CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

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

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

$$f(n) = 4log(2^{n^3}) + n^2 - 16$$

$$f(n) = \begin{cases} log(n) & \textbf{if } n \textbf{ is prime} \\ 16n & \textbf{if } n > 10 \textbf{ and is even} \\ 19nlog(n) & \textbf{otherwise} \end{cases}$$

Part 1 - Summations and Bounds

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

$$f(n) = 4log(2^{n^3}) + n^2 - 16$$

$$f(n) = \begin{cases} log(n) & \textbf{if } n \textbf{ is prime} \\ 16n & \textbf{if } n > 10 \textbf{ and is even} \\ 19nlog(n) & \textbf{otherwise} \end{cases}$$

Part 1 - Summations and Bounds

$f(n) = \sum_{i=1}^{n^4} \boxed{\sum_{j=1}^{n} 42i}$ ← *i* is constant with respect to *j*. Notice how this summation expands to (42i + 42i + 42i + …), the terms don't change

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

$$f(n) = \begin{cases} log(n) & \textbf{if } n \textbf{ is prime} \\ 16n & \textbf{if } n > 10 \textbf{ and is even} \\ 19nlog(n) & \textbf{otherwise} \end{cases}$$

Part 1 - Summations and Bounds

$f(n) = \sum_{i=1}^{n^4} \boxed{\sum_{j=1}^{n} 42i}$ ← *i* is constant with respect to *j*. Notice how this summation expands to (42i + 42i + 42i + …), the terms don't change

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

$$f(n) = \begin{cases} log(n) & \textbf{if } n \textbf{ is prime} \\ 16n & \textbf{if } n > 10 \textbf{ and is even} \\ 19nlog(n) & \textbf{otherwise} \end{cases}$$

← ***O*** is the upper bound (nlogn), **Ω** is the lower bound (logn), but **Θ** does not exist because tight ***O*** does not equal tight **Ω**

Part 1 - Summations and Bounds

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

$$f(n) = 6n^3 + 14n - 2$$

$$g(n) = 2n^3$$

Part 1 - Summations and Bounds

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

$$f(n) = 6n^3 + 14n - 2$$

$$g(n) = 2n^3$$

Consider the following inequalities:

$6\,n^3 \leq c_1 * 2\,n^3$ ← This is true if c_1 = 3 and n_0 = 0

$14\,n \leq c_2 * 2\,n^3$ ← This is true if c_2 = 7 and n_0 = 0

$-2 \leq c_3 * 2\,n^3$ ← This is true if c_3 = 1 and n_0 = 0

So if we set c to 11 and n_0 to 0, we have:
$f(n) \leq c * g(n)$ for all $n > n_0$, therefore f(n) 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)

Part 2 - Data Structure Choice

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

Part 3 - Stacks and Queues

```
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"
```

Part 3 - Stacks and Queues

```
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"
```

Part 3 - Stacks and Queues

```
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"
```

Part 3 - Stacks and Queues

```
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) $\Theta(1)$

Part 3 - Stacks and Queues

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? $\boldsymbol{O(n)}$

So amortized runtime of push is $\boldsymbol{O(1)}$

...it can be the same as the unqualified runtime. Will never be worse.

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 "foo" is always O(n)

Why?

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 "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)

$$T_{append}(n) = \begin{cases} n & \text{if used} = \text{n} \\ \boxed{1 \quad \text{otherwise}} \end{cases}$$

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)

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

...but if we can't make that assumption: *O*(n)

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")
```

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)

Part 5 - Misc

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

Part 5 - Misc

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

If we hypothesize that the runtime of this recursive algorithm is O(n log(n)), then our base case proof must be:

T(1) ≤ c * (1) log (1)

...or T(2) ≤ c * (2) log (2)

Part 5 - Misc

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

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

$T(n/2) \leq c * (n/2) \log(n/2)$

Part 6 - Graphs

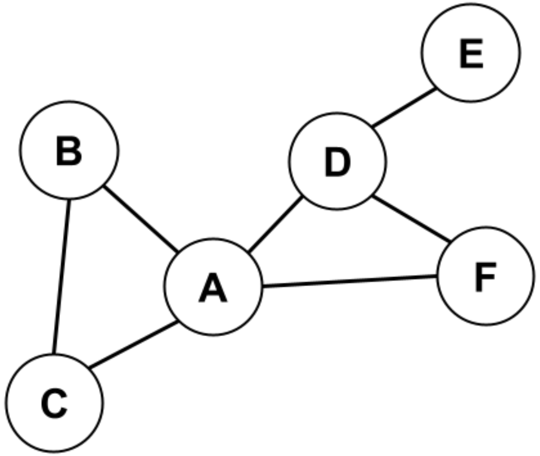
Find a spanning tree that would be produced by DFS or BFS from A

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 $\boldsymbol{O}(n^2)$ and $\Omega(\log n)$?

Is $f(n) = 3n$ in $\boldsymbol{O}(n^2)$?

Part 7 - Extra Credit

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

Is f(n) = 3n in $\boldsymbol{O}(n^2)$? **Yes.** $\boldsymbol{O}(n^2)$ 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 $\boldsymbol{O}(n^2)$ and $\boldsymbol{\Omega}(\log n)$?

Is $f(n) = 3n$ in $\boldsymbol{\Omega}(\log n)$?

Part 7 - Extra Credit

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

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