The members of the project:

1-Joseph Emad Fawzy Shafik

Id: 12400353

Class: 11

2-Ziad Ahmed Ibrahim

Id: 12400336

Class: 11

**The project is about this idea:**

**Find smallest number of two arrays**

You are given two unsorted arrays of integers a and b of the same sizes, sort them using two different sorting (Merge Sort & Quick Sort) algorithms then put the smallest number in the same index in third array c (Note: The numbers should be different in array a and b). Then Search about number in array c and output the number is found in index i in array a or b. Print not Found if the number is not exist in both arrays. Then calculate Big (O) of your program.

So the code that we have developed by java is this:

import java.util.Scanner;

public class SmallestNumberArrays {

    // Merge Sort

    public static void mergeSort(int[] arr, int l, int r) {

        if (l < r) {

            int m = (l + r) / 2;

            mergeSort(arr, l, m);

            mergeSort(arr, m + 1, r);

            merge(arr, l, m, 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;

        int k = l;

        while (i < n1 && j < n2) {

            if (L[i] <= R[j]) {

                arr[k] = L[i];

                i++;

            } else {

                arr[k] = R[j];

                j++;

            }

            k++;

        }

        while (i < n1) {

            arr[k] = L[i];

            i++;

            k++;

        }

        while (j < n2) {

            arr[k] = R[j];

            j++;

            k++;

        }

    }

    // Quick Sort

    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 iBegin, int jEnd) {

    int i = iBegin;

    int j = jEnd;

    int pivLoc = i;

    while (true) {

        while (arr[pivLoc] <= arr[j] && pivLoc != j) {

            j--;

        }

        if (pivLoc == j) {

            break;

        } else if (arr[pivLoc] > arr[j]) {

            // Swap arr[j] and arr[pivLoc]

            int temp = arr[j];

            arr[j] = arr[pivLoc];

            arr[pivLoc] = temp;

            pivLoc = j;

        }

        while (arr[pivLoc] >= arr[i] && pivLoc != i) {

            i++;

        }

        if (pivLoc == i) {

            break;

        } else if (arr[pivLoc] < arr[i]) {

            // Swap arr[i] and arr[pivLoc]

            int temp = arr[i];

            arr[i] = arr[pivLoc];

            arr[pivLoc] = temp;

            pivLoc = i;

        }

    }

    return pivLoc;

}

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        // Input

        System.out.print("Enter the size of arrays: ");

        int n = sc.nextInt();

        int[] a = new int[n];

        int[] b = new int[n];

        int[] c = new int[n];

        System.out.println("Enter elements of array a:");

        for (int i = 0; i < n; i++) {

            a[i] = sc.nextInt();

        }

        System.out.println("Enter elements of array b:");

        for (int i = 0; i < n; i++) {

            b[i] = sc.nextInt();

        }

        // Sort arrays

        mergeSort(a, 0, n - 1);

        quickSort(b, 0, n - 1);

        // Build array c with smallest elements

        for (int i = 0; i < n; i++) {

            c[i] = Math.min(a[i], b[i]);

        }

        // Output array c

        System.out.print("Array c: ");

        for (int i = 0; i < n; i++) {

            System.out.print(c[i] + " ");

        }

        System.out.println();

        // Search

        System.out.print("Enter number to search for: ");

        int searchNum = sc.nextInt();

        boolean found = false;

        // Search in array a

        for (int i = 0; i < n; i++) {

            if (a[i] == searchNum) {

                System.out.println("Found at index " + i + " in array a");

                found = true;

                break;

            }

        }

        // Search in array b if not found in a

        if (!found) {

            for (int i = 0; i < n; i++) {

                if (b[i] == searchNum) {

                    System.out.println("Found at index " + i + " in array b");

                    found = true;

                    break;

                }

            }

        }

        // If not found

        if (!found) {

            System.out.println("Not Found");

        }

        // Time Complexity Information

        System.out.println("\nTime Complexities:");

        System.out.println("Merge Sort on a: O(n log n)");

        System.out.println("Quick Sort on b: O(n log n) (average case)");

        System.out.println("Building c array: O(n)");

        System.out.println("Searching: O(n)");

        sc.close();

    }

}