Asymptotics of Weighted Quick Unions

Here is a video walkthrough of all parts of this problem.

For this problem, we will be addressing the asymptotics of Weighted Quick Unions! For all big Ω and big O bounds, give the *tightest* bound possible.

(a) Suppose we have a Weighted Quick Union (WQU) without path compression with N elements.

1. What is the runtime, in big Ω and big O, of `isConnected`?

   Ω(______), O(______)

2. What is the runtime, in big Ω and big O, of `connect`?

   Ω(______), O(______)

Solution:

1. $\Omega(1)$, $O(log(N))$
2. $\Omega(1)$, $O(log(N))$

(b) Suppose for the following problem we add the method `addToWQU` to the WQU class. Simply put, the method takes in a list of `elements` and randomly `connects` elements together. Assume that all the `elements` are disconnected before the method call, and the `connect` method works as described in lecture.

```
void addToWQU(int[] elements) {
        int[][] pairs = pairs(elements);
        pairs = shuffle(pairs);
        for (int[] pair: pairs) {
            connect(pair[0], pair[1]);
        }
}
```

In a bit more detail, the `pairs` method accepts an array and returns an ordered array of *all* unique pairs, where each pair is a 2 element array. For instance,

```
pairs(new int[]{1, 2, 3})
```

would return

```
{{1, 2}, {1, 3}, {2, 3}}
```

The `shuffle` method shuffles the ordering of the elements, and returns a new array. For instance,

```
shuffle(new int[]{{1, 2}, {1, 3}, {2, 3}})
```

might return

```
{{1, 3}, {2, 3}, {1, 2}}
```

Assume, for simplicity, that `pairs` and `shuffle` run in constant time (admittedly this couldn't be the case, but assume so for the sake of this problem).

What is the runtime of `addToWQU` in big O? For this and all remaining subparts you may write your answer in terms of `N`, where `N` is `elements.length`.

`addToWQU` runtime: O(______)

Solution:
`addToWQU` runtime: $O(N^2 log(N))$

For the remainder of this problem, suppose we are using the modified version of `addToWQU` as defined below. Note the only difference is the added if condition.

```
void addToWQU(int[] elements) {
        int[][] pairs = pairs(elements);
        pairs = shuffle(pairs);
        for (int[] pair: pairs) {
            if (size() == elements.length) {
                return;
            }
            connect(pair[0], pair[1]);
        }
}
```

Assume the method `size` calculates the size of the largest connected component and runs in constant time (this can be easily implemented with adding an instance variable to the class).

(c) What is the runtime of `addToWQU` in big Ω and big O?

Ω(______), O(______)

Solution:
$\Omega(N)$, $O(N^2 log(N))$

(d) Let us define a **matching size connection** as `connecting` two trees, i.e. components in a WQU, together of matching size. For instance, suppose we have two trees, one with values 1 and 2, and another with the values 3 and 4. Calling `connect(1, 4)` is a matching size connection since both trees are the same size.

What is the **minimum** and **maximum** number of matching size connections that can occur after executing `addToWQU`. Assume `N`, i.e. `elements.length`, is a power of two. Your answers should be exact.

minimum: _____, maximum: _____

Solution:
minimum: 1, maximum: `N - 1`