr;
        WorstCase(int n) {
            arr = new int[n];
            for (int i = n - 1; i >= 0; i--) {
                arr[i] = n - i;
            }
        }
    }
```

```
    static class AverageCase {
        private int[] arr;
        AverageCase(int n) {
            arr = new int[n];
            Random rand = new Random();
            for (int i = 0; i < n; i++) {
                arr[i] = rand.nextInt(n);
            }
        }
    }

    public static void main(String[] args) {
        int size;
        System.out.print("Enter length of array :- ");
        Scanner scanner = new Scanner(System.in);
        size = scanner.nextInt();
        scanner.close();
        AverageCase avgArray = new AverageCase(size);
        BestCase bestArray = new BestCase(size);
        WorstCase worstArray = new WorstCase(size);
        Sort(bestArray.arr);
        Sort(avgArray.arr);
        Sort(worstArray.arr);
    }
}
```

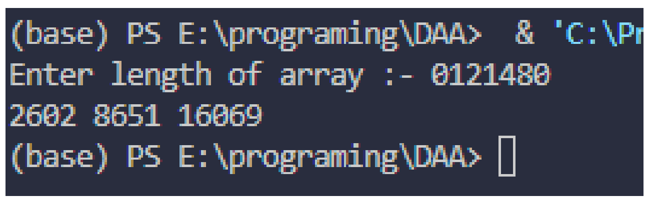
1. **<u>Selection Sort</u>**

   Code :-

```
    static void selectionSort(int arr[]) {
        long begin = System.currentTimeMillis();
        for (int i = 0; i < arr.length; i++) {
            int minIndex = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
```

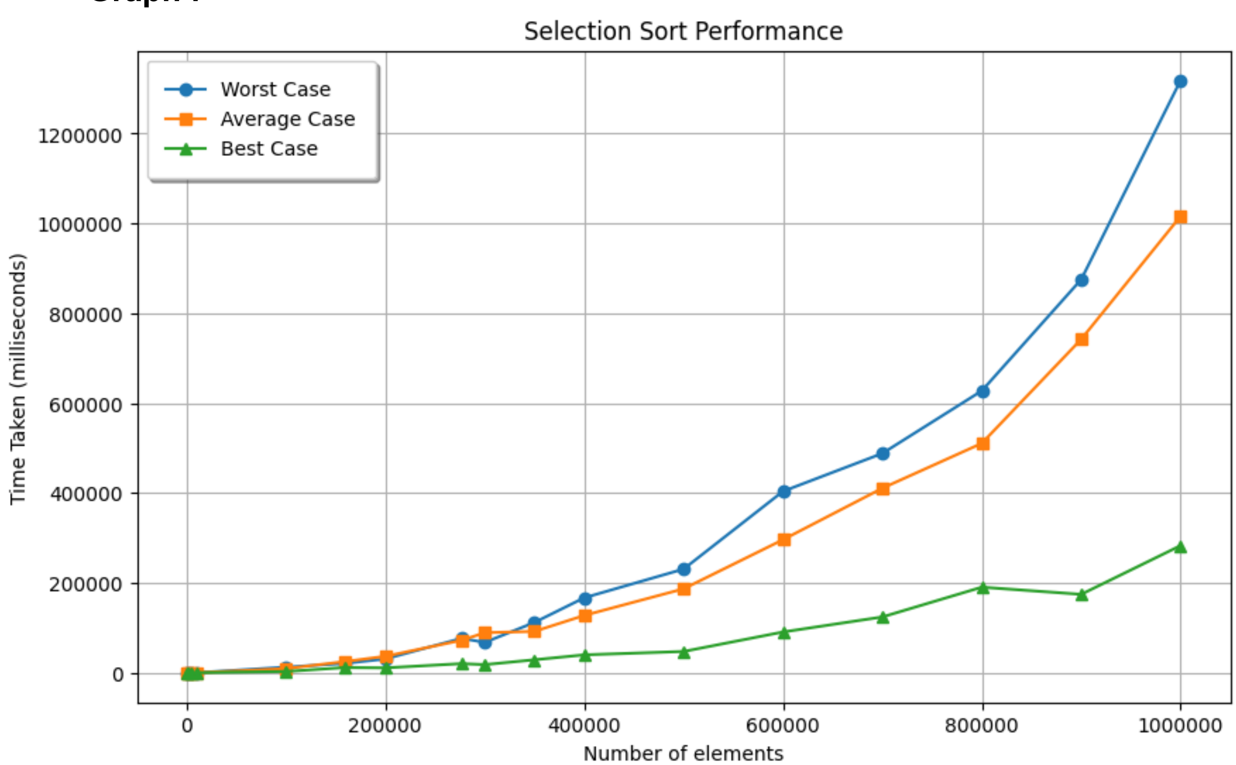
```
            if (minIndex != i) {
                int temp = arr[i];
                arr[i] = arr[minIndex];
                arr[minIndex] = temp;
            }
        }

        long end = System.currentTimeMillis();
        long time = end - begin;
        System.out.print(time + " ");
    }
```

Output :-



```
(base) PS E:\programing\DAA>  & 'C:\Pr
Enter length of array :- 0121480
2602 8651 16069
(base) PS E:\programing\DAA> 
```

Graph :-



2. **Bubble Sort**

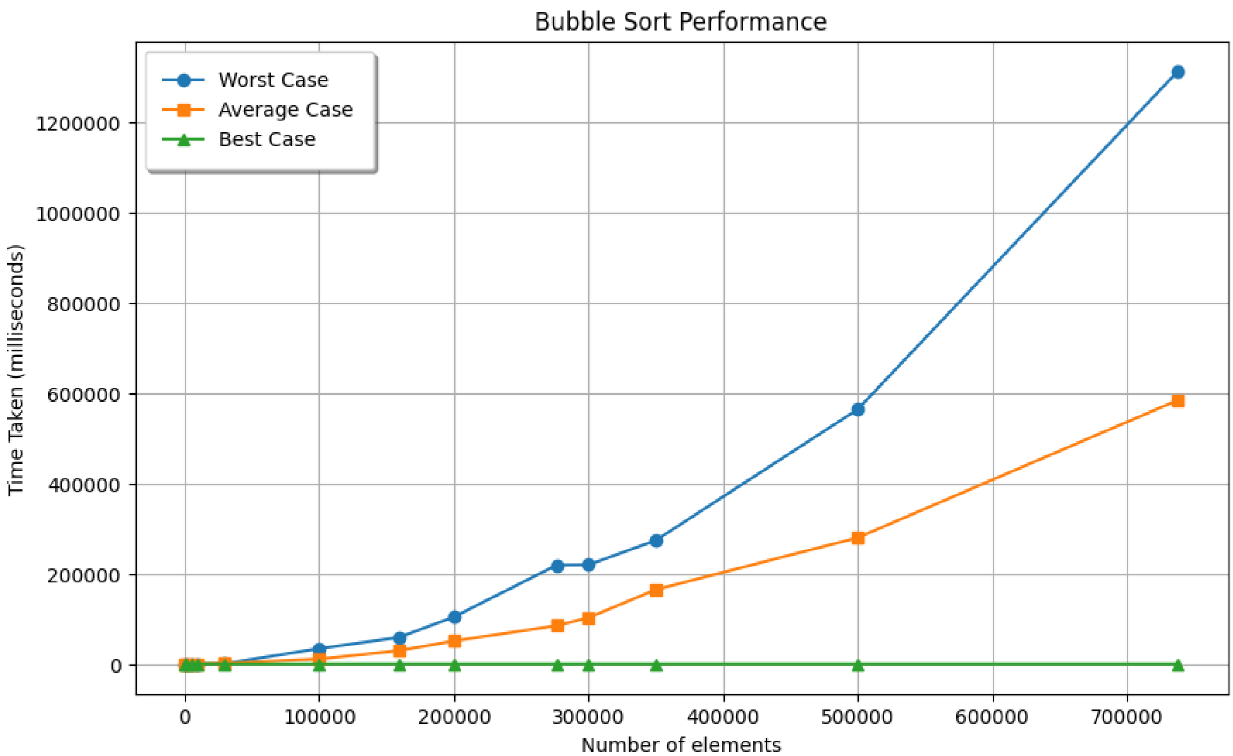
Code :-

```
static void bubbleSort(int arr[]) {
    long begin = System.currentTimeMillis();
    int n = arr.length;
    boolean swapped;
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }

    long end = System.currentTimeMillis();
    long time = end - begin;
    System.out.print(time + " ");
}
```

Output :-

```
eComplexity'
Enter length of array :- 0121480
12 43930 15186
(base) PS E:\programing\DAA>
```

Graph :-



3. **<u>Insertion Sort</u>**

Code :-

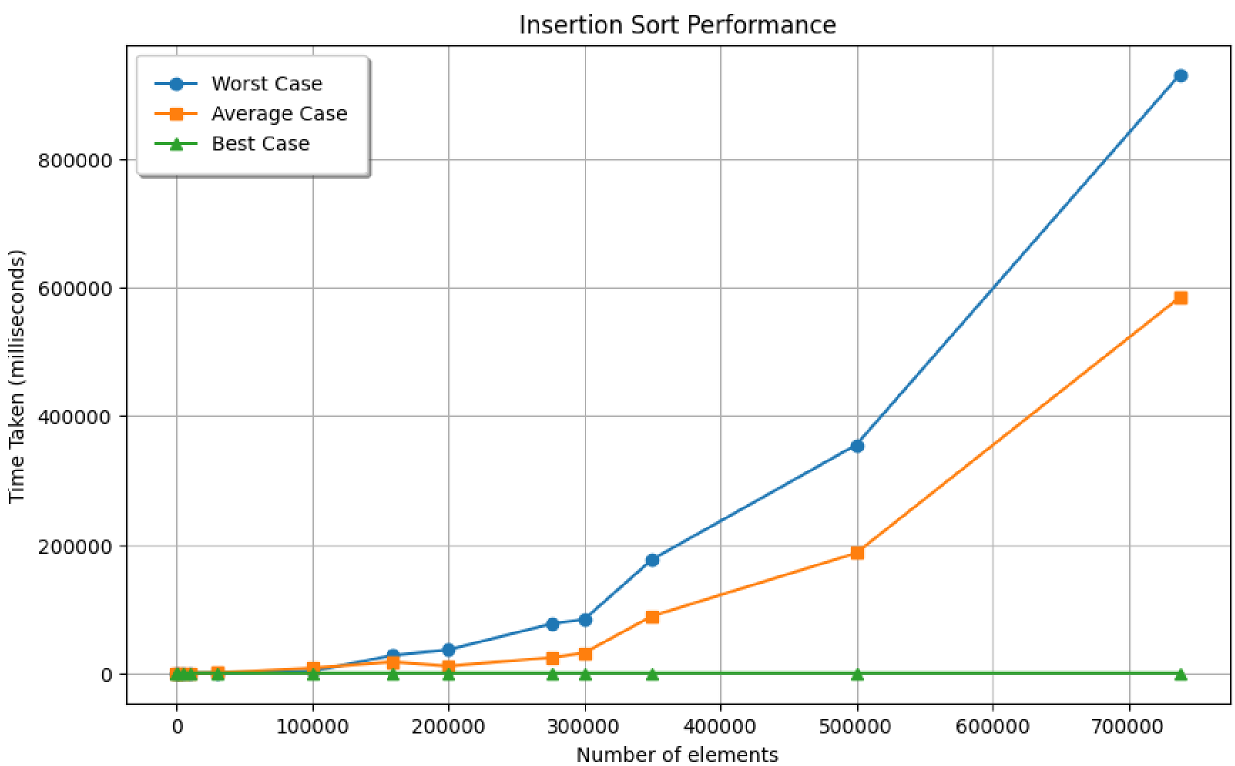
```
static void insertionSort( int arr[]){
    long start = System.currentTimeMillis();
    for(int i=1; i<arr.length; i++){
        int j = i -1 ;
        int temp = arr[i];
        while(j>=0 && arr[j] > temp){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
    long end = System.currentTimeMillis();
    long time = end - start;
```

```
        System.out.print(time + " ");

    }
```

Output :-

```
TimeComplexity'
Enter length of array :- 0121480
14 19450 6253
(base) PS E:\programing\DAA>
```

Graph :-



Lab - 2

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

Template :-

```
import java.util.Random;
import java.util.Scanner;

public class SearchTimeComplexity {

    static void printArr(int arr[]){
        for(int i = 0 ; i<arr.length ; i++){
            System.out.print(arr[i] + " ");
        }
        System.out.println("");
    }

    static void Search(int arr[], int key){
        // Change This function with the proper functions given below
    }

  static class BestCase {
        int[] array;
        BestCase(int[] arr, int key) {
            array = arr;
            array[0] = key;
        }
    }

    static class WorstCase {
        int[] array;
        WorstCase(int[] arr, int key) {
            array = arr;
            array[arr.length / 2] = key;
        }
    }

    static class AverageCase {
```

```
    int[] array;
    AverageCase(int[] arr, int key) {
        Random random = new Random();
        int randomIndex = random.nextInt(arr.length);
        array = arr;
        array[randomIndex] = key;
    }
}

static class ArrayGenerator {
    int[] arr;
    ArrayGenerator(int n) {
        arr = new int[n];
        for (int i = 1; i < n; i++) {
            arr[i] = i;
        }
    }
}


static class ArrayGenerator {
    int[] arr;
    ArrayGenerator(int n) {
        arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = i;
        }
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the size of array :-  ");
    int size = scanner.nextInt();

    scanner.close();

    ArrayGenerator array = new ArrayGenerator(size);
    Random random = new Random();
```

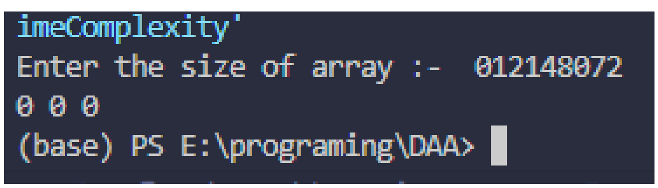
```
        int randNo = random.nextInt(size);

        Search(array.arr, (size/2)+1);
        Search(array.arr, randNo);
        Search(array.arr, size - 1);
    }
}
```

1. **Linear Search**

   Code :-

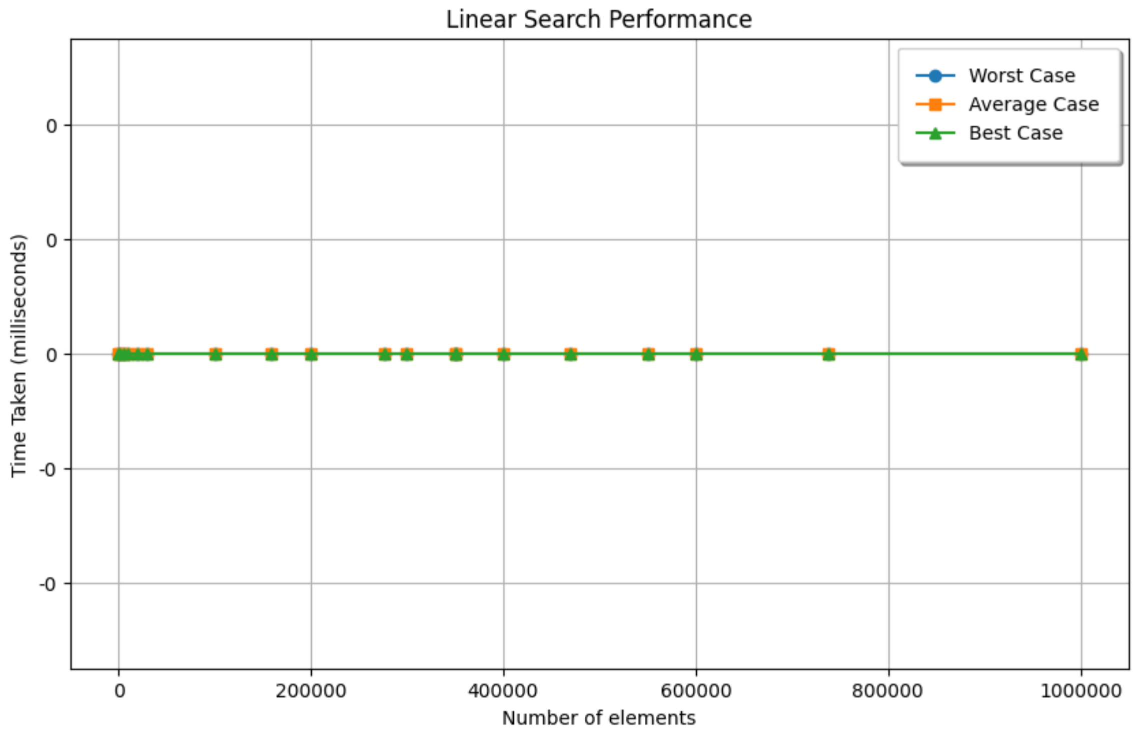
```
static void linearSearch(int arr[],int key){
    long start = System.currentTimeMillis();
    for(int i=0 ; i<arr.length; i++){
        if(arr[i] == key){
            long end = System.currentTimeMillis();
            long time = end - start;
            System.out.print(time + " ");
            arr[i] = 0;
            return ;
        }
    }
}
```

   Output :-



```
imeComplexity'
Enter the size of array :-  012148072
0 0 0
(base) PS E:\programing\DAA>
```

   Graph :-



2. **<u>BinarySearch</u>**

Code :-

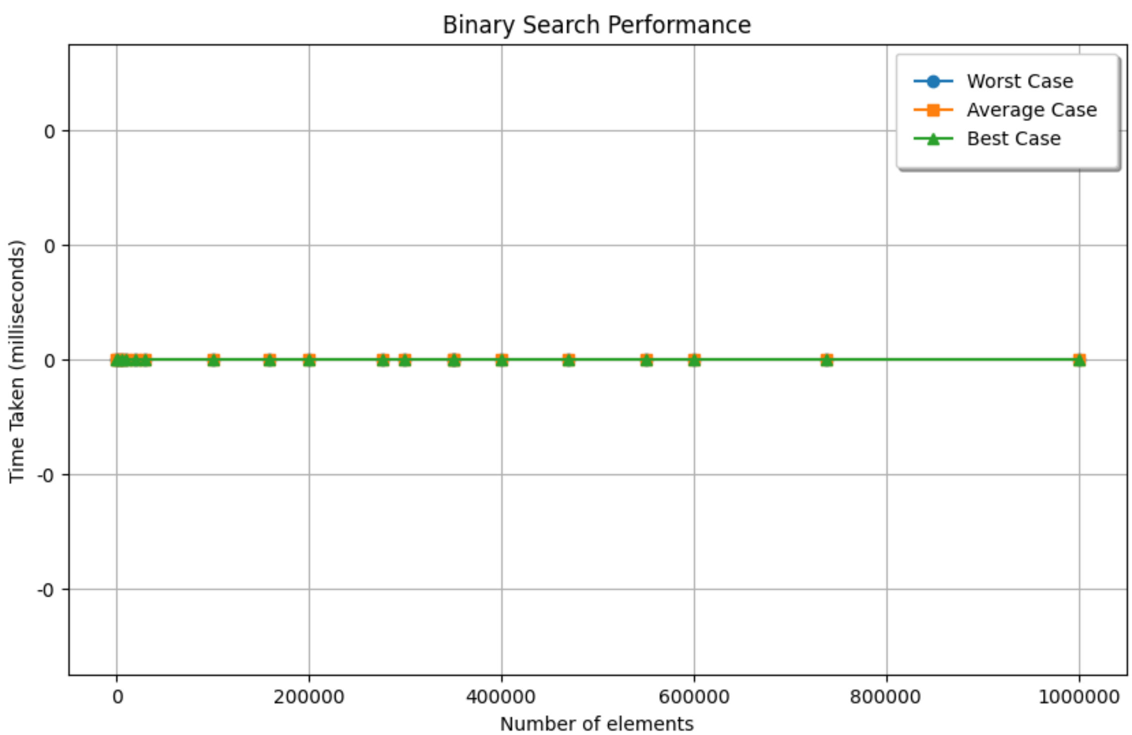
```
static void binarySearch(int arr[], int key){
    int s = 0;
    int e = arr.length - 1;
    long start = System.currentTimeMillis();
    while(s <= e){
        int mid = s + (e-s)/2;

        if(arr[mid] == key){
            long end = System.currentTimeMillis();
            long time = end - start;
            System.out.print(time + " ");
            return ;
        }
        else if(arr[mid] > key){
            e = mid - 1 ;
        }
        else{
            s = mid +1 ;
```

```
        }
    }
}
```

Output :-

```
imeComplexity'
Enter the size of array :-  01214807
0 0 0
(base) PS E:\programing\DAA>
```

Graph :-



Lab - 3

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

Template:-

```
import java.util.Random;
import java.util.Scanner;

public class SortTimeComplexity {

    static void print(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }

    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int 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) {
            // Change This function with the proper functions given below
    }
```

```
static void SortTime(int[] arr){
    long start = System.currentTimeMillis();
    Sort(arr, 0, arr.length - 1);
    long end = System.currentTimeMillis();
    long time = end - start;
    System.out.print(time + " ");
 }


 static class BestCase {
    int[] arr;
    BestCase(int n) {
        arr = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            arr[i] = rand.nextInt(n);
        }
    }
 }

 static class WorstCase {
    int[] arr;
    WorstCase(int n) {
        arr = new int[n];
        for (int i = n - 1; i >= 0; i--) {
            arr[i] = n - i;
        }
    }
 }

 static class AverageCase {
    int[] arr;
    AverageCase(int n) {
        arr = new int[n];
        Random rand = new Random();
        for (int i = 0; i < n; i++) {
            arr[i] = rand.nextInt(n);
        }
    }
 }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of array ");
        int n = sc.nextInt();
        sc.close();


        AverageCase avgCase = new AverageCase(n);
        BestCase bestCase = new BestCase(n);
        WorstCase worstCase = new WorstCase(n);

        SortTime(bestCase.arr);
        SortTime(avgCase.arr);
        SortTime(worstCase.arr);
    }
}
```

1. **<u>Quick Sort</u>**
   Code :-

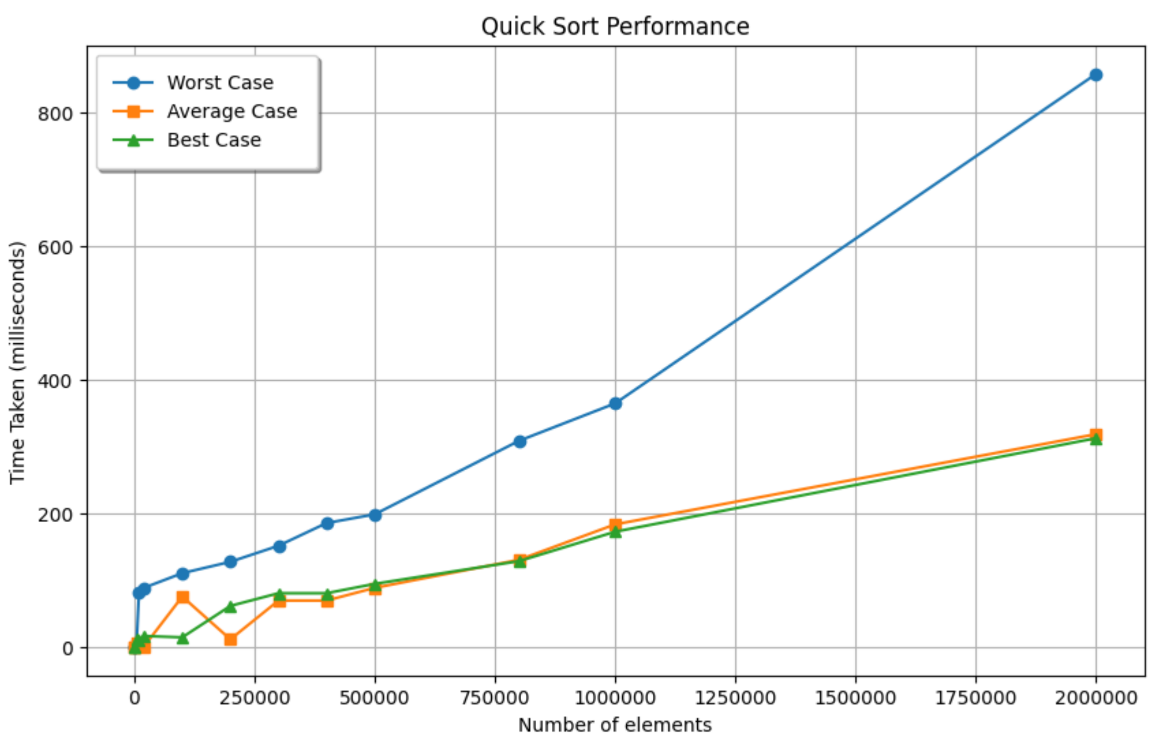
```
    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
```

```
        return i + 1;
    }

    static void quickSort(int arr[], int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }
```

Output :-

```
609e4f90b4a0db0bc\redhat.java\jdt_ws\DAA
Enter the size of array
3000
2 1 5
(base) PS E:\programing\DAA>
```

Graph :-



2. <u>**Merge Sort**</u>

Code :-

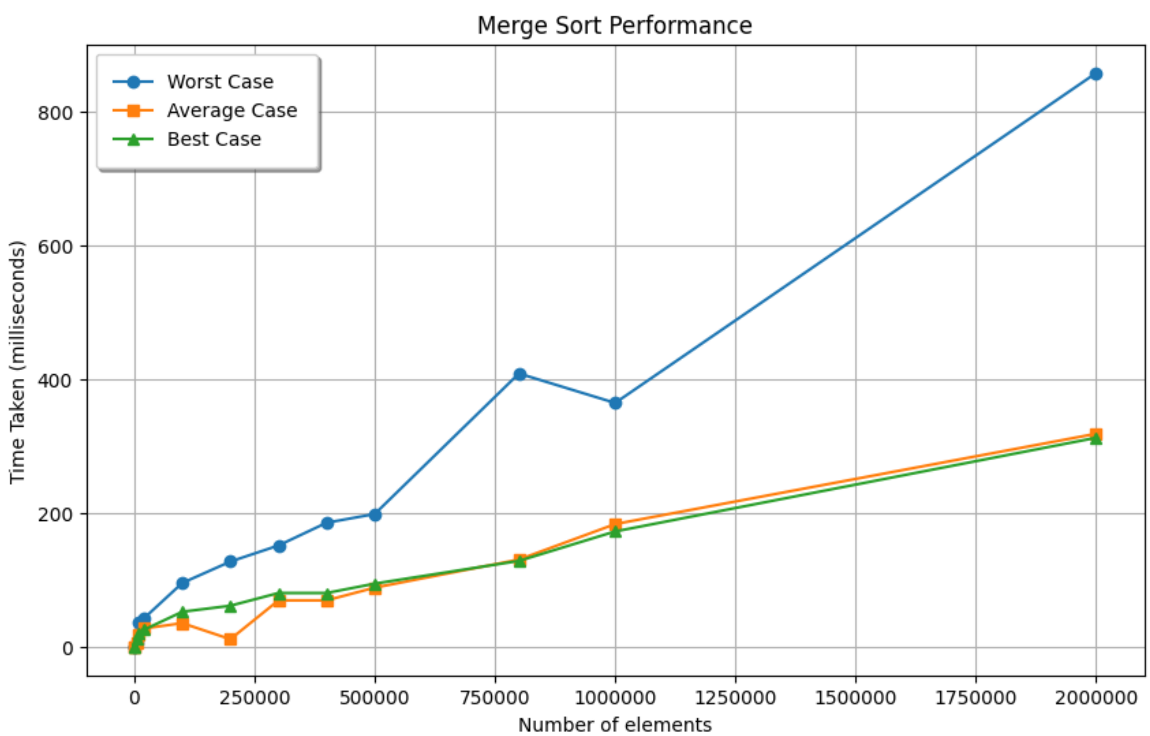
```
static void merge(int arr[],int s,int e){
    int mid = (s+e)/2;
    int len1 = mid - s +1;
    int len2 = e- mid;
    int[] arr1 = new int[len1];
    int[] arr2 = new int[len2];
    int ind = s;
    for(int i=0;i<len1;i++){
        arr1[i] = arr[ind++];
    }
    ind = mid+1;
    for(int i=0;i<len2;i++){
        arr2[i] = arr[ind++];
    }
    int index1=0;
    int index2=0;
    ind = s;
    while(index1 < len1 && index2<len2){
        if(arr1[index1] > arr2[index2]){
            arr[ind++] = arr2[index2++];
        }
        else{
            arr[ind++] = arr1[index1++];
        }
    }

    while (index1<len1)
    {
        arr[ind++] = arr1[index1++];
    }
    while (index2<len2)
    {
        arr[ind++] = arr2[index2++];
    }
}

static void mergeSort(int arr[] ,int s, int e){
    if(s >= e){
        return;
```

```
        }
        int mid = (s+e)/2;
        mergeSort(arr, s, mid);
        mergeSort(arr, mid+1, e);
        merge(arr,s,e);
    }
```

Output :-

```
a\jdt_ws\DAA_42f11d6d\bin' 'MergeSortTimeComplexity'
Enter the size of array
100000
39 94 18
```

Graph :-



Lab - 4

Aim: To implement the Strassen's algorithm using an array as data structure and analyzing its time complexity.

Code :-

```
import java.util.Scanner;

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

         double[][] M1 = multiply(add(A11, A22), add(B11, B22));
         double[][] M2 = multiply(add(A21, A22), B11);
         double[][] M3 = multiply(A11, sub(B12, B22));
         double[][] M4 = multiply(A22, sub(B21, B11));
         double[][] M5 = multiply(add(A11, A12), B22);
         double[][] M6 = multiply(sub(A21, A11), add(B11, B12));
         double[][] M7 = multiply(sub(A12, A22), add(B21, B22));
```

```
        double[][] C11 = add(sub(add(M1, M4), M5), M7);
        double[][] C12 = add(M3, M5);
        double[][] C21 = add(M2, M4);
        double[][] 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 double[][] sub(double[][] A, double[][] B) {
    int n = A.length;
    double[][] C = new double[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 double[][] add(double[][] A, double[][] B) {
    int n = A.length;
    double[][] C = new double[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(double[][] P, double[][] 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(double[][] C, double[][] P, int iB, int jB) {
    for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
```

```java
        for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
            P[i2][j2] = C[i1][j1];
}

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.println("Strassen Multiplication Algorithm\n");
    StrassenMatrixTimeComplexity s = new StrassenMatrixTimeComplexity();

    System.out.print("Enter order n :");
    int N = scan.nextInt();

    double[][] A = new double[N][N];
    double[][] B = new double[N][N];

    System.out.println("\nEnter " + N + " order matrix A :");

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            A[i][j] = scan.nextDouble();

    System.out.println("\nEnter " + N + " order matrix B :");

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            B[i][j] = scan.nextDouble();

    scan.close();

    System.out.println("\nMatrix A =>");

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            System.out.print(A[i][j] + " ");
        System.out.println();
    }

    System.out.println("\nMatrix B =>");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
```

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

        long startTime = System.nanoTime();
        double[][] C = s.multiply(A, B);
        long endTime = System.nanoTime();
        long timeElapsed = endTime - startTime;

        System.out.println("\nProduct of matrices A and B : ");

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(C[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("\nTime taken to multiply matrices A and B : " +
timeElapsed + " nanoseconds");
    }
}
```

Output :-

```
spot\bin\java.exe' '-cp' 'C:\Users\hp\AppData\Roaming\Code\User\workspaceStorag
0db0bc\redhat.java\jdt_ws\DAA_42f11d6d\bin' 'StrassenMatrixTimeComplexity'
Strassen Multiplication Algorithm

Enter order n :4

Enter 4 order matrix A :
5.0 -3.0 7.12 -11.54
-12.43   4    -00.43  02
-0.0 5.23 14.0 3.0
34.0 32.43 -27.69 3.0

Enter 4 order matrix B :
3.65 -42.99 13.09 12.0
5.53 54.53 1.0 -1.0
1.0 2.0 8.0 9.0
2.4 13.5 14.5 4.0

Matrix A =>
5.0 -3.0 7.12 -11.54
-12.43 4.0 -0.43 2.0
-0.0 5.23 14.0 3.0
34.0 32.43 -27.69 3.0

Matrix B =>
3.65 -42.99 13.09 12.0
5.53 54.53 1.0 -1.0
1.0 2.0 8.0 9.0
2.4 13.5 14.5 4.0

Product of matrices A and B :
-18.9159999999996 -520.0899999999999 -47.92 80.92000000000002
-18.8795000000000064 778.6257000000007 -133.1487 -149.03000000000003
50.12189999999987 353.69190000000003 160.73000000000042 132.7700000000001
282.9479 291.86789999999996 299.4699999999999 138.36000000000092

Time taken to multiply matrices A and B : 604900 nanoseconds
(base) PS E:\programing\DAA>
```

Lab - 5

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

Template:-

```
import java.util.Random;
import java.util.Scanner;

public class HeapSortTimeComplexity {

    static void print(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }

     static void Sort(int arr[]) {
          // Change This function with the proper functions given below
    }

    static void SortTime(int[] arr) {
        long start = System.currentTimeMillis();
        heapSort(arr);
        long end = System.currentTimeMillis();
        long time = end - start;
        System.out.print(time + " ");
    }

    static class BestCase {
        int[] arr;
        BestCase(int n) {
            arr = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = i;
            }
        }
    }
```

```
    static class WorstCase {
        int[] arr;
        WorstCase(int n) {
            arr = new int[n];
            for (int i = 0; i < n; i++) {
                arr[i] = n - i;
            }
        }
    }

    static class AverageCase {
        int[] arr;
        AverageCase(int n) {
            arr = new int[n];
            Random rand = new Random();
            for (int i = 0; i < n; i++) {
                arr[i] = rand.nextInt(n);
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of array ");
        int n = sc.nextInt();
        sc.close();

        AverageCase avgCase = new AverageCase(n);
        BestCase bestCase = new BestCase(n);
        WorstCase worstCase = new WorstCase(n);

        SortTime(bestCase.arr);
        SortTime(avgCase.arr);
        SortTime(worstCase.arr);
    }
}
```

1. <u>**Heap Sort**</u>

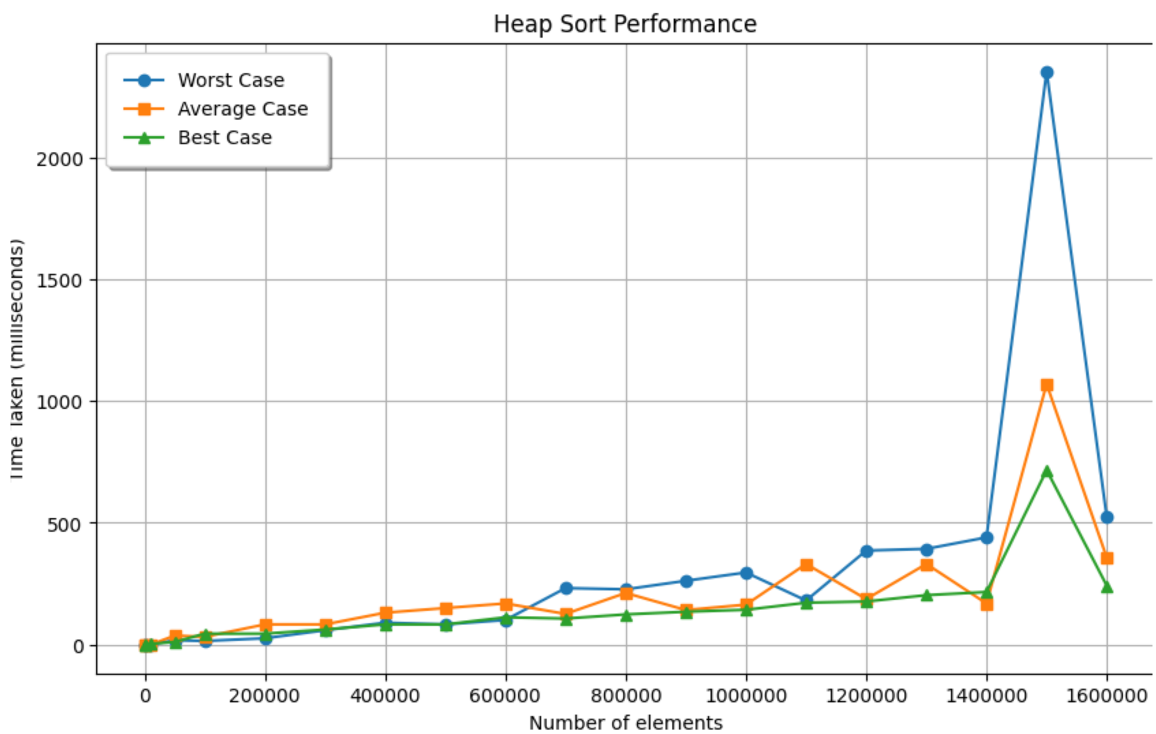
   Code :-

```
static void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        int swap = arr[i];
        arr[i] = arr[largest];
        arr[largest] = swap;

        heapify(arr, n, largest);
    }
}

static 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);
    }
}
```

Output :-

```
(base) PS E:\programing\DAA>  & 'C:\Pro
42f11d6d\bin' 'HeapSortTimeComplexity'
Enter the size of array
1600000
242 523 357
(base) PS E:\programing\DAA> 
```

Graph :-



Lab - 6

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

1. **Huffman Encoding**

Code :-

```
import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;

class HuffmanNode implements Comparable<HuffmanNode> {
    char data;
    int frequency;
    HuffmanNode left, right;

    public HuffmanNode(char data, int frequency) {
        this.data = data;
        this.frequency = frequency;
    }

    @Override
    public int compareTo(HuffmanNode other) {
        return Integer.compare(this.frequency, other.frequency);
    }
}

public class HuffmanCoding {

    private static HuffmanNode buildHuffmanTree(Map<Character, Integer>
frequencyMap) {
        PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();

        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet())
{
            minHeap.offer(new HuffmanNode(entry.getKey(),
entry.getValue()));
        }
```

```
        while (minHeap.size() > 1) {
            HuffmanNode left = minHeap.poll();
            HuffmanNode right = minHeap.poll();

            HuffmanNode internalNode = new HuffmanNode('$', left.frequency
+ right.frequency);
            internalNode.left = left;
            internalNode.right = right;

            minHeap.offer(internalNode);
        }

        return minHeap.poll();
    }

    private static void generateHuffmanCodes(HuffmanNode root, String
code, Map<Character, String> huffmanCodes) {
        if (root == null) {
            return;
        }

        if (root.left == null && root.right == null) {
            huffmanCodes.put(root.data, code);
        }

        generateHuffmanCodes(root.left, code + "0", huffmanCodes);
        generateHuffmanCodes(root.right, code + "1", huffmanCodes);
    }

    private static String encodeMessage(String message, Map<Character,
String> huffmanCodes) {
        StringBuilder encodedMessage = new StringBuilder();
        for (char ch : message.toCharArray()) {
            encodedMessage.append(huffmanCodes.get(ch));
        }
        return encodedMessage.toString();
    }

    private static String decodeMessage(String encodedMessage,
HuffmanNode root) {
```

```
        StringBuilder decodedMessage = new StringBuilder();
        HuffmanNode current = root;

        for (char bit : encodedMessage.toCharArray()) {
            if (bit == '0') {
                current = current.left;
            } else {
                current = current.right;
            }

            if (current.left == null && current.right == null) {
                decodedMessage.append(current.data);
                current = root;
            }
        }

        return decodedMessage.toString();
    }

    public static void main(String[] args) {
        String message = "hello world";

        Map<Character, Integer> frequencyMap = new HashMap<>();
        for (char ch : message.toCharArray()) {
            frequencyMap.put(ch, frequencyMap.getOrDefault(ch, 0) + 1);
        }

        long startTime = System.nanoTime();
        HuffmanNode root = buildHuffmanTree(frequencyMap);
        long endTime = System.nanoTime();
        long buildTime = endTime - startTime;

        Map<Character, String> huffmanCodes = new HashMap<>();
        generateHuffmanCodes(root, "", huffmanCodes);

        startTime = System.nanoTime();
        String encodedMessage = encodeMessage(message,
huffmanCodes);
        endTime = System.nanoTime();
        long encodeTime = endTime - startTime;
```

```java
        startTime = System.nanoTime();
        String decodedMessage = decodeMessage(encodedMessage, root);
        endTime = System.nanoTime();
        long decodeTime = endTime - startTime;

        System.out.println("Original Message: " + message);
        System.out.println("Encoded Message: " + encodedMessage);
        System.out.println("Decoded Message: " + decodedMessage);
        System.out.println("Huffman Tree Build Time: " + buildTime + " ns");
        System.out.println("Encoding Time: " + encodeTime + " ns");
        System.out.println("Decoding Time: " + decodeTime + " ns");
    }
}
```

Output :-

```
Original Message: hello world
Encoded Message: 01011101010110000111111000110011
Decoded Message: hello world
Huffman Tree Build Time: 2624300 ns
Encoding Time: 92300 ns
Decoding Time: 11100 ns
(base) PS E:\programing\DAA>
```

2. **<u>Minimum Spanning Tree</u>**

Code:-

```java
import java.util.Arrays;
class Graph {
    private int vertices;
    private int[][] graph;
    public Graph(int vertices) {
        this.vertices = vertices;
        this.graph = new int[vertices][vertices];
    }
    public void addEdge(int u, int v, int weight) {
        this.graph[u][v] = weight;
        this.graph[v][u] = weight;
    }
```

```
    public int minKey(int[] key, boolean[] mstSet) {
        int minVal = Integer.MAX_VALUE;
        int minIndex = -1;
        for (int v = 0; v < this.vertices; v++) {
            if (!mstSet[v] && key[v] < minVal) {
                minVal = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }

    public int[] primMST() {
        int[] key = new int[this.vertices];
        int[] parent = new int[this.vertices];
        boolean[] mstSet = new boolean[this.vertices];
        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(parent, -1);
        key[0] = 0;
        for (int i = 0; i < this.vertices - 1; i++) {
            int u = minKey(key, mstSet);
            mstSet[u] = true;
            for (int v = 0; v < this.vertices; v++) {
                if (this.graph[u][v] > 0 && !mstSet[v] && key[v] > this.graph[u][v])
{
                    key[v] = this.graph[u][v];
                    parent[v] = u;
                }
            }
        }
        return parent;
    }
}

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

```
        best.addEdge(3, 0, 2);
        best.addEdge(0, 2, 2);
        best.addEdge(1, 3, 2);
        long startB = System.currentTimeMillis();
        int[] bestParent = best.primMST();
        long endB = System.currentTimeMillis();
        String bestTime = String.format("%.6f", (endB - startB) / 1000.0);
        Graph worst = new Graph(4);
        worst.addEdge(0, 1, 1);
        worst.addEdge(1, 2, 2);
        worst.addEdge(2, 3, 3);
        worst.addEdge(3, 0, 4);
        worst.addEdge(0, 2, 5);
        worst.addEdge(1, 3, 6);
        long startW = System.currentTimeMillis();
        int[] worstParent = worst.primMST();
        long endW = System.currentTimeMillis();
        String worstTime = String.format("%.6f", (endW - startW) / 1000.0);
        System.out.println("Best case:");
        System.out.println("Time taken: " + bestTime);
        System.out.println("MST Parent array: " +
Arrays.toString(bestParent));

        System.out.println("\nWorst case:");
        System.out.println("Time taken: " + worstTime);
        System.out.println("MST Parent array: " +
Arrays.toString(worstParent));
    }
}
```

Output :-

```
Best case:
Time taken: 0.000000
MST Parent array: [-1, 0, 0, 0]

Worst case:
Time taken: 0.000000
MST Parent array: [-1, 0, 1, 2]
(base) PS E:\programing\DAA>
```

Lab - 7

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

1. **Dijkstra's Algorithm**

Code:-

```
import java.util.Arrays;
import java.util.PriorityQueue;

class Graph {
    private int vertices;
    private int[][] adjacencyMatrix;

    public Graph(int vertices) {
        this.vertices = vertices;
        this.adjacencyMatrix = new int[vertices][vertices];
    }

    public void addEdge(int source, int destination, int weight) {
        this.adjacencyMatrix[source][destination] = weight;
        this.adjacencyMatrix[destination][source] = weight;
    }

    public int[] dijkstra(int startVertex) {
        int[] distance = new int[vertices];
        Arrays.fill(distance, Integer.MAX_VALUE);

        distance[startVertex] = 0;
        PriorityQueue<Node> priorityQueue = new PriorityQueue<>();
        priorityQueue.add(new Node(startVertex, 0));

        while (!priorityQueue.isEmpty()) {
            int currentVertex = priorityQueue.poll().vertex;

            for (int adjacentVertex = 0; adjacentVertex < vertices;
adjacentVertex++) {
                int edgeWeight =
adjacencyMatrix[currentVertex][adjacentVertex];
```

```
                if (edgeWeight > 0) {
                    int newDistance = distance[currentVertex] + edgeWeight;

                    if (newDistance < distance[adjacentVertex]) {
                        distance[adjacentVertex] = newDistance;
                        priorityQueue.add(new Node(adjacentVertex,
newDistance));
                    }
                }
            }
        }

        return distance;
    }

    static class Node implements Comparable<Node> {
        int vertex;
        int distance;

        public Node(int vertex, int distance) {
            this.vertex = vertex;
            this.distance = distance;
        }

        @Override
        public int compareTo(Node other) {
            return Integer.compare(this.distance, other.distance);
        }
    }
}

public class DijkstraAlgorithm {

    public static void main(String[] args) {
        Graph graph = new Graph(5);
        graph.addEdge(0, 1, 2);
        graph.addEdge(0, 2, 4);
        graph.addEdge(1, 2, 1);
        graph.addEdge(1, 3, 7);
```

```
        graph.addEdge(2, 4, 3);
        graph.addEdge(3, 4, 1);

        long startBest = System.nanoTime();
        int[] bestCaseDistance = graph.dijkstra(0);
        long endBest = System.nanoTime();
        double bestTime = (endBest - startBest) / 1e6;

        long startWorst = System.nanoTime();
        int[] worstCaseDistance = graph.dijkstra(0);
        long endWorst = System.nanoTime();
        double worstTime = (endWorst - startWorst) / 1e6;

        System.out.println("Best case:");
        System.out.println("Shortest distances: " +
Arrays.toString(bestCaseDistance));
        System.out.println("Time taken: " + bestTime + " ms");

        System.out.println("\nWorst case:");
        System.out.println("Shortest distances: " +
Arrays.toString(worstCaseDistance));
        System.out.println("Time taken: " + worstTime + " ms");
    }
}
```

Output: -

```
Best case:
Shortest distances: [0, 2, 3, 7, 6]
Time taken: 2.7695 ms

Worst case:
Shortest distances: [0, 2, 3, 7, 6]
Time taken: 0.0099 ms
(base) PS E:\programing\DAA>
```

2. **<u>Bellman Ford Algorithm</u>**

Code:-

```
import java.util.Arrays;

class Graph {
    private int vertices;
    private int edges;
    private Edge[] edgeList;

    public Graph(int vertices, int edges) {
        this.vertices = vertices;
        this.edges = edges;
        this.edgeList = new Edge[edges];
        for (int i = 0; i < edges; i++) {
            this.edgeList[i] = new Edge();
        }
    }

    public void addEdge(int source, int destination, int weight, int
edgeIndex) {
        this.edgeList[edgeIndex].source = source;
        this.edgeList[edgeIndex].destination = destination;
        this.edgeList[edgeIndex].weight = weight;
    }

    public void bellmanFord(int startVertex) {
        int[] distance = new int[vertices];
        Arrays.fill(distance, Integer.MAX_VALUE);
        distance[startVertex] = 0;

        for (int i = 0; i < vertices - 1; i++) {
            for (int j = 0; j < edges; j++) {
                int u = edgeList[j].source;
                int v = edgeList[j].destination;
                int weight = edgeList[j].weight;

                if (distance[u] != Integer.MAX_VALUE && distance[u] + weight <
distance[v]) {
                    distance[v] = distance[u] + weight;
```

```
            }
        }
    }

    for (int j = 0; j < edges; j++) {
        int u = edgeList[j].source;
        int v = edgeList[j].destination;
        int weight = edgeList[j].weight;

        if (distance[u] != Integer.MAX_VALUE && distance[u] + weight <
distance[v]) {
            System.out.println("Graph contains negative weight cycle");
            return;
        }
    }
    System.out.println("Shortest distances from vertex " + startVertex + ":
" + Arrays.toString(distance));
  }

  static class Edge {
      int source, destination, weight;
  }
}

public class BellmanFordAlgorithm {

  public static void main(String[] args) {
      int vertices = 5;
      int edges = 8;

      Graph graph = new Graph(vertices, edges);

      graph.addEdge(0, 1, -1, 0);
      graph.addEdge(0, 2, 4, 1);
      graph.addEdge(1, 2, 3, 2);
      graph.addEdge(1, 3, 2, 3);
      graph.addEdge(1, 4, 2, 4);
      graph.addEdge(3, 2, 5, 5);
      graph.addEdge(3, 1, 1, 6);
      graph.addEdge(4, 3, -3, 7);
```

```
        long startBest = System.nanoTime();
        graph.bellmanFord(0);
        long endBest = System.nanoTime();
        double bestTime = (endBest - startBest) / 1e6;

        System.out.println("\nTime taken for best case: " + bestTime + " ms");
    }
}
```

Output: -

```
Shortest distances from vertex 0: [0, -1, 2, -2, 1]

Time taken for best case: 91.7914 ms
(base) PS E:\programing\DAA>
```

Lab - 8

Aim: Implement NQueen's problem using Backtracking.

Code:-

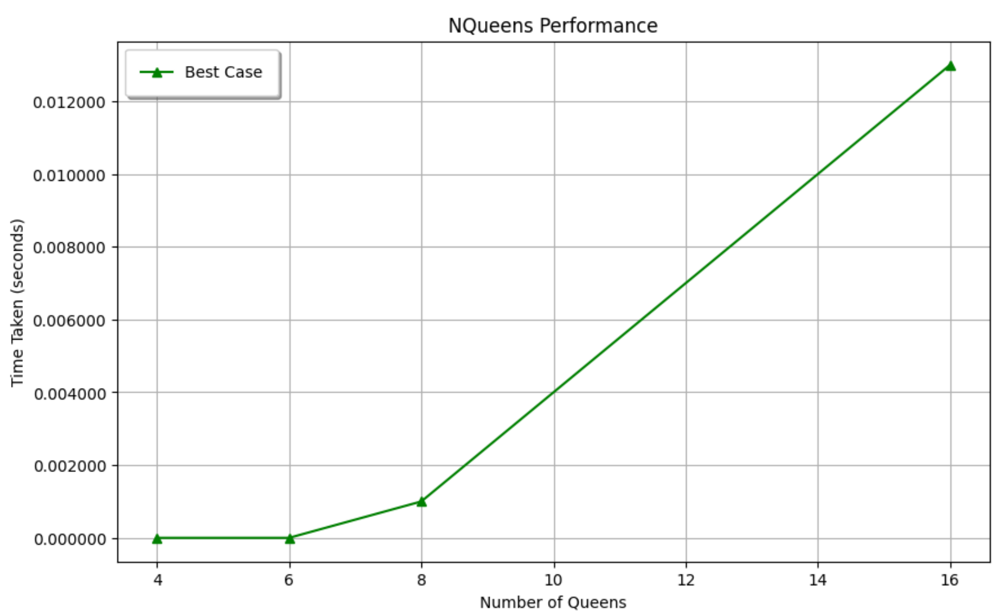
```
public class NQueens {
    public static boolean isSafe(int[][] board, int row, int col, int n) {
        for (int i = 0; i < col; i++) {
            if (board[row][i] == 1) {
                return false;
            }
        }
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
        for (int i = row, j = col; i < n && j >= 0; i++, j--) {
            if (board[i][j] == 1)
                return false;
        }
        return true;
    }
    public static boolean solveNQueensUtil(int[][] board, int col, int n) {
        if (col >= n) {
            return true;
        }
        for (int i = 0; i < n; i++) {
            if (isSafe(board, i, col, n)) {
                board[i][col] = 1;
                if (solveNQueensUtil(board, col + 1, n))
                    return true;
                board[i][col] = 0;}
        }
        return false;
    }
    public static void solveNQueens(int n) {
        int[][] board = new int[n][n];
        if (!solveNQueensUtil(board, 0, n)) {
            System.out.println("Solution does not exist");
```

```
            return;}
    }
    public static void main(String[] args) {
        int[] queens = {4, 6, 8, 16};
        double[] times = new double[queens.length];
        for (int i = 0; i < queens.length; i++) {
            long startTime = System.currentTimeMillis();
            solveNQueens(queens[i]);
            long endTime = System.currentTimeMillis();
            times[i] = (endTime - startTime) / 1000.0;
        }
        System.out.println("Number of queens        Time taken");
        for (int i = 0; i < queens.length; i++) {
            System.out.printf("%d                        %.6f%n", queens[i], times[i]);
        }
    }
}
```

Output: -

```
Number of queens          Time taken
4                           0.000000
6                           0.000000
8                           0.001000
16                           0.013000
(base) PS E:\programing\DAA>
```

Graph: -



Lab - 9

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

1) **Matrix Chain Multiplication**

Code:-

```
import java.util.Arrays;
public class MatrixChainMultiplication {
    public static void matrixChainMultiplication(int[] p) {
        int n = p.length - 1;
        int[][] m = new int[n + 1][n + 1];
        int[][] s = new int[n + 1][n + 1];
        for (int length = 2; length <= n; length++) {
            for (int i = 1; i <= n - length + 1; i++) {
                int j = i + length - 1;
                m[i][j] = Integer.MAX_VALUE;
                for (int k = i; k < j; k++) {
                    int cost = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                    if (cost < m[i][j]) {
                        m[i][j] = cost;
                        s[i][j] = k;
                    }
                }
            }
        }
        printOptimalParentheses(s, 1, n);
    }
    public static void printOptimalParentheses(int[][] s, int i, int j) {
        if (i == j) {
            System.out.print("A" + i);
        } else {
            System.out.print("(");
            printOptimalParentheses(s, i, s[i][j]);
            printOptimalParentheses(s, s[i][j] + 1, j);
            System.out.print(")");
        }
    }
```

```
    public static void runExperiment(int[] matrixDimensions) {
        long startTime = System.currentTimeMillis();
        matrixChainMultiplication(matrixDimensions);
        long endTime = System.currentTimeMillis();
        double elapsedTime = (endTime - startTime) / 1000.0;
        System.out.println("Matrix Dimensions: " +
Arrays.toString(matrixDimensions));
        System.out.println("Time taken: " + String.format("%.6f", elapsedTime) +
"\n");
    }
    public static void main(String[] args) {
        runExperiment(new int[]{30, 35, 15, 5, 10, 20, 25});
        runExperiment(new int[]{10, 20, 30, 40, 30});
        runExperiment(new int[]{5, 10, 3, 12, 5, 50, 6});
    }
}
```

Output:-

```
((A1(A2A3))((A4A5)A6))Matrix Dimensions: [30, 35, 15, 5, 10, 20, 25]
Time taken: 0.095000

(((A1A2)A3)A4)Matrix Dimensions: [10, 20, 30, 40, 30]
Time taken: 0.004000

((A1A2)((A3A4)(A5A6)))Matrix Dimensions: [5, 10, 3, 12, 5, 50, 6]
Time taken: 0.001000

(base) PS E:\programing\DAA>
```

2) <u>Longest Common Subsequence</u>

Code:-

```
public class LongestCommonSubsequence {
    public static char[] longestCommonSubsequence(String X, String Y) {
        int m = X.length();
        int n = Y.length();
        int[][] lcsTable = 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) {
                    lcsTable[i][j] = 0;
                } else if (X.charAt(i - 1) == Y.charAt(j - 1)) {
```

```java
                lcsTable[i][j] = lcsTable[i - 1][j - 1] + 1;
            } else {
                lcsTable[i][j] = Math.max(lcsTable[i - 1][j], lcsTable[i][j - 1]);
            }
        }
    }
    char[] lcs = new char[lcsTable[m][n]];
    int i = m, j = n, index = lcs.length - 1;
    while (i > 0 && j > 0) {
        if (X.charAt(i - 1) == Y.charAt(j - 1)) {
            lcs[index] = X.charAt(i - 1);
            i--;
            j--;
            index--;
        } else if (lcsTable[i - 1][j] > lcsTable[i][j - 1]) {
            i--;
        } else {
            j--;
        }
    }
    return lcs;
}
public static void main(String[] args) {
    String XBest = "ANKURRAWAT";
    String YBest = "AKRRAT";
    String XWorst = "OCALMMEOCA";
    String YWorst = "XYZ";
    String XAverage = "ANKJ";
    String YAverage = "XLKQYJ";
    long startBest = System.currentTimeMillis();
    char[] resultBest = longestCommonSubsequence(XBest, YBest);
    long endBest = System.currentTimeMillis();
    String timeBest = String.format("%.6f", (endBest - startBest) / 1000.0);
    String resultStringBest = resultBest.length == 0 ? "Non common
subsequence" : new String(resultBest);
    System.out.println("Best Case:");
    System.out.println("Longest Common Subsequence: " + resultStringBest);
    System.out.println("Time taken: " + timeBest + "\n");
    long startWorst = System.currentTimeMillis();
    char[] resultWorst = longestCommonSubsequence(XWorst, YWorst);
```

```
        long endWorst = System.currentTimeMillis();
        String timeWorst = String.format("%.6f", (endWorst - startWorst) / 1000.0);
        String resultStringWorst = resultWorst.length == 0 ? "Non common subsequence" : new String(resultWorst);
        System.out.println("Worst Case:");
        System.out.println("Longest Common Subsequence: " + resultStringWorst);
        System.out.println("Time taken: " + timeWorst + "\n");
        long startAverage = System.currentTimeMillis();
        char[] resultAverage = longestCommonSubsequence(XAverage, YAverage);
        long endAverage = System.currentTimeMillis();
        String timeAverage = String.format("%.6f", (endAverage - startAverage) / 1000.0);
        String resultStringAverage = resultAverage.length == 0 ? "Non common subsequence" : new String(resultAverage);
        System.out.println("Average Case:");
        System.out.println("Longest Common Subsequence: " + resultStringAverage);
        System.out.println("Time taken: " + timeAverage + "\n");
    }
}
```

Output:-

```
Best Case:
Longest Common Subsequence: AKRRAT
Time taken: 0.000000

Worst Case:
Longest Common Subsequence: Non common subsequence
Time taken: 0.000000

Average Case:
Longest Common Subsequence: KJ
Time taken: 0.000000

(base) PS E:\programing\DAA>
```

Lab - 10

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

Code:-

```
import java.util.ArrayList;
import java.util.List;
public class StringMatchingAlgorithms {
    public static class NaiveStringMatchingResult {
        public List<Integer> occurrences;
        public String elapsedTime;
        public NaiveStringMatchingResult(List<Integer> occurrences, String elapsedTime)
{
            this.occurrences = occurrences;
            this.elapsedTime = elapsedTime;
        }
    }
    public static NaiveStringMatchingResult naiveStringMatching(String text, String
pattern) {
        List<Integer> occurrences = new ArrayList<>();
        int n = text.length();
        int m = pattern.length();
        long startTime = System.currentTimeMillis();
        for (int i = 0; i <= n - m; i++) {
            if (text.substring(i, i + m).equals(pattern)) {
                occurrences.add(i);
            }
        }
        long endTime = System.currentTimeMillis();
        double elapsedTime = (endTime - startTime) / 1000.0;
        return new NaiveStringMatchingResult(occurrences, String.format("%.6f",
elapsedTime));
    }
    public static class RabinKarpResult {
        public List<Integer> occurrences;
        public String elapsedTime;
        public RabinKarpResult(List<Integer> occurrences, String elapsedTime) {
            this.occurrences = occurrences;
            this.elapsedTime = elapsedTime;
```

```
        }
    }
    public static RabinKarpResult rabinKarp(String text, String pattern, int d, int q) {
        List<Integer> occurrences = new ArrayList<>();
        int n = text.length();
        int m = pattern.length();
        long startTime = System.currentTimeMillis();
        int hPattern = 0;
        int hWindow = 0;
        for (int i = 0; i < m; i++) {
            hPattern = (hPattern * d + pattern.charAt(i)) % q;
            hWindow = (hWindow * d + text.charAt(i)) % q;}
        for (int i = 0; i <= n - m; i++) {
            if (hPattern == hWindow && text.substring(i, i + m).equals(pattern))
                occurrences.add(i);
            if (i < n - m) {
                hWindow = (d * (hWindow - text.charAt(i) * pow(d, m - 1)) + text.charAt(i + m))
% q;
                if (hWindow < 0)
                    hWindow += q;
            }
        }
        long endTime = System.currentTimeMillis();
        double elapsedTime = (endTime - startTime) / 1000.0;
        return new RabinKarpResult(occurrences, String.format("%.6f", elapsedTime));
    }
    public static int[] computePrefixFunction(String pattern) {
        int m = pattern.length();
        int[] pi = new int[m];
        int k = 0;
        for (int q = 1; q < m; q++) {
            while (k > 0 && pattern.charAt(k) != pattern.charAt(q)) {
                k = pi[k - 1];
            }
            if (pattern.charAt(k) == pattern.charAt(q)) {
                k++;
            }
            pi[q] = k;}
        return pi;}
    public static class KnuthMorrisPrattResult {
```

```
        public List<Integer> occurrences;
        public String elapsedTime;
        public KnuthMorrisPrattResult(List<Integer> occurrences, String elapsedTime) {
            this.occurrences = occurrences;
            this.elapsedTime = elapsedTime;
        }
    }
    public static KnuthMorrisPrattResult knuthMorrisPratt(String text, String pattern) {
        List<Integer> occurrences = new ArrayList<>();
        int n = text.length();
        int m = pattern.length();
        long startTime = System.currentTimeMillis();
        int[] pi = computePrefixFunction(pattern);
        int q = 0;
        for (int i = 0; i < n; i++) {
            while (q > 0 && pattern.charAt(q) != text.charAt(i)) {
                q = pi[q - 1];}
            if (pattern.charAt(q) == text.charAt(i)) {
                q++;}
            if (q == m) {
                occurrences.add(i - m + 1);
                q = pi[q - 1];}
        }
        long endTime = System.currentTimeMillis();
        double elapsedTime = (endTime - startTime) / 1000.0;
        return new KnuthMorrisPrattResult(occurrences, String.format("%.6f",
elapsedTime));
    }
    public static void main(String[] args) {
        String textBest = "TodayIsTheBestDayToCodeBecauseItIsRaining";
        String patternBest = "BestDayToCode";
        String textWorst = "RepetitionRepetitionRepetitionRepetitionRepetition";
        String patternWorst = "PatternPatternPattern";
        String textAverage =
"FindingMeaningfulStringsForStringMatchingCanBeChallengingButItIsImportant";
        String patternAverage = "MeaningfulStringsForStringMatching";
        NaiveStringMatchingResult naiveResultBest = naiveStringMatching(textBest,
patternBest);
        NaiveStringMatchingResult naiveResultWorst = naiveStringMatching(textWorst,
patternWorst);
```

```
        NaiveStringMatchingResult naiveResultAverage =
naiveStringMatching(textAverage, patternAverage);
        System.out.println("Naive String Matching:");
        System.out.println("Best Case - Occurrences: " + naiveResultBest.occurrences);
        System.out.println("Best Case - Time taken: " + naiveResultBest.elapsedTime + "
seconds");
        System.out.println("Worst Case - Occurrences: " + naiveResultWorst.occurrences);
        System.out.println("Worst Case - Time taken: " + naiveResultWorst.elapsedTime +
" seconds");
        System.out.println("Average Case - Occurrences: " +
naiveResultAverage.occurrences);
        System.out.println("Average Case - Time taken: " +
naiveResultAverage.elapsedTime + " seconds\n");
        RabinKarpResult rabinKarpResultBest = rabinKarp(textBest, patternBest, 256,
101);
        RabinKarpResult rabinKarpResultWorst = rabinKarp(textWorst, patternWorst, 256,
101);
        RabinKarpResult rabinKarpResultAverage = rabinKarp(textAverage,
patternAverage, 256, 101);
        System.out.println("Rabin-Karp Algorithm:");
        System.out.println("Best Case - Occurrences: " +
rabinKarpResultBest.occurrences);
        System.out.println("Best Case - Time taken: " + rabinKarpResultBest.elapsedTime
+ " seconds");
        System.out.println("Worst Case - Occurrences: " +
rabinKarpResultWorst.occurrences);
        System.out.println("Worst Case - Time taken: " +
rabinKarpResultWorst.elapsedTime + " seconds");
        System.out.println("Average Case - Occurrences: " +
rabinKarpResultAverage.occurrences);
        System.out.println("Average Case - Time taken: " +
rabinKarpResultAverage.elapsedTime + " seconds\n");
        KnuthMorrisPrattResult kmpResultBest = knuthMorrisPratt(textBest, patternBest);
        KnuthMorrisPrattResult kmpResultWorst = knuthMorrisPratt(textWorst,
patternWorst);
        KnuthMorrisPrattResult kmpResultAverage = knuthMorrisPratt(textAverage,
patternAverage);
        System.out.println("Knuth-Morris-Pratt Algorithm:");
        System.out.println("Best Case - Occurrences: " + kmpResultBest.occurrences);
```

```
        System.out.println("Best Case - Time taken: " + kmpResultBest.elapsedTime + " seconds");
        System.out.println("Worst Case - Occurrences: " + kmpResultWorst.occurrences);
        System.out.println("Worst Case - Time taken: " + kmpResultWorst.elapsedTime + " seconds");
        System.out.println("Average Case - Occurrences: " + kmpResultAverage.occurrences);
        System.out.println("Average Case - Time taken: " + kmpResultAverage.elapsedTime + " seconds");
    }
    private static int pow(int d, int e) {
        int result = 1;
        for (int i = 0; i < e; i++) {
            result *= d;}
        return result;}
}
```

Output: -

```
Naive String Matching:
Best Case - Occurrences: [10]
Best Case - Time taken: 0.000000 seconds
Worst Case - Occurrences: []
Worst Case - Time taken: 0.000000 seconds
Average Case - Occurrences: [7]
Average Case - Time taken: 0.000000 seconds

Rabin-Karp Algorithm:
Best Case - Occurrences: []
Best Case - Time taken: 0.000000 seconds
Worst Case - Occurrences: []
Worst Case - Time taken: 0.000000 seconds
Average Case - Occurrences: []
Average Case - Time taken: 0.000000 seconds

Knuth-Morris-Pratt Algorithm:
Best Case - Occurrences: [10]
Best Case - Time taken: 0.000000 seconds
Worst Case - Occurrences: []
Worst Case - Time taken: 0.000000 seconds
Average Case - Occurrences: [7]
Average Case - Time taken: 0.000000 seconds
(base) PS E:\programing\DAA>
```