$\mathcal{O}(\cdot)$ (*"Big–Oh"*) notation

CMSC 250

Logistics / Reminders

- All grades for graded assignments on ELMS.
  - Stats in yesterday night's announcement.
- HW 06 posted yesterday!
  - Quite long; please begin asap!
- Today: Big-Oh notation.
- Thursday: intro to proofs

A loop example

```
for(i = 1 to n)
   for(j=1 to i)
      print "Hello";
```

- How fast will this run (as a function of n?)

Roughly n steps

Roughly n^2 steps

Roughly $n \cdot \log_2 n$ steps

Something Else

A loop example

```
for(i = 1 to n)
   for(j=1 to i)
      print “Hello”;
```

- How fast will this run (as a function of n?)

Roughly n steps	Roughly n^2 steps	Roughly $n \cdot \log_2 n$ steps	Something Else

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} \approx n^2$$

What do we care about?

- Do we care about $\frac{n^2}{2} + \frac{n}{2}$ as opposed to "roughly" n^2?
  - No, (mathematical reason): because n^2 grows much faster than the other terms and thus dominates the expression.
  - No, (CompSci reason): The constant of ${}^1/_2$ doesn't matter because of Moore's Law: Every year and a half, computers double in speed. The formula of the partial sum S_n of the arithmetic progression, of course, doesn't change.

How to define “roughly” n^2?

- We won’t care about multiplicative constants of n^2, like $\frac{n^2}{2}, \frac{n^2}{3}, 2 \cdot n^2$.
- We also don’t care about “small” values of n.
- How do we pin down our apathy formally?
  - With ``**Big-oh”** notation: $\mathcal{O}(\cdot)$ (LaTeX: `\mathcal{O}`)
- We know that 132 also covers “Big-Oh” notation.
  - We define it **rigorously** using **quantifiers.**

Formal definition of "Big-Oh"

- Let $f(n), g(n)$ be functions of n.
- We say one of the following:
  - $f(n)$ is $\mathcal{O}(g(n))$ (common and the Epp way)
  - $f(n) = \mathcal{O}(g(n))$ (also common)
  - $f(n) \in \mathcal{O}(g(n))$ (rare)
  - $f(n) \leq \mathcal{O}(g(n))$ (avoid this)

if, and only if,

$$(\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^{>0})[(\forall n \geq n_0)[f(n) \leq c \cdot g(n)]]$$

Formal definition of “Big-Oh”

- Let $f(n), g(n)$ be functions of n.
- We say one of the following:
  - $f(n)$ is $\mathcal{O}(g(n))$ (common and the Epp way)
  - $f(n) = \mathcal{O}\big(g(n)\big)$ (also common)
  - $f(n) \in \mathcal{O}(g(n))$ (rare)
  - $f(n) \leq \mathcal{O}(g(n))$ (avoid this)

That’s why we suggest that you avoid this notation: easy to confuse the inequalities.

if, and only if,

$$(\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^{>0})[(\forall n \geq n_0)[f(n) \leq c \cdot g(n)]]$$

Formal definition of "Big-Oh"

- Let $f(n), g(n)$ be functions of n.
- We say one of the following:
  - $f(n)$ is $\mathcal{O}(g(n))$ (the Epp way)
  - $f(n) = \mathcal{O}(g(n))$
  - $f(n) \leq \mathcal{O}(g(n))$ (avoid this)

if, and only if,

That's why we suggest that you avoid this: easy to confuse the inequalities.

$$(\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^{>0})[(\forall n \geq n_0)[f(n) \leq c \cdot g(n)]]$$

- Intuitively: Starting with a natural number n_0, the graph of $c \cdot g(n)$ bounds the graph of $f(n)$ from above.

Choice of n_0, c

$$(\exists n_0 \in \mathbb{N}, c \in \mathbb{R}^{>0})[(\forall n \geq n_0)[f(n) \leq c \cdot g(n)]]$$

- $n \geq n_0$ says: We won't care about small values of n (let n grow as much as we want).
- $c > 0$ says: The constants (like ${}^1/_3$, ${}^1/_2$, 5) don't matter (but it's better if they're small).

Example of finding n_0, c

- Proof that $3n^2 - 4n + 100$ is $\mathcal{O}(n^2)$ (find appropriate n_0, c)
  - For $c = 3$,

$$
\begin{aligned}
&3n^2 - 4n + 100 \leq 3n^2 \\
\Leftrightarrow\ & n \geq 25 \\
\Rightarrow\ & n_0 = 25
\end{aligned}
$$

- We found a pair (n_0, c) so we are done

Example of finding n_0, c

- Proof that $3n^2 - 4n + 100$ is $\mathcal{O}(n^2)$ (find appropriate n_0, c)
  - For $c = 3$, $3n^2 - 4n + 100 \leq 3n^2 \Leftrightarrow n \geq 25 \Rightarrow n_0 = 25$
  - We found a pair (n_0, c) so we are done
- $3n^2 - 4n + 1000$ is $\mathcal{O}(n^3)$

True

False

Something Else

Example of finding n_0, c

- Proof that $3n^2 - 4n + 1000$ is $\mathcal{O}(n^2)$ (find appropriate n_0, c)
  - For $c = 3$, $3n^2 - 4n + 100 \le 3n^2 \Leftrightarrow n \ge 25 \Rightarrow n_0 = 25$
  - We found a pair (n_0, c) so we are done

- $3n^2 - 4n + 1000$ is $\mathcal{O}(n^3)$

True | False | Something Else

For $c = 3, 3n^2 - 4n + 100 \le 3 \cdot n^3 \Leftrightarrow 3n^3 - 3n^2 + 4n - 100 \ge 0$

Valid for $n \ge 4$

True, but stupid

- $3n^2 - 4n + 1000$ is $\mathcal{O}(n^3)$: Statement is true, but stupid!
  - Reason: We can **easily** make a **stronger** statement. E.g:

$$3n^2 - 4n + 1000 \text{ is } \mathcal{O}(n^2)$$

  - Here by **stronger** we mean “more accurate”, or “tighter”

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ F

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ F
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ F
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$ F

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ F
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$ F
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^5)$

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^5)$ *T A I*

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^5)$ *T A I*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{5.1})$

True and interesting

True but Stupid

False

Polynomials

- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^4)$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{4.9})$ *F*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^5)$ *T A I*
- $6n^5 - 4n^4 - 3n^2 + 1000$ is $\mathcal{O}(n^{5.1})$ T B S

True and interesting

True but Stupid

False

Polynomial vs Exponential

- n^{10} is $\mathcal{O}(2^n)$?

n	n^{10}	2^n
10	10,000,000,000	1,024
20	10,240,000,000,000	1,048,576
30	590,490,000,000,000	1,073,741,824
40	10,485,760,000,000,000	1,099,511,627,776
$\vdots$	$\vdots$	$\vdots$

True and interesting

True but Stupid

False

Polynomial vs Exponential

- n^{10} is $\mathcal{O}(2^n)$

n	n^{10}	2^n
10	10,000,000,000	1,024
20	10,240,000,000,000	1,048,576
30	590,490,000,000,000	1,073,741,824
40	10,485,760,000,000,000	1,099,511,627,776
$\vdots$	$\vdots$	$\vdots$

True and interesting

True but Stupid

False

For c = 1, $n_0 \geq 59$, OR
For c = 10, $n_0 \geq 55$ OR….

- Can prove by
  1. Induction (messy because of $(n+1)^{10}$ term)
  2. Calculus (next slide)

Proof by calculus that n^{10} is $\mathcal{O}(2^n)$

- For $c = 1$, $n^{10} < 2^n \Leftrightarrow 10 \cdot \log_2 n < n \Leftrightarrow 10 < \frac{n}{\log_2 n}$
- By L' Hospital's Rule, $\frac{n}{\log_2 n} \mapsto +\infty$, therefore this inequality is true.
- For which choices of n and above? We don't tell you. The only thing we tell you is that $10 < \frac{n}{\log_2 n}$ **at some point** n_0.
  - An example of a **non-constructive proof of existence:** $\exists n_0$ such that $(\forall n \geq n_0)[n^{10} < 2^n]$: we do not tell you which this n_0 is nor do we give an algorithm that constructs it!

Polynomial Summary

- Let $a, b \in \mathbb{N}^{\geq 1}$ with $a < b$. The following hold:

1. Any polynomial of degree a is $\mathcal{O}(n^a)$ (True and Interesting)
2. n^a is $\mathcal{O}(n^b)$ (True but stupid)
3. n^a is $\mathcal{O}(b^n)$ *(for b > 1)*

- Note: The above holds even if $a, b \in \mathbb{Q}^{>0}$!

Logs

- $\log^{10^3}(n)$ is $\mathcal{O}(n^{1/20})$

True and interesting

True but Stupid

False

Logs

- $\log^{10^3}(n)$ is $\mathcal{O}(n^{1/20})$
- Let's pick $c = 1$ and solve through Calculus like before:

$$\log^{10^3} n \leq n^{1/20} \Leftrightarrow \log(\log^{10^3} n) < \frac{\log n}{20} \Leftrightarrow$$

$$10^3 \log(\log \mathrm{n}) < \frac{\log n}{20} \Leftrightarrow 20 * 10^3 < \frac{\log n}{\log(\log n)}$$

- $\lim_{n \mapsto +\infty} \frac{\log n}{\log(\log n)} = +\infty$ (convince yourselves), so the above inequality holds.

True and interesting

True but Stupid

False

Log Summary

- Let $a, b \in \mathbb{N}, b > 0$. Then:

$$\log^a n \text{ is } \mathcal{O}\left(n^b\right)$$

- If $a \leq b$,

$$\log^a n \text{ is } \mathcal{O}\left(\log^b n\right)$$

Log Summary

- Let $a, b \in \mathbb{N}, b > 0$. Then:

$$\log^a n \text{ is } \mathcal{O}\left(n^b\right)$$

Could also work for $a, b \in \mathbb{Q}^{\geq 0}$

- If $a \leq b$,

$$\log^a n \text{ is } \mathcal{O}\left(\log^b n\right)$$

“Big-Omega” notation ($\Omega(\cdot)$)

- Let $n \in \mathbb{N}$ and $f(n), g(n)$ be functions. Then,

$$f(n) = \Omega\big(g(n)\big) \Leftrightarrow g(n) = \mathcal{O}(f(n))$$

“Big-Omega” notation ($\mathcal{\Omega}(\cdot)$)

LaTeX: \mathcal{\Omega}

- Let $n \in \mathbb{N}$ and $f(n), g(n)$ be functions. Then,

$$f(n) = \mathcal{\Omega}\big(g(n)\big) \Leftrightarrow g(n) = \mathcal{O}(f(n))$$

Polynomial Summary (Ω)

- Let $a, b \in \mathbb{N}^{>0}$ with $a < b$. The following hold:

1. Any polynomial of degree a is $\Omega(n^a)$ (True and Interesting)
2. n^b is $\Omega(n^a)$ (True but stupid)
3. b^n is $\Omega(n^a)$ *(for b > 1)*

Log Summary ($\boldsymbol{\Omega}$)

- Let $a, b \in \mathbb{N}, b > 0$. Then:

$$n^b \text{ is } \Omega(\log^a n)$$

- If $a \leq b$,

$$\log^b n \text{ is } \Omega(\log^a n\ )$$

“Big Theta” notation ($\Theta(\cdot)$)

- Let $n \in \mathbb{N}$ and $f(n), g(n)$ be functions. Then,

$$f(n) = \Theta(g(n)) \Leftrightarrow \left[\left(f(n) = \mathcal{O}(g(n))\right) \wedge \left(f(n) = \Omega(g(n))\right)\right]$$

“Big Theta” notation ($\mathcal{\Theta}(\cdot)$)

LaTeX: \mathcal{\Theta}

- Let $n \in \mathbb{N}$ and $f(n), g(n)$ be functions. Then,

$$f(n) = \mathcal{\Theta}(g(n)) \Leftrightarrow \left[\left(f(n) = \mathcal{O}(g(n))\right) \wedge \left(f(n) = \Omega(g(n))\right)\right]$$

Polynomial Summary (Θ)

- Let $a, b \in \mathbb{N}^{>0}$ with $a < b$. The following hold:

1. Any polynomial of degree a is $\Theta(n^a)$ (True and Interesting)
2. *n^b is **not** $\Theta(n^a)$ (Since it's not $O(n^a)$)*
3. *b^n is **not** $\Theta(n^a)$ for b > 1, since it is not $O(n^a)$*
4. If $f(n)$ and $g(n)$ are polynomials of the same degree, then

$$f(n) = \Theta(g(n))$$

Log Summary ($\boldsymbol{\Theta}$)

- Let $a, b \in \mathbb{N}, b > 0$. Then:

n^b is **not** $\Theta(\log^a n)$ (since it's not $\Theta(\log^a n)$)

Applications

- Used to analyze program runtime.
- Algorithm design:
  1. Get the order term ($\mathcal{O}$) as low as possible.
  2. Wittle down the constants as low as possible.
- Example: median finding
  - $\mathcal{O}(n \cdot \log n)$: 1950s
  - $\mathcal{O}(n \cdot \log(\log n))$: 1970
  - $\mathcal{O}(n)$ where the constant c is pretty bad: 1973
  - $3n$: 1976

Applications

- Used to analyze program runtime.
- Algorithm design:
  1. Get the order term ($\mathcal{O}$) as low as possible.
  2. Wittle down the constants as low as possible.

We will do this right now! ☺

- Example: median finding
  - $\mathcal{O}(n \cdot \log n)$: 1950s
  - $\mathcal{O}(n \cdot \log(\log n))$: 1970
  - $\mathcal{O}(n)$ where the constant c is pretty bad: 1973
  - $3n$: 1976

Algorithms for coins

- We want to write a computer program that takes as input:
  1. An amount of money equal to n cents.
  2. Two (2) coins with denominations x and y cents.

  and answers "Can I pay n cents using only coins of denomination x and y?

Algorithms for coins

- We want to write a computer program that takes as input:
  1. An amount of money equal to n cents.
  2. Two (2) coins with denominations x and y cents.

  and answers "Can I pay n cents using only coins of denomination x and y?
- Mathematically, given $n, x, y \in \mathbb{N}$, x, y co-prime,

$$(\exists ?\, a, b \in \mathbb{N})[n = ax + by]$$

First algorithm

- **Assume $x < y$ for simplicity**

$boolean\ function\ f_1(x, y, n)\{$

$d_{\max} = \lfloor n/y \rfloor$ // The whole #times y =max(x, y) ”fits” into n

$for(d = 0 : d_{max})\{$

$if((n - d * y) \equiv 0 \pmod{x})\{$ // Assume unit cost

return **true**;

$\}$

$\}$

return **false**;

$\}$

First algorithm

- **Assume $x < y$ for simplicity**

$boolean\ function\ f_1(x, y, n)\{$

$d_{\max} = \lfloor {}^{n}/_{y} \rfloor$ // The whole #times y =max(x, y) "fits" into n

$for(d = 0{:}\, d_{max})\{$

$if((n - d * y) \equiv 0\ (\,mod\ x))\{$ // Assume unit cost

return **true**;

$\}$

$\}$

return **false**;

$\}$

f_1 is:

$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
$\mathcal{O}(\log n)$	Something Else

First algorithm

- **Assume $x < y$ for simplicity**

$boolean\ function\ f_1(x, y, n)\{$

$d_{\max} = \lfloor {}^{n}/_{y} \rfloor$ // The whole #times y =max(x, y) ”fits” into n

$for(d = 0{:}\, d_{max})\{$

$if((n - d * y) \equiv 0\ (\ mod\ x))\{$ // Assume unit cost

return **true**;

}

}

return **false**;

}

The loop will run <u>at most</u> $\mathrm{d}_{\max} = \lfloor {}^{n}/_{y} \rfloor$ times!

f_1 is:

$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
$\mathcal{O}(\log n)$	Something Else

(Ungraded) homework for you!

- Modify the previous program such that it outputs (returns, prints, whatever) **all** the different possibilities for $a, b \in \mathbb{N}$ such that $n = ax + by$
- Does anything change in terms of $\mathcal{O}(\cdot)$?
  - Can we make any statements about $\Omega(\cdot)$ or $\Theta(\cdot)$?

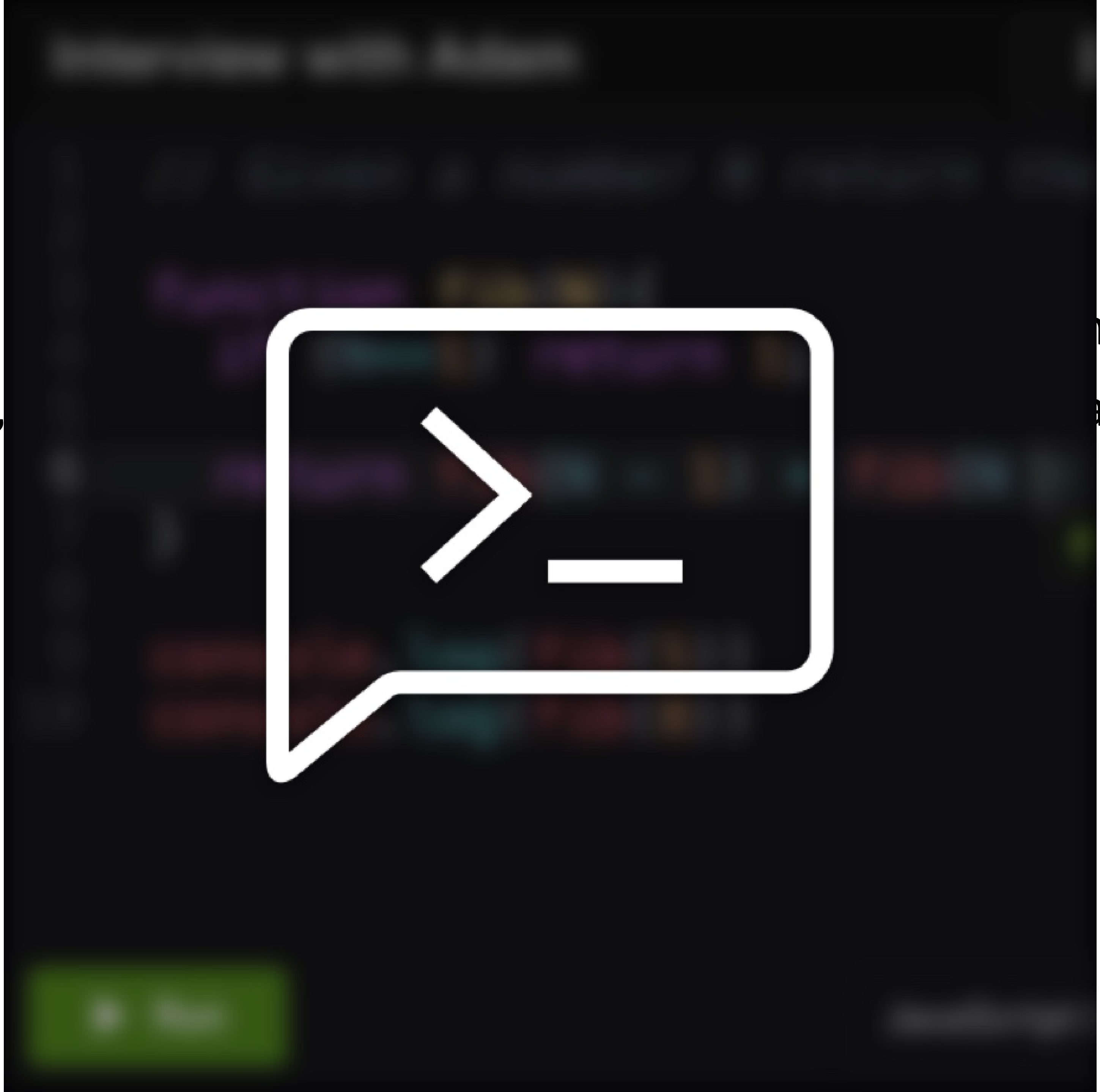
Generalization

- Can we build a program that does a similar thing for 3 coins?
- Given $n, x, y, z \in \mathbb{N}$, x, y, z co-prime, the program should answer

$$(\exists ?\, a, b, c \in \mathbb{N})[n = ax + by + cz]$$

- We are also interested in the **efficiency** of this program ($\mathcal{O}(\cdot)$)!

- Can we bu ns?
- Given $n, x,$ answer
- We are als $(\cdot))$!



Second algorithm

- Assume $x < y < z$ for simplicity

$boolean\ function\ \ f_2(x, y, z, n)\{$

$e_{\max} = \lfloor {}^{n}/_{z} \rfloor$ // The whole #times $z = \max(x, y, z)$ "fits" into n

$for(e = 0 : e_{max})\{$

$d_{max} = \lfloor {}^{n-e}/_{y} \rfloor$ // The whole # times $y = \max(x, y)$ "fits" into $n - e$

$for(d = 0 : d_{max})\{$

$if((n - ez - dy) \equiv 0\ (\,mod\ x))\{$ // Assume unit cost

return **true**;

$\}$

$\}$

return **false**;

$\}$

Second algorithm-runtime

- **Assume $x < y < z$ for simplicity**

$boolean\ function\ \ f_2(x, y, z, n)\{$

$e_{\max} = \lfloor {}^{n}/_{z} \rfloor$ // The whole #times $z = \max(x, y, z)$ "fits" into n

$for(e = 0 : e_{max})\{$

$d_{max} = \lfloor {}^{n-e}/_{y} \rfloor$ // The whole # times $y = \max(x, y)$ "fits" into $n - e$

$for(d = 0 : d_{max})\{$

$if((n - ez - dy) \equiv 0\ (\ mod\ x))\{$ // Assume unit cost

return **true**;

$\}$

$\}$

return **false**;

$\}$

f_2 is:

$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
$\mathcal{O}(\log n)$	Something Else

Second algorithm-runtime

- **Assume $x < y < z$ for simplicity**

$boolean\ function\ \ f_2(x, y, z, n)\{$

$e_{\max} = \lfloor {}^{n}/_{z} \rfloor$ // The whole #times $z = \max(x, y, z)$ "fits" into n

$for(e = 0 : e_{max})\{$

$d_{max} = \lfloor {}^{n-e}/_{y} \rfloor$ // The whole # times $y = \max(x, y)$ "fits" into $n - e$

$for(d = 0 : d_{max})\{$

$if((n - ez - dy) \equiv 0\ (\ mod\ x))\{$ // Assume unit cost

return **true**;

$\}$

$\}$

return **false**;

$\}$

Both loops will run at most ${}^{n^2}/_{zy}$ times!

f_2 is:

$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
$\mathcal{O}(\log n)$	Something Else

(Ungraded) homework for you

1. Modify algorithm to produce the NUMBER of ways n can be written as a sum of x's and y's.
2. Modify to produce HOW n can be written as x's and y's.

Second algorithm-runtime

- f_2 is $\mathcal{O}(n^2)$
  - constant = $\frac{1}{zy}$
  - Good if z, y large
    - Intuition: for large z, y, $e_{max} = \lfloor {}^{n}/_{z} \rfloor$ and $d_{max} = \lfloor {}^{(n-e)}/_{y} \rfloor$ are small, so the loops don't run for many steps!
  - Bad if z, y small ☹

Second algorithm-runtime

- f_2 is $\mathcal{O}(n^2)$
  - constant = $\frac{1}{zy}$
  - Good if z, y large
    - Intuition: for large z, y, $e_{max} = \lfloor {}^{n}/_{z} \rfloor$ and $d_{max} = \lfloor {}^{(n-e)}/_{y} \rfloor$ are small, so the loops don't run for many steps!
  - Bad if z, y small ☹

CAN WE DO BETTER?

Second algorithm-runtime

- f_2 is $\mathcal{O}(n^2)$
  - constant = $\frac{1}{zy}$
  - Good if z, y large
    - Intuition: for large z, y, $e_{max} = \lfloor {}^{n}/_{z} \rfloor$ and $d_{max} = \lfloor {}^{(n-e)}/_{y} \rfloor$ are small, so the loops don't run for many steps!
  - Bad if z, y small ☹

CAN WE DO BETTER?

Third algorithm

- We will present another algorithm for the three coin problem, which runs in $\mathcal{O}(n)$
- It uses memoization/dynamic programming.
- Interesting point: In CS, often easier to solve a harder problem that the problem at hand.
  - In this case, we can **efficiently** find ***all*** i between 0 and n such that i can be written as $a_i x + b_i y + c_i z$ for $a_i, b_i, c_i \in \mathbb{N}$!

Third algorithm – Take 1

- **Assume $x < y < z$ for simplicity**

$$
\begin{array}{l}
A[0] = \textbf{\textit{true}} \\
for\ (i = 1\ to\ n)\{ \\
\qquad if\ A[i-x] = \textbf{\textit{true}}\ or\ A[i-y] = \textbf{\textit{true}}\ or\ A[i-z] = \textbf{\textit{true}} \\
\qquad\qquad A[i] = \textbf{\textit{true}} \\
\qquad else \\
\qquad\qquad A[i] = \textbf{\textit{false}} \\
\}
\end{array}
$$

Third algorithm – Take 1

- **Assume $x < y < z$ for simplicity**

$$A[0] = \boldsymbol{true}$$
$$for\ (i = 1\ to\ n)\{$$
$$\quad if\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$$
$$\quad\quad A[i] = \boldsymbol{true}$$
$$\quad else$$
$$\quad\quad A[i] = \boldsymbol{false}$$
$$\}$$

- This is susceptible to out of bounds errors immediately…☹
  - So we need to improve it.

Third algorithm – Take 2

- If $i - x < 0$ or $i - y < 0$ or $i - z < 0$, we should have $A[i - x] = false$ or $A[i - y] = false$ or $A[i - z] = false$
- Since we assume that $x < y < z$, we have that the smallest possible value we can reach for any i is $i - z$.

Third algorithm – Take 2

$for\ (i = -z\ to\ -1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[0] = \boldsymbol{true}$

$for\ (i = 1\ to\ n)\{$

$\quad If\ A[i-x] = \boldsymbol{true}\ or\ A[i-y] = \boldsymbol{true}\ or\ A[i-z] = \boldsymbol{true}$

$\qquad A[i] = \boldsymbol{true}$

$\quad else$

$\qquad A[i] = \boldsymbol{false}$

$\}$

Third algorithm – Take 2

$$
\begin{aligned}
&for\ (i = -z\ to\ -1)\{ \\
&\qquad A[i] = \boldsymbol{false} \\
&\} \\
&A[0] = \boldsymbol{true} \\
&for\ (i = 1\ to\ n)\{ \\
&\qquad If\ A[i-x] = \boldsymbol{true}\ or\ A[i-y] = \boldsymbol{true}\ or\ A[i-z] = \boldsymbol{true} \\
&\qquad\qquad A[i] = \boldsymbol{true} \\
&\qquad else \\
&\qquad\qquad A[i] = \boldsymbol{false} \\
&\}
\end{aligned}
$$

Unfortunately, we cannot really index into negative positions of an array…. ☹

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

But we can always shift everything by z! ☺

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ n + z)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\qquad A[i] = \boldsymbol{true}$

$\quad else$

$\qquad A[i] = \boldsymbol{false}$

$\}$

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ n + z)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\quad\quad A[i] = \boldsymbol{true}$

$\quad else$

$\quad\quad A[i] = \boldsymbol{false}$

$\}$

But we can always shift everything by z! ☺

This makes our array from size n to size $n + z$.

Since $n \gg z$, space is still $O(n)$!

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ n + z)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\quad\quad A[i] = \boldsymbol{true}$

$\quad else$

$\quad\quad A[i] = \boldsymbol{false}$

$\}$

But we can always shift everything by z! ☺
This makes our array from size n to size $n + z$.
Since $n \gg z$, space is still $O(n)$!

$n \gg z$ also explains why this loop is correct

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ n + z)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\quad\quad A[i] = \boldsymbol{true}$

$\quad else$

$\quad\quad A[i] = \boldsymbol{false}$

$\}$

But we can always shift everything by z! ☺

F	F	F	F	T	F	?	?
0	1	…	$z - 1$	z	$z + 1$	…	$n + z$

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ n + z)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\qquad A[i] = \boldsymbol{true}$

$\quad else$

$\qquad A[i] = \boldsymbol{false}$

$\}$

Runtime =... ?

$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
$\mathcal{O}(n \cdot \log n)$	Something Else

Third algorithm – Take 3

$for\ (i = 0\ to\ z - 1)\{$

$\quad A[i] = \boldsymbol{false}$

$\}$

$A[z] = \boldsymbol{true}$

$for\ (i = z + 1\ to\ z + n)\{$

$\quad If\ A[i - x] = \boldsymbol{true}\ or\ A[i - y] = \boldsymbol{true}\ or\ A[i - z] = \boldsymbol{true}$

$\qquad A[i] = \boldsymbol{true}$

$\quad else$

$\qquad A[i] = \boldsymbol{false}$

$\}$

$\boldsymbol{z} + \boldsymbol{1} + \boldsymbol{n} = \mathcal{O}(n)$ **assignments (since $\boldsymbol{n} \gg \boldsymbol{z}$)!**

$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
$\mathcal{O}(n \cdot \log n)$	Something Else