UM-SJTU Joint Institute

VE281 Data Structure and Algorithm

Project 2 Report

Tianze Wang, 515370910202

Contents

I Introduction 1

II Appendix 1

I Introduction

We've implemented different data structure to store our stock, client and order.

II Appendix

Please refer to this part for detailed function.

```
#include <iostream>
#include <queue>
#include <getopt.h>
#include <sstream>
#include <set>
#include <map>


using namespace std;

enum BUY_OR_SELL {
    BUY, SELL
};

struct order {
    int TIMESTAMP;
    int id;
    string CLIENT_NAME;
    BUY_OR_SELL buy_or_sell;
    string EQUITY_SYMBOL;
    int PRICE;
    int QUANTITY;
    int DURATION;
    bool isdone = false;
};

struct order_compare {
    bool operator()(order &a, order &b) {
        if (a.TIMESTAMP == b.TIMESTAMP) return a.id < b.id;
        else return a.TIMESTAMP < b.TIMESTAMP;
    }
};

struct compare_buy_order {
    bool operator()(order &a, order &b) {
        if (a.PRICE == b.PRICE) return a.id < b.id;
        else return a.PRICE > b.PRICE;
    }

    bool operator()(const order &a, const order &b) {
        if (a.PRICE == b.PRICE) return a.id < b.id;
        else return a.PRICE > b.PRICE;
    }
};

struct compare_sell_order {
    bool operator()(order &a, order &b) {
        if (a.PRICE == b.PRICE) return a.id < b.id;
        else return a.PRICE < b.PRICE;
    }

    bool operator()(const order &a, const order &b) {
```

```
        if (a.PRICE == b.PRICE) return a.id < b.id;
        else return a.PRICE < b.PRICE;
    }
};

struct time_traveler_order {
    string equity_name;
    int buy_price = 0;
    int sell_price = 0;
    int buy_time = -1;
    int sell_time = -1;
    int ttt_id = 0;
};

struct compare_ttt {
    bool operator()(time_traveler_order &a, time_traveler_order &b) const {
        return a.ttt_id < b.ttt_id;
    }

    bool operator()(const time_traveler_order &a, const time_traveler_order &b) const {
        return a.ttt_id < b.ttt_id;
    }
};

struct Client {
    string client_name = "";
    int stock_buy = 0;
    int stock_sell = 0;
    int amount_traded = 0;
};

struct compare_Client {
    bool operator()(Client &a, Client &b) const {
        return a.client_name < b.client_name;
    }

    bool operator()(const Client &a, const Client &b) const {
        return a.client_name < b.client_name;
    }
};

struct Equity {
    int id;
    int buy_time;
    int sell_time;
    int buy_price;
    int sell_price;
    string Equity_NAME;
};

//template <typename less = std::less(),
struct Big_Order {
    string EQUITY_SYMBOL;
    set<order, compare_buy_order> Buy;
    set<order, compare_sell_order> Sell;
//  multiset<int> Price_dealt;
    priority_queue<int, vector<int>, std::less<int>> max_queue;
    priority_queue<int, vector<int>, std::greater<int>> min_queue;
};

struct compare_big_order {
    bool operator()(Big_Order &a, Big_Order &b) {
        return a.EQUITY_SYMBOL < b.EQUITY_SYMBOL;
    }
```

```
    bool operator()(const Big_Order &a, const Big_Order &b) {
        return a.EQUITY_SYMBOL < b.EQUITY_SYMBOL;
    }
};


int main(int argc, char *argv[]) {
    bool verbose = false, median = false, midpoint = false, transfers = false, ttt = false;
    set<time_traveler_order, compare_ttt> ttt_set;
    set<time_traveler_order, compare_ttt>::iterator ttt_setit;
    time_traveler_order *ttt_Ptr;
    time_traveler_order ttt_temp;
    int tttid = 0;
    while (1) {
        static struct option long_option[] = {{"median",    no_argument,       NULL, 'm'},
                                              {"verbose",   no_argument,       NULL, 'v'},
                                              {"midpoint",  no_argument,       NULL, 'p'},
                                              {"transfers", no_argument,       NULL, 't'},
                                              {"ttt",       required_argument, NULL, 'g'},
                                              {0, 0, 0,                              0}};
        auto c = getopt_long(argc, argv, "mvptg:", long_option, NULL);
        if (c == -1) break;
        if (c == 'v') {
            verbose = true;
        }
        else if (c == 'm') {
            median = true;
        }
        else if (c == 'p') {
            midpoint = true;
        }
        else if (c == 't') {
            transfers = true;
        }
        else if (c == 'g') {
            ttt = true;
            ttt_temp.ttt_id = tttid;
            ttt_temp.equity_name = optarg;
            ttt_set.insert(ttt_temp);
            tttid++;
        }
    }
    int current_timestamp = 0, id = 0;
    int timestamp;
    string client_name;
    string buy_or_sell1;
    string equity_symbol;
    int price;
    int quantity;
    int duration;
    char note;// used to indicate # and $
    set<order, compare_buy_order> Buy;
    set<order, compare_sell_order> Sell;
//    multiset<order, std::less>::iterator Buy_iter, Sell_iter;
//    multiset<order, order_compare> OrderALL;
//    multiset<order, order_compare>::iterator it;
    set<Big_Order, compare_big_order> OrderAll;
    set<Big_Order, compare_big_order> Order_in;
    set<Big_Order, compare_big_order>::iterator it, it2;
    Big_Order *AllPtr;
    set<order, compare_buy_order>::iterator BuyIt;
    set<order, compare_sell_order>::iterator SellIt;
    set<order, compare_buy_order> *BuyPtr;
    set<order, compare_sell_order> *SellPtr;
    set<int>::iterator Medianitr;
```

```
    set<Client, compare_Client> BigClient;
    set<Client, compare_Client>::iterator BigClientit;
    Client *ClientPtr;
    order *SellOrderPtr;
    order *BuyOrderPtr;
    char c;
    // Market info:
    int Commission_Earnings = 0;
    int Money_Transferred = 0;
    int Number_of_Completed_Trades = 0;
    int Number_of_share = 0;
    // Read
    stringstream ss;
    while (!cin.eof()) {
//        if (c == '\n') break;
//    while ((cin >> timestamp)) {
        string str1;
        getline(cin, str1);
        if (str1.empty()) break;
        ss.clear();
        ss.str(str1);
        order Read_temp;
        ss >> timestamp >> client_name >> buy_or_sell1 >> equity_symbol >> note >> price >> note >>
     quantity
           >> duration;
        Read_temp.id = id;
        Read_temp.TIMESTAMP = timestamp;
        Read_temp.CLIENT_NAME = client_name;
        Read_temp.PRICE = price;
        Read_temp.EQUITY_SYMBOL = equity_symbol;
        Read_temp.QUANTITY = quantity;
        Read_temp.DURATION = duration;
        if (buy_or_sell1 == "BUY") {
            Read_temp.buy_or_sell = BUY;
            Sell.insert(Read_temp);
        }
        else {
            Read_temp.buy_or_sell = SELL;
            Buy.insert(Read_temp);
        }
        id++;
        for (it = OrderAll.begin(); it != OrderAll.end(); it++) {
            AllPtr = const_cast<Big_Order *> (&(*it));
            SellPtr = const_cast<set<order, compare_sell_order> *> (&(it->Sell));
            BuyPtr = const_cast<set<order, compare_buy_order> *> (&(it->Buy));
        }

        // Clients
        bool Clientfound = false;
        for (BigClientit = BigClient.begin(); BigClientit != BigClient.end(); BigClientit++) {
            ClientPtr = const_cast<Client *> (&(*BigClientit));
            if (BigClientit->client_name == Read_temp.CLIENT_NAME) {
                Clientfound = true;
            }
        }
        if (!Clientfound) {
            Client temp_client;
            temp_client.client_name = client_name;
            BigClient.insert(temp_client);
        }

        // ttt tag
        if (ttt) {
            for (ttt_setit = ttt_set.begin(); ttt_setit != ttt_set.end(); ttt_setit++) {
                if (ttt_setit->equity_name == Read_temp.EQUITY_SYMBOL) {
```

```
                    ttt_Ptr = const_cast<time_traveler_order *> (&(*ttt_setit));
                    if (Read_temp.buy_or_sell == SELL && (ttt_Ptr->buy_time == -1 ||
                                                          Read_temp.PRICE < ttt_Ptr->buy_price)) {
                        ttt_Ptr->buy_price = Read_temp.PRICE;
                        ttt_Ptr->buy_time = Read_temp.TIMESTAMP;

                    }
                    else if (Read_temp.buy_or_sell == BUY && ttt_Ptr->buy_time == -1) {
                        break;
                    }
                    else if (Read_temp.buy_or_sell == BUY && (ttt_Ptr->sell_time == -1 ||
                                                              Read_temp.PRICE > ttt_Ptr->sell_price)) {
                        ttt_Ptr->sell_time = Read_temp.TIMESTAMP;
                        ttt_Ptr->sell_price = Read_temp.PRICE;
                    }
                }
            }
        }
        //Output Median and Midpoint
        if (timestamp != current_timestamp) {
            if (median) {
                int median_num;

                for (it = OrderAll.begin(); it != OrderAll.end(); ++it) {
                    /* debug output
                    int iiii = 0;
                    for (auto it_2 = it->Price_dealt.begin(); it_2 != it->Price_dealt.end(); it_2++) {
                        cout << iiii << " times "<< *it_2 << " with timestamp " << timestamp << endl;
                        iiii++;
                    }
                    */
//                    if (it->Price_dealt.size() != 0) {
//                        Medianitr = it->Price_dealt.begin();
//                        if ((it->Price_dealt).size() % 2 == 0) {
//                            for (int i = 0; i < (it->Price_dealt).size() / 2; i++) {
//                                ++Medianitr;
//                            }
////                          median_num = ((*Medianitr) + (*(--Medianitr))) / 2;
//                            median_num = *Medianitr;
//                            median_num += *(--Medianitr);
//                            median_num /= 2;
//                        }
//                        else {
//                            for (int i = 0; i < (it->Price_dealt).size() / 2; i++) {
//                                ++Medianitr;
//                            }
//                            median_num = *Medianitr;
//                        }
//                        cout << "Median match price of " << it->EQUITY_SYMBOL << " at time " <<
    current_timestamp
//                             << " is $" << median_num << endl;
//                    }
                    if (it->min_queue.size() + it->max_queue.size() != 0) {
                        if ((it->min_queue.size()+it->max_queue.size())%2==1) median_num = it->max_queue.
    top();
                        else median_num = (it->max_queue.top() + it->min_queue.top())/2;
                        cout << "Median match price of " << it->EQUITY_SYMBOL << " at time " <<
    current_timestamp
                             << " is $" << median_num << endl;
                    }
                }

            }

            if (midpoint) {
```

```
            int midpoint_num;
            for (it = OrderAll.begin(); it != OrderAll.end(); it++) {
                if (it->Buy.empty() || it->Sell.empty()) {
                    cout << "Midpoint of " << it->EQUITY_SYMBOL << " at time " << current_timestamp
                         << " is undefined" << endl;
                }
                else {
                    midpoint_num = ((*(it->Buy).begin()).PRICE + (*(it->Sell).begin()).PRICE) / 2;
                    cout << "Midpoint of " << it->EQUITY_SYMBOL << " at time " << current_timestamp
<< " is $"
                         << midpoint_num << endl;
                }
            }
        }
    }

    current_timestamp = timestamp;

    //Deal_With_Expired_Order();
    //Erase from temp
    /*
     *
    for (SellIt = Sell.begin(); SellIt != Sell.end(); SellIt++) {
        if (SellIt->DURATION != -1 && SellIt->DURATION + SellIt->TIMESTAMP <= current_timestamp) {
            Sell.erase(SellIt);
        }
    }

    for (BuyIt = Buy.begin(); BuyIt != Buy.end(); BuyIt++) {
        if (BuyIt->DURATION != -1 && BuyIt->DURATION + BuyIt->TIMESTAMP <= current_timestamp) {
            Buy.erase(BuyIt);
        }
    }
    *
    */

    //Erase from OrderAll


    for (it = OrderAll.begin(); it != OrderAll.end(); it++) {
        AllPtr = const_cast<Big_Order *> (&(*it));
        SellPtr = const_cast<set<order, compare_sell_order> *> (&(it->Sell));
        BuyPtr = const_cast<set<order, compare_buy_order> *> (&(it->Buy));
        for (SellIt = SellPtr->begin(); SellIt != SellPtr->end();) {
            if (SellIt->DURATION != -1 && SellIt->DURATION + SellIt->TIMESTAMP <= current_timestamp)
{
                SellIt = SellPtr->erase(SellIt);
            }
            else {
                ++SellIt;
            }
        }
        for (BuyIt = BuyPtr->begin(); BuyIt != BuyPtr->end();) {
            if (BuyIt->DURATION != -1 && BuyIt->DURATION + BuyIt->TIMESTAMP <= current_timestamp) {
                BuyIt = BuyPtr->erase(BuyIt);
            }
            else {
                ++BuyIt;
            }

        }
    }

    // Match the order:
    //  Logics: 1. Deal with Buy order
```

```
        //          2. Check whether the orders in Sell can be matched with buy (!isdone)
        //          3. Divide into two cases: all bought/ partial bought
        //          4. If all bought, two cases: once or several times
        //          5. If partial, add the remaining part to the order book.
        bool fonud_equity = 0;
        for (it = OrderAll.begin(); it != OrderAll.end(); it++) {
            AllPtr = const_cast<Big_Order *> (&(*it));
            SellPtr = const_cast<set<order, compare_sell_order> *> (&(it->Sell));
            BuyPtr = const_cast<set<order, compare_buy_order> *> (&(it->Buy));
            if (it->EQUITY_SYMBOL == equity_symbol) { // Match and Dealt
                fonud_equity = true;
                // Match and Dealt the order. already excluded the expired items deletion before.

                // Case A: Buy order comes
                // First judge the existing of Sell order

                /* Can get Optimized by using a new data structure only to store the current trading*/

                if (Read_temp.buy_or_sell == BUY) {
                    while (!AllPtr->Sell.empty()) {
                        SellOrderPtr = const_cast<order *> (&(*SellPtr->begin()));
                        SellIt = SellPtr->begin();
//                        AllPtr->Price_dealt.insert(SellIt->PRICE);
                        // Then judge in loog, for Sell QUAN < Buy QUAN, should stop as long as the temp
    is done
                        if (Read_temp.isdone) break;
                        else if (!SellIt->isdone) {
                            // Case A.1, Sell's QUAN >= Buy's QUAN, which is always the final case.
                            if (SellIt->QUANTITY >= Read_temp.QUANTITY && Read_temp.PRICE >= SellIt->
    PRICE) {
                                SellOrderPtr->QUANTITY -= Read_temp.QUANTITY;

                                // Store information about clients
                                // 1. The coming Buyer
                                bool Clientfound = false;
                                for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                    ClientPtr = const_cast<Client *> (&(*BigClientit));
                                    if (BigClientit->client_name == Read_temp.CLIENT_NAME) {
                                        Clientfound = true;
                                        ClientPtr->amount_traded -= quantity * SellIt->PRICE;
                                        ClientPtr->stock_buy += quantity;
                                        break;
                                    }
                                }
                                if (!Clientfound) {
                                    Client temp_client;
                                    temp_client.client_name = Read_temp.CLIENT_NAME;
                                    temp_client.stock_buy += quantity;
                                    temp_client.amount_traded -= quantity * SellIt->PRICE;
                                    BigClient.insert(temp_client);
                                }

                                // 2. The existing seller
                                Clientfound = false;
                                for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                    ClientPtr = const_cast<Client *> (&(*BigClientit));
                                    if (BigClientit->client_name == SellOrderPtr->CLIENT_NAME) {
                                        Clientfound = true;
                                        ClientPtr->amount_traded += quantity * SellIt->PRICE;
                                        ClientPtr->stock_sell += quantity;
                                        break;
                                    }
                                }
```

```
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = SellOrderPtr->CLIENT_NAME;
                                temp_client.stock_sell += quantity;
                                temp_client.amount_traded += quantity * SellIt->PRICE;
                                BigClient.insert(temp_client);
                            }


                            // Remove the done Sell order
                            if (SellIt->QUANTITY == 0) {
                                SellOrderPtr->isdone = true;
                                // Erase from data structure
                                AllPtr->Sell.erase(SellIt);
                            }
                            // Output numbers
                            Number_of_share += quantity;
                            Money_Transferred += SellIt->PRICE * quantity;
                            Commission_Earnings += SellIt->PRICE * quantity / 100;
                            Commission_Earnings += SellIt->PRICE * quantity / 100;
                            Number_of_Completed_Trades += 1;
                            // Verbose Output
                            if (verbose) {
                                cout << client_name << " purchased " << quantity << " shares of " <<
    equity_symbol;
                                cout << " from " << SellIt->CLIENT_NAME << " for $" << SellIt->PRICE
    << "/share"
                                     << endl;
                            }
//                            AllPtr->Price_dealt.insert(SellIt->PRICE);
                            // insert price

                            if ((AllPtr->min_queue.size() + AllPtr->max_queue.size()) % 2 == 0) {
                                if (AllPtr->min_queue.empty()) AllPtr->max_queue.push(SellIt->PRICE);
                                else if (SellIt->PRICE <= AllPtr->min_queue.top()) {
                                    AllPtr->max_queue.push(SellIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->min_queue.top();
                                    AllPtr->min_queue.pop();
                                    AllPtr->max_queue.push(tempp);
                                    AllPtr->min_queue.push(SellIt->PRICE);
                                }
                            }
                            else {
                                if (SellIt->PRICE >= AllPtr->max_queue.top()) {
                                    AllPtr->min_queue.push(SellIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->max_queue.top();
                                    AllPtr->max_queue.pop();
                                    AllPtr->max_queue.push(SellIt->PRICE);
                                    AllPtr->min_queue.push(tempp);
                                }
                            }

                            Read_temp.QUANTITY = 0;
                            Read_temp.isdone = true;
                        }
                            // Case A.2, Sell's QUAN < Buy's QUAN, will recursive to Case A.1 or to
    Case A.3
                        else if (SellIt->QUANTITY < Read_temp.QUANTITY && Read_temp.PRICE >= SellIt->
    PRICE) {

                            // Store information about clients
```

```
                            // 1. The coming Buyer
                            bool Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == Read_temp.CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->amount_traded -= SellIt->QUANTITY * SellIt->PRICE;
                                    ClientPtr->stock_buy += SellIt->QUANTITY;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = Read_temp.CLIENT_NAME;
                                temp_client.amount_traded -= SellIt->QUANTITY * SellIt->PRICE;
                                temp_client.stock_buy += SellIt->QUANTITY;
                                BigClient.insert(temp_client);
                            }

                            // 2. The existing seller
                            Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == SellOrderPtr->CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->stock_sell += SellIt->QUANTITY;
                                    ClientPtr->amount_traded += SellIt->QUANTITY * SellIt->PRICE;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = SellOrderPtr->CLIENT_NAME;
                                temp_client.stock_sell += SellIt->QUANTITY;
                                temp_client.amount_traded += SellIt->QUANTITY * SellIt->PRICE;
                                BigClient.insert(temp_client);
                            }


                            // Output numbers
                            Number_of_share += SellIt->QUANTITY;
                            Money_Transferred += SellIt->PRICE * SellIt->QUANTITY;
                            Commission_Earnings += SellIt->PRICE * SellIt->QUANTITY / 100;
                            Commission_Earnings += SellIt->PRICE * SellIt->QUANTITY / 100;
                            Number_of_Completed_Trades += 1;
                            if (verbose) {
                                cout << client_name << " purchased " << SellIt->QUANTITY << " shares
    of "
                                     << equity_symbol;
                                cout << " from " << SellIt->CLIENT_NAME << " for $" << SellIt->PRICE
    << "/share"
                                     << endl;
                            }
//                            AllPtr->Price_dealt.insert(SellIt->PRICE);


                            if ((AllPtr->min_queue.size() + AllPtr->max_queue.size()) % 2 == 0) {
                                if (AllPtr->min_queue.empty()) AllPtr->max_queue.push(SellIt->PRICE);
                                else if (SellIt->PRICE <= AllPtr->min_queue.top()) {
                                    AllPtr->max_queue.push(SellIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->min_queue.top();
```

```
                                    AllPtr->min_queue.pop();
                                    AllPtr->max_queue.push(tempp);
                                    AllPtr->min_queue.push(SellIt->PRICE);
                                }
                            }
                            else {
                                if (SellIt->PRICE >= AllPtr->max_queue.top()) {
                                    AllPtr->min_queue.push(SellIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->max_queue.top();
                                    AllPtr->max_queue.pop();
                                    AllPtr->max_queue.push(SellIt->PRICE);
                                    AllPtr->min_queue.push(tempp);
                                }
                            }

                            quantity -= SellIt->QUANTITY;
                            Read_temp.QUANTITY -= SellIt->QUANTITY;
                            SellOrderPtr->QUANTITY = 0;
                            SellOrderPtr->isdone = true;
                            AllPtr->Sell.erase(SellIt);
                        }
                        else break;
//                          else if (SellIt->QUANTITY == Read_temp.QUANTITY && Read_temp.PRICE > SellIt
    ->PRICE) {
//                              SellOrderPtr->QUANTITY = 0;
//
//                          }
                    }
                }
                // Case A.3
                if (!Read_temp.isdone && Read_temp.DURATION != 0) {
                    AllPtr->Buy.insert(Read_temp);
                }
            }
                //Case B: Sell Order Comes
            else {
//                    for (BuyIt = (it->Buy).begin(); BuyIt != (it->Buy).end(); BuyIt++) {
                while (!AllPtr->Buy.empty()) {
                    BuyOrderPtr = const_cast<order *> (&(*BuyPtr->begin()));
                    BuyIt = BuyPtr->begin();
//                      AllPtr->Price_dealt.insert(BuyIt->PRICE);
                    if (Read_temp.isdone) break;
                    else if (!BuyIt->isdone) {
                        if (BuyIt->QUANTITY >= Read_temp.QUANTITY && Read_temp.PRICE <= BuyIt->PRICE)
    {

                            // Store information about clients
                            // 1. The coming Seller
                            bool Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == Read_temp.CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->amount_traded += quantity * BuyIt->PRICE;
                                    ClientPtr->stock_sell += quantity;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = Read_temp.CLIENT_NAME;
                                temp_client.amount_traded += quantity * BuyIt->PRICE;
```

```
                                temp_client.stock_sell += quantity;
                                BigClient.insert(temp_client);
                            }

                            // 2. The existing Buyer
                            Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
    BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == BuyOrderPtr->CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->stock_buy += quantity;
                                    ClientPtr->amount_traded -= quantity * BuyIt->PRICE;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = BuyOrderPtr->CLIENT_NAME;
                                temp_client.stock_buy += quantity;
                                temp_client.amount_traded -= quantity * BuyIt->PRICE;
                                BigClient.insert(temp_client);
                            }

                            BuyOrderPtr->QUANTITY -= Read_temp.QUANTITY;
                            // Output numbers
                            Number_of_share += quantity;
                            Money_Transferred += BuyIt->PRICE * quantity;
                            Commission_Earnings += BuyIt->PRICE * quantity / 100;
                            Commission_Earnings += BuyIt->PRICE * quantity / 100;
                            Number_of_Completed_Trades += 1;

                            // Verbose Output, what if one purchase is separated into 2 parts?
                            if (verbose) {
                                cout << BuyIt->CLIENT_NAME << " purchased " << quantity << " shares
    of "
                                     << equity_symbol;
                                cout << " from " << client_name << " for $" << BuyIt->PRICE << "/
    share"
                                     << endl;
                            }
//                              AllPtr->Price_dealt.insert(BuyIt->PRICE);

                            if ((AllPtr->min_queue.size() + AllPtr->max_queue.size()) % 2 == 0) {
                                if (AllPtr->min_queue.empty()) AllPtr->max_queue.push(BuyIt->PRICE);
                                else if (BuyIt->PRICE <= AllPtr->min_queue.top()) {
                                    AllPtr->max_queue.push(BuyIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->min_queue.top();
                                    AllPtr->min_queue.pop();
                                    AllPtr->max_queue.push(tempp);
                                    AllPtr->min_queue.push(BuyIt->PRICE);
                                }
                            }
                            else {
                                if (BuyIt->PRICE >= AllPtr->max_queue.top()) {
                                    AllPtr->min_queue.push(BuyIt->PRICE);
                                }
                                else {
                                    auto tempp = AllPtr->max_queue.top();
                                    AllPtr->max_queue.pop();
                                    AllPtr->min_queue.push(tempp);
                                    AllPtr->max_queue.push(BuyIt->PRICE);
                                }
```

```
                            }

                            Read_temp.QUANTITY = 0;
                            Read_temp.isdone = true;
                            if (BuyIt->QUANTITY == 0) {
                                BuyOrderPtr->isdone = true;
                                AllPtr->Buy.erase(BuyIt);
                            }
                        }
                            // Case B.2
                        else if (BuyIt->QUANTITY < Read_temp.QUANTITY && Read_temp.PRICE <= BuyIt->
PRICE) {

                            // Store information about clients
                            // 1. The coming Seller
                            bool Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == Read_temp.CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->amount_traded += BuyIt->QUANTITY * BuyIt->PRICE;
                                    ClientPtr->stock_sell += BuyIt->QUANTITY;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = Read_temp.CLIENT_NAME;
                                temp_client.amount_traded += BuyIt->QUANTITY * BuyIt->PRICE;
                                temp_client.stock_sell += BuyIt->QUANTITY;
                                BigClient.insert(temp_client);
                            }

                            // 2. The existing Buyer
                            Clientfound = false;
                            for (BigClientit = BigClient.begin(); BigClientit != BigClient.end();
BigClientit++) {
                                ClientPtr = const_cast<Client *> (&(*BigClientit));
                                if (BigClientit->client_name == BuyOrderPtr->CLIENT_NAME) {
                                    Clientfound = true;
                                    ClientPtr->amount_traded -= BuyIt->QUANTITY * BuyIt->PRICE; //
wrong sign

                                    ClientPtr->stock_buy += BuyIt->QUANTITY;
                                    break;
                                }
                            }
                            if (!Clientfound) {
                                Client temp_client;
                                temp_client.client_name = BuyOrderPtr->CLIENT_NAME;
                                temp_client.amount_traded -= BuyIt->QUANTITY * BuyIt->PRICE;
                                temp_client.stock_buy += BuyIt->QUANTITY;
                                BigClient.insert(temp_client);
                            }


                            Read_temp.QUANTITY -= BuyIt->QUANTITY;
                            // Output numbers
                            Number_of_share += BuyIt->QUANTITY;
                            Money_Transferred += BuyIt->PRICE * BuyIt->QUANTITY;
                            Commission_Earnings += BuyIt->PRICE * BuyIt->QUANTITY / 100;
                            Commission_Earnings += BuyIt->PRICE * BuyIt->QUANTITY / 100;
                            Number_of_Completed_Trades += 1;
                            if (verbose) {
                                cout << BuyIt->CLIENT_NAME << " purchased " << BuyIt->QUANTITY << "
```

```
    shares of ”
                                        << equity_symbol;
                                    cout << ” from ” << client_name << ” for $” << BuyIt->PRICE << ”/
    share”
                                        << endl;
                                }
//                                  AllPtr->Price_dealt.insert(BuyIt->PRICE);

                                if ((AllPtr->min_queue.size() + AllPtr->max_queue.size()) % 2 == 0) {
                                    if (AllPtr->min_queue.empty()) AllPtr->max_queue.push(BuyIt->PRICE);
                                    else if (BuyIt->PRICE <= AllPtr->min_queue.top()) {
                                        AllPtr->max_queue.push(BuyIt->PRICE);
                                    }
                                    else {
                                        auto tempp = AllPtr->min_queue.top();
                                        AllPtr->min_queue.pop();
                                        AllPtr->max_queue.push(tempp);
                                        AllPtr->min_queue.push(BuyIt->PRICE);
                                    }
                                }
                                else {
                                    if (BuyIt->PRICE >= AllPtr->max_queue.top()) {
                                        AllPtr->min_queue.push(BuyIt->PRICE);
                                    }
                                    else {
                                        auto tempp = AllPtr->max_queue.top();
                                        AllPtr->max_queue.pop();
                                        AllPtr->min_queue.push(tempp);
                                        AllPtr->max_queue.push(BuyIt->PRICE);
                                    }
                                }

                                quantity -= BuyIt->QUANTITY;
                                BuyOrderPtr->QUANTITY = 0;
                                BuyOrderPtr->isdone = 1;
                                AllPtr->Buy.erase(BuyIt);
                            }
                            else break;
                        }
                    }
                    if (!Read_temp.isdone && Read_temp.DURATION != 0) {
                        AllPtr->Sell.insert(Read_temp);
                    }
                }
                // Case C: Reamins some undealt orders, should be put back

            }
            // Remain another case that dealts without

//              else { // Not found and create
//                  Big_Order bigorderTemp;
//                  if (buy_or_sell1 == ”BUY”) {
//                      bigorderTemp.Buy.insert(Read_temp);
//                  }
//                  else {
//                      bigorderTemp.Sell.insert(Read_temp);
//                  }
//                  bigorderTemp.EQUITY_SYMBOL = Read_temp.EQUITY_SYMBOL;
//                  OrderAll.insert(bigorderTemp);
//              }
        }

        if (!fonud_equity) {
            Big_Order bigorderTemp;
            if (buy_or_sell1 == ”BUY”) {
```

```
                bigorderTemp.Buy.insert(Read_temp);
            }
            else {
                bigorderTemp.Sell.insert(Read_temp);
            }
            bigorderTemp.EQUITY_SYMBOL = Read_temp.EQUITY_SYMBOL;
            OrderAll.insert(bigorderTemp);
        }
    }


    if (median) {
        int median_num;

        for (it = OrderAll.begin(); it != OrderAll.end(); ++it) {
            /* debug output
            int iiii = 0;
            for (auto it_2 = it->Price_dealt.begin(); it_2 != it->Price_dealt.end(); it_2++) {
                cout << iiii << " times "<< *it_2 << " with timestamp " << timestamp << endl;
                iiii++;
            }
            */
//              if (it->Price_dealt.size() != 0) {
//                  Medianitr = it->Price_dealt.begin();
//                  if ((it->Price_dealt).size() % 2 == 0) {
//                      for (int i = 0; i < (it->Price_dealt).size() / 2; i++) {
//                          ++Medianitr;
//                      }
////                          median_num = ((*Medianitr) + (*(--Medianitr))) / 2;
//                      median_num = *Medianitr;
//                      median_num += *(--Medianitr);
//                      median_num /= 2;
//                  }
//                  else {
//                      for (int i = 0; i < (it->Price_dealt).size() / 2; i++) {
//                          ++Medianitr;
//                      }
//                      median_num = *Medianitr;
//                  }
//                  cout << "Median match price of " << it->EQUITY_SYMBOL << " at time " <<
    current_timestamp
//                       << " is $" << median_num << endl;
//              }
            if (it->min_queue.size() + it->max_queue.size() != 0) {
                if ((it->min_queue.size()+it->max_queue.size())%2==1) median_num = it->max_queue.top();
                else median_num = (it->max_queue.top() + it->min_queue.top())/2;
                cout << "Median match price of " << it->EQUITY_SYMBOL << " at time " << current_timestamp
                     << " is $" << median_num << endl;
            }
        }

    }

    if (midpoint) {
        int midpoint_num;
        for (it = OrderAll.begin(); it != OrderAll.end(); it++) {
            if (it->Buy.empty() || it->Sell.empty()) {
                cout << "Midpoint of " << it->EQUITY_SYMBOL << " at time " << current_timestamp << " is
    undefined"
                     << endl;
            }
            else {
                midpoint_num = ((*(it->Buy).begin()).PRICE + (*(it->Sell).begin()).PRICE) / 2;
                cout << "Midpoint of " << it->EQUITY_SYMBOL << " at time " << current_timestamp << " is $
    "
```

```
                    << midpoint_num << endl;
            }
        }
    }



    // At the end of day, output
    cout << "———End of Day———" << endl;
    cout << "Commission Earnings: $" << Commission_Earnings << endl;
    cout << "Total Amount of Money Transferred: $" << Money_Transferred << endl;
    cout << "Number of Completed Trades: " << Number_of_Completed_Trades << endl;
    cout << "Number of Shares Traded: " << Number_of_share << endl;

    if (transfers) {
        for (BigClientit = BigClient.begin(); BigClientit != BigClient.end(); BigClientit++) {
            cout << BigClientit->client_name << " bought " << BigClientit->stock_buy << " and sold "
                 << BigClientit->stock_sell << " for a net transfer of $" << BigClientit->amount_traded
     << endl;
        }
    }

    if (ttt) {
        for (ttt_setit = ttt_set.begin(); ttt_setit != ttt_set.end(); ttt_setit++) {
            cout << "Time travelers would buy " << ttt_setit->equity_name << " at time: " << ttt_setit->
     buy_time
                 << " and sell it at time: " << ttt_setit->sell_time << endl;
        }
    }
}

//void Add_to_Sell (order Temp, )
```