PROJECT 2

GANTA PRIYA

1. Problem Statement

Merging Sorted Lists
See textbook, Section 5.3

You are given an array a[] of numbers, where a[i] is the size of the i-th list to merge. You have to produce the sequence in which to merge the lists, and the total cost of merging all the lists. ***Implementation Notes: Use a heap data structure. (You don't have to implement your own heal structure, you can simply use the inbuilt one in Java/C#.)***

Write-up explaining key implementation characteristics (EXAMPLE)

Assume we possess three lists in the following sizes: 5M, 10M, and 15M. Let's examine two potential merge sequences

1. If we merge the lists of sizes 10M and 15M first, This merger requires 10M +15M -1, that is, approximately 25M comparisons in the worst case. Now we have two lists of sizes 5M and 25M merging them gives 30M approximately. So the total comparisons are 25M from the first merge and 30M from the second merge so in total 25M + 30M= 55M. So the total cost is 55M
2. If we merge the lists of sizes 5M and 10M first, This merger requires 5M +10M -1, that is, approximately 15M comparisons in the worst case. Now we have two lists of sizes 15M and 15M merging them gives 30M approximately. So the total comparisons are 15M from the first merge and 30M from the second merge so in total 15M + 30M= 45M . So the total cost here is 45M

We note that the order in which the mergers occur materially affects the quantity of comparisons needed to combine. By selecting the ideal merging order of 5M and 10M, which resulted in 45M comparisons overall as opposed to 55M in the non-optimal sequence. So merging two smallest lists first gave us optimal solution 45M which can be obtained by using Greedy Algorithm .This illustrates how important it is to merge lists in the right order in order to reduce total cost.

ALGORITHM USED

We use Greedy Algorithm as the the sum of the cost of merging two lists of sizes, a and b, is a+b. This expense is incurred each time a merge is completed. So here, the greedy strategy always combines the two smallest lists first. You reduce the cost of each merge step and, as a result, the total cost of merging all lists by doing this. So this greedy method is the best choice for obtaining optimal solutions.

2. Theoretical Analysis

Application of greedy algorithm in the code

1. Choosing the two smallest lists: Using a min-heap (priority queue), the greedy method is carried out.The heap allows us to efficiently retrieve the two smallest lists at any given step. By using a min-heap, the smallest list is always at the top of the heap, and the second smallest list can be retrieved by removing the smallest one.

2. The smallest lists are combined first: At the start of each loop iteration, the two shortest lists are removed (polled) from the heap. This ensures that the lists that will be combined will be the most affordable ones at that specific moment.

3. Minimizing the Total Merge Cost: After merging the two smallest lists, their sizes are added to the running total cost and the merged list is reinserted into the heap. This process continues until all lists are merged, ensuring the overall cost is minimized by always merging the smallest lists first.

Outer loop: The loop runs $n - 1$ times where n is the number of lists, because we are reducing the number of elements in the heap by 1 after each merge. So its runs for $O(n)$ times.
Inner loop (heap operations): Extract min (poll()): This operation takes $O(log\, n)$ time, as removing the smallest element from a heap involves a log time operation. Insert back (add): This also takes $O(log\, n)$, as inserting an element into a heap maintains its logarithmic height. Each iteration performs two extract operations and one insert operation, each taking $O(log\, n)$, so each iteration takes $O(log\, n)$

Total Time Complexity: Since the outer loop runs $n - 1$ times and each iteration involves $O(log\, n)$ operations, the total time complexity is: $O(n\, log\, n)$

Mathematical Expression

For the outer loop : $\sum_{i=1}^{n-1} 1 = n - 1$ (This is essentially $O(n)$)

For the inner loop : Inner loop performs operations in $O(\, log\, n)$ the total cost across all iterations can be expressed as $T_{inner\ total} = \sum_{i=1}^{n-1}$ $O(\, log\, n) = (n - 1)\, .\, log\, n\ = O(n\, log\, n)$

The total time complexity : $T(n)\ = \sum_{i=1}^{n-1} O\ (\, log\, n) = O(\, n\, log\, n)$

3. Experimental Analysis

3.1 Program Listing

```
for (int size : listSizes) {
  minHeap.add(size);
```

```
        }
        int totalCost = 0;
        while (minHeap.size() > 1) {
            int first = minHeap.poll();
            int second = minHeap.poll();
            int mergedSize = first + second;
            minHeap.add(mergedSize);
            totalCost += mergedSize;
        }
        return totalCost;
    }
    public static void main(String[] args) {
        int[] arraySizes = {100, 1000, 10000, 100000, 1000000};
            System.out.println("Total cost of merging " + n + " lists: " + totalCost + "M");
            System.out.println("Execution time for " + n + " lists: " + duration + " nanoseconds");
        }
```

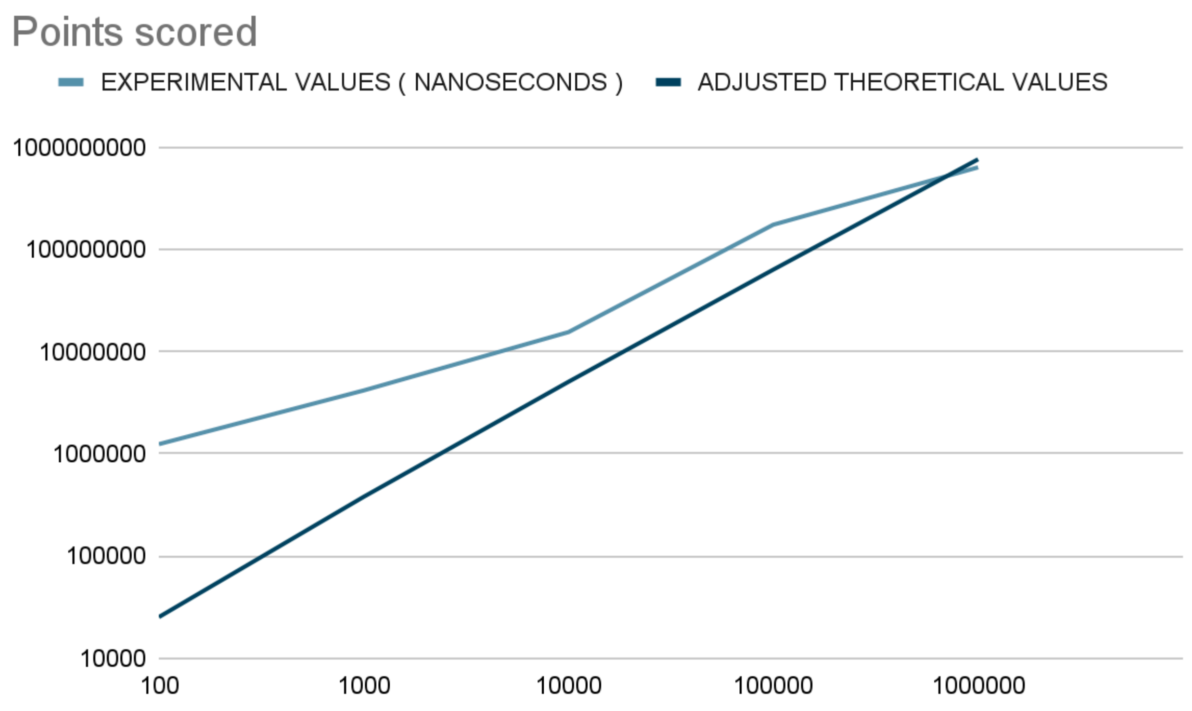
3.2 Data Normalization Notes

Since there are no measurement units in theoretical analysis, we derive a constant called the scaling constant. By dividing the mean of the experimental result by the mean of the theoretical result, one can obtain this scaling constant.The adjusted theoretical values are equal to the scaling constant times the theoretical values, and the constant scaling is 38.21044461

3.3 Output Numerical Data

n	EXPERIMENTAL VALUES (NANOSECONDS)	THEORETICAL VALUES	SCALING CONSTANT	ADJUSTED THEORETICAL VALUES
100	1246279	664.39		25386.63729
1000	4177738	9965.78		380796.8847
10000	15536524	132877.12		5077293.834
100000	174575704	1660964.05		63466174.83
1000000	635007504	19931568.57		761594096.8
	830543749	21736039.91	38.21044461	

3.4 Graph



3.5 Graph Observations

The graph shows that the adjusted theoretical values are steadily growing, and experimental values are intersecting them at n values between 10^5 and 10^6.

4 Conclusion

The greedy algorithm effectively minimizes merge costs by merging the smallest lists first. The time complexity of $O(n \log n)$ demonstrates efficiency, even with larger datasets. Execution times confirmed the logarithmic growth of runtime relative to input size.

Github link:- https://github.com/gantapriya55/Merging-sorted-lists/upload