| | |
|---|---|
| **Started on** | Wednesday, 21 February 2024, 7:49 PM |
| **State** | Finished |
| **Completed on** | Wednesday, 21 February 2024, 8:48 PM |
| **Time taken** | 58 mins 23 secs |
| **Marks** | 30.00/30.00 |
| **Grade** | **10.00** out of 10.00 (**100**%) |

**Question 1**

Correct

Mark 10.00 out of 10.00

The previous challenges covered Insertion Sort, which is a simple and intuitive sorting algorithm with a running time of $O(n^2)$. In these next few challenges, we're covering a *divide-and-conquer* algorithm called Quicksort (also known as *Partition Sort*). This challenge is a modified version of the algorithm that only addresses partitioning. It is implemented as follows:

**Step 1: Divide**

Choose some pivot element, $p$, and partition your unsorted array, $arr$, into three smaller arrays: $left$, $right$, and $equal$, where each element in $left < p$, each element in $right > p$, and each element in $equal = p$.

**Example**

$arr = [5, 7, 4, 3, 8]$

In this challenge, the pivot will always be at $arr[0]$, so the pivot is $5$.

$arr$ is divided into $left = \{4, 3\}$, $equal = \{5\}$, and $right = \{7, 8\}$.
Putting them all together, you get $\{4, 3, 5, 7, 8\}$. There is a flexible checker that allows the elements of $left$ and $right$ to be in any order. For example, $\{3, 4, 5, 8, 7\}$ is valid as well.

Given $arr$ and $p = arr[0]$, partition $arr$ into $left$, $right$, and $equal$ using the *Divide* instructions above. Return a 1-dimensional array containing each element in $left$ first, followed by each element in $equal$, followed by each element in $right$.

**Function Description**

Complete the *quickSort* function in the editor below.

quickSort has the following parameter(s):

- *int arr[n]:* $arr[0]$ is the pivot element

**Returns**

- *int[n]:* an array of integers as described above

**Input Format**

The first line contains $n$, the size of $arr$.
The second line contains $n$ space-separated integers $arr[i]$ (the unsorted array). The first integer, $arr[0]$, is the pivot element, $p$.

**Constraints**

- $1 \leq n \leq 1000$
- $-1000 \leq arr[i] \leq 1000$ where $0 \leq i < n$
- All elements are distinct.

**Sample Input**

```
STDIN       Function
-----       --------
5           arr[] size n =5
4 5 3 7 2   arr =[4, 5, 3, 7, 2]
```

**Sample Output**

```
3 2 4 5 7
```

**Explanation**

$arr = [4, 5, 3, 7, 2]$ *Pivot*: $p = arr[0] = 4$.
$left = \{\}$; $equal = \{4\}$; $right = \{\}$

$arr[1] = 5 > p$, so it is added to $right$.
$left = \{\}$; $equal = \{4\}$; $right = \{5\}$

$arr[2] = 3 < p$, so it is added to $left$.
$left = \{3\}$; $equal = \{4\}$; $right = \{5\}$

$arr[3] = 7 > p$, so it is added to $right$.
$left = \{3\}; equal = \{4\}; right = \{5, 7\}$

$arr[4] = 2 < p$, so it is added to $left$.
$left = \{3, 2\}; equal = \{4\}; right = \{5, 7\}$

Return the array $\{32457\}$.

The order of the elements to the left and right of $4$ needs to match the order of the original list.
**For example:**

| Input | Result |
|---|---|
| 5<br>4 5 3 7 2 | 3 2 4 5 7 |

**Answer:** (penalty regime: 0 %)

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

using namespace std;

vector<int> quickSort(vector<int> arr) {

    //Defining the pivot element
    int pivot = arr[0];

    //Defining the arrays to store,
    // values < pivot in array 'left',
    // values > pivot in array 'right',
    // values = pivot in array 'equal'
    vector<int> left, right, equal, returnArr;

    int arr_size = arr.size();

    for (int i = 0; i < arr_size; i++){
        if(arr[i] < pivot){
            //element is added to left array if value<pivot
            left.push_back(arr[i]);

        }else if(arr[i] > pivot){
            //element is added to right array if value>pivot
            right.push_back(arr[i]);

        }else{
            //element is added to equal array if value=pivot
            equal.push_back(arr[i]);
        }
    }

    int left_size = left.size();

    //Continue adding elements to the returnArray, from left, equal, right arrays respectively
    for (int i =0; i<left_size; i++){
        returnArr.push_back(left[i]);
    }

    int equal_size = equal.size();

    for (int i =0; i<equal_size; i++){
        returnArr.push_back(equal[i]);
    }

    int right_size = right.size();

    for (int i =0; i<right_size; i++){
        returnArr.push_back(right[i]);
    }

    //Returning the final array
```

```
    return returnArr;
}

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

    //Reading the input
    vector<int> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    //Calling the quickSort function
    vector<int> returnArr = quickSort(arr);

    //Outputting the returned array
    int new_len = returnArr.size();
    for (int i = 0; i < new_len; i++) {
        cout << returnArr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

| | Input | Expected | Got | |
|---|---|---|---|---|
| ✔ | 5<br>4 5 3 7 2 | 3 2 4 5 7 | 3 2 4 5 7 | ✔ |

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

Whenever George asks Lily to hang out, she's busy doing homework. George wants to help her finish it faster, but he's in over his head! Can you help George understand Lily's homework so she can hang out with him?

Consider an array of $n$ distinct integers, $arr = [a[0], a[1], \ldots, a[n-1]]$. George can swap any two elements of the array any number of times. An array is *beautiful* if the sum of $|arr[i] - arr[i-1]|$ among $0 < i < n$ is minimal.

Given the array $arr$, determine and return the minimum number of swaps that should be performed in order to make the array *beautiful*.

**Example**

$arr = [7, 15, 12, 3]$

One minimal array is $[3, 7, 12, 15]$. To get there, George performed the following swaps:

```
Swap      Result
      [7, 15, 12, 3]
3 7   [3, 15, 12, 7]
7 15  [3, 7, 12, 15]
```

It took $2$ swaps to make the array beautiful. This is minimal among the choices of beautiful arrays possible.

**Function Description**

Complete the *lilysHomework* function in the editor below.

lilysHomework has the following parameter(s):

- *int arr[n]:* an integer array

**Returns**

- *int:* the minimum number of swaps required

**Input Format**

The first line contains a single integer, $n$, the number of elements in $arr$. The second line contains $n$ space-separated integers, $arr[i]$.

**Constraints**

- $1 \leq n \leq 10^5$
- $1 \leq arr[i] \leq 2 \times 10^9$

**Sample Input**

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

**Sample Output**

```
2
```

**Explanation**

Define $arr' = [1, 2, 3, 5]$ to be the beautiful reordering of $arr$. The sum of the absolute values of differences between its adjacent elements is minimal among all permutations and only two swaps ($1$ with $2$ and then $2$ with $5$) were performed.

**For example:**

| Input | Result |
|---|---|
| 4<br>2 5 3 1 | 2 |
| 5<br>3 4 2 5 1 | 2 |

**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 'lilysHomework' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

int lilysHomework(vector<int> arr) {

    vector<pair<int, int>> pair1, pair2;
    int size = arr.size();

    for (int i = 0; i < size; i++){
        pair1.push_back({arr[i], i});
    }

    for (int i = 0; i < size; i++){
        pair2.push_back({arr[i], i});
    }

    //Obtaining the sorted arrays
    sort(pair1.begin(), pair1.end());
    sort(pair2.rbegin(), pair2.rend());

    int swaps_pair1 = 0;
    int swaps_pair2 = 0;

    //Finding the no of swaps needed in pair array 1 and pair array 2 respectively
    for (int i = 0; i < size; i++){
        if (pair1[i].second != i) {
            swap(pair1[i], pair1[pair1[i].second]);
            swaps_pair1++;
            i--;
        }
    }
    for (int i = 0; i < size; i++){
        if (pair2[i].second != i) {
            swap(pair2[i], pair2[pair2[i].second]);
            swaps_pair2++;
            i--;
        }
    }

    //Returning the minimum from those two no of swaps
    return min(swaps_pair1, swaps_pair2);
}

int main()
{
    string n_temp;
    getline(cin, n_temp);
```

```
60
61      int n = stoi(ltrim(rtrim(n_temp)));
62
63      string arr_temp_temp;
64      getline(cin, arr_temp_temp);
65
66      vector<string> arr_temp = split(rtrim(arr_temp_temp));
67
68      vector<int> arr(n);
69
70      for (int i = 0; i < n; i++) {
71          int arr_item = stoi(arr_temp[i]);
72
73          arr[i] = arr_item;
74      }
75
76      int result = lilysHomework(arr);
77
78      cout << result << "\n";
79
80      return 0;
81  }
82
83  string ltrim(const string &str) {
84      string s(str);
85
86      s.erase(
87          s.begin(),
88          find_if(s.begin(), s.end(), not1(ptr_fun<int, int>(isspace)))
89      );
90
91      return s;
92  }
93
94  string rtrim(const string &str) {
95      string s(str);
96
97      s.erase(
98          find_if(s.rbegin(), s.rend(), not1(ptr_fun<int, int>(isspace))).base(),
99          s.end()
100     );
101
102     return s;
103 }
104
105 vector<string> split(const string &str) {
106     vector<string> tokens;
107
108     string::size_type start = 0;
109     string::size_type end = 0;
110
111     while ((end = str.find(" ", start)) != string::npos) {
112         tokens.push_back(str.substr(start, end - start));
113
114         start = end + 1;
115     }
116
117     tokens.push_back(str.substr(start));
118
119     return tokens;
120 }
121
```

| | Input | Expected | Got | |
|---|---|---|---|---|
| ✔ | 4<br>2 5 3 1 | 2 | 2 | ✔ |

| | Input | Expected | Got | |
|---|---|---|---|---|
| ✔ | 5<br>3 4 2 5 1 | 2 | 2 | ✔ |

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

Sorting is useful as the first step in many different tasks. The most common task is to make finding things easier, but there are other uses as well. In this case, it will make it easier to determine which pair or pairs of elements have the smallest absolute difference between them.

**Example**

$arr = [5, 2, 3, 4, 1]$

Sorted, $arr' = [1, 2, 3, 4, 5]$. Several pairs have the minimum difference of $1$: $[(1, 2), (2, 3), (3, 4), (4, 5)]$. Return the array $[1, 2, 2, 3, 3, 4, 4, 5]$.

**Note**

As shown in the example, pairs may overlap.

Given a list of unsorted integers, $arr$, find the pair of elements that have the smallest absolute difference between them. If there are multiple pairs, find them all.

**Function Description**

Complete the *closestNumbers* function in the editor below.

closestNumbers has the following parameter(s):

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

**Returns**

- *int[]:* an array of integers as described

**Input Format**

The first line contains a single integer $n$, the length of $arr$.
The second line contains $n$ space-separated integers, $arr[i]$.

**Constraints**

- $2 \leq n \leq 200000$
- $-10^7 \leq arr[i] \leq 10^7$
- All $a[i]$ are unique in $arr$.

**Output Format**

**Sample Input 0**

```
10
-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854
```

**Sample Output 0**

```
-20 30
```

**Explanation 0**

*(30) - (-20) = 50*, which is the smallest difference.

**Sample Input 1**

```
12
-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 -520 -470
```

**Sample Output 1**

```
-520 -470 -20 30
```

**Explanation 1**

*(-470) - (-520) = 30 - (-20) = 50*, which is the smallest difference.

**Sample Input 2**

```
4
5 4 3 2
```

**Sample Output 2**

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

**Explanation 2**

Here, the minimum difference is *1*. Valid pairs are *(2, 3)*, *(3, 4)*, and *(4, 5)*.

Submissions:

196

Max Score:

25

Difficulty:

Easy

Rate This Challenge:

More

C++

**For example:**

| Input | Result |
|---|---|
| 10<br>-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 | -20 30 |
| 12<br>-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 -520 -470 | -520 -470 -20 30 |
| 4<br>5 4 3 2 | 2 3 3 4 4 5 |

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

vector<int> closestNumbers(vector<int> arr) {

    //Obtaining the sorted input array
    sort(arr.begin(), arr.end());

    //Initializing minimum difference
    int minDiff = INT_MAX;

    //Result vector
    vector<int> returnArr;

    int size = arr.size();

```

```
    for (int i = 1; i < size; i++) {

        //Finding the absolute minimum difference
        int diff = abs(arr[i] - arr[i - 1]);

        //Storing pairs with the difference
        if (diff < minDiff) {
            minDiff = diff;
            returnArr.clear();
            returnArr.push_back(arr[i - 1]);
            returnArr.push_back(arr[i]);

        } else if (diff == minDiff) {
            returnArr.push_back(arr[i - 1]);
            returnArr.push_back(arr[i]);
        }
    }

    return returnArr;
}

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;
    }

    vector<int> result = closestNumbers(arr);

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

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

    cout << "\n";

    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 | |
|---|---|---|---|---|
| ✔ | 10<br>-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 | -20 30 | -20 30 | ✔ |
| ✔ | 12<br>-20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 -520 -470 | -520 -470 -20 30 | -520 -470 -20 30 | ✔ |
| ✔ | 4<br>5 4 3 2 | 2 3 3 4 4 5 | 2 3 3 4 4 5 | ✔ |

Passed all tests! ✔

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

Correct

Marks for this submission: 10.00/10.00.