The Maximum Subarray

```cpp
#include <bits/stdc++.h>

using namespace std;

string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

/*
 * Complete the 'maxSubarray' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

vector<int> maxSubarray(vector<int> arr) {
    int maxSub = arr[0], curr = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        curr = max(arr[i], curr + arr[i]);
        maxSub = max(maxSub, curr);
    }

    int maxSeq = arr[0];
    int sumPos = 0;
    bool hasPos = false;
    for (int x : arr) {
        if (x > 0) {
            sumPos += x;
            hasPos = true;
        }
        maxSeq = max(maxSeq, x);
    }
    if (hasPos) maxSeq = sumPos;

    return {maxSub, maxSeq};
}

int main()
{
    ofstream fout(getenv("OUTPUT_PATH"));

    string t_temp;
    getline(cin, t_temp);
    int t = stoi(ltrim(rtrim(t_temp)));

    for (int t_itr = 0; t_itr < t; t_itr++) {
        string n_temp;
        getline(cin, n_temp);

        int n = stoi(ltrim(rtrim(n_temp)));

        string arr_temp_temp;
        getline(cin, arr_temp_temp);

        vector<string> arr_temp = split(rtrim(arr_temp_temp));

        vector<int> arr(n);

        for (int i = 0; i < n; i++) {
            int arr_item = stoi(arr_temp[i]);

            arr[i] = arr_item;
        }

        vector<int> result = maxSubarray(arr);

        for (size_t i = 0; i < result.size(); i++) {
            fout << result[i];

            if (i != result.size() - 1) {
                fout << " ";
            }
        }

        fout << "\n";
    }

    fout.close();

    return 0;
}

string ltrim(const string &str) {
    string s(str);

    s.erase(
        s.begin(),
        find_if(s.begin(), s.end(), not1(ptr_fun<int,
int>(isspace)))
    );

    return s;
}

string rtrim(const string &str) {
    string s(str);

    s.erase(
        find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(),
        s.end()
    );

    return s;
}

vector<string> split(const string &str) {
    vector<string> tokens;

    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));

        start = end + 1;
    }

    tokens.push_back(str.substr(start));

    return tokens;
}
```