CS 61B
Spring 2024

Asymptotics, Disjoint Sets

Exam-Level 05: February 19, 2024

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.

```java
private void f1(int N) {
    for (int i = 1; i < N; i++) {
        for (int j = 1; j < i; j++) {
            System.out.println("shreyas 1.0");
        }
    }
}
```

$\Theta(\_\_\_)$

```java
private void f2(int N) {
    for (int i = 1; i < N; i *= 2) {
        for (int j = 1; j < i; j++) {
            System.out.println("shreyas 2.0");
        }
    }
}
```

$\Theta(\_\_\_)$

Solution:

```java
private void f1(int N) {
    for (int i = 1; i < N; i++) {
        for (int j = 1; j < i; j++) {
            System.out.println("shreyas 1.0");
        }
    }
}
```

$\Theta(N^2)$

```java
private void f2(int N) {
    for (int i = 1; i < N; i *= 2) {
        for (int j = 1; j < i; j++) {
            System.out.println("shreyas 2.0");
        }
    }
}
```

$\Theta(N)$

Explanation (1): The inner loop does up to i work each time, and the outer loop increments i each time. Summing over each loop, we get that $1 + 2 + 3 + 4 + \ldots + N = \Theta(N^2)$.

Explanation (2): The inner loop does i work each time, and we double i each time until reaching N. $1 + 2 + 4 + 8 + \ldots + N = \Theta(N)$

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. **Break ties by choosing the smaller integer to be the root.**

```
        i:   0   1   2   3   4   5   6   7   8   9
        ------------------------------------------
A. a[i]:   1   2   3   0   1   1   1   4   4   5
B. a[i]:   9   0   0   0   0   0   9   9   9 -10
C. a[i]:   1   2   3   4   5   6   7   8   9 -10
D. a[i]: -10   0   0   0   0   1   1   1   6   2
E. a[i]: -10   0   0   0   0   1   1   1   6   8
```

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

Solution: There are three criteria here that invalidates a representation:
(1) If there is a cycle in the parent-link.
(2) For each parent-child link, the tree rooted at the parent is smaller than the tree rooted at the child before the link (you would have merged the other way around).
(3) The height of the tree is greater than $\log_2 n$, where n is the number of elements.

Therefore, we have the following verdicts.
A. Impossible: has a cycle 0-1, 1-2, 2-3, and 3-0 in the parent-link representation.
B. Impossible: the nodes 1, 2, 3, 4, and 5 must link to 0 when 0 is a root; hence, 0 would not link to 9 because 0 is the root of the larger tree.
C. Impossible: tree rooted at 9 has height 9 $> \log_2 10$.
D. Possible: 8-6, 7-1, 6-1, 5-1, 9-2, 3-0, 4-0, 2-0, 1-0.
E. Impossible: tree rooted at 0 has height 4 $> \log_2 10$.
F. Impossible: tree rooted at 0 has height 3 $> \log_2 7$.

3 Asymptotics of Weighted Quick Unions

Note: 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. $\Omega(1)$, $O(\log(N))$

In the best-case, if we're checking if a and b are connected, a is the root, and b is a node directly below the root. This means we only have to traverse one edge of the tree, which is constant time. In the worst-case, we have to traverse the entire height of the tree, and Weighted Quick Union gives us a worst-case height of $\log N$, hence the upper-bound of $O(\log N)$. Similar logic applies to the `connect` method.

(b) Suppose we add the method `addToWQU` to a WQU without path compression. The method takes in a list of `elements` and `connects` them in a random order, stopping when all elements are connected. Assume that all the `elements` are disconnected before the method call.

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

The `pairs` method takes in a list of `elements` and generates all possible pairs of elements in a random order. For example, `pairs([1, 2, 3])` might return `[[1, 3], [2, 3], [1, 2]]` or `[[1, 2], [1, 3], [2, 3]]`.

The `size` method calculates the size of the largest component in the WQU.

Assume that `pairs` and `size` run in constant time.

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

Ω(______), O(______)

Hint: Consider the number of calls to `connect` *in the best case and worst case. Then, consider the best/worst case time complexity for one call to* `connect`*.*

Solution: $\Omega(N)$, $O(N^2\log(N))$
Note that the if-statement terminates the method when the disjoint set becomes fully connected. The best case occurs when there is a sequence of pairs such that each connect() operation takes constant time and the tree becomes connected as quickly as possible. This will happen if we have a sequence $(0,1),(0,2),...,(0,N-1)$, which consists of $N-1$ operations each taking constant time (ie. the best case for `connect` from part a). Note that long running-times occur when an element (e.g. 0) is not connected for many operations, and in the worst-case, 0 is not connected until the last N operations. This results in a tree of height $\log N$ and requires up to N^2-N+1 iterations.

(c) Let us define a **matching size connection** as `connecting` two components in a WQU of equal 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 have 2 elements.

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`
The minimum number occurs for the sequence above, where there is only one matching size connection: (0, 1). The maximum number is a bit more tricky, but occurs if we pairwise-connect the elements together, then pairwise connect those, and so on. An example for N=8 elements is as follows: (0, 1), (2, 3), (4, 5), (6, 7), (0, 2), (4, 6), (0, 4). In general, there are $N/2$ matching-size connections of size 1, $N/4$ matching-size connections of size 2, and so on, up until one matching-size connection of size $N/2$. This is the sum $N/2+N/4+N/8+...+2+1$, which simplifies to $N-1$.