# DIVIDE AND CONQUER

1. Inversion Count

```java
import java.util.Scanner;
public class Inversion_count {
    public static int merge (int arr[], int left, int mid, int right) {
        int i = left, j = mid, k = 0;
        int invCount = 0;
        int temp[] = new int [(right - left + 1)];
        while ((i < mid) && (j <= right)) {
            if (arr[i] <= arr[j]) {
                temp[k] = arr[i];
                k++;
                i++;
            } else {
                temp[k] = arr[j];
                invCount += (mid - i);
                k++;
                j++;
            }
        }
        while (i < mid) {
            temp[k] = arr[i];
            k++;
            i++;
        }
        while (j <= right) {
            temp[k] = arr[j];
            k++;
            j++;
        }
        for (i = left, k = 0; i <= right; i++, k++) {
            arr[i] = temp[k];
        }
        return invCount;
    }
    private static int mergeSort (int arr[], int left, int right) {
        int invCount = 0;
        if (right > left) {
            int mid = (right + left) / 2;
            invCount = mergeSort(arr, left, mid);
            invCount += mergeSort(arr, mid + 1, right);
            invCount += merge(arr, left, mid + 1, right);
        }
        return invCount;
    }

    public static int getInversions (int arr[]) {
        int n = arr.length;
        return mergeSort(arr, 0, n-1);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
```

```
        System.out.print("Enter Size of an Array: ");
        int size = sc.nextInt();
        System.out.print("Enter Elements: ");
        int arr[] = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Inversion Count = " + getInversions(arr));
    }
}
```

2. Majority Elements

```
import java.util.Scanner;
public class Majority_elements {
    private static int countInRange (int nums[], int num, int lo, int hi) {
        int count = 0;
        for (int i = lo; i <= hi; i++) {
            if (nums[i] == num) {
                count ++;
            }
        }
        return count;
    }
    private static int majorityElementRec (int nums[], int lo, int hi) {
        if (lo == hi) {          // BASE CASE
            return nums[lo];
        }
        // reverse on left and right halves of this slice
        int mid = (hi - lo) / 2 + lo;
        int left = majorityElementRec(nums, lo, mid);
        int right = majorityElementRec(nums, mid + 1, hi);
        // if the two halves agree on majority element, then return it
        if (left == right) {
            return left;
        }
        // otherwise, count each element and return the "winner"
        int leftCount = countInRange(nums, left, lo, hi);
        int rightCount = countInRange(nums, right, lo, hi);
        return leftCount > rightCount ? left : right;
    }
    public static int majorityElement (int nums[]) {
        return majorityElementRec(nums, 0, nums.length-1);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter size of an Array: ");
        int size = sc.nextInt();
        System.out.print("Enter Elements: ");
        int arr[] = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Majority Element is " + majorityElement(arr));
    }
}
```

3. Merge Sort

```java
import java.util.Scanner;

public class mergeSort {

    public static void merge (int arr[], int si, int mid, int ei) {
        int temp[] = new int[ei-si+1];
        int i = si; // iterator for left part
        int j = mid + 1; // iterator for right part
        int k = 0; // iterator for temp array
        while (i <= mid && j <= ei) {
            if (arr[i] < arr[j]) {
                temp[k] = arr[i];
                i++;
            }
            else {
                temp[k] = arr[j];
                j++;
            }
            k++;
        }
        // Left Part
        while (i <= mid) {
            temp[k++] = arr[i++];
        }
        // Right Part
        while (j <= ei) {
            temp[k++] = arr[j++];
        }
        // Copying temp to original array
        for (k = 0, i = si; k < temp.length; k++, i++) {
            arr[i] = temp[k];
        }
    }

    public static void merge_sort (int arr[], int si, int ei) {
        if (si >= ei) {
            return;
        }
        int mid = si + (ei - si)/2; // (si + ei) / 2
        merge_sort(arr, si, mid); // left part
        merge_sort(arr, mid + 1, ei); // right part
        merge(arr, si, mid, ei);
    }

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

    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter size of an Array: ");
        int size = sc.nextInt();
```

```
        System.out.print("Enter elements: ");
        int arr[] = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("After Performing Merge Sort: ");
        merge_sort(arr, 0,size-1);
        printArr(arr);
    }
}
```

4. Quick Sort

```
import java.util.Scanner;

public class quickSort {

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

    public static int partition (int arr[], int si, int ei) {
        int pivot = arr[ei];
        int i = si - 1; // to make place for elements smaller than pivot.
        for (int j = si; j < ei; j++) {
            if (arr[j] <= pivot) {
                i++;
                // swapping
                int temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
            }
        }
        i++;
        int temp = pivot;
        arr[ei] = arr[i];
        arr[i] = temp;
        return i;
    }

    public static void quickSorting (int arr[], int si, int ei) {
        if (si >= ei) {
            return;
        }
        // Last Element
        int pIdx = partition (arr, si, ei);
        quickSorting(arr, si, pIdx - 1); // left
        quickSorting(arr, pIdx + 1, ei); // right
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter size of an Array: ");
```

```
        int size = sc.nextInt();
        System.out.print("Enter Elements: ");
        int arr[] = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("After Performing Quick Sort: ");
        quickSorting(arr, 0, size-1);
        printArr(arr);
    }
}
```

5. Search in Rotated Sorted Array

```
import java.util.Scanner;

public class SearchInRotatedSorted {

    public static int search (int arr[], int target, int si, int ei) {
        if (si > ei) {
            return -1;
        }
        // KAAM
        int mid = si + (ei - si) / 2; // OR (si + ei) / 2
        // CASE FOUND
        if (arr[mid] == target) {
            return mid;
        }
        // mid on line1
        if (arr[si] <= arr[mid]) {
            // case a: left
            if (arr[si] <= target && target <= arr[mid]) {
                return search(arr, target, si, mid-1);
            }
            // case b: right
            else {
                return search(arr, target, mid + 1, ei);
            }
        }
        // mid on line2
        else {
            // case c: right
            if (arr[mid] <= target && target <= arr[ei]) {
                return search(arr, target, mid + 1, ei);
            }
            // case d: left
            else {
                return search(arr, target, si, mid - 1);
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter size of an Array: ");
        int size = sc.nextInt();
```

```
        System.out.print("Enter Elements: ");
        int arr[] = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter Target: ");
        int target = sc.nextInt();
        int tarIdx = search(arr, target, 0, size-1);
        System.out.println("Index of the Target is: " + tarIdx);
    }
}
```

6. Sorted Strings

```
import java.util.Scanner;

public class sortingStrings {

    public static boolean isAlphabeticallySmaller (String str1, String str2) {
        if (str1.compareTo(str2) < 0) {
            return true;
        }
        return false;
    }

    public static String[] merge (String arr1[], String arr2[]) {
        int m = arr1.length;
        int n = arr2.length;
        String[] arr3 = new String[m + n];
        int idx = 0;
        int i = 0;
        int j = 0;
        while (i < m && j < n) {
            if (isAlphabeticallySmaller(arr1[i], arr2[j])) {
                arr3[idx] = arr1[i];
                i++;
            }
            else {
                arr3[idx] = arr2[j];
                j++;
            }
            idx++;
        }
        while (i < m) {
            arr3[idx] = arr1[i];
            i++;
            idx++;
        }
        while (j < n) {
            arr3[idx] = arr2[j];
            j++;
            idx++;
        }
        return arr3;
    }
```

```
    public static String[] mergeSort (String arr[], int lo, int hi) {
        if (lo == hi) {
            String[] A = { arr[lo] };
            return A;
        }
        int mid = lo + (hi - lo) / 2;
        String[] arr1 = mergeSort(arr, lo, mid);
        String[] arr2 = mergeSort(arr, mid + 1, hi);
        String[] arr3 = merge(arr1, arr2);
        return arr3;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("How many no. of Strings: ");
        int size  = sc.nextInt();
        System.out.print("Enter Strings: ");
        String arr[] = new String[size];
        for (int i = 0; i < size; i++) {
            arr[i] = sc.next();
        }
        System.out.println("After Sorting the Strings: ");
        String[] a = mergeSort(arr, 0, size-1);
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
}
```