PARALLEL ALGORITHMS
ASSIGNMENT 3

Submitted by TEAM 7 (Rama Charan Pavan, Sri Charan, Prataprao)

1.

1) Divide the unsorted array into n partitions, each partition contains 1 element. Here the one element is considered as sorted.
2) Repeatedly merge partitioned units to produce new sublists until there is only 1 sublist remaining. This will be the sorted list at the end.
3) Merge sort is a fast, stable sorting routine with guaranteed O(n*log(n)) efficiency. When sorting arrays, merge sort requires additional scratch space proportional to the size of the input array. Merge sort is relatively simple to code and offers performance typically only slightly below that of quicksort.

Sequential Merge-sort code: Time complexity : O(nlogn)

```
public class MergeSorting {

   private int[] array;
   private int[] tempMergArr;
   private int length;

   public static void main(String a[]){

      int[] inputArr = {45,23,11,89,77,98,4,28,65,43};
      MergeSorting ms = new MergeSorting ();
      ms.sort(inputArr);
      for(int i:inputArr){
         System.out.print(i);
         System.out.print(" ");
      }
   }

   public void sort(int inputArr[]) {
      this.array = inputArr;
      this.length = inputArr.length;
      this.tempMergArr = new int[length];
      doMergeSort(0, length - 1);
   }

   private void doMergeSort(int lowerIndex, int higherIndex) {

      if (lowerIndex < higherIndex) {
         int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
```

```
            doMergeSort(lowerIndex, middle);
            doMergeSort(middle + 1, higherIndex);
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }

    private void mergeParts(int lowerIndex, int middle, int higherIndex) {

        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] <= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }

    }
}
```

Output:

4 11 23 28 43 45 65 77 89 98

Pseudo Code for Parallel Merge-Sort code: time complexity: O(nlogn)

```
Take Method odd_even_merge(a,b) =
if (a > 1)
then
    b = {odd_even_merge(a,b)
      : a in [odd_elts(a),even_elts(a)]
      ; b in [odd_elts(b),even_elts(b)]};
      odd, even = b[0],b[1];
   in take (even,1) ++
     flatten({[min(o,e),max(o,e)]
        : o in drop(odd,-1)
           ; e in drop(even,1)}) ++
     Take(odd,-1)
else
   if (a[0] < b[0]) then a++b else b++a;
```

Output:

Input Given ([2,8,9,11,12,18,22,22]

==➔ [1,5,6,11,13,14,19,21]

2. Comparision with other Sorting Techniques with **Merge Sort**:

1. **Insertion Sort**:

Sequential: O (n log n) for Merge Sort and O (n^2) for Insertion Sort.

Parallel: O (log n) for Odd Even Merge and O (n log n) for Pipeline Insertion Sort.

2. **Selection Sort**:

Sequential: O (n log n) for Merge Sort and O (n^2) for Selection Sort.

Parallel: O (log n) for Odd Even Merge and O (n^2) for Selection Sort.

3. **Quick Sort**:

Sequential: O (n log n) for Merge Sort and O (n log n) for Quick Sort.

Parallel: O (log n) for Odd Even Merge and O (log n) for Quick Sort.

4. **Bubble Sort:**

Sequential: O (n log n) for Merge Sort and O (n^2) for Bubble Sort.

Parallel: O (log n) for Odd Even Merge and O (n log n) for Bubble Sort.

5. **Tree Sort:**

Sequential: O (n log n) for Merge Sort and O (n) for Tree Sort.

Parallel: O (log n) for Odd Even Merge and O (log n) for Tree Sort.

6. **Counting Sort:**

Sequential: O (n log n) for Merge Sort and O (n^2) for counting Sort.

Parallel: O (log n) for Odd Even Merge and O (n^2) for Counting Sort.

By the following comparision of Time complexity above we can say that Merge Sort is best when compared with other sorting techniques.