W7 - Data Structure & Algorithms

Problem 1: Sorting Large Arrays with Merge Sort and Quick Sort

Overview: Implement Merge Sort and Quick Sort algorithms to sort an array of 1,000,000 random integers. Compare the running times of both algorithms. Discuss the practicality of using Selection Sort or Bubble Sort for sorting an array of this size.

Merge Sort Algorithm:

- **Divide and Conquer Approach**: Recursively divide the array into halves, sort each half, and then merge them.
- **Stable and Efficient for Large Data Sets**: Particularly efficient for sorting large arrays.

Quick Sort Algorithm:

- **Divide and Conquer with a Pivot**: Choose a pivot element and partition the array around the pivot. Recursively apply the same logic to the partitions.
- **Generally Faster, but Less Stable**: Often faster than Merge Sort, but can degrade to O(n^2) in the worst case.

Algorithm Implementations in Java:

1. **Merge Sort**:

```
public class MergeSort {

    public static void sort(int[] array) {
        if (array.length < 2) {
            return;
        }
        int mid = array.length / 2;
        int[] left = new int[mid];
        int[] right = new int[array.length - mid];

        System.arraycopy(array, 0, left, 0, mid);
        System.arraycopy(array, mid, right, 0, array.length - mid);

        sort(left);
        sort(right);

        merge(array, left, right);
    }

    private static void merge(int[] array, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) {
                array[k++] = left[i++];
            } else {
                array[k++] = right[j++];
            }
        }
        while (i < left.length) {
            array[k++] = left[i++];
        }
        while (j < right.length) {
            array[k++] = right[j++];
        }
    }
}
```

2. **Quick Sort**:

```
public class QuickSort {

   public static void sort(int[] array) {
       quickSort(array, 0, array.length - 1);
   }

   private static void quickSort(int[] array, int start, int end) {
       if (start < end) {
           int partitionIndex = partition(array, start, end);
           quickSort(array, start, partitionIndex - 1);
           quickSort(array, partitionIndex + 1, end);
       }
   }

   private static int partition(int[] array, int start, int end) {
       int pivot = array[end];
       int i = (start - 1);

       for (int j = start; j < end; j++) {
           if (array[j] <= pivot) {
               i++;
               int temp = array[i];
               array[i] = array[j];
               array[j] = temp;
           }
       }

       int temp = array[i + 1];
       array[i + 1] = array[end];
       array[end] = temp;
       return i + 1;
   }
}
```

3. **Generating and Sorting the Array**:
    - **Generate Random Array**: Create an array of 1,000,000 random integers.
    - **Sort Using Both Algorithms**: Use Merge Sort and Quick Sort to sort the array.
    - **Measure Running Times**: Compare the time taken by each algorithm to sort the array.

```
import java.util.Random;

public class SortingComparison {
   public static void main(String[] args) {
       int[] array = new int[1000000];
       Random random = new Random();
       // Fill the array with random integers
       for (int i = 0; i < array.length; i++) {
           array[i] = random.nextInt(Integer.MAX_VALUE);
       }
       // Copy the array for fair comparison
       int[] arrayForMergeSort = array.clone();
       int[] arrayForQuickSort = array.clone();
       // Measure time for Merge Sort
       long startTime = System.currentTimeMillis();
       MergeSort.sort(arrayForMergeSort);
       long endTime = System.currentTimeMillis();
       System.out.println("Merge Sort Time: " + (endTime - startTime) + " ms");
```

```
        // Measure time for Quick Sort
        startTime = System.currentTimeMillis();
        QuickSort.sort(arrayForQuickSort);
        endTime = System.currentTimeMillis();
        System.out.println("Quick Sort Time: " + (endTime - startTime) + " ms");
    }
}
```

Problem 2

↳ 2 cells hidden