

ASSIGNMENT 6 –QUICK SORT

```
import java.util.*; // Import Scanner for input and Random for random pivot selection

public class QuickSortComparison {

    // Step counters for deterministic and randomized quick sort
    static int stepDet = 0; // Counts steps in deterministic quick sort
    static int stepRand = 0; // Counts steps in randomized quick sort

    // Swap function to exchange two elements in an array
    static void swap(int[] arr, int i, int j) {
        int temp = arr[i]; // Store one value temporarily
        arr[i] = arr[j]; // Replace first value with second
        arr[j] = temp; // Replace second with stored value
    }

    // ----- Deterministic Quick Sort -----

    // Partition function using the last element as pivot (deterministic)
    static int partitionDet(int[] arr, int low, int high) {
        int pivot = arr[high]; // Choose pivot as last element
        int i = low - 1; // Pointer for smaller element

        // Traverse array to rearrange elements
        for (int j = low; j < high; j++) {
            stepDet++; // Count comparison step
            if (arr[j] < pivot) { // If element smaller than pivot
                swap(arr, ++i, j); // Swap it to the left side
                stepDet++; // Count swap step
            }
        }

        swap(arr, i + 1, high); // Move pivot to correct position
        stepDet++; // Count swap step
        return i + 1; // Return partition index
    }

    // Recursive deterministic quick sort function
    static void quickSortDet(int[] arr, int low, int high) {
        stepDet++; // Count recursion call
        if (low < high) { // Base condition
            int pi = partitionDet(arr, low, high); // Partition array
            quickSortDet(arr, low, pi - 1); // Sort left part
            quickSortDet(arr, pi + 1, high); // Sort right part
        }
    }

    // ----- Randomized Quick Sort -----

    // Partition function (same as deterministic but counted separately)
```

```

static int partitionRand(int[] arr, int low, int high) {
    int pivot = arr[high]; // Pivot as last element after random swap
    int i = low - 1; // Pointer for smaller element

    // Rearrange elements
    for (int j = low; j < high; j++) {
        stepRand++; // Count comparison
        if (arr[j] < pivot) {
            swap(arr, ++i, j); // Swap elements
            stepRand++; // Count swap step
        }
    }

    swap(arr, i + 1, high); // Move pivot to its correct place
    stepRand++; // Count swap step
    return i + 1; // Return partition index
}

// Function to select random pivot before partitioning
static int randomPartition(int[] arr, int low, int high) {
    Random rand = new Random(); // Create random object
    int randIndex = low + rand.nextInt(high - low + 1); // Choose random pivot index
    swap(arr, randIndex, high); // Swap random pivot with last element
    stepRand++; // Count swap step
    return partitionRand(arr, low, high); // Continue with partitioning
}

// Recursive randomized quick sort function
static void quickSortRand(int[] arr, int low, int high) {
    stepRand++; // Count recursion call
    if (low < high) { // Base condition
        int pi = randomPartition(arr, low, high); // Get partition index using random pivot
        quickSortRand(arr, low, pi - 1); // Sort left side
        quickSortRand(arr, pi + 1, high); // Sort right side
    }
}

// ----- Utility Function -----

// Function to print elements of array
static void printArray(int[] arr) {
    for (int value : arr)
        System.out.print(value + " "); // Print each element
    System.out.println(); // Move to new line
}

// ----- Main Function -----
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in); // Create Scanner for user input

    // Step 1: Input number of elements

```

```

System.out.print("Enter number of elements: ");
int n = sc.nextInt();

// Step 2: Create and input array
int[] arr = new int[n];
System.out.print("Enter " + n + " elements: ");
for (int i = 0; i < n; i++)
    arr[i] = sc.nextInt(); // Read each element

// Step 3: Copy array for both sorting methods
int[] detArr = arr.clone(); // Clone for deterministic sort
int[] randArr = arr.clone(); // Clone for randomized sort

// Step 4: Perform deterministic quick sort
quickSortDet(detArr, 0, n - 1);

// Step 5: Perform randomized quick sort
quickSortRand(randArr, 0, n - 1);

// Step 6: Display deterministic quick sort result
System.out.println("\nSorted Array (Deterministic Quick Sort): ");
printArray(detArr);
System.out.println("Total Steps: " + stepDet);

// Step 7: Display randomized quick sort result
System.out.println("\nSorted Array (Randomized Quick Sort): ");
printArray(randArr);
System.out.println("Total Steps: " + stepRand);

sc.close(); // Close Scanner to prevent resource leak
}
}

```

OUTPUT:

Enter number of elements: 4

Enter 4 elements: 7 2 5 1

Sorted Array (Deterministic Quick Sort):

1 2 5 7

Total Steps: 19

Sorted Array (Randomized Quick Sort):

1 2 5 7

Total Steps: 16

==== Code Execution Successful ===