Started on	Tuesday, 20 February 2024, 10:25 PM
State	Finished
Completed on	Tuesday, 20 February 2024, 11:25 PM
Time taken	1 hour
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 4 5 3 7 2	3 2 4 5 7

Answer: (penalty regime: 0 %)

```cpp
#include <iostream>
#include <vector>

using namespace std;

// Function to perform quicksort on a vector of integers
vector<int> quickSort(vector<int> arr) {
    // Get the length of the array
    int len = arr.size();

    // Base case: if the array has 1 or fewer elements, it is already sorted
    if (len <= 1)
        return arr;

    // Choose the first element of the array as the pivot
    int pivot = arr[0];
    vector<int> left, right, equal;

    // Partition the array into three parts: elements less than the pivot, equal to the pivot, and greater
    for (int i = 0; i < len; i++) {
        if (arr[i] < pivot){
            left.push_back(arr[i]); // Elements less than the pivot go to the left partition
        }
        else if (arr[i] > pivot){
            right.push_back(arr[i]); // Elements greater than the pivot go to the right partition
        }
        else{
            equal.push_back(arr[i]); // Elements equal to the pivot go to the equal partition
        }
    }

    // Concatenate the left, equal, and right partitions to form the sorted array
    left.insert(left.end(), equal.begin(), equal.end());
    left.insert(left.end(), right.begin(), right.end());
    return left; // Return the sorted array
}

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

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

    // Sort the array using quicksort
    vector<int> sortedArr = quickSort(arr);

    // Display the sorted array
```

	Input	Expected	Got	
✓	5 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 2 5 3 1	2
5 3 4 2 5 1	2

Answer: (penalty regime: 0 %)

Reset answer

```cpp
#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) {
    int min_swaps = INT_MAX; // Initialize the minimum number of swaps to a large value
    vector<int> sorted(arr);
    sort(sorted.begin(), sorted.end());

    // Iterate through the array twice: once in ascending order and once in descending order
    for (size_t rev = 0; rev < 2; ++rev) {
        int current_swaps = 0; // Initialize the current number of swaps to 0

        // Reverse the sorted array if needed
        if (rev) {
            reverse(sorted.begin(), sorted.end());
        }

        vector<int> arr_new(arr);
        unordered_map<int, size_t> value_to_position; // Map to store the position of each element in the o

        // Populate the map with the positions of elements in the original array
        for (size_t i = 0; i < arr.size(); ++i) {
            value_to_position[arr[i]] = i;
        }

        // Iterate through the array to perform swaps
        for (size_t i = 0; i < arr.size(); ++i) {
            // If the element is already in its correct position, continue to the next element
            if (arr_new[i] == sorted[i]) {
                continue;
            }

            // Perform the swap
            int current_element = arr_new[i];
            int sorted_element = sorted[i];
            swap(arr_new[i], arr_new[value_to_position[sorted_element]]);
            ++current_swaps; // Increment the swap count

            // Update the position of the swapped elements in the map

```

	Input	Expected	Got	
✓	4 2 5 3 1	2	2	✓
✓	5 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 -20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854	-20 30
12 -20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 -520 -470	-520 -470 -20 30
4 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) {
    // Sort the array in ascending order
    sort(arr.begin(), arr.end());

    // Initialize variables to store the minimum difference and the resulting pairs
    int minDiff = abs(arr[1] - arr[0]); // Initialize minDiff with the difference between the first two ele
    vector<int> result;

    // Iterate through the array to find the minimum difference
    int len=arr.size();
    for (int i = 1; i < len - 1; ++i) {
        int diff = abs(arr[i + 1] - arr[i]); // Calculate the difference between adjacent elements
        if (diff < minDiff) {
            minDiff = diff; // Update minDiff if a smaller difference is found
        }
    }

    // Iterate through the array again to find pairs with the minimum difference
    for (int i = 0; i < len - 1; ++i) {
        int diff = abs(arr[i + 1] - arr[i]); // Calculate the difference between adjacent elements
        if (diff == minDiff) {
            result.push_back(arr[i]); // Add the pair to the result vector
            result.push_back(arr[i + 1]);
        }
    }

    return result;
}

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

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

    string arr_temp_temp;
```

	Input	Expected	Got	
✓	10 -20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854	-20 30	-20 30	✓
✓	12 -20 -3916237 -357920 -3620601 7374819 -7330761 30 6246457 -6461594 266854 -520 -470	-520 -470 -20 30	-520 -470 -20 30	✓
✓	4 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.