**Analysis and Complexity**

For assignment 5, I chose to implement it using a vector to represent my heap. That is due to it being more flexible than a static array. I will be going over the functions I have used in this assignment one by one analysing them and giving their complexities

A computer screen shot of a program code

Description automatically generated

**heapifyMax:** It is the heapify algorithm that we were asked to implement in this assignment. It checks whether the parent is indeed the max between its children. It first checks the max between the children and then compares the larger child to the parent. If the parent is larger nothing is changed, however, if the child is larger then we swap them and continue to work on this recursively until we reach the leave nodes.

This is similar to heapifyMin except it checks for smallest.

Complexity:O(logn)

A screen shot of a computer code

Description automatically generated

**buildMaxHeap:** This function uses heapify in order to build a max heap. The parent will have index I, left child 2\*i+1 right child 2\*i+2. Parent will always be greater than its children. Moreover, the root in a max heap will always be the largest value in the array. Again, this is also similar to the function buildMinHeap.

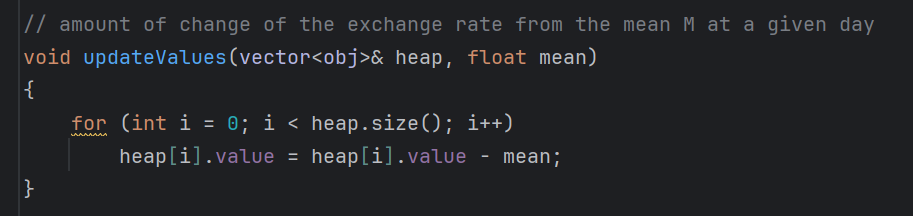
Complexity: O(n)

A screenshot of a computer

Description automatically generated

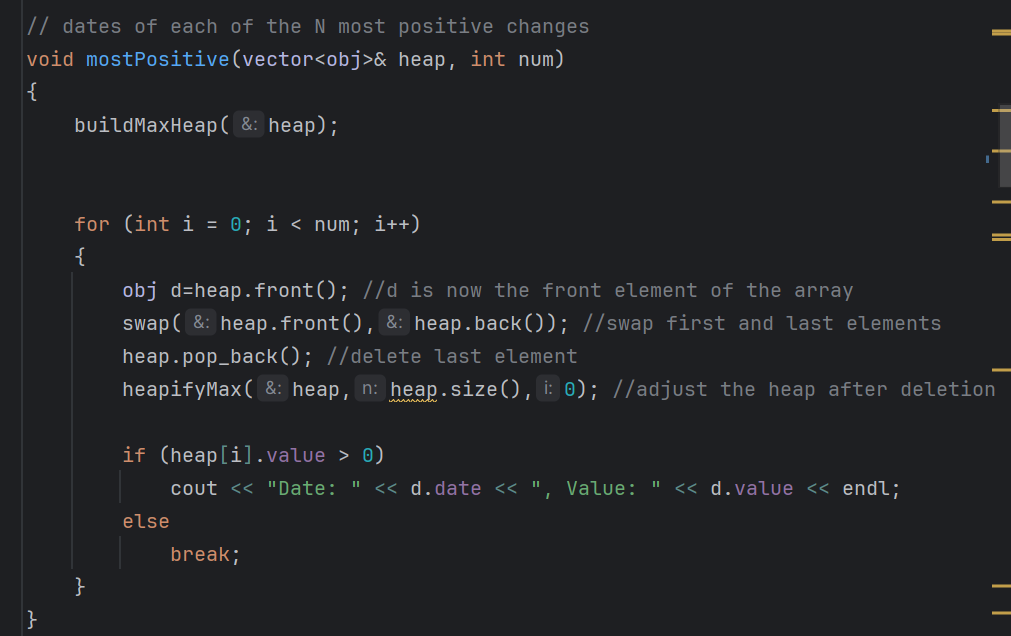
**getData:** I took the data in excel and put it in a text file called input.txt. This function takes the data in the file and stores it in the vector and inserts it line by line.

Complexity: O(n)



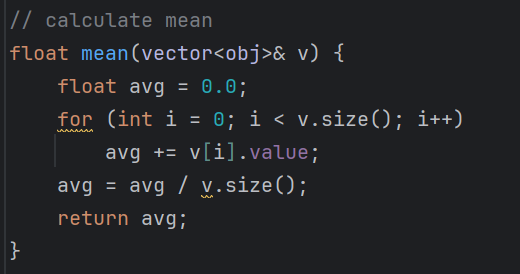
**updateValues:** This function loops over the whole vector and updates its values as the task asks us. Value-mean

Complexity: O(n)



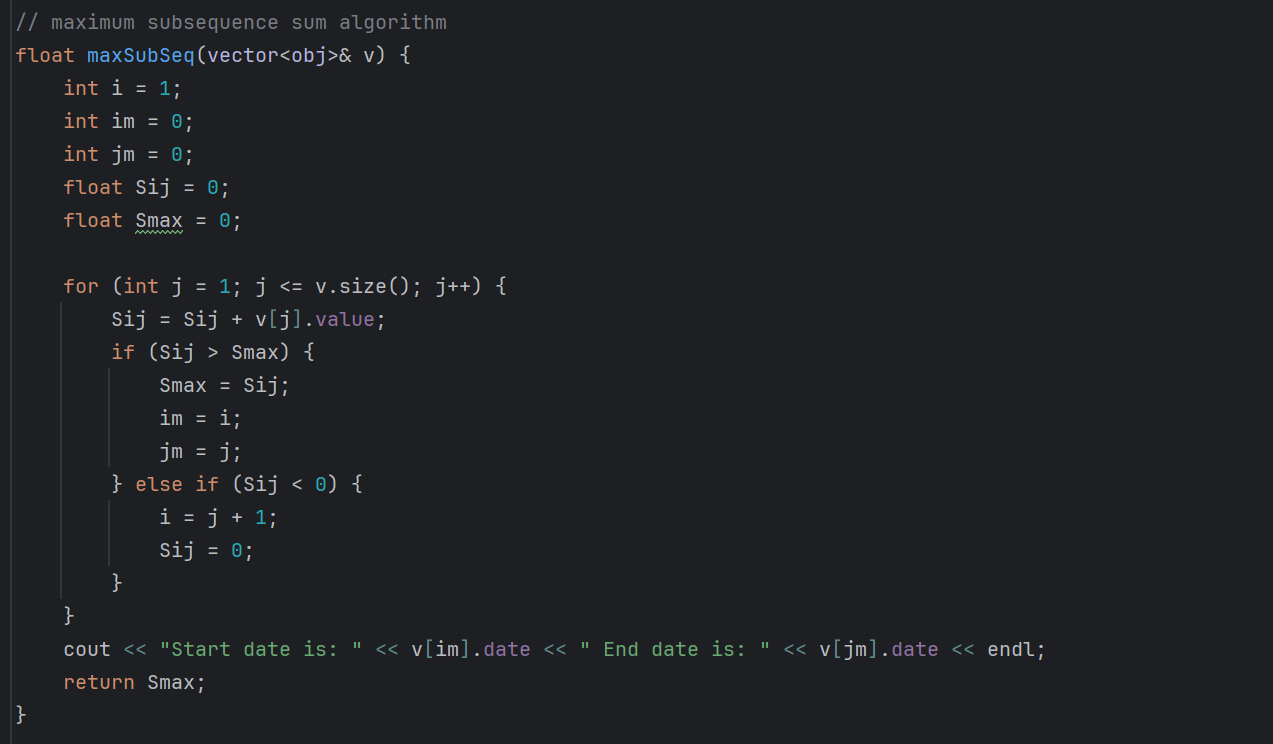
**mostPositive**: This function first builds a max heap. Moreover, we said a max heap’s root element is always the largest element in the vector. Therefore, we print the root, then remove it. Afterwards, we call heapify to adjust the heap and find the new root and continue this loop. This is similar to mostNegative

Complexity: O(nlogn)



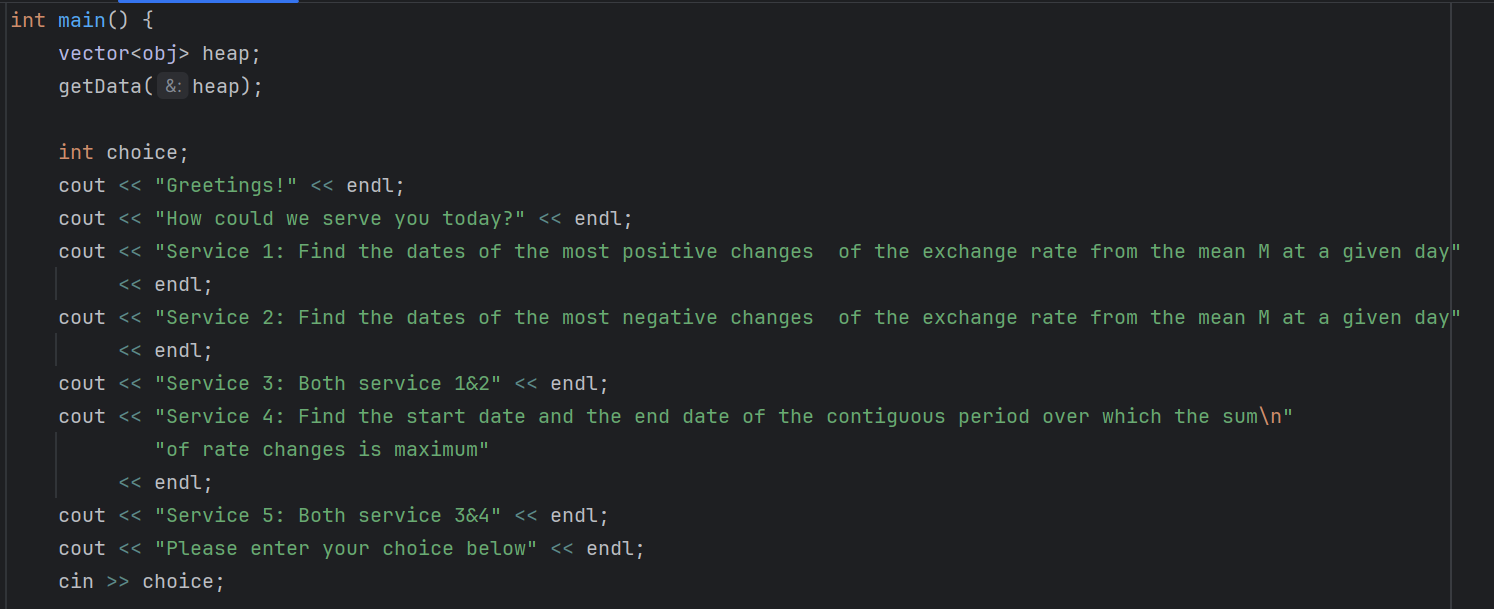
**mean:** This function calculates the mean by finding the sum of all element’s values and dividing by the size.

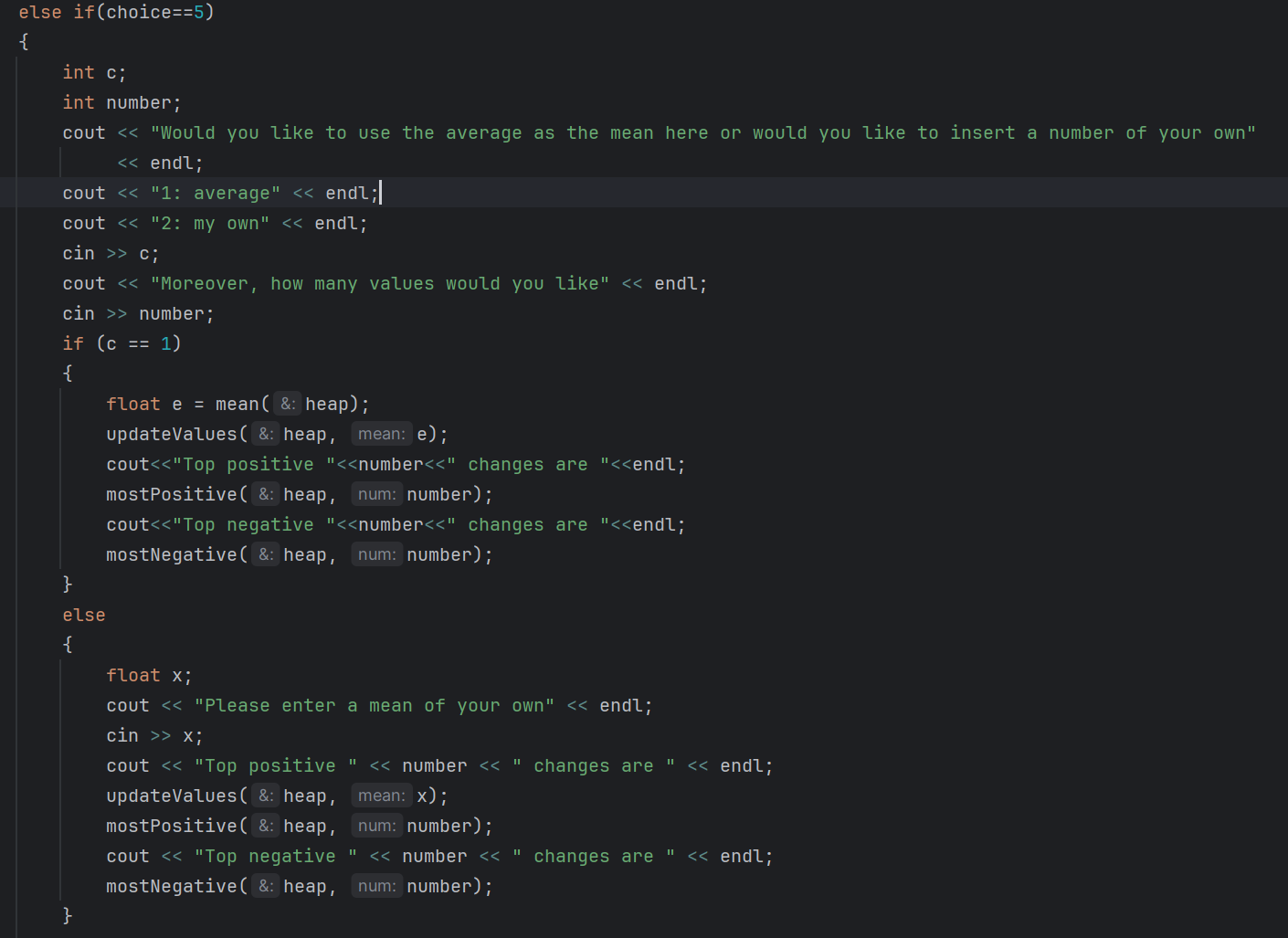
Complexity: O(n)

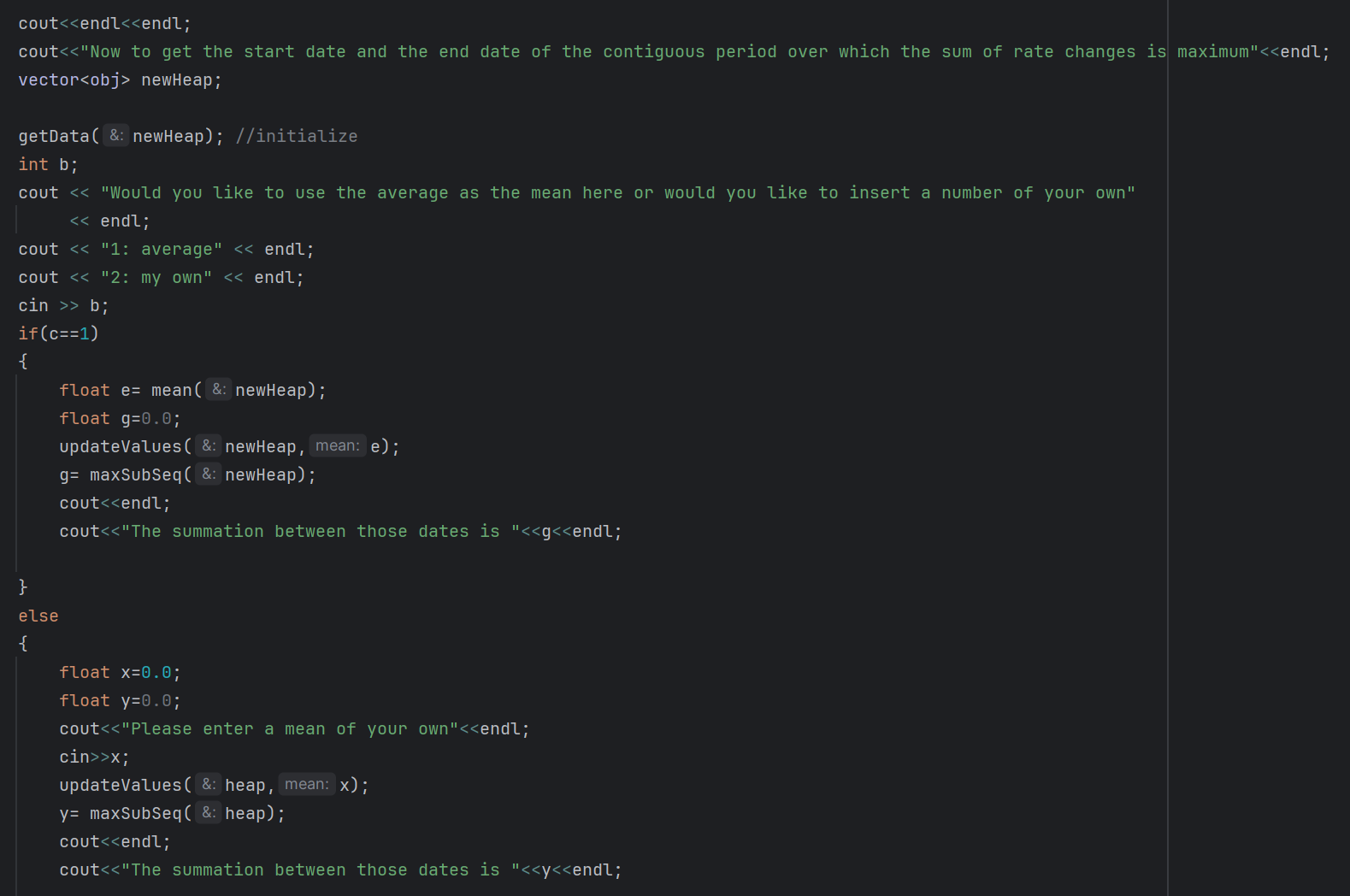


**maxSubSeq**: This function pseudocode was given by Dr.Goneid, I wrote in c++ and added the print date section. Complexity: O(n) ->given by Dr.Goneid

**Main**







Now we will calculate the complexity for service 5 since that is what the assignment asks for.

As you can see it calls:

getData, which has complexity O(n)

updateValues, which has complexity O(n)

mostPositive, which has complexity O(nlogn)

mostNegative which has complexity O(nlogn)

then calls getData again, which has complexity O(n)

it also calls updateValues again, which has complexity O(n)

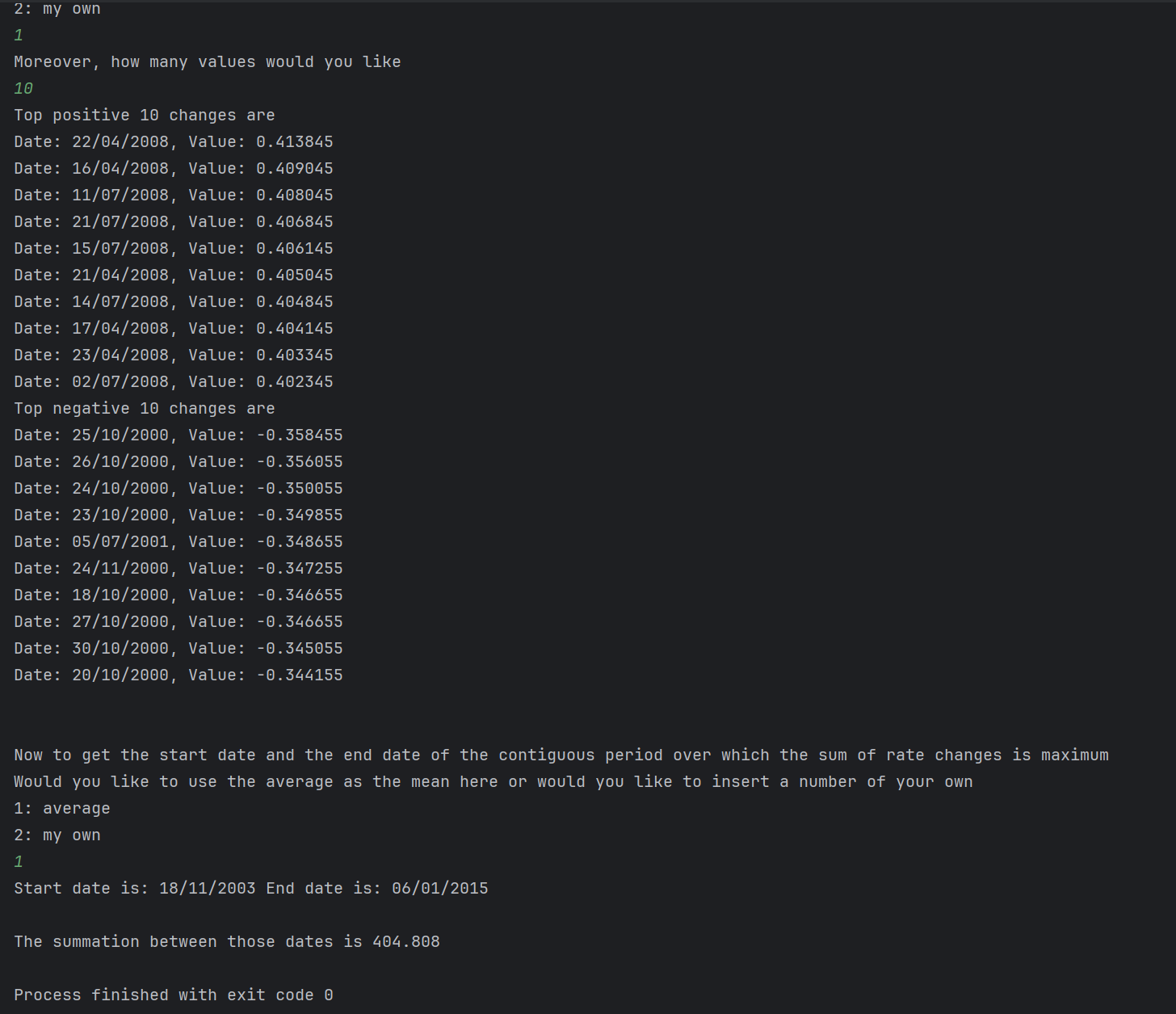
lastly it calls maxSubSeq, which has complexity O(n)

Therefore when adding them all together we receive time complexity of O(nlogn)

**Output**

A screenshot of a computer

Description automatically generated



**Remarks**

I initially planned to split this into a main and a heap class. Nevertheless, we spoke to Dr.Goneid and he said it is fine to put the functions in the main and not implement a class. Therefore, I went ahead and did that. Moreover, if you notice I used two vectors in service 5, that is because max heap and min heap mess up the order of the dates and that is vital when finding the start and end date for the max subsequence sum. Hence, I made a new initial vector and worked with it to solve problem 2 in the assignment. As making a new vector wouldn’t hurt the time complexity since readin from the file is O(n) and the total complexity of the program is O(nlogn) which is a higher complexity. Lastly, in my code I put an extra touch by allowing the user to pick a exchange rate of his choice to test these functions instead of using the mean. Feel free to try it by running my code. Hope you liked my work!