QF633 C++ for Financial Engineering

Fernando Oktavianes

01474014

1. **Approach 1: Add and then sort.**
   - ***Time Complexity:***
     - *OrderManager::Add*

       The overall time complexity for OrderManager::Add is O(N log N). The reason for this is because the order_.push_back() is O(1). Then, the std::stable_sort is O(N log N). So, the overall time complexity is O(1) + O(N log N) which results in O(N log N). I provided the reason why I am using std::stable_sort instead of std::sort in rational section.
     - *OrderManager::Update*

       The overall time complexity for OrderManager::Update is O(N). The reason for this is because the std::find_if will iterate the elements inside the vector which indicate the time complexity is O(N). Then, the order_.erase is O(1) because we just changing one value with known index inside the vector. So, the overall time complexity is O(N) + O(1) which results in O(N).
   - ***Rationale to maintain desired ordering:***

     If we just implement std::sort in Add section, the vector will be sorted in ascending manner, but this method will have some issue with handling duplicate because the order of duplicate elements might change. In order to avoid this issue, I implemented std::stable_sort in my code to ensure the stability and priority of order in the order book. I also implemented lambda function to determine the sorting rules. The logic here is deciding the sorting based on the price and if different orders have same price, there will be additional step to sort based on the order_id. In case of Update section, the ordering is already maintained due to the arrangement from Add. So, there is no need to concern about the ordering here.

2. **Approach 2: Add while preserving the ordering.**

- ***Time Complexity:***
  - *OrderManager::Add*

    The overall time complexity for OrderManager::Add is O(N). The reason for this is because the time complexity for std::binary_search is O(log N). Then, the std::lower_bound is also O(log N). Lastly, order_.insert() is O(N). So, the overall time complexity is O(log N) + O(log N) + O(N) which results in O(N).
  - *OrderManager::Update*

    The overall time complexity for OrderManager::Update is O(N). The reason for this is because the std::find_if will iterate the elements inside the vector which indicate the time complexity is O(N). Then, the order_.erase is O(1) because we just changing one value with known index inside the vector. So, the overall time complexity is O(N) + O(1) which results in O(N).
- ***Rationale to maintain desired ordering:***

  I implemented std::binary_search as an error handling to handle the case of orders with same order_id which is not possible to happen. In the case this happen, this means that the algorithm interpret same order twice and we need to adjust our algorithm to handle this case. In order to maintain the ordering in Add section, I implemented a lambda function inside the std::lower_bound in my code to ensure the stability and priority of order in the order book. I also implemented lambda function to determine the sorting rules. The logic here is deciding the sorting based on the price and if different orders have same price, there will be additional step to sort based on the order_id. In case of Update section, the ordering is already maintained due to the arrangement from Add. So, there is no need to concern about the ordering here.