Sp18 CS 61B Discussion 7

Welcome!

Wayne Li

wli2@berkeley.edu

https://wayne-li2.github.io/

Announcements

- Project 2, Part 2 released

Quiz Instructions

- If you haven't yet, please also **neatly** put your email address **outside the name box** if you want to be emailed!
- Bubble number **41**.

Aside

Brainfuck! (CS 164)

- Turing-complete programming language
- Written by Urban Muller in 1993
    - Original README says “Who can program anything useful with it? :)”

Brainfuck! (CS 164)

- Original goal: Write the smallest compiler possible.
    - Original design: 240 bytes.
        - Today, some only down to 100 bytes.

Brainfuck! (CS 164)

- 8 commands and an instruction pointer.

Brainfuck! (CS 164)

Character	Meaning
>	increment the data pointer (to point to the next cell to the right).
<	decrement the data pointer (to point to the next cell to the left).
+	increment (increase by one) the byte at the data pointer.
-	decrement (decrease by one) the byte at the data pointer.
.	output the byte at the data pointer.
,	accept one byte of input, storing its value in the byte at the data pointer.
[	if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it *forward* to the command after the *matching*] command.
]	if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it *back* to the command after the *matching* [command.

Brainfuck! (CS 164)

- Hello World program:
-
```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>
>.>---.+++++++..+++.>>.<-.<.+++.------.--------.>>+.>++.
```

JSFuck! (CS 161)

- JSFuck is a derivative of Brainfuck.
- Any JavaScript interpreter can interpret JSFuck.
- The character “y”:
  - `(+[![]]+[+(+!+[]+(!+[]+[])[!+[]+!+[]+!+[]]+(+!+[])+(+[])+(+[])+(+[]))])[+!+[]+[+[]]]`

Why JSFuck? (CS 161)

- Usually, computers can detect malicious JavaScript code from websites via common attack syntax.
- But with JSFuck encoding, these features are nearly undetectable!
- Allows malicious code to bypass detection systems.

Why JSFuck? (CS 161)

- https://arstechnica.com/information-technology/2016/02/ebay-has-no-plans-to-fix-severe-bug-that-allows-malware-distribution/

Moral of the Story

- **Programmers like to swear?**

References

- JSFuck source code: https://github.com/aemkei/jsfuck

Asymptotics

Asymptotics is the whole reason for 61B

- If we did not care about efficiency of our code, why use different data structures at all?

Key Idea

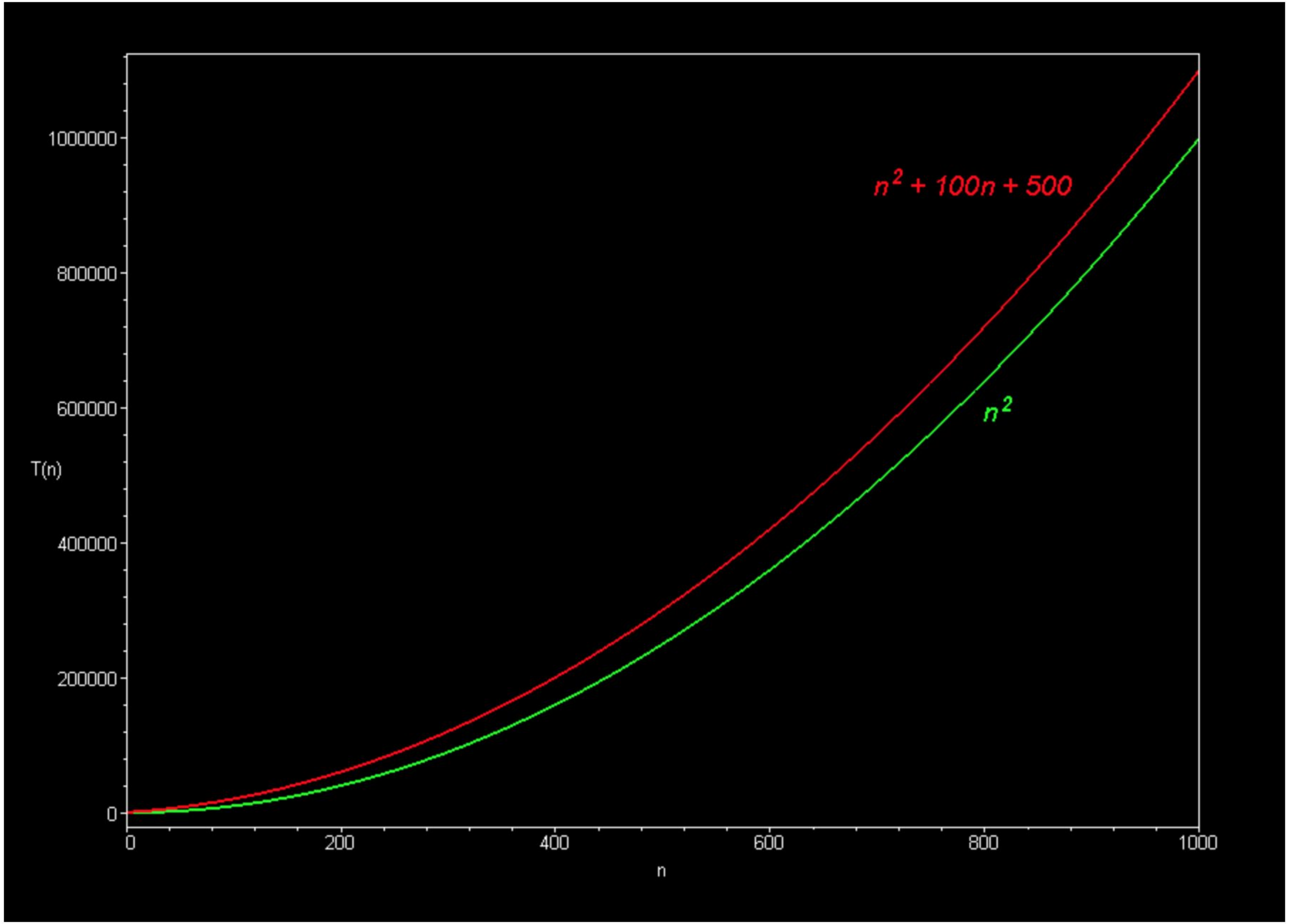
- Asymptotics describe how a function **grows** as input size grows.

Key Idea (The most key-est of them all)

- Asymptotics are **approximations.**
- To put it another way, we are not finding the runtime of a function.
- We are finding the **family of runtimes** that this function belongs to.

Key Idea

- **Drop your constants and lower order terms.**
  - Number one way to lose points on an asymptotic exam question.



$n^2 + 100n + 500$
n^2
T(n)
n
0
200000
400000
600000
800000
1000000
0
200
400
600
800
1000

Notation: Big O, Big Omega, Big Theta

- Goal: Look at program complexity for large input
- Notations:
  - Big O - bounds above (often used for worst case)
  - Big Omega - bounds below (often used for best case)
  - Big Theta - bounds above and below (both best and worst case)

O (Big O)

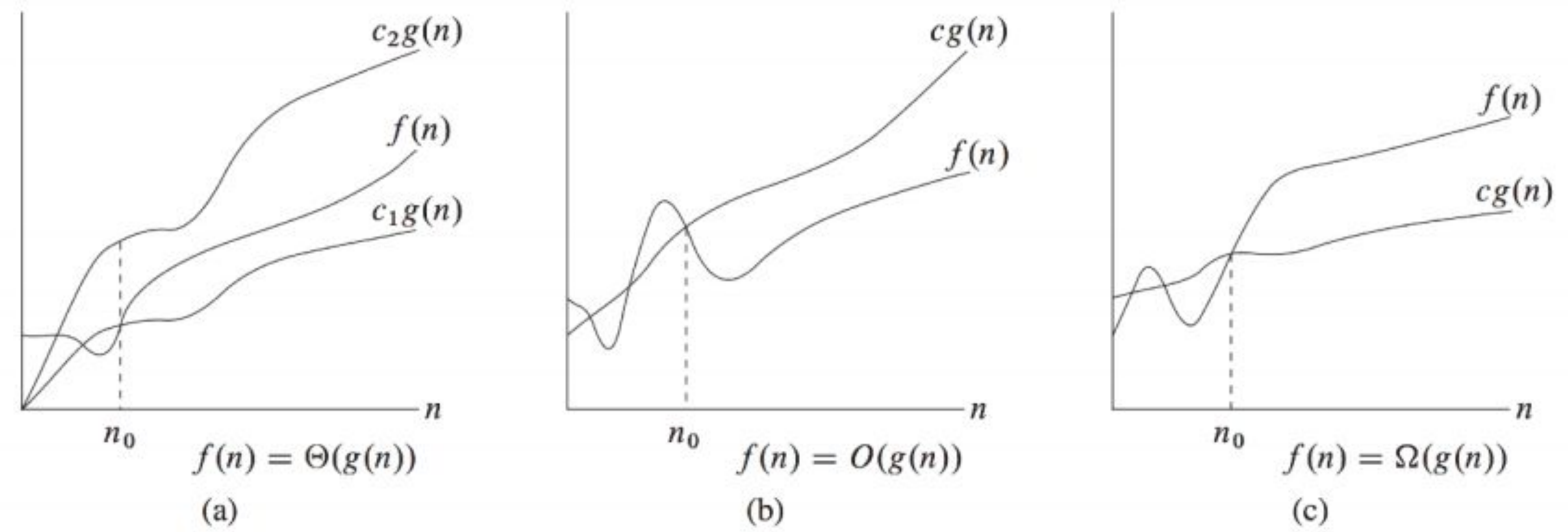
- Let $f(n)$ and $g(n)$ be positive real numbers on inputs of size n
- $f \in O(g)$ if there is a constant $c > 0$ s.t. $f(n) <= c\, g(n)$
- Upper bounded by $g(n)$ when n gets significantly large.
- Bound does not have to be tight.

Ω (Big Omega)

- Let f(n) and g(n) be positive real numbers on inputs of size n
- f ∈ O(g) if there is a constant c > 0 s.t. f(n) >= c g(n)
- Lower bounded by g(n) when n gets significantly large.
- Bound does not have to be tight.

Θ (Big Theta)

- Let $f(n)$ and $g(n)$ be positive real numbers on inputs of size n
- $f \in \Theta(g)$ if there is a constant $c1 > 0$ and $c2 > 0$ s.t.
  - $C1\ g(n) <= f(n) <= c2\ g(n)$ for all $c1 <= c2$
- Tightly bounded by $g(n)$ when n gets significantly large.
- • $f \in \Omega(g)$ and $f \in O(g)$



Introduction to Algorithms, Cormen, Leiserson, Rivest, Stein

Key Idea: Be **TIGHT**

- If my function runs in $\mathbf{f(n) = n}$ time, then technically I could say that it is bounded by $\mathbf{O(n^{100})}$.
    - No credit on an exam!
    - Also defeats the purpose of asymptotics.
- Be as tight as possible. Say $\mathbf{\Theta(n)}$.

Best / Worst Case

- If an exam asks for best / worst case, give a **THETA** bound.
  - Why? Think about what the words “best” and “worst” mean.

Multiple Input Size

- Sometimes, we will ask you for a runtime in terms of A, B, and C.
- Ex: Given runtime $O(M^2 + N)$:
    - Can I drop N?

Multiple Input Size

- Recall last time when I told you, you could drop **lower-order terms** and **constants?**
- What about the runtime $O(M^2 + N)$?
    - Can I drop N?

Multiple Input Size

- Recall last time when I told you, you could drop **lower-order terms** and **constants?**
- What about the runtime $O(M^2 + N)$?
  - Can I drop N?
- If **M** and **N** are **independent**, then you cannot drop N.
  - If dependent, then after conversion, reevaluate.

Common Patterns

- $1 + 2 + 3 + 4 + \dots + n \rightarrow O(n^2)$
  - Summation is equal to $n(n + 1) / 2$
- $1 + 2 + 4 + 8 + \dots + n \rightarrow O(n)$
  - To see this, take a look at the **8**.
  - Then, take a look at the numbers to the left of **8.**
  - Sums up to 7. Therefore we sum to **2n -> O(n)**

Recursive Runtime

Key Idea

- Draw a recursion tree, then write a summation!

Putting it Together

1. Find the work done per node in terms of input size.
2. Find the work done per layer in terms of input size.
    a. Number of nodes x work done per node.
3. Find the height of the tree.
4. Find the work done by tree.
    a. Summation of all layers and work done per layer.

Onto Discussion