Report - assignment 1

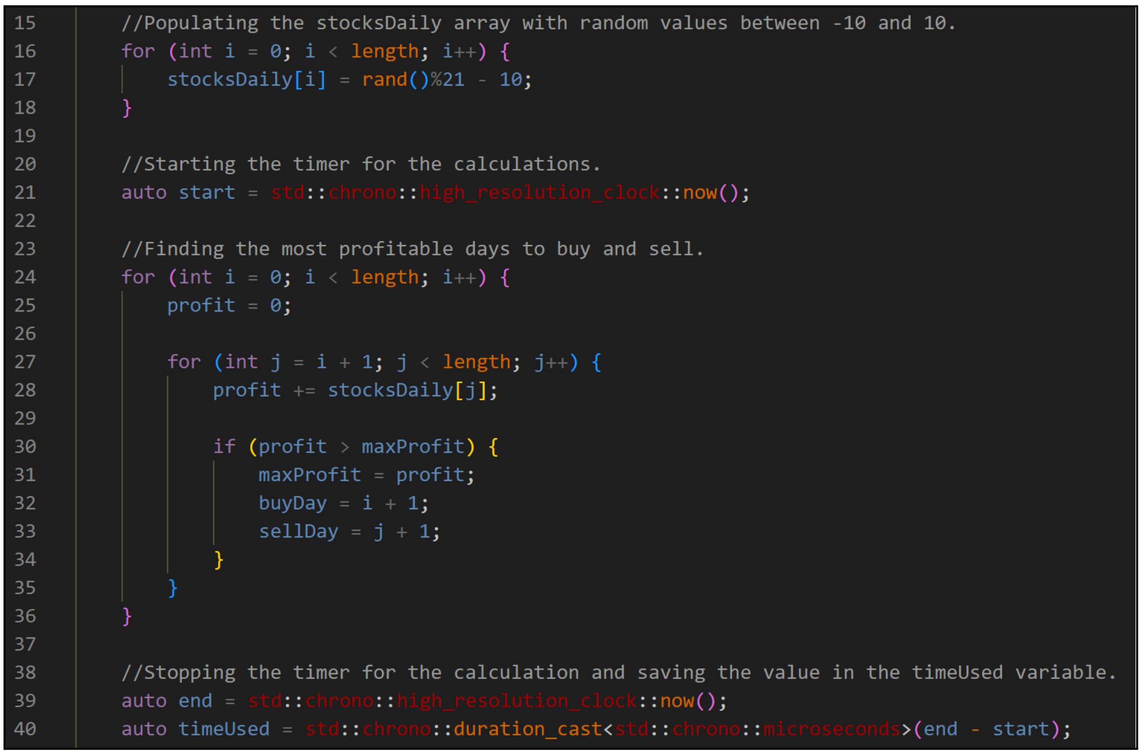
Camilla K. Birkelund IDATT2101 25. August 2022

Assignment description

The assignment was to create an algorithm to find the value of the most profitable day to sell and buy stocks from a given data set. In addition, the program should display the amount of time used to execute the algorithm, which should be compared to the theoretical complexity of the program.

Implementation

The algorithm is implemented by using two nested for-loops, where the outer loop iterates through all possible days to buy. The inner loop iterates through all possible days to sell, which will be all the remaining days in the array. The two nested for-loops gives a theoretical complexity of O(n^2), which means that the time used to run the program is proportional to n^2.



```cpp
    //Populating the stocksDaily array with random values between -10 and 10.
    for (int i = 0; i < length; i++) {
        stocksDaily[i] = rand()%21 - 10;
    }

    //Starting the timer for the calculations.
    auto start = std::chrono::high_resolution_clock::now();

    //Finding the most profitable days to buy and sell.
    for (int i = 0; i < length; i++) {
        profit = 0;

        for (int j = i + 1; j < length; j++) {
            profit += stocksDaily[j];

            if (profit > maxProfit) {
                maxProfit = profit;
                buyDay = i + 1;
                sellDay = j + 1;
            }
        }
    }

    //Stopping the timer for the calculation and saving the value in the timeUsed variable.
    auto end = std::chrono::high_resolution_clock::now();
    auto timeUsed = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
```

Figure 1: Screenshot of the algorithm. The implementation of the algorithm in c++ can be found in the file "stocksAssignment.cpp".

To measure the amount of time used, the “chrono” library in c++ was imported. The timer is started when running the program, and stops immediately after the results have been found.

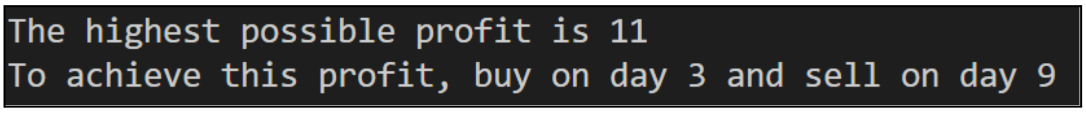
Testing

To test the syntax and semantics of the algorithm, the following set of 10 days with varying values were used to confirm that the algorithm provides the expected results.

Day	1	2	3	4	5	6	7	8	9	10
Stock values	-9	-6	-1	9	-2	0	0	-1	5	0

Table 1: The values given to test the algorithm.

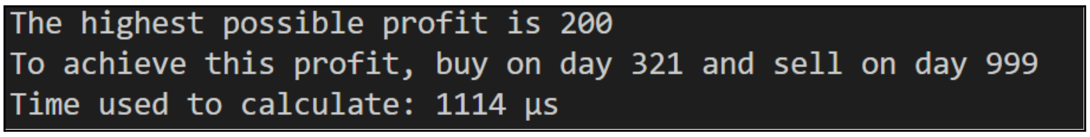
The expected result for the test is a maximum profit of 11, which is the result of buying on day 3, and selling on day 9. When running the algorithm, the result is as expected.



```
The highest possible profit is 11
To achieve this profit, buy on day 3 and sell on day 9
```

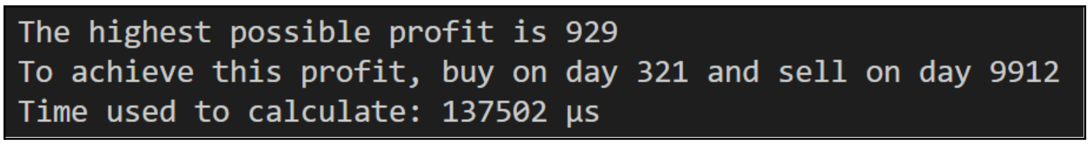
Figure 2: Results of the test of the algorithm’s semantics and syntax.

To test the theoretical complexity of the algorithm, the time was measured using different sized arrays to compare the values. To get a decent time measurement while running an algorithm of O(n^2) complexity on a computer, n should at least be set to 1000. The first iteration was therefore set to have an array of 1000 elements. This provided a run time of 1114 microseconds.



```
The highest possible profit is 200
To achieve this profit, buy on day 321 and sell on day 999
Time used to calculate: 1114 µs
```

Figure 3: The result of the algorithm’s iteration of 1000 elements.



```
The highest possible profit is 929
To achieve this profit, buy on day 321 and sell on day 9912
Time used to calculate: 137502 µs
```

Figure 4: The result of the algorithm’s iteration over 10000 elements.

To compare, the next iteration included an array of 10000 elements, which is ten times the previous amount. Theoretically, this means that the next iteration should take about 100 times longer than the previous one. The measurement for this iteration was 137502 microseconds, which is not exactly 100 times longer, but close enough due to the fact that some uncertainties are expected when working with such short amounts of time.

Further improvements

The current implementation works well for the array sizes that have been used in the examples, but the run time can become an issue with arrays that contain a considerably larger amount of elements. A way to avoid this problem is to implement the algorithm to have a linear theoretical complexity (i.e. a complexity of O(n)). This can be implemented by creating separate for-loops instead of nested ones.