

1 Asymptotics Introduction

Give the runtime of the following functions in Θ notation. Your answer should be as simple as possible with no unnecessary leading constants or lower order terms.

```
private void f1(int N) {
    for (int i = 1; i < N; i++) {
        for (int j = 1; j < i; j++) {
            System.out.println("hello tony");
        }
    }
}
```

$\Theta(N^2)$

```
private void f2(int N) {
    for (int i = 1; i < N; i *= 2) {
        for (int j = 1; j < i; j++) {
            System.out.println("hello hannah");
        }
    }
}
```

}

$\Theta(N)$ $1+2+4+\dots+8 = \Theta(N)$

2 Disjoint Sets

For each of the arrays below, write whether this could be the array representation of a weighted quick union with path compression and explain your reasoning.

i: 0 1 2 3 4 5 6 7 8 9

A. a[i]: 1 2 3 0 1 1 1 4 4 5 X

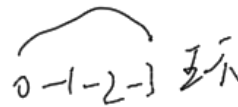
B. a[i]: 9 0 0 0 0 0 9 9 9 -10 X

C. a[i]: 1 2 3 4 5 6 7 8 9 -10 X

D. a[i]: -10 0 0 0 0 1 1 1 6 2 X

E. a[i]: -10 0 0 0 0 1 1 1 6 8 X

F. a[i]: -7 0 0 1 1 3 3 -3 7 X



Handwritten notes explaining the reasoning for the 'X' marks:

- For A: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ is a path, but 3 is the root. 9 is the root of 0's path.
- For B: 12 is the root of 4, 5, 6, 7, 8. 9 is the root of 0's path.
- For C: 12 is the root of 4, 5, 6, 7, 8. 9 is the root of 0's path.
- For D: 12 is the root of 4, 5, 6, 7, 8. 9 is the root of 0's path.
- For E: 12 is the root of 4, 5, 6, 7, 8. 9 is the root of 0's path.
- For F: 12 is the root of 4, 5, 6, 7, 8. 9 is the root of 0's path.

3 Asymptotics of Weighted Quick Unions

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`?

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

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

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

- (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.

```

1 void addToWQU(int[] elements) {
2     int[][] pairs = pairs(elements);
3     pairs = shuffle(pairs);
4     for (int[] pair: pairs) {
5         connect(pair[0], pair[1]);
6     }
7 }
```

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,

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

would return

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

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

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

might return

```
1 {{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: $O(N^2 \log N)$

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.

```

1 void addToWQU(int[] elements) {
2     int[][] pairs = pairs(elements);
3     pairs = shuffle(pairs);
4     for (int[] pair: pairs) {
5         if (size() == elements.length) {
6             return;
7         }
8         connect(pair[0], pair[1]);
9     }
10 }
```

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 ?

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

- (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: $\frac{N}{2}$, maximum: $N-1$

$$\frac{N}{2} + \frac{N}{4} + \dots + 1$$