Midterm Recap
Announcements

● PA2 is due Sunday
● WA2 will be released ASAP (you will have a full week after break to complete it as normal, but I'll release it as early as I can for those that want to start early)
● Midterm grading should be completed today, announcement will be on Piazza when grades get posted
● I will not curve/adjust individual assignments until the end of the semester, and there is no guarantee of that
● Answer keys are posted
Part 1 - Summations and Bounds

\[ f(n) = \sum_{i=1}^{n^4} \sum_{j=1}^{n} 42i \]

\[ f(n) = 4\log(2^n^3) + n^2 - 16 \]

\[ f(n) = \begin{cases} 
\log(n) & \text{if } n \text{ is prime} \\
16n & \text{if } n > 10 \text{ and is even} \\
19n\log(n) & \text{otherwise}
\end{cases} \]
Part 1 - Summations and Bounds

\[ f(n) = \sum_{i=1}^{n^4} \sum_{j=1}^{n} 42i \quad \text{← } i \text{ is constant with respect to } j. \text{ Notice how this summation expands to } (42i + 42i + 42i + \ldots), \text{ the terms don't change} \]

\[ f(n) = 4\log(2^{n^3}) + n^2 - 16 \]

\[ f(n) = \begin{cases} 
\log(n) & \text{if } n \text{ is prime} \\
16n & \text{if } n > 10 \text{ and is even} \\
19n\log(n) & \text{otherwise}
\end{cases} \]
Part 1 - Summations and Bounds

\[ f(n) = \sum_{i=1}^{n^4} \sum_{j=1}^{n} 42i \quad \leftarrow i \text{ is constant with respect to } j. \text{ Notice how this summation expands to } (42i + 42i + 42i + \ldots ), \text{ the terms don't change} \]

\[ f(n) = 4\log(2^{n^3}) + n^2 - 16 \quad \leftarrow \log(2^X) = X. \text{ So the first term is just } 4n^3 \]

\[ f(n) = \begin{cases} 
\log(n) & \text{if } n \text{ is prime} \\
16n & \text{if } n > 10 \text{ and is even} \\
19n\log(n) & \text{otherwise}
\end{cases} \]
Part 1 - Summations and Bounds

\[ f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} 42i \quad \text{← } i \text{ is constant with respect to } j. \text{ Notice how this summation expands to } (42i + 42i + 42i + \ldots ), \text{ the terms don't change} \]

\[ f(n) = 4\log(2^{n^3}) + n^2 - 16 \quad \text{← } \log(2^X) = X. \text{ So the first term is just } 4n^3 \]

\[ f(n) = \begin{cases} 
  \log(n) & \text{if } n \text{ is prime} \\
  16n & \text{if } n > 10 \text{ and is even} \\
  19n\log(n) & \text{otherwise} 
\end{cases} \]

← \( O \) is the upper bound \((n\log n)\), \( \Omega \) is the lower bound \((\log n)\), but \( \Theta \) does not exist because tight \( O \) does not equal tight \( \Omega \)
Part 1 - Summations and Bounds

Prove $f(n) \in O(g(n))$ by finding $c$ and $n_0$

\[ f(n) = 6n^3 + 14n - 2 \]
\[ g(n) = 2n^3 \]
Part 1 - Summations and Bounds

Prove \( f(n) \in O(g(n)) \) by finding \( c \) and \( n_0 \)

Consider the following inequalities:

\[
6 \ n^3 \leq c_1 \times 2 \ n^3 \quad \leftarrow \text{This is true if } c_1 = 3 \text{ and } n_0 = 0 \\
14 \ n \leq c_2 \times 2 \ n^3 \quad \leftarrow \text{This is true if } c_2 = 7 \text{ and } n_0 = 0 \\
-2 \leq c_3 \times 2 \ n^3 \quad \leftarrow \text{This is true if } c_3 = 1 \text{ and } n_0 = 0
\]

So if we set \( c \) to 11 and \( n_0 \) to 0, we have:

\[ f(n) \leq c \times g(n) \text{ for all } n > n_0, \text{ therefore } f(n) \text{ is in } O(g(n)) \]

Note: There are infinite valid values for \( c \) and \( n_0 \)

The limit test does not work here (does not find \( c \) and \( n_0 \))
For the Social Media Question: The problem described a **Graph**. The algorithm the problem wanted to perform was BFS, which is asymptotically faster when our graph is implemented with an **Adjacency List**
Part 2 - Data Structure Choice

For the Photo Question: The distinguishing factor here was that we knew we only needed a constant amount of space! No need for the extra bits in ArrayBuffer or LinkedLists. Just an Array meets all our needs, which is a Seq.

In general: The simplest data structure that meets your needs efficiently, is probably the best
FOR QUEUES:
val seq = new MysterySequence()
seq.addSomething("S")
seq.addSomething("P")
seq.addSomething("A")
seq.addSomething("C")
print(seq.removeSomething()) ← Prints "S"
print(seq.removeSomething()) ← Prints "P"
print(seq.removeSomething()) ← Prints "A"
seq.addSomething("E")
print(seq.removeSomething()) ← Prints "C"
seq.addSomething("N")
print(seq.removeSomething()) ← Prints "E"
FOR QUEUES:
val seq = new MysterySequence()
seq.addSomething("S")
seq.addSomething("P")
seq.addSomething("A")
seq.addSomething("C")
print(seq.removeSomething()) ← Prints "S"
print(seq.removeSomething()) ← Prints "P"
print(seq.removeSomething()) ← Prints "A"
seq.addSomething("E")
print(seq.removeSomething()) ← Prints "C"
seq.addSomething("N") ← "N" is leftover
print(seq.removeSomething()) ← Prints "E"
FOR STACKS:
val seq = new MysterySequence()
seq.addSomething("S")
seq.addSomething("P")
seq.addSomething("A")
seq.addSomething("C")
print(seq.removeSomething()) ← Prints "C"
print(seq.removeSomething()) ← Prints "A"
print(seq.removeSomething()) ← Prints "P"
seq.addSomething("E")
print(seq.removeSomething()) ← Prints "E"
seq.addSomething("N")
print(seq.removeSomething()) ← Prints "N"
FOR STACKS:
val seq = new MysterySequence()
seq.addSomething("S") ← "S" is leftover
seq.addSomething("P")
seq.addSomething("A")
seq.addSomething("C")
print(seq.removeSomething()) ← Prints "C"
print(seq.removeSomething()) ← Prints "A"
print(seq.removeSomething()) ← Prints "P"
seq.addSomething("E")
print(seq.removeSomething()) ← Prints "E"
seq.addSomething("N")
print(seq.removeSomething()) ← Prints "N"
Part 3 - Stacks and Queues

Stacks: Last In First Out (LIFO)
- Push (put item on top of the stack) \( \Theta(1) \) (or amortized \( O(1) \))
- Pop (take item off top of stack) \( \Theta(1) \)
- Top (peek at top of stack) \( \Theta(1) \)

Queues: First in First Out (FIFO)
- Enqueue (put item on the end of the queue) \( \Theta(1) \) (or amortized \( O(1) \))
- Dequeue (take item off the front of the queue) \( \Theta(1) \)
- Head (peek at the item in the front of the queue) \( \Theta(1) \)
If $n$ calls to a function take $O(T(n))$...

We say the **Amortized Runtime** is $O(T(n) / n)$

How long does it take to do $n$ pushes in a LinkedList based Stack? $O(n)$

So amortized runtime of push is $O(1)$

...it can be the same as the unqualified runtime. Will never be worse.
array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")

Unqualified Worst-Case to insert "foo" is always $O(n)$

Why?
array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")

**Unqualified Worst-Case to insert "foo" is always O(n)**

**Why?** You have to shift the elements to make space.
Part 4 - Arrays and Linked Lists

array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")

Unqualified Worst-Case to insert "bar" if we assume buffer is not full: $O(1)$

\[ T_{append}(n) = \begin{cases} 
  n & \text{if used} = n \\
  1 & \text{otherwise} 
\end{cases} \]
Part 4 - Arrays and Linked Lists

array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")

Unqualified Worst-Case to insert "bar" if we assume buffer is not full: \( O(1) \)

\[
T_{append}(n) = \begin{cases} 
  n & \text{if } \text{used} = n \\
  1 & \text{otherwise}
\end{cases}
\]

...but if we can't make that assumption: \( O(n) \)
array.insert(idx = x, elem = "foo")
array.insert(idx = array.length, elem = "bar")
list.insert(idx = list.length, elem = "baz")

For a singly linked list, we must iterate from head: $O(n)$
For a doubly linked list, we have a reference to tail: $O(1)$
ADTs just describe **what** you can do with the data

The data structure is the actual implementation of those capabilities.

**ADTS:** Seq, Buffer, Stack, Queue, Graph

**Data Structures:** Array, ArrayBuffer, LinkedList, EdgeList, AdjList, AdjMatrix
If we hypothesize that the runtime of this recursive algorithm is $O(n \log(n))$, then our base case proof must be:

$$T(1) \leq c \cdot (1) \log (1)$$

...or $T(2) \leq c \cdot (2) \log (2)$
Part 5 - Misc

If we hypothesize that the runtime of this recursive algorithm is $O(n \log(n))$, then our inductive assumption would be:

$$T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq 1 \\
2 \cdot T(\frac{n}{2}) + \Theta(1) + \Theta(n) & \text{otherwise}
\end{cases}$$

If we hypothesize that the runtime of this recursive algorithm is $O(n \log(n))$, then our inductive assumption would be:

$$T(n/2) \leq c \cdot (n/2) \log(n/2)$$
Part 6 - Graphs

Find a spanning tree that would be produced by DFS or BFS from A

Spanning subgraph: Must include all nodes

Tree: No loops

DFS: Must include BC and DF

BFS: Must not include BC and DF
Part 7 - Extra Credit

Is it possible to have a function, $f(n)$, that is in both $O(n^2)$ and $\Omega(\log n)$?

Is $f(n) = 3n$ in $O(n^2)$?
Is it possible to have a function, $f(n)$, that is in both $O(n^2)$ and $\Omega(\log n)$?

Is $f(n) = 3n$ in $O(n^2)$? **Yes.** $O(n^2)$ bounds $3n$ from above. Not tightly, but it's still a bound.
Part 7 - Extra Credit

Is it possible to have a function, $f(n)$, that is in both $O(n^2)$ and $\Omega(\log n)$?

Is $f(n) = 3n$ in $\Omega(\log n)$?
Is it possible to have a function, \( f(n) \), that is in both \( \Theta(n^2) \) and \( \Omega(\log n) \)?

Is \( f(n) = 3n \) in \( \Omega(\log n) \)? **Yes.** \( \Omega(\log n) \) bounds \( 3n \) from below. Not tightly, but it's still a bound.