Dutch national flag problem

The solution to the **Dutch national flag problem** is probably what we need to 'properly deal' with the many-duplicates test cases that previously plagued the older implementation of randomized Quicksort algorithm that was displayed in VisuAlgo since 2011-2021 (fixed in January 2022 using a hacking solution: throwing elements equal to the pivot to the left/right region with 50-50 chance)...

This problem is proposed by Edsger Dijkstra (he is a Dutch/the Netherlands). The flag of the Netherlands consists of three colors: red, white, and blue.

Given integers (or anything else that can be compared), the task is to arrange them such that all integers that are strictly smaller than the pivot is grouped in the red/top/left section, all integers that are EQUAL to the pivot is grouped in the white/middle section, and all integers that are strictly greater than the pivot is grouped in the blue/bottom/right section.

The solution to this problem is of interest for designing randomized Quicksort algorithm that can properly handle duplicates.

1. Fill in the blanks

(8 marks)

Suppose that you have implemented this C++17 function below

```
pair<int, int> threewaypartition(int[] a, int lo, int hi) {
  int p = a[lo]; // p = a[lo] (first index in [lo..hi]) is always the pivot
  // see below later
}
```

where given an integer array **a** (most likely not sorted) and two indices **lo** and **hi** (**lo <= hi**), this `threewaypartition` will partition **a[lo..hi]** into 3 regions: [<p, ==p, >p] and returns two indices, the starting (**st**) and ending (**en**) indices of integers that are ==p. For this implementation, the pivot is always **p = a[lo]**.

For example, on `a = [3, 2, 3, 4, 3, 3, 1, 5, 7]`, `threewaypartition(a, 0, 8)` will return [2, 5] (that is, st = 2 and en = 5) because after partition, we will have 2 integers: 2, 1 or 1, 2 strictly smaller than pivot p = a[0] = 3, and 3 integers strictly greater than 3: 4, 5, 7 (in any order, there are 3! = 6 possibilities). Indices [2, 3, 4, 5] will contain those 4 copies of same-valued 3.

Now the sub-questions.

If `tc1 = {1, 2, 3, 4, 5, 6, 7}`, then `auto [st1, en1] = threewaypartition(tc1, 0, 6)` will return st1 = __1__ and en1 = __2__.

If `tc2 = {4, 2, 3, 1, 5, 6, 7}`, then `auto [st2, en2] = threewaypartition(tc2, 0, 6)` will return st2 = __3__ and en2 = __4__.

If `tc3 = {4, 2, 3, 4, 1, 5, 6, 7, 4}`, then `auto [s3t, en3] = threewaypartition(tc3, 0, 8)` will return st3 = __5__ and en3 = __6__.

If `tc4 = {7, 7, 7, 7, 7, 7, 7}`, then `auto [st4, en4] = threewaypartition(tc4, 0, 6)` will return st4 = __7__ and en4 = __8__.

Enter the correct answer below.

1 Please enter a number for this text box.

2 Please enter a number for this text box.

3 Please enter a number for this text box.

4 Please enter a number for this text box.

5 Please enter a number for this text box.

6 Please enter a number for this text box.

7 Please enter a number for this text box.

8 Please enter a number for this text box.

2.

This Dutch national flag problem is "classic" and will be integrated in future CS2040C classes and in VisuAlgo /sorting module by default after this semester is over.

To minimize cheating because this question is so easy to Google, in this midterm, you will not be asked to come up with the algorithm to solve the Dutch national flag problem, but rather, you will be tested on how to turn this specific pseudo-code (from Wikipedia):

```
procedure three-way-partition(A : array of values, mid : value):
    i ← 0
    j ← 0
    k ← size of A - 1

    while j <= k:
        if A[j] < mid:
            swap A[i] and A[j]
            i ← i + 1
            j ← j + 1
        else if A[j] > mid:
            swap A[j] and A[k]
            k ← k - 1
        else:
            j ← j + 1
```

into a fully working C++17 code that has the following function signature (your code must be working correctly; you are allowed to test your code locally):

```
pair<int, int> threewaypartition(int a[], int lo, int hi) {
  int p = a[lo]; // p = a[lo] (first index in [lo..hi]) is always the pivot
  int st = -1, en = -1; // overwrite this with your answer
  // translate the pseudo-code above to a working C++17 code of this style
  // use the essay box for this part
  // your code will be manually graded based on correctness
  // be careful of the subtle differences between the Wikipedia version and ours



  return {st, en}; // return st/en ind(ices) of a that are == pivot as a pair
}
```

(15 marks)

Enter your answer here

Character Limit: 2000

3. Fill in the blanks

(4 marks)

The three-way-partition as described in Wikipedia

```
procedure three-way-partition(A : array of values, p : value):
```

runs in O(__1__) regardless of the type of input of array A (sorted, nearly-sorted - whichever your definition, totally random, etc). Note that A contains **n** integers, thus describe your analysis in terms of **n**.

Enter the correct answer below.

1

4. Fill in the blanks

(6 marks)

Now that you have implemented `threewaypartition`, use it to fix the current implementation of Quicksort as currently shown in VisuAlgo (as of 16 February 2022), this question will no longer be relevant in the future.

```
// upgraded version of https://visualgo.net/en/sorting?slide=12-8
void quickSort(int a[], int low, int high) {
  if (low < high) {
    auto [st, en] = threewaypartition(a, low, high);
    // WARNING: do NOT put any space in your answer
    // use variable st or en in your answer, not any other variable names
    quickSort(a, low, ___1___);
    quickSort(a, ___2___, high);
  }
}
```

Enter the correct answer below.

1 [ ]

2 [ ]

5. Now if randomized Quicksort is using `threewaypartition` as discussed in this section, then what is now the **best** case input for such implementation?

There is one unique best answer among the possibly overlapping options below. Pick that best answer.

(7 marks)

- when all n integers are already sorted in ascending order (no duplicates allowed).
- when all n integers are equal
- when all n integers are already sorted in non-decreasing order (duplicates allowed).
- when all n integers are nearly sorted (very low inversion count) in non-decreasing order (duplicates allowed).
- when all n integers are nearly sorted (very low inversion count) in ascending order (no duplicates allowed).
- when all n integers are distinct

Placing 32-bit unsigned Integers into Boxes

You are given a list of **n** distinct 32-bit unsigned integers. Note: unsigned 32-bit integers = non-negative integers [0..2^32-1].

You are also given a sequence of **n** fill-in-the-blank boxes with **n**-1 preset inequality signs inserted in-between two successive boxes. These inequality signs are placed in a character array `signs` of size **n**-1. Notice that there are two possible inequality options ('<' or '>') between two successive boxes.

Your job in this section is to design an algorithm to place these **n** distinct integers into **n** fill-in-the-blank boxes so that all those **n-1** inequalities are all satisfied.

Example 1: If you are given **n** = 4 distinct integers {2, 5, 1, 0} and `signs` = {'<', '>', '<'}, then it describes this problem: [box1] < [box2] > [box3] < [box4]. One possible placement is [0] < [5] > [1] < [2].

Example 2: If you are given the same **n** = 4 distinct integers {2, 5, 1, 0} but `signs` = {'>', '>', '<'}, then it describes this problem: [box1] > [box2] > [box3] < [box4]. One possible placement is [5] > [2] > [0] < [1].

6. Fill in the blanks

(9 marks)

To test your understanding of this simple problem, let's try these test cases.

n = 2, the integers are {7, 10}, so [1] > [2].

n = 3, the integers are {7, 11, 8}, so [11] (already filled in for you for this example) > [3] < [4].

n = 5, the integers are {3, 7, 5, 6, 1}, so [5] < [6] < [7] < [8] < [9].

Enter the correct answer below.

1 [ ] Please enter a number for this text box.

2 [ ] Please enter a number for this text box.

3 [ ] Please enter a number for this text box.

4 [ ] Please enter a number for this text box.

5 [ ] Please enter a number for this text box.

6 [ ] Please enter a number for this text box.

7 [ ] Please enter a number for this text box.

8 Please enter a number for this text box.

9 Please enter a number for this text box.

7. For Subtask 1, we simplify the problem and assume that **n** = 2 (always 2), i.e., you are given 2 random distinct non-negative integers and 2 fill-in-the-blank boxes with either < or > in between the two boxes.

Design an O(**1**) algorithm to fully solve this Subtask 1 regardless of the input.

(3 marks)

Enter your answer here

Characters Limit: 200

8. For Subtask 1, **n** = 2, is the answer always unique (there is only one possible placement of **n** distinct integers to the **n** boxes so that all **n**-1 inequalities are satisfied)?

(1 mark)

- No
- Yes

9. For Subtask 2, **n** = 3 (always 3), i.e., you are given 3 random distinct non-negative integers and 3 fill-in-the-blank boxes with either < or > in between each pair of the three boxes, thus it can be [ ] < [ ] < [ ], [ ] < [ ] > [ ], [ ] > [ ] < [ ], or [ ] > [ ] > [ ].

Design an O(**1**) algorithm to fully solve this Subtask 2 regardless of the input.

(4 marks)

Enter your answer here

Characters Limit: 200

10. For Subtask 2, **n** = 3, is the answer always unique (there is only one possible placement of **n** distinct integers to the **n** boxes so that all **n**-1 inequalities are satisfied)?

(1 mark)

- No
- Yes

11. For Subtask 3, 2 <= **n** <= 200,000. However, this time all the **n-1** inequalities in between of two successive fill-in-the-blank boxes are **always** <.

Design an O(**n**) algorithm to fully solve this Subtask 3 regardless of the input (**n** random distinct non-negative integers) for full marks. If you can only come up with O(**n log n**) or O(**n^2**) or any other weaker time complexities, you will only get partial marks.

(7 marks)

Enter your answer here

Character Limit: 1000

12. Design the proper full algorithm that will work for any **n** random input distinct 32-bit unsigned integers and any **n-1** inequalities (mix of '<' and '>') between successive boxes.

Then, analyze the time complexity of your algorithm in terms of **n**.

Hint: The best algorithm to solve this problem should run under 1s (<= 10^8 operations) for 2 <= **n** <= 200,000.

PS: This question will first be graded based on correctness, and if ties, by the runtime speed of correct algorithm. That is, a correct but slow algorithm worth more partial marks than a fast but totally wrong algorithm. Look at the previous subtasks to give you some ideas on how to proceed.

(15 marks)

Enter your answer here

Character Limit: 2000

Find Next Larger (on the right)

You are given an array **a** of **n** distinct integers and each integer is positive and not more than 2^31-1.

Your job is as follows: For each integer **x** in **a**, identify the 0-based index of the next larger integer of **x** that resides on the right of **x** in **a**. If a particular integer **x** has no next larger integer, report -1 for that integer **x**.

For example, if a = {1, 4, 3, 5, 2}, we should output {1, 3, 3, -1, -1} because:
1 (index 0) has 4 (index 1) as its next larger integer on its right,
4 (index 1) and 3 (index 2) both have 5 (index 3) as their next larger integer on their right,
5 (index 3) has nothing on its right that is larger than 5, so we put -1 for 5.
2 (index 4) is the last element, it has nothing on its right, so we also put -1 for 2.

13. Fill in the blanks

(5 marks)

To test your understanding, please answer the following test cases.

If a = {5, 4, 3, 2, 1}, then the output should be {-1, -1, __1__, -1, -1}.

If a = {1, 2, 3, 4, 5}, then the output should be {1, 2, __2__, __3__, -1}.

If a = {21, 7, 6, 20}, then the output should be {-1, __4__, __5__, -1}.

Enter the correct answer below.

1 Please enter a number for this text box.

2 Please enter a number for this text box.

3 Please enter a number for this text box.

4 Please enter a number for this text box.

5 Please enter a number for this text box.

14. Design an algorithm that will work for any **n** random distinct integers and each integer is positive and not more than 2^31-1.

Then, analyze the time complexity of your algorithm in terms of **n**.

Hint: The best algorithm to solve this problem should run under 1s (<= 10^8 operations) for 2 <= **n** <= 200,000 and *it uses a stack* data structure to achieve this performance.

PS: This question will first be graded based on correctness, and if ties, by the runtime speed of correct algorithm. That is, a correct but slow algorithm worth more partial marks than a fast but totally wrong algorithm

(15 marks)