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 shortest lists are combined first: This is so because, during each iteration of the loop, it removes the two shortest lists from the heap. This way, it is ensured that the lists which will be merged shall be at that particular time the cheapest.

3. Minimum Total Merge Cost: The heap replaces the two smallest lists with the sum of their sizes added to the running total cost, and the merged list is reinserted into the heap. This continues till all lists are merged, ensuring an overall minimum cost since the smallest lists will always be chosen for a merge.

Outer loop: No. of times the loop runs is $n - 1$, where n = no. of lists. This is because after every merge we decrease the heap size by 1. Thus, it runs for $O(n)$ times.

Inner loop (heap operations): Extract min, poll(): It takes $O(log\, n)$ because removing the smallest element from a heap is a log-time operation. Insert back (add): This also takes $O(log\, n)$, since upon insertion of an element in a heap, it keeps it at its logarithmic height. So every iteration, there are two extract operations and one insert operation, each taking $O(log\, n)$; therefore, every iteration takes $O(log\, n)$

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

```
public class MergeSequenceTest {
```

```
 public static int mergeSequence(int[] sizes) {

       PriorityQueue<Integer> heap = new PriorityQueue<>();

       for (int size : sizes) {
         heap.add(size);
}
      public static void main(String[] args) {
int[] testSizes = {100, 1000, 10000, 100000, 1000000};

for (int n : testSizes) {
    int[] listSizes = generateRandomArray(n);
    System.out.println("n = " + n + ": Total merge cost = " + totalCost + ", Time = " + timeInSeconds + " seconds");
}
```

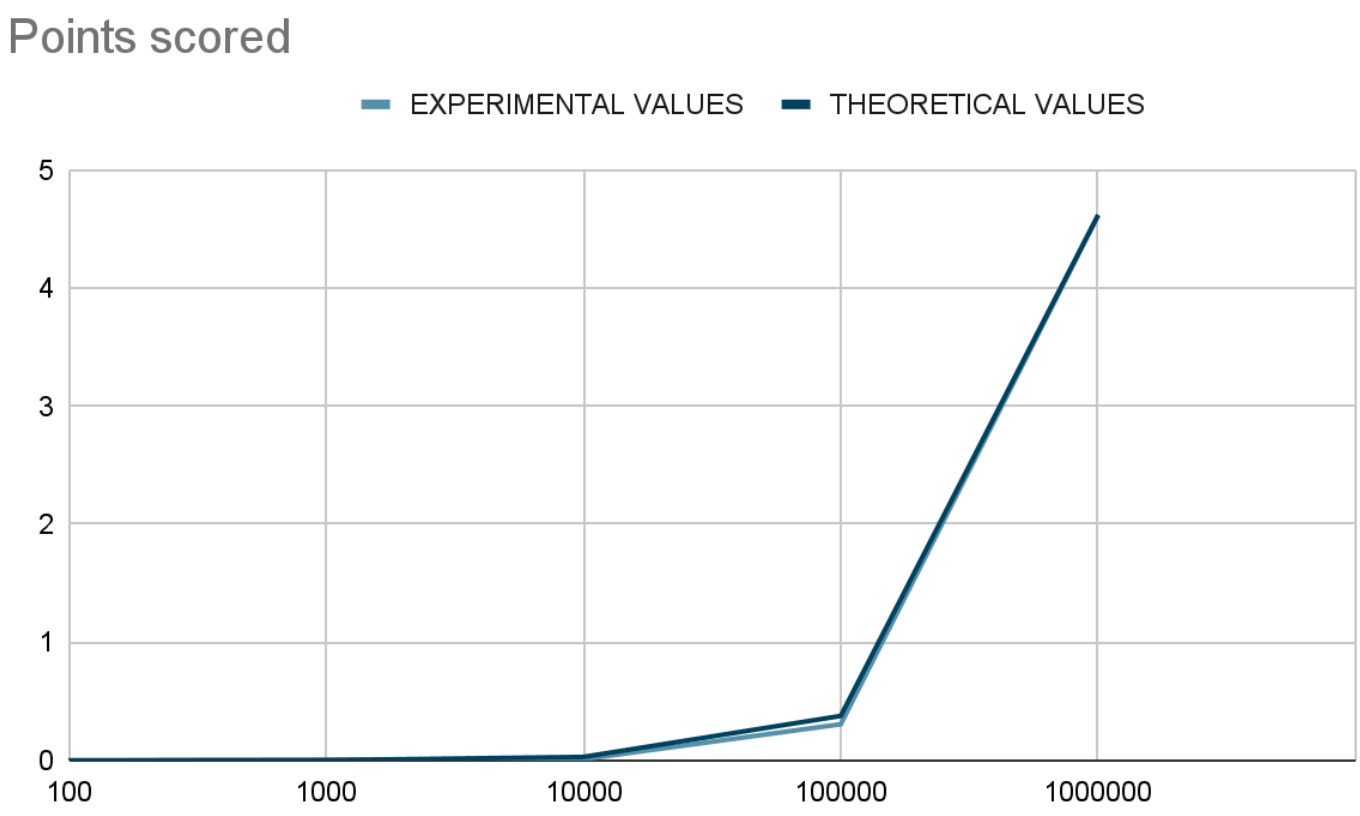
3.2 Data Normalization Notes

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 0.00000022727231

3.3 Output Numerical Data

n	EXPERIMENTAL VALUES (SECONDS)	THEORETICAL VALUES	SCALING CONSTANT	ADJUSTED THEORETICAL VALUES
100	0.001	664.39		0.00015099745
1000	0.006	9965.78		0.002264945842
10000	0.011	132877.12		0.03019929001
100000	0.307	1660964.05		0.3774911365
1000000	4.615	19931568.57		4.529893635
	0.988	4347207.982	0.00000022727231	

3.4 Graph



3.5 Graph Observations

Both experimental and adjusted theoretical values have the same trend initially when the values are at low in the beginning. However, the adjusted theoretical values increase progressively, and between 10^5 and 10^6, the values go upwards. The experimental values similarly go up, and the graphs of the two functions overlap between 10^5 and 10^6 to show a point of equality.

4. Conclusion

The algorithm significantly reduces the total merge cost by pursuing a greedy strategy of always merging the two smallest lists; its overall time complexity is O(n log n). This efficiency holds remarkably well for larger datasets.