Student Name: Pulkit
UID: 23BCS11733
Branch: CSE
Section/Group: KRG-3B
Semester: 5th
Date of Performance: 21 JUL
Subject Name: Design and Analysis of Algorithm (23CSH-301)
Experiment No: 1

AIM: Operations on Array

1. Insertion in Array

Algorithm:

- Start from the end and shift elements one position right.
- Insert the new element at the given position.
- Increase the size of the array.

Java Code:

```
public class ArrayOperations {
    public static int insertAtPosition(int[] arr, int size, int pos, int value)
        {
        if (pos < 0 || pos > size || size >= arr.length) {
            System.out.println("Invalid␣position");
            return size;
        }
        for (int i = size; i > pos; i--) {
            arr[i] = arr[i - 1];
        }
        arr[pos] = value;
        return size + 1;
    }
}
```

2. Deletion in Array

Algorithm:

- Start from the given position.
- Shift all elements left to overwrite the target.
- Decrease the size of the array.

Java Code:

```
public class ArrayOperations {
    public static int deleteAtPosition(int[] arr, int size, int pos) {
        if (pos < 0 || pos >= size) {
            System.out.println("Invalid␣position");
            return size;
        }
        for (int i = pos; i < size - 1; i++) {
            arr[i] = arr[i + 1];
        }
        return size - 1;
    }
}
```

3. Merge Sort

Algorithm:

- Divide the array into two halves.
- Recursively sort both halves.
- Merge the sorted halves.

Java Code:

```
public class MergeSort {
    public static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int mid = l + (r - l) / 2;
            mergeSort(arr, l, mid);
            mergeSort(arr, mid + 1, r);
            merge(arr, l, mid, r);
        }
    }

    public 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, k = l;

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

```
                arr[k++] = R[j++];
        }

        while (i < n1)
            arr[k++] = L[i++];
        while (j < n2)
            arr[k++] = R[j++];
    }
}
```

4. Quick Sort

Algorithm:

- Select a pivot element.
- Partition the array around the pivot.
- Recursively sort the left and right parts.

Java Code:

```
public class QuickSort {
    public 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);
        }
    }

    public 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;
    }
}
```