Started on	Tuesday, 6 February 2024, 8:47 PM
State	Finished
Completed on	Tuesday, 6 February 2024, 9:40 PM
Time taken	53 mins 4 secs
Marks	30.00/30.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 10.00 out of 10.00

Sorting

One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.

Insertion Sort

These challenges will cover *Insertion Sort*, a simple and intuitive sorting algorithm. We will first start with a nearly sorted list.

Insert element into sorted list

Given a sorted list with an unsorted number e in the rightmost cell, can you write some simple code to insert e into the array so that it remains sorted?

Since this is a learning exercise, it won't be the most efficient way of performing the insertion. It will instead demonstrate the brute-force method in detail.

Assume you are given the array $arr = [1, 2, 4, 5, 3]$ indexed $0 \ldots 4$. Store the value of $arr[4]$. Now test lower index values successively from 3 to 0 until you reach a value that is lower than $arr[4]$, at $arr[1]$ in this case. Each time your test fails, copy the value at the lower index to the current index and print your array. When the next lower indexed value is smaller than $arr[4]$, insert the stored value at the current index and print the entire array.

Example

$n = 5$
$arr = [1, 2, 4, 5, 3]$
Start at the rightmost index. Store the value of $arr[4] = 3$. Compare this to each element to the left until a smaller value is reached. Here are the results as described:

```
1 2 4 5 5
1 2 4 4 5
1 2 3 4 5
```

Function Description

Complete the *insertionSort1* function in the editor below.

insertionSort1 has the following parameter(s):

- *n*: an integer, the size of arr
- *arr*: an array of integers to sort

Returns

- *None:* Print the interim and final arrays, each on a new line. No return value is expected.

Input Format

The first line contains the integer n, the size of the array arr.
The next line contains n space-separated integers $arr[0] \ldots arr[n-1]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000$

Output Format

Print the array as a row of space-separated integers each time there is a shift or insertion.

Sample Input

```
5
2 4 6 8 3
```

Sample Output

```
2 4 6 8 8
2 4 6 6 8
2 4 4 6 8
2 3 4 6 8
```

Explanation

3 is removed from the end of the array.
In the 1^{st} line $8 > 3$, so 8 is shifted one cell to the right.
In the 2^{nd} line $6 > 3$, so 6 is shifted one cell to the right.
In the 3^{rd} line $4 > 3$, so 4 is shifted one cell to the right.
In the 4^{th} line $2 < 3$, so 3 is placed at position 1.

Next Question

In the next question, we will complete the insertion sort.

For example:

Input	Result
5 2 4 6 8 3	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8

Answer: (penalty regime: 0 %)

Reset answer

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

using namespace std;

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

/*
 * Complete the 'insertionSort1' function below.
 *
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER_ARRAY arr
 */

int value; //Variable to store the value
int hole; //Variable to store the index

void insertionSort1(int n, vector<int> arr) {
    value = arr[n-1]; //Storing the rightmost value of the given array
    hole = n-1; //Starting from the rightmost index

    //Comparing the values to the left until the smaller value
    while(hole > 0 && arr[hole-1] > value){
        arr[hole] = arr[hole-1]; //Shifting one element to the right
        hole--; //Reducing the index by 1
        for(int i = 0; i < n; i++){
            cout << arr[i] << " "; //Printing array after each iteration
        }
        cout << endl; //Introducing a line break
    }

    arr[hole] = value; //Inserting the value in the exact position
    for (int i = 0; i < n; i++){
        cout << arr[i] << " "; //Printing final sorted array
    }
}

int main()
{
    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);
```

```
    vector<int> arr(n);

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

        arr[i] = arr_item;
    }

    insertionSort1(n, arr);

    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;
}
```

	Input	Expected	Got	
✔	5 2 4 6 8 3	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8	2 4 6 8 8 2 4 6 6 8 2 4 4 6 8 2 3 4 6 8	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.

Question 2

Correct

Mark 10.00 out of 10.00

In *Insertion Sort Part 1*, you inserted one element into an array at its correct sorted position. Using the same approach repeatedly, can you sort an entire array?

Guideline: You already can place an element into a sorted array. How can you use that code to build up a sorted array, one element at a time? Note that in the first step, when you consider an array with just the first element, it is already sorted since there's nothing to compare it to.

In this challenge, print the array after each iteration of the insertion sort, i.e., whenever the next element has been inserted at its correct position. Since the array composed of just the first element is already sorted, begin printing after placing the second element.

Example.

$n = 7$
$arr = [3, 4, 7, 5, 6, 2, 1]$

Working from left to right, we get the following output:

```
3 4 7 5 6 2 1
3 4 7 5 6 2 1
3 4 5 7 6 2 1
3 4 5 6 7 2 1
2 3 4 5 6 7 1
1 2 3 4 5 6 7
```

Function Description

Complete the *insertionSort2* function in the editor below.

insertionSort2 has the following parameter(s):

- *int n:* the length of arr
- *int arr[n]:* an array of integers

Prints

At each iteration, print the array as space-separated integers on its own line.

Input Format

The first line contains an integer, n, the size of arr.
The next line contains n space-separated integers $arr[i]$.

Constraints

$1 \leq n \leq 1000$
$-10000 \leq arr[i] \leq 10000, 0 \leq i < n$

Output Format

Print the entire array on a new line at every iteration.

Sample Input

```
STDIN           Function
-----           --------
6               n = 6
1 4 3 5 6 2     arr = [1, 4, 3, 5, 6, 2]
```

Sample Output

```
1 4 3 5 6 2
1 3 4 5 6 2
1 3 4 5 6 2
1 3 4 5 6 2
1 2 3 4 5 6
```

Explanation

Skip testing 1 against itself at position 0. It is sorted.
Test position 1 against position 0: $4 > 1$, no more to check, no change.
Print arr
Test position 2 against positions 1 and 0:

- $3 < 4$, new position may be 1. Keep checking.
- $3 > 1$, so insert 3 at position 1 and move others to the right.

Print arr
Test position 3 against positions $2, 1, 0$ (as necessary): no change.
Print arr
Test position 4 against positions $3, 2, 1, 0$: no change.
Print arr
Test position 5 against positions $4, 3, 2, 1, 0$, insert 2 at position 1 and move others to the right.
Print

For example:

Input	Result
6 1 4 3 5 6 2	1 4 3 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 2 3 4 5 6

Answer: (penalty regime: 0 %)

Reset answer

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

using namespace std;

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

/*
 * Complete the 'insertionSort2' function below.
 *
 * The function accepts following parameters:
 *  1. INTEGER n
 *  2. INTEGER_ARRAY arr
 */

int value; //Variable to store the value
int hole; //Variable to store the index

void insertionSort2(int n, vector<int> arr) {

    //Iterating over the array from first index
    for (int i =1; i<n; i++){
        value = arr[i]; //Getting the value at the current position
        hole = i; //Getting the current index

        while (hole>0 && arr[hole-1]>value){
            arr[hole] = arr[hole-1];
            hole = hole-1;
        }

        arr[hole] = value; //Inserting value to the exact position

        for(int i = 0; i < n; i++){
            cout << arr[i] << " "; //Printing array after each iteration
        }
        cout << endl; //Introducing a line break
    }
}

int main()
{
    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;
    }

    insertionSort2(n, arr);

    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;
}

```

	Input	Expected	Got	
✔	6 1 4 3 5 6 2	1 4 3 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 2 3 4 5 6	1 4 3 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 3 4 5 6 2 1 2 3 4 5 6	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.

Question 3

Correct

Mark 10.00 out of 10.00

Starting with a 1-indexed array of zeros and a list of operations, for each operation add a value to each the array element between two given indices, inclusive. Once all operations have been performed, return the maximum value in the array.

Example

$n = 10$
$queries = [[1, 5, 3], [4, 8, 7], [6, 9, 1]$

Queries are interpreted as follows:

```
   a b k
   1 5 3
   4 8 7
   6 9 1
```

Add the values of k between the indices a and b inclusive:

```
index->  1 2 3  4  5 6 7 8 9 10
   [0,0,0, 0, 0,0,0,0,0, 0]
   [3,3,3, 3, 3,0,0,0,0, 0]
   [3,3,3,10,10,7,7,7,0, 0]
   [3,3,3,10,10,8,8,8,1, 0]
```

The largest value is 10 after all operations are performed.

Function Description

Complete the function *arrayManipulation* in the editor below.

arrayManipulation has the following parameters:

- *int n* - the number of elements in the array
- *int queries[q][3]* - a two dimensional array of queries where each *queries[i]* contains three integers, *a*, *b*, and *k*.

Returns

- *int* - the maximum value in the resultant array

Input Format

The first line contains two space-separated integers n and m, the size of the array and the number of operations.
Each of the next m lines contains three space-separated integers a, b and k, the left index, right index and summand.

Constraints

- $3 \leq n \leq 10^7$
- $1 \leq m \leq 2 * 10^5$
- $1 \leq a \leq b \leq n$
- $0 \leq k \leq 10^9$

Sample Input

```
5 3
1 2 100
2 5 100
3 4 100
```

Sample Output

```
200
```

Explanation

After the first update the list is `100 100 0 0 0`.
After the second update list is `100 200 100 100 100`.
After the third update list is `100 200 200 200 100`.

The maximum value is 200.

For example:

Input	Result
5 3 1 2 100 2 5 100 3 4 100	200

Answer: (penalty regime: 0 %)

Reset answer

```
#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> arr(n + 1); // Initialize the array with size n+1 (zero valued array)

    int size = queries.size(); //Size of the 'queries' 2D vector

    // Process each query and update the array accordingly
    for (int i = 0; i < size; i++) {
        int a = queries[i][0];
        int b = queries[i][1];
        int k = queries[i][2];

        arr[a - 1] += k; // Add k to the element at index a-1
        arr[b] -= k;     // Subtract k from the element at index b
    }

    long max_value = 0;
    long current_value = 0;

    // Calculate the cumulative sum to get the final values after all operations
    for (int i = 0; i < n; i++) {
        current_value += arr[i];
        max_value = max(max_value, current_value);
    }

    return max_value;
}

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);
```

```
66
67          vector<string> queries_row_temp = split(rtrim(queries_row_temp_temp));
68
69          for (int j = 0; j < 3; j++) {
70              int queries_row_item = stoi(queries_row_temp[j]);
71
72              queries[i][j] = queries_row_item;
73          }
74      }
75
76      long result = arrayManipulation(n, queries);
77
78      cout << result << "\n";
79
80      fout.close();
81
82      return 0;
83  }
84
85  string ltrim(const string &str) {
86      string s(str);
87
88      s.erase(
89          s.begin(),
90          find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
91      );
92
93      return s;
94  }
95
96  string rtrim(const string &str) {
97      string s(str);
98
99      s.erase(
100         find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
101         s.end()
102     );
103
104     return s;
105 }
106
107 vector<string> split(const string &str) {
108     vector<string> tokens;
109
110     string::size_type start = 0;
111     string::size_type end = 0;
112
113     while ((end = str.find(" ", start)) != string::npos) {
114         tokens.push_back(str.substr(start, end - start));
115
116         start = end + 1;
117     }
118
119     tokens.push_back(str.substr(start));
120
121     return tokens;
122 }
123
```

	Input	Expected	Got	
✔	5 3 1 2 100 2 5 100 3 4 100	200	200	✔

Passed all tests! ✔

▸ **Show/hide question author's solution (Cpp)**

Correct

Marks for this submission: 10.00/10.00.