AM5801: Computational Lab

Assignment 4

Date: September 4, 2025

Deadline: September 10, 2025 Max mark: 50

1. A restaurant owner follows a greedy revenue policy: for every three consecutive pending orders, the average revenue must be at least a threshold value x. When a new order arrives, the owner may insert it anywhere in the list of current pending orders (before the first order, between any two orders, or after the last order). If there is at least one valid position such that the new list of orders satisfies the revenue policy, the order is accepted and inserted at the first valid position found. If no such position exists, the order is rejected.

   For example, if n = 4 and the orders are [10.0, 8.25, 9.75, 5.0] with x = 7.5 and the new order revenue r = 5.5, then inserting 5.5 at the end fails the revenue condition, but inserting at position 1 produces [10.0, 5.5, 8.25, 9.75, 5.0], which satisfies the requirement, so the order is accepted at that position. On the other hand, if n = 3 and the orders are [6.0, 7.0, 6.5] with x = 7.0 and the new order revenue r = 5.0, then no insertion position satisfies the requirement, so the order is rejected.

   (a) Write a program to implement this greedy order insertion strategy. The program should first read the number of pending orders n, the revenue values of the current n pending orders, the target threshold value x, and the revenue value r of the new order, and then display the list. **(5)**

   (b) Next, the program should check all possible insertion positions, test the average revenue condition for every three consecutive orders, and accept the order at the first valid position if possible, otherwise reject it. **(10)**

   (c) Finally, the program should display the updated list of orders if the new order is accepted, or the message "Order rejected" if no valid position exists. Along with this, provide a short explanation either in comments or separately on why the method represents a greedy approach. **(10)**

2. The Fibonacci sequence is defined as

$$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2), \quad \text{for all } n \geq 2.$$

   A naive recursive algorithm for computing $F(n)$ leads to exponential time complexity because the same subproblems are recomputed many times. Dynamic programming avoids this inefficiency by storing the results of subproblems and reusing them. Write a program to calculate how many recursive calls (or additions) would have been required by the naive recursive method for the same n, and compare this with the actual number of computations performed using Dynamic programming. Comment on the efficient improvement achieved, if any. **(10)**

3. An inversion in a list is defined as a pair of elements (a_i, a_j) such that $i < j$ but $a_i > a_j$. For example, in the list $[2, 4, 1, 3, 5]$, the inversions are $(2, 1)$, $(4, 1)$, and $(4, 3)$, giving a total of 3 inversions.

   A brute-force method would require checking all pairs of elements, which takes $O(n^2)$ time. However, merge sort can be modified to count inversions efficiently in $O(n \log n)$ time by integrating the inversion count into the merge step of the divide-and-conquer process.

   Write a program that uses a merge sort based approach to count the total number of inversions in a list. The program should first read the number of elements n and the list of n integers, and then display the list before sorting. It should then extend the recursive divide-and-conquer structure of merge sort so that during the merging process, it also counts the number of inversions encountered when elements from the right sublist are placed before elements in the left sublist. Finally, the program should display both the sorted list and the total inversion count, along with a short explanation either in comments or separately on why the inversion counting method exemplifies the power of divide-and-conquer. **(15)**