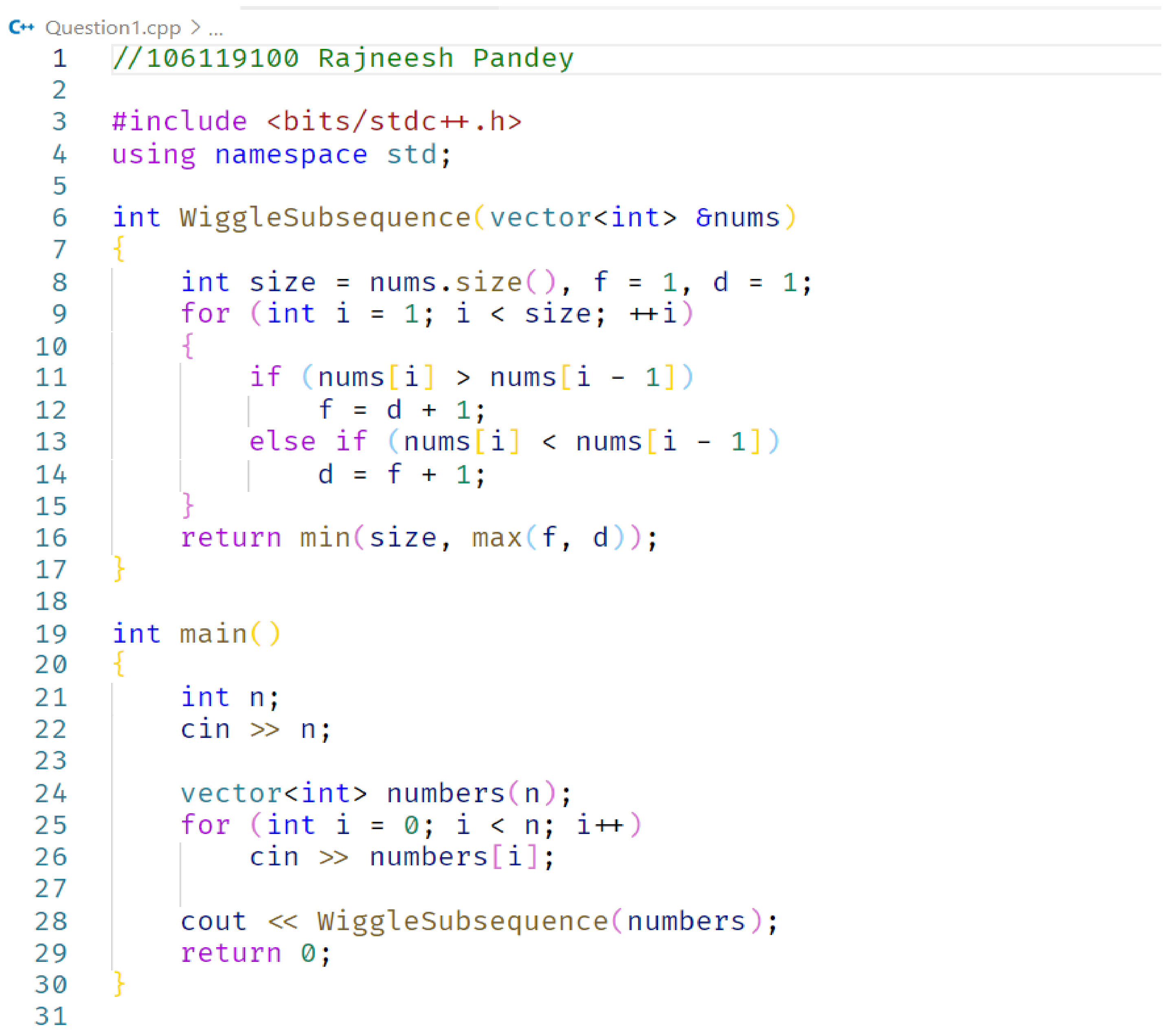
12/04/2021 Algorithms Lab Internal Assessment-1 **106119100**

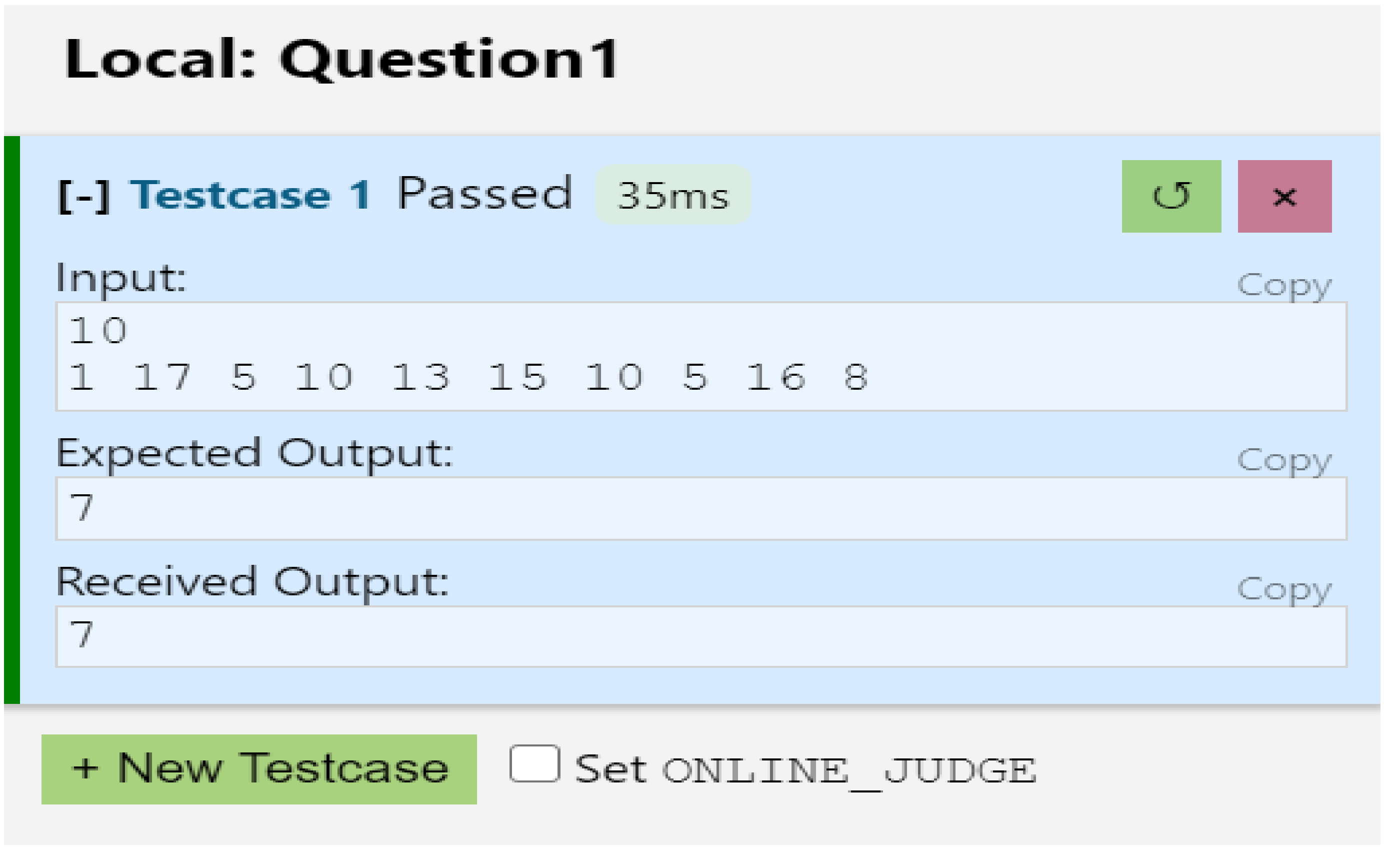
Rajneesh Pandey

Question 1

For this Problem, I used Greedy Approach. This problem is equivalent to finding the number of alternating max. and min. peaks in the array. Since, if we choose any other intermediate number to be a part of the current wiggle subsequence, the maximum length of that wiggle subsequence will always be less than or equal to the one obtained by choosing only the consecutive max. and min. elements.



```cpp
//106119100 Rajneesh Pandey

#include <bits/stdc++.h>
using namespace std;

int WiggleSubsequence(vector<int> &nums)
{
    int size = nums.size(), f = 1, d = 1;
    for (int i = 1; i < size; ++i)
    {
        if (nums[i] > nums[i - 1])
            f = d + 1;
        else if (nums[i] < nums[i - 1])
            d = f + 1;
    }
    return min(size, max(f, d));
}

int main()
{
    int n;
    cin >> n;

    vector<int> numbers(n);
    for (int i = 0; i < n; i++)
        cin >> numbers[i];

    cout << WiggleSubsequence(numbers);
    return 0;
}

```



Local: Question1

[-] Testcase 1 Passed 35ms

Input:

```
10
1 17 5 10 13 15 10 5 16 8
```

Expected Output:

```
7
```

Received Output:

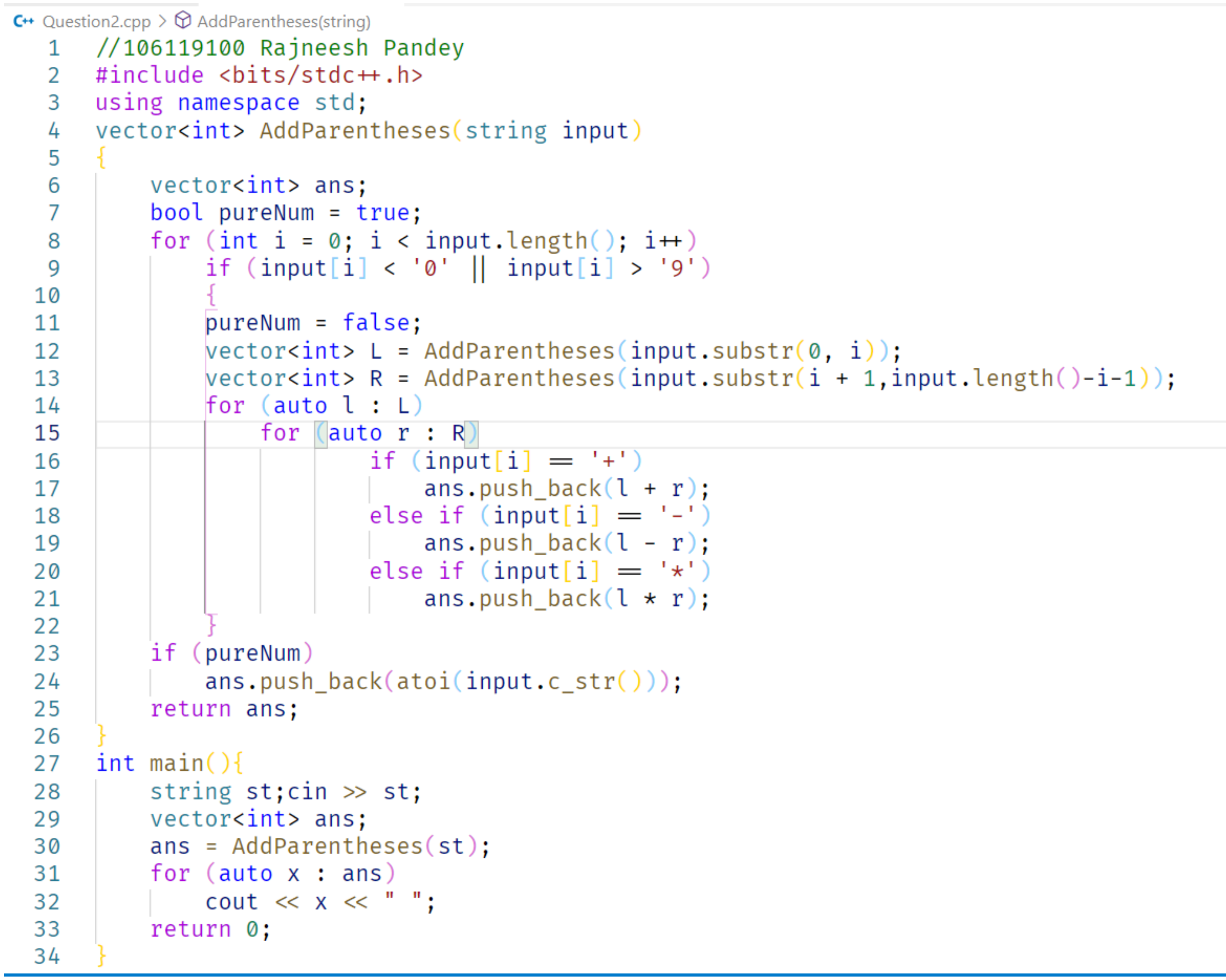
```
7
```

Question 2

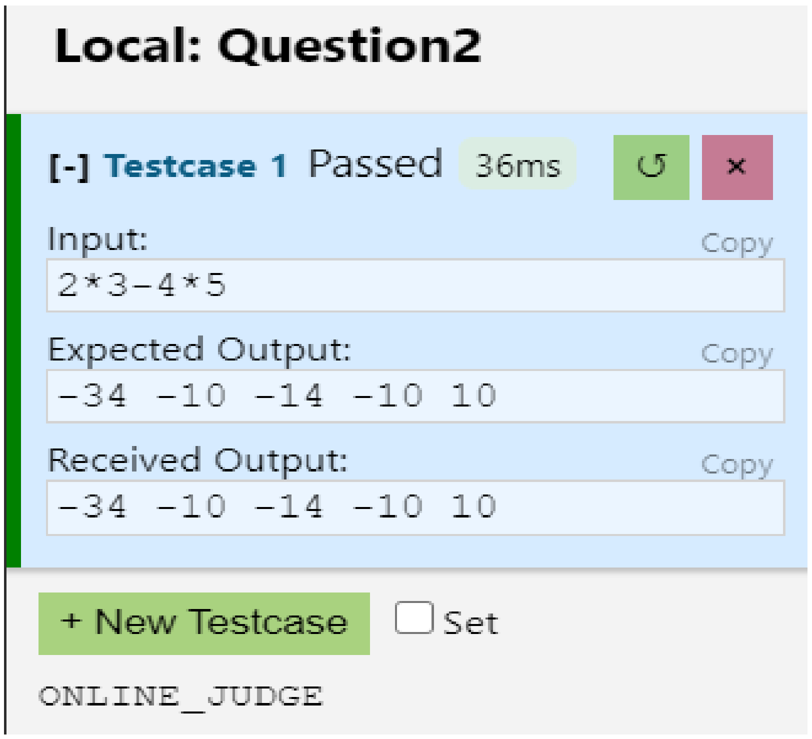
We can simply observe the recursion formula as follows:

(base) case 1: if there is 0 symbol,
we simply add input to result list;
case 2: if there is 1 symbol,
e.g., 1 s 2, we calculate the input and the result add to result list;
case 3: if there are 2 symbols,
e.g., 1 s 2 s 3, we can add parentheses when we meet a symbol as below:
1 s (2 s 3) and (1 s 2) s 3.
In this way, 1, (2 s 3), (1 s 2), and 3 can all be solved based on the above base cases, and we combine the result accordingly and add to result list.
...
for case n, we separate the expression into 2 parts at each position of symbol,
and solve the two parts separately following the same pattern, and combine them accordingly.

The time complexity is O(2^n) exponential, for n is the number of operators in the input. That's because each function call calls itself twice unless it has been recursed n times.



```cpp
//106119100 Rajneesh Pandey
#include <bits/stdc++.h>
using namespace std;
vector<int> AddParentheses(string input)
{
    vector<int> ans;
    bool pureNum = true;
    for (int i = 0; i < input.length(); i++)
        if (input[i] < '0' || input[i] > '9')
        {
        pureNum = false;
        vector<int> L = AddParentheses(input.substr(0, i));
        vector<int> R = AddParentheses(input.substr(i + 1,input.length()-i-1));
        for (auto l : L)
            for (auto r : R)
                if (input[i] == '+')
                    ans.push_back(l + r);
                else if (input[i] == '-')
                    ans.push_back(l - r);
                else if (input[i] == '*')
                    ans.push_back(l * r);
        }
    if (pureNum)
        ans.push_back(atoi(input.c_str()));
    return ans;
}
int main(){
    string st;cin >> st;
    vector<int> ans;
    ans = AddParentheses(st);
    for (auto x : ans)
        cout << x << " ";
    return 0;
}
```


Local: Question2

[-] Testcase 1 Passed 36ms

Input:

```
2*3-4*5
```

Expected Output:

```
-34 -10 -14 -10 10
```

Received Output:

```
-34 -10 -14 -10 10
```