# Asymptotics & Disjoint Sets

Exam Prep 06

# Announcements

- Weekly Survey 5 due on **Monday 9/25**
- Proj 1C due on **Monday 9/25**
- Lab 6 due on **Friday 9/29**
- HW 2 due on **Wednesday 10/4**

# Content Review

# Asymptotics

Asymptotics allow us to evaluate the performance of programs using math. We ignore all constants and only care about the total work done in regards to the input as it grows very large (usually defined as N)

Big O - The upper bound in terms of the input. In other words, if a function has big O in f(x), we say that it could grow at most as fast as f(x), but it could grow more slowly.

Big Ω - The lower bound in terms of the input. In other words, if a function has big Ω in f(x), we say that it could grow at least as slowly as f(x), but it could grow more quickly.

Big Θ - The tightest bound, which only exists when the tightest upper bound and the tightest lower bound converge to the same value.

# Common Orders of Growth

- $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(c^n)$
- Alternatively: constant < logarithmic < linear < nlogn < quadratic < exponential
- Desmos example here
    - Constants don't matter in the long run!
- Fun sums:

$$1 + 2 + 3 + \ldots + N = \Theta(N^2)$$

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

# Tightest Bound?

- Refers to the most "specific" bound possible
- Ex. Given `f(n) = 2n + 5`, we could say that `f(n)` $\in$ `O(`$n^n$`)`, but that doesn't tell us very much
  - a lot of functions are upper bounded by `O(`$n^n$`)` (grows really fast!)
  - A better, tighter bound would be `f(n)` $\in$ $\Theta$`(n)`

# Best vs. Worst Case

- NOT based on size of the input; more about inputs that cause specific behavior
- Represented with tight bound Θ because they should be consistent (always run in the same time)
- Look out for branching statements, loop conditions, breaks
- Best/worst case vs. lower/upper bound analogy: how much does it cost to eat at a restaurant?
    - Best/worst-case: "the cheapest thing on the menu is $5 and the most expensive is $50"
    - Lower/upper bound: "every item is at least $5 and at most $50" (credit: Alex Schedel)

```
public static void example(int N) {
     while (N > 0) {
          if (func(N)) {
               break;
          }
          N -= 1;
     }
}
```

Best case: Θ(1), where N = some int for which func(N) is immediately true

Worst case: Θ(N), where N = some int for which func(N), func(N - 1), …, func(1) are all false

# Disjoint Sets, also known as Union Find

```
public interface DisjointSet {
     void connect (x, y); // Connects nodes x and y (you may also see union)
     boolean isConnected(x, y); // Returns true if x and y are connected
}
```

QuickFind uses an array of integers to track which set each element belongs to.

QuickUnion stores the parent of each node rather than the set to which it belongs and merges sets by setting the parent of one root to the other.

WeightedQuickUnion does the same as QuickUnion except it decides which set is merged into which by size (merge smaller into larger), reducing stringiness.

WeightedQuickUnion with Path Compression sets the parent of each node to the set's root whenever find() is called on it.

# Disjoint Sets Asymptotics

```
public interface DisjointSet {
     void connect (x, y); // Connects nodes x and y (you may also see union)
     boolean isConnected(x, y); // Returns true if x and y are connected
}
```

| Implementation | Constructor | connect() | isConnected() |
|---|---|---|---|
| QuickUnion | Θ(N) | O(N) | O(N) |
| QuickFind | Θ(N) | O(N) | O(1) |
| Weighted Quick Union | Θ(N) | O(log N) | O(log N) |
| WQU with Path Compression | Θ(N) | O(log N)<br>Θ(1)-ish amortized | O(log N)<br>Θ(1)-ish amortized |

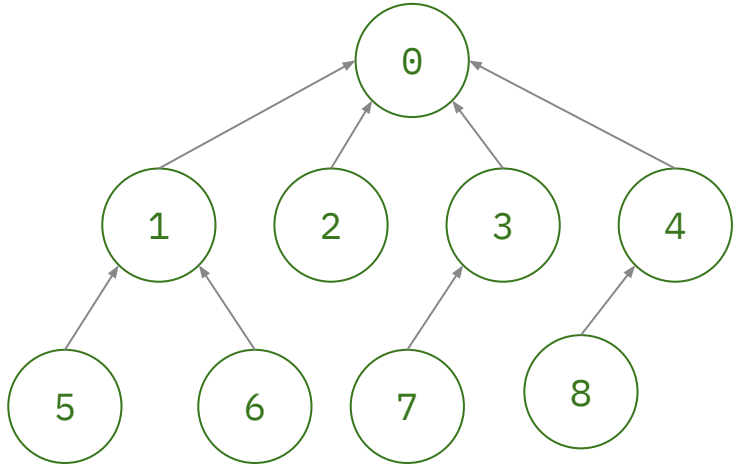
* we don't really talk about QU/QF in application, more to show the asymptotic motivation for WQU

# Disjoint Sets Representation

- We can use a single array to represent our disjoint set when implementing `connect()` optimally (ie. `WeightedQuickUnion`)

- `arr[i]` contains the parent of element i in the set; the index of a root contains `-(# elements in set rooted at that index)`

```
[-9, 0, 0, 0, 0, 1, 1, 3, 4]
```



# Worksheet

# 1A Asymptotics Introduction

```
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");
        }
    }
}
```

# 1A Asymptotics Introduction

```
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");
        }
    }
}
```

| i | 1 | 2 | … | N-2 | N-1 |
|---|---|---|---|---|---|
| Work per i | 0 | 1 | … | N-3 | N-2 |

$1 + 2 + 3 + 4 + \ldots N-1 = \Theta(N^2)$

# 1B Asymptotics Introduction

```
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");
        }
    }
}
```

# 1B Asymptotics Introduction

```
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");
        }
    }
}
```

| i | 1 | 2 | 4 | … | N/4 | N/2 |
|---|---|---|---|---|---|---|
| Work per i | 0 | 1 | 3 | … | N/4 - 1 | N/2 - 1 |

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

# 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.

| | 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 |

# 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.

| | 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 |

A. Impossible: cycle 0-1, 1-2, 2-3, and 3-0 in the parent-link representation.
B. Impossible: 1, 2, 3, 4, and 5 must link to 0 when 0 is a root; 0 would not link to 9 because 0 is the root of the larger tree.
C. Impossible: tree rooted at 9 has height 9 >lg 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 >lg 10.
F. Impossible: tree rooted at 0 has height 3 >lg 7.

# 3A Asymptotics of Weighted Quick Unions

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

# 3A Asymptotics of Weighted Quick Unions

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(1), O(log(N))
2. What is the runtime, in big Ω and big *O*, of `connect`? Ω(1), O(log(N))

# 3B Asymptotics of Weighted Quick Unions

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

# 3B Asymptotics of Weighted Quick Unions

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

Best case: $\Omega(N)$
Connect all elements in first N calls, with $O(1)$ `connect`

Worst case: $O(N^2 \log(N))$
One isolated element until last N calls → $N^2$ - N iterations, log N `connect`

# 3C Asymptotics of Weighted Quick Unions

Define a **matching size connection** as `connecting` two trees, i.e. components in a WQU, together of matching 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.

# 3C Asymptotics of Weighted Quick Unions

Define a **matching size connection** as `connecting` two trees, i.e. components in a WQU, together of matching 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.

Min: 1
Only one matching-size connection at beginning

Max: N - 1
Connect equal-size tree pairs until all are connected

# CS 61B Fall 2023

# Asymptotics, Disjoint Sets

Exam-Level 6: September 25, 2023

## 1 Asymptotics Introduction

Give the runtime of the following functions in $\Theta$ 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("shreyas 1.0");
        }
    }
}
```

$\Theta(N^2)$

$0+1+2+\ldots+N-2$

$= \frac{(N-1)(N-2)}{2} = \Theta(N^2)$

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

$\Theta(N)$

$0+1+3+\ldots+\frac{N}{4}-1+\frac{N}{2}-1$

$\hookrightarrow 1+2+4+\ldots+\frac{N}{2}-\log_2 N$

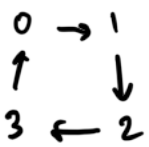
$\frac{1(1-2^{\log_2 N})}{1-2} - \log_2 N = N-1-\log_2 N = \Theta(N)$

## 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.

| | 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 |

A ✗

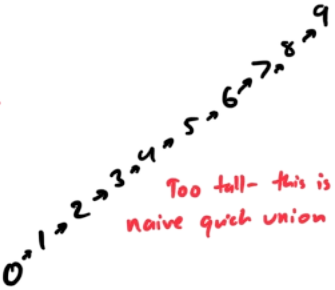



Cycle

B ✗

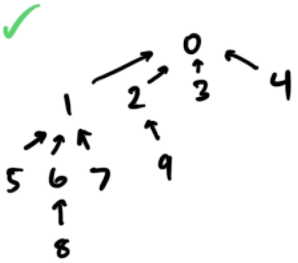
Should be 9→0 based on weight


C ✗

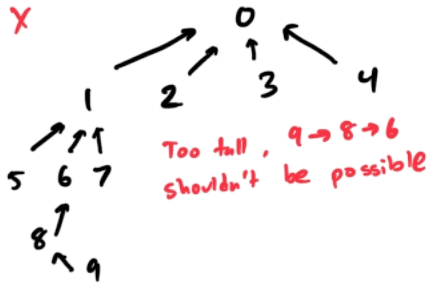



Too tall- this is naive quick union

D ✓

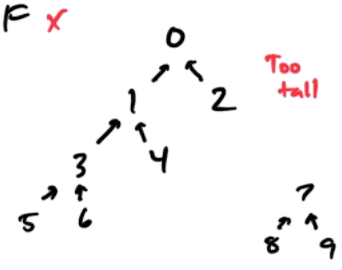



E ✗




Too tall, 9→8→6 shouldn't be possible

F ✗




Too tall

# 3 Asymptotics of Weighted Quick Unions

Note: for all big $\Omega$ 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 $\Omega$ and big $O$, of `isConnected`?

   $\Omega(\underline{\quad 1 \quad})$, $O(\underline{\log N})$

   Best case - elements are at the top together

   Worst case - have to go through the entire tree, height $\log N$

2. What is the runtime, in big $\Omega$ and big $O$, of `connect`?

   $\Omega(\underline{\quad 1 \quad})$, $O(\underline{\log N})$

(b) Suppose for the following problem we add the method `addToWQU` to the WQU class. 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.

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

Worst Case:

$N$ items $\rightarrow \binom{N-1}{2} = \frac{(N-1)!}{(N-3)!\,2!} = \frac{(N-1)(N-2)}{2}$

Potentially $\times 2$ if pairs don't have to be distinct $= \frac{N^2-3N+2}{2}$ pairs excluding the last item

$\downarrow$

Each of these triggering $\log N$ lookup

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 $\Omega$ and big $O$?

$\Omega(\underline{\quad N \quad})$, $O(\underline{N^2 \log N})$

(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: $\underline{\quad 1 \quad}$, maximum: $\underline{N-1}$

The best case for (b)

Pairwise combining

0,1 2,3 ... N-1,N

$\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$

$\vdots$

Reverse the order

$\frac{N}{2} + \frac{N}{4} + \ldots + 1 = \frac{1\left(1-2^{\log_2 N}\right)}{1-2} = N-1$