Leetcode739 Daily Temperatures:

Code:

class Solution {

public:

vector<int> dailyTemperatures(vector<int>& temperatures) {

int len = temperatures.size();

vector<int> day(len,0);

stack<int>st{};

for(int i=0;i<len;i++){

if(!st.empty()) {

int num = st.top();

while(temperatures[num]<temperatures[i]&&!st.empty()){

st.pop();

day[num] = i-num;

if(!st.empty()) num = st.top();

}

}

st.push(i);

}

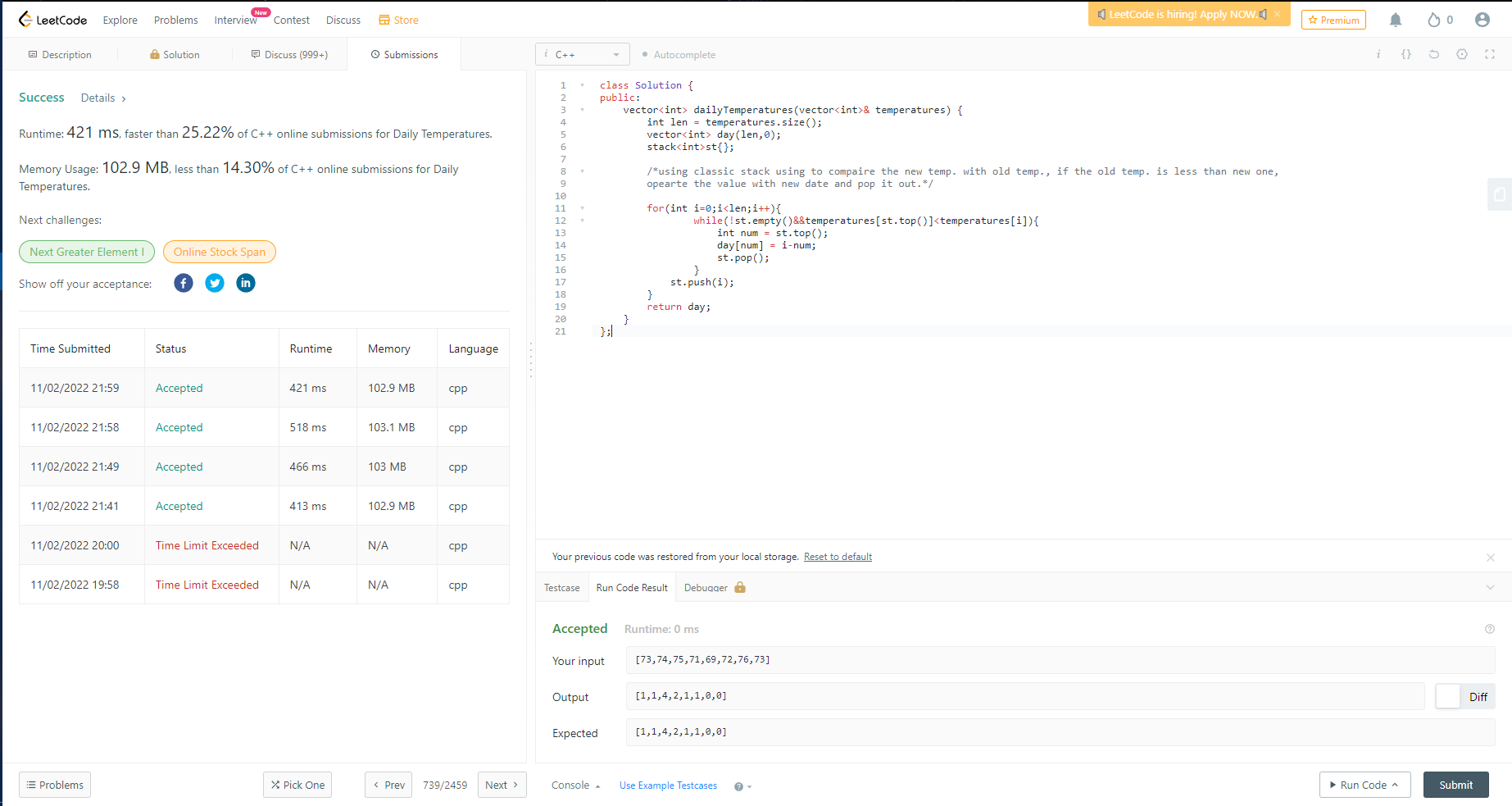
return day;

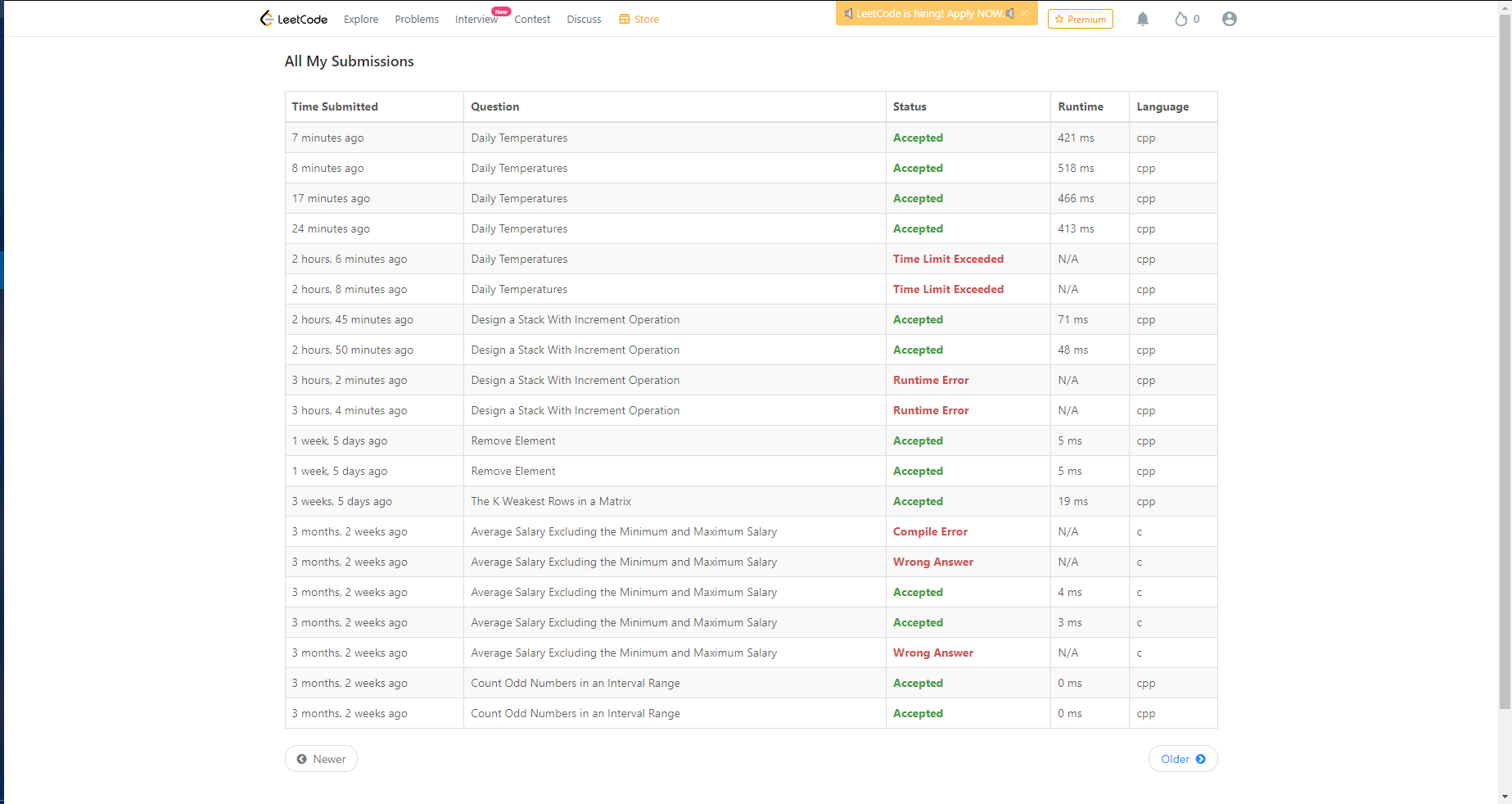
}

};

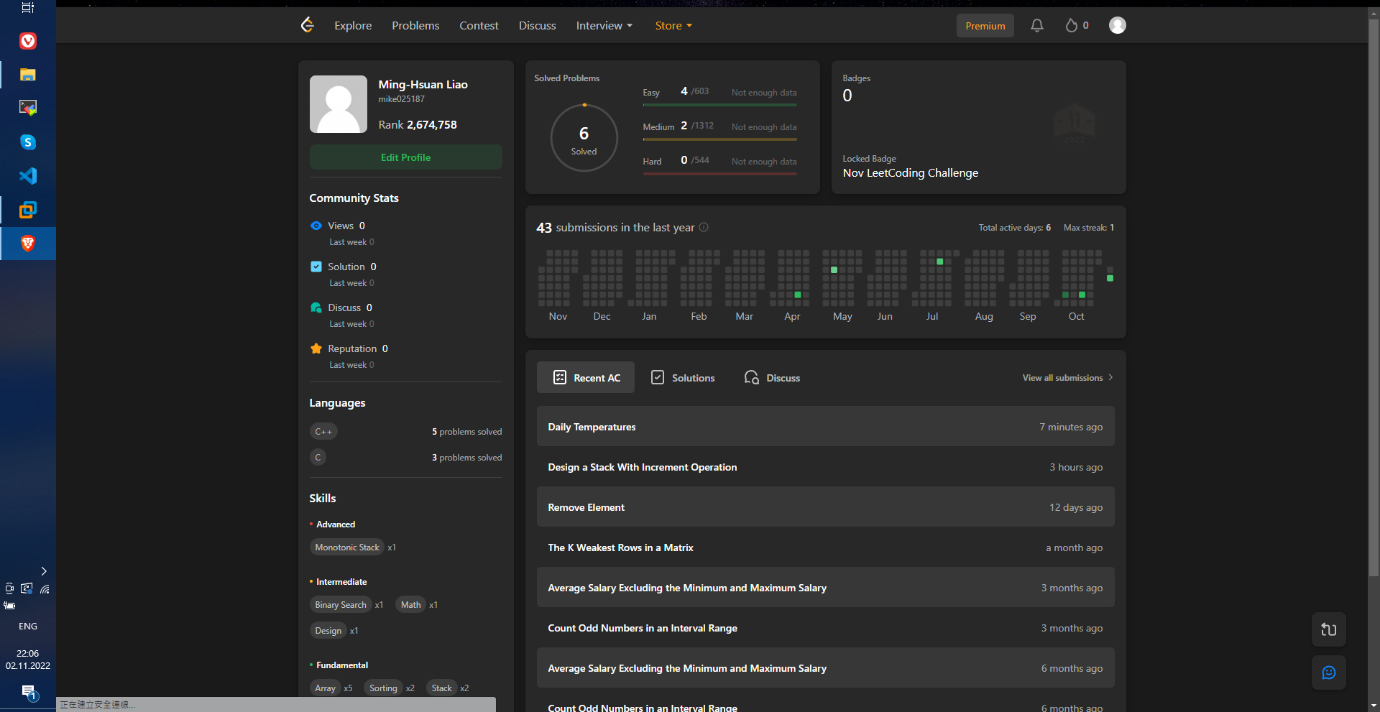
**Time complexity** is O(n) because we use n step to push element into stack, and if it need to pop out the worst case will be n step for popping out. Thus, the total number of steps will be about 2n, the time complexity will be O(n).

**Space complexity** is O(n) because we initialize 2 vector and one stack with the worst case will be these initialization each n, thus total 3n. Therefore the space complexity will be O(n) according to definition of big O.



Submission:

Personal profile:



Leetcode1381 Design a Stack With Increment Operation:

class CustomStack {

vector<int> st; // using vector to store the value array

int capacity;

public:

CustomStack(int maxSize) {

capacity = maxSize; //give a fixed maximun number for stack

}

void push(int x) {

if(st.size() == capacity) return; // determine if our stack size is more than its capacity, than we return directly do nothing

st.push\_back(x); //otherwise insert the value into out stack

}

int pop() {

if(st.empty()) return -1; //determine if our stack dont have anything, than we need to return directly doing nothing

int len = st.size(); //get the current location of pop of stack

int num = st[len-1]; // using a variable to store the value, which will be poped out later and pop the value out

st.pop\_back();

return num;

}

void increment(int k, int val) {

if(st.empty()) return; // check if the stak is empty, than it will do nothing

k > st.size() ? k = st.size() : 1;// compare the max capacity and k for loop to add value into the stack, which the location is under the compared number

for(int i =0; i<k;i++) {

st[i] += val;

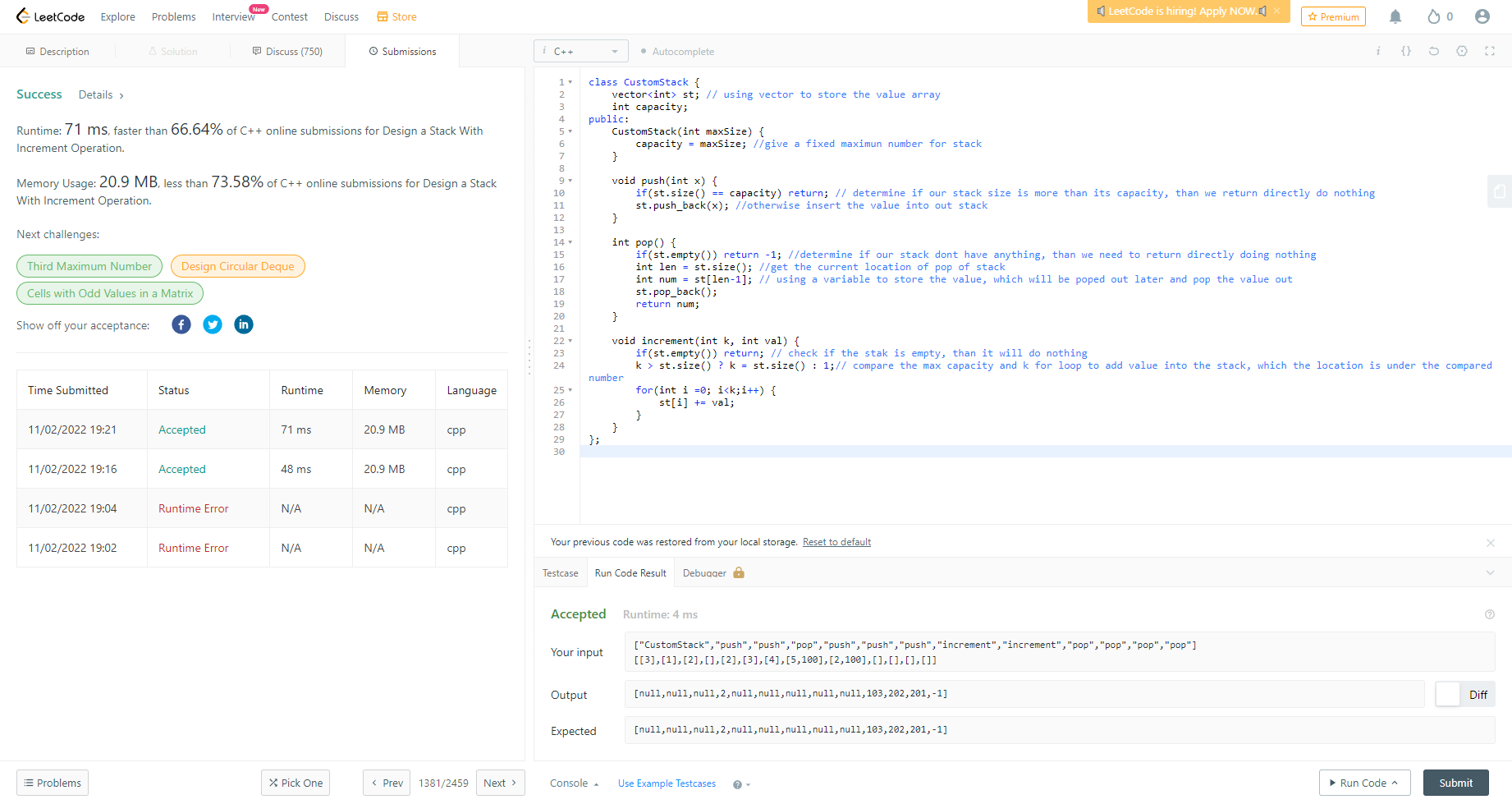
}

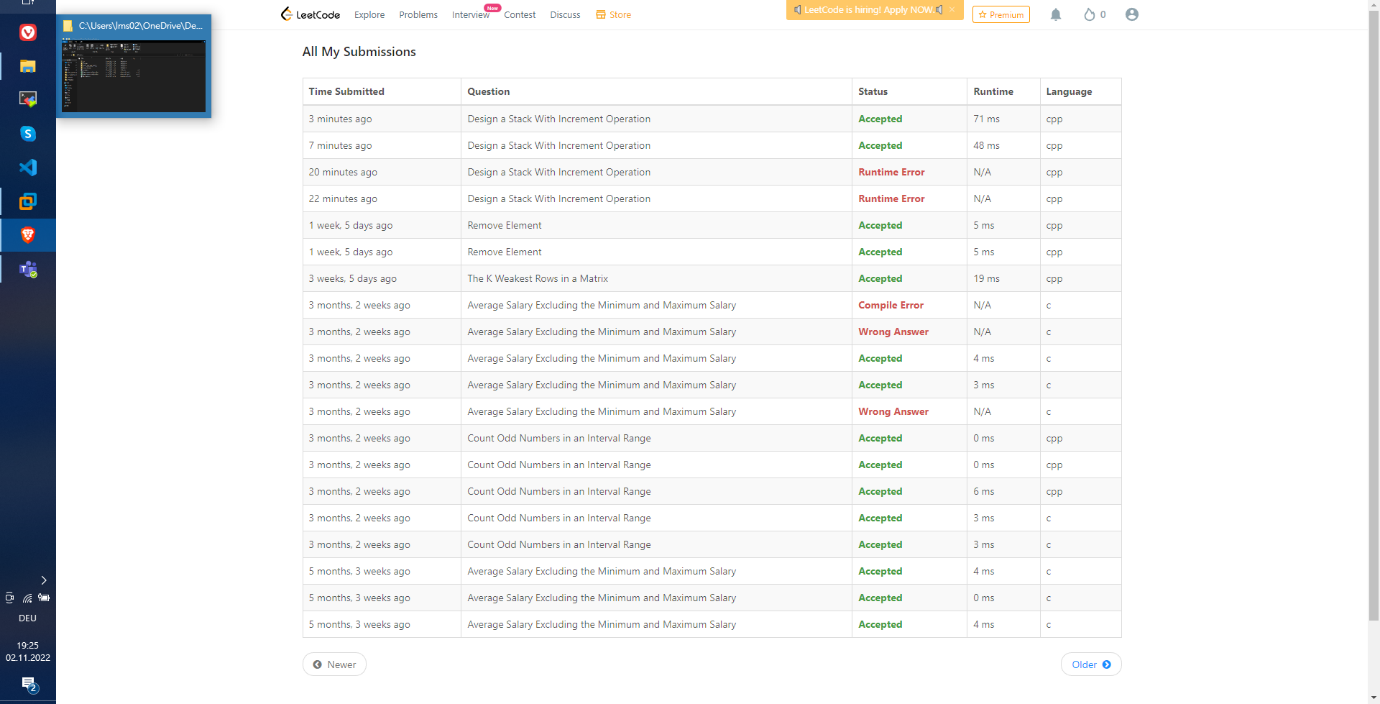
}

};

**Time complexity** in each function will be CustomStack: O(1); push: O(1); pop: O(1); increment: O(n). The front three functions is only almost one step for each command, thus the big O for each three function is O(1). The last function is O(n) because it use a for loop for n times to make addition operation.

**Space complexity** in first three functions is O(1) because it don’t need any loop or recursive operation to do it. But the last function’s space complexity is O(n) because it need in the worst situation at least n address to store the vector array st.



Submission:

Personal profile: