UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

NAAC GRADE A+ ACCREDITED UNIVERSITY

# Worksheet 2

**Student Name:** Rahul Saxena **UID:** 24MCI10204
**Branch:** MCA AI/ML **Section/Group:** 24MAM3/B
**Semester:** FIRST **Date of Performance:** 12/08/2024
**Subject Name** DESIGN & ANALYSIS OF ALGORITHMS LAB
**Subject Code:** 24CAP-612

---

**Q. Using Open, implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

1. **Aim/Overview of the practical:**

   To implement the basic Merge Sort algorithm, measure the time taken to sort arrays of different sizes, and plot a graph for the time taken versus the number of elements.

2. **Task to be done:**
   - Implement a basic Merge Sort algorithm.
   - Measure the time taken for sorting arrays of different sizes.
   - Plot the time taken versus the number of elements in Python.

3. **Algorithm/Flowchart:**
   - Let arr[] be the array to be sorted. The function mergeSort(arr, l, r) sorts the array from index l to r.
   - **Merge Sort**:
     - If l >= r: The array has one or no elements, so it is already sorted.
     - Else:
       - Calculate mid = (l + r) / 2.
       - Recursively call mergeSort(arr, l, mid) to sort the left half.
       - Recursively call mergeSort(arr, mid + 1, r) to sort the right half.
       - Call merge(arr, l, mid, r) to merge the two sorted halves.
   - **Merge Function**:
     - Take two sub-arrays: one from index l to mid and another from mid + 1 to r.
     - Compare the elements of both sub-arrays and place the smaller element into the main array.

- Continue merging until all elements from both sub-arrays are placed back in the main array.

**4. Code for experiment/practical:**

```
import java.util.Random;

class SimpleMergeSort {
    public static void merge(int[] arr, int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;
        int[] left = new int[n1];
        int[] right = new int[n2];
        for (int i = 0; i < n1; ++i)
            left[i] = arr[l + i];
        for (int i = 0; i < n2; ++i)
            right[i] = arr[m + 1 + i];
        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (left[i] <= right[j]) {
                arr[k] = left[i];
                i++;
            } else {
                arr[k] = right[j];
                j++;
            }
            k++;
        }
        while (i < n1) {
            arr[k] = left[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = right[j];
            j++;
            k++;
        }
    }
    // Recursive Merge Sort
    public static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;

            // Recursively sort the two halves
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }
```

```
public static long timeSort(int[] arr) {
    long startTime = System.nanoTime();
    mergeSort(arr, 0, arr.length - 1);
    long endTime = System.nanoTime();
    return endTime - startTime;
}
public static void main(String[] args) {
    Random random = new Random();
    int[] sizes = {10000, 20000, 50000, 100000, 200000};

    // Execute sorting for different sizes and print the time taken
    for (int n : sizes) {
        int[] arr = random.ints(n, 0, 1000000).toArray();
        long timeTaken = timeSort(arr);
        System.out.println("Array Size: " + n + " Time Taken: " + timeTaken / 1e6 + " ms");
    }
}
```

**Python Code for Graph**

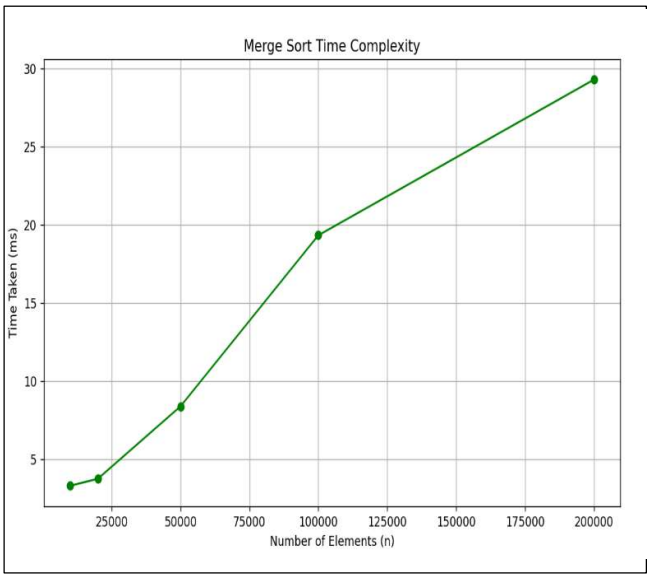
```
import matplotlib.pyplot as plt

# Array sizes
sizes = [10000, 20000, 50000, 100000, 200000]
times = [3.3237, 3.7634, 8.3947, 19.3282, 29.2751]
plt.figure(figsize=(10, 6))
plt.plot(sizes, times, marker='o', linestyle='-', color='g')
# Adding labels and title
plt.xlabel('Number of Elements (n)')
plt.ylabel('Time Taken (ms)')
plt.title('Time Taken for Merge Sort vs Number of Elements')
# Display the graph
plt.grid(True)
plt.show()
```

## 5. Result/Output/Writing Summary:

```
Array Size: 10000 Time Taken: 3.3237 ms
Array Size: 20000 Time Taken: 3.7634 ms
Array Size: 50000 Time Taken: 8.3947 ms
Array Size: 100000 Time Taken: 19.3282 ms
Array Size: 200000 Time Taken: 29.2751 ms

Process finished with exit code 0
```



**Learning outcomes (What I have learnt):**

1. **Working of Quick sort.**
2. **Implementation of Quick sort using python.**
3. **Graph plotting using matplotlib library.**

**Evaluation Grid:**

| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
|---|---|---|---|
| 1. | Demonstration and Performance (Pre Lab Quiz) | | 5 |
| 2. | Worksheet | | 10 |
| 3. | Post Lab Quiz | | 5 |