Array Manipulation

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

using namespace std;

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

/*
 * Complete the 'arrayManipulation' function below.
 *
 * The function is expected to return a LONG_INTEGER.
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. 2D_INTEGER_ARRAY queries
 */

long arrayManipulation(int n, vector<vector<int>> queries) {
    vector<long> diff(n+2,0);
    for(auto&q:queries){
        int a=q[0],b=q[1],k=q[2];
        diff[a]+=k;
        diff[b+1]-=k;
    }
    long mx=0,cur=0;
    for(int i=1;i<=n;i++){
        cur+=diff[i];
        mx=max(mx,cur);
    }
    return mx;
}

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

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);
```

```
    int m = stoi(first_multiple_input[1]);

    vector<vector<int>> queries(m);

    for (int i = 0; i < m; i++) {
        queries[i].resize(3);

        string queries_row_temp_temp;
        getline(cin, queries_row_temp_temp);

        vector<string> queries_row_temp =
split(rtrim(queries_row_temp_temp));

        for (int j = 0; j < 3; j++) {
            int queries_row_item = stoi(queries_row_temp[j]);

            queries[i][j] = queries_row_item;
        }
    }

    long result = arrayManipulation(n, queries);

    fout << result << "\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;
}
```