S. KIMMEL

Time Complexity

The worst-case time complexity of an algorithm A is

$T_A : D \to \mathbb{N}$, for $D \subseteq \mathbb{N}$

$T_A(n)$ = # of operations performed by algorithm A in worst case on input size n.

If know T_A and know # operations/sec, can determine runtime

Linear Search

- Input: $(a_1, a_2, \dots a_n)$, x ⟸ Input size is n
- Output: j if $a_j = x$, 0 otherwise

```
1) i = 1                           ← +1
2) while (i ≤ n and x ≠ a_i)   } ←
3)        i = i+1
4) if i ≤ n:
        return i        ] too exhausted to continue
5) else:
        return 0
```

- Checking if $x = a_i$ takes at most $\log_2 x$ operations b/c might need to check $\log_2 x$ bits
- Checking if $i \leq n$ might take at most $\log_2 n$ bits
- Adding 1 to i might involve flipping $\log_2 n$ bits

$n(\log_2 x + 2\log_2 n)$

☆ This is bad! Difficult to count operations even on simplest alg.

☆ Usually consider $i \leq n$, $a_i = x$, return, $+, *$, etc to be constant # of operations

S. KIMMEL

Issues: (Brainstorm)

- too fine-grained / detailed
  - different computers might do operations differently
  - when n gets large, don't care about 100000 vs 100001
- too difficult to count every operation

Big-O to Rescue!

special notation to describe how functions grow

def: Let $f, g: \mathbb{Z} \to \mathbb{R}^{+/0}$ or $f, g: \mathbb{R} \to \mathbb{R}^{+/0}$, $\mathbb{R}^{+/0} = \{x : x \geq 0 \wedge x \in \mathbb{R}\}$

- Then $\underbrace{f(x) \text{ is } O(g(x))}$ if $\exists\, k, C \in \mathbb{R}^{+} : \forall\, x \geq k$,

$$f(x) \leq C g(x).$$

"f of x is big-oh of g of x"

"big omega"

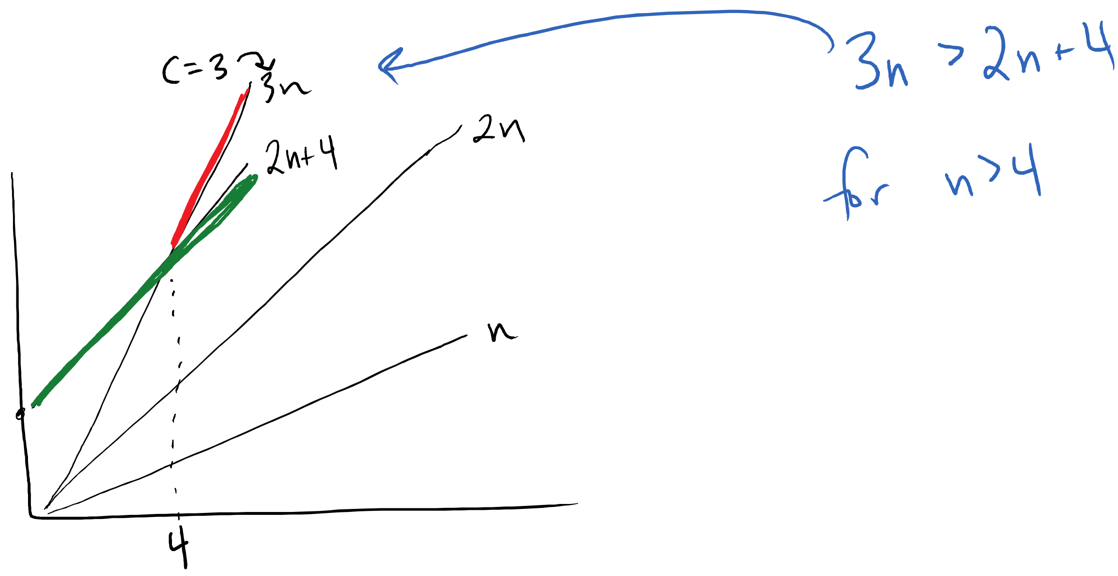
- $f(x)$ is $\Omega(g(x))$ if $\exists\, k, C \in \mathbb{R} : \forall\, x \geq k$

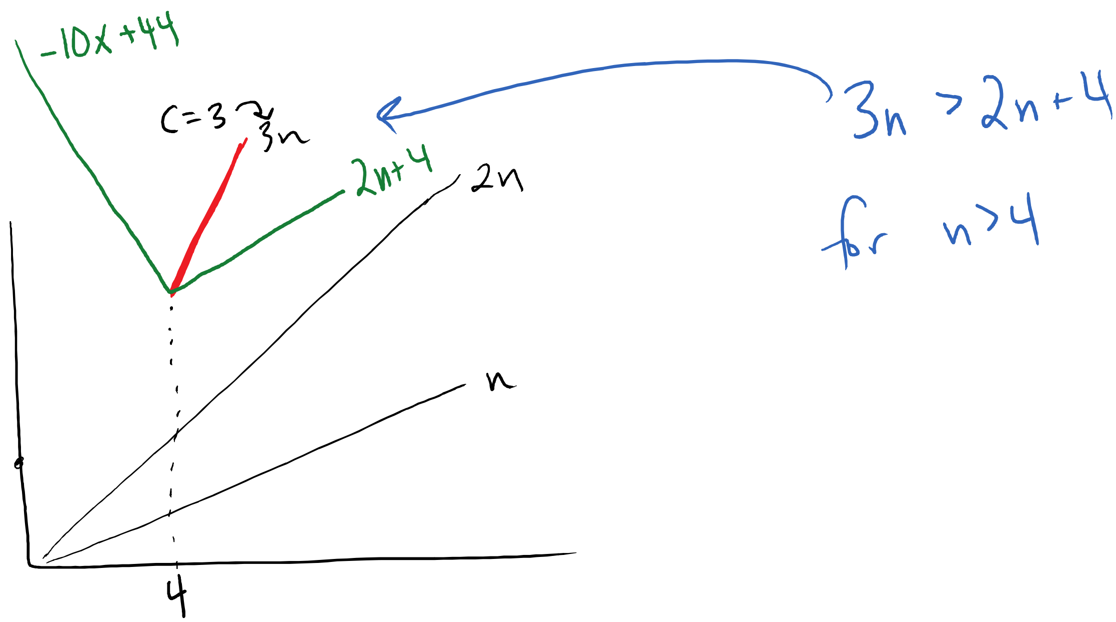
$$f(x) \geq C g(x)$$

- $f(x)$ is $\Theta(g(x))$ if $f(x) = O(g(x))$ and $f(x) = \Omega(g(x))$

big theta

S. KIMMEL





Discuss

For time complexity, we only care about large input sizes, and we only care about the scaling, not the detailed function.

Why do we want these two things for time complexity? How does big-O notation capture these two desiderata? (Which idea corresponds to C and which to k?)

S.KIMMEL

Discuss

For time complexity, we only care about large input sizes, and we only care about the scaling, not the detailed function.

Why do we want these two things for time complexity? How does big-O notation capture these two desiderata? (Which idea corresponds to C and which to k?)

scaling

large input sizes

Slide Problems

A

```
While 0≤n≤100 do
    n=n-1
end
Print "All Done"
```

$T_A(n) = O(1)$ with

$k = 101$, $C = 2$

Input	Time
1	2
2	3
3	4
4	5
⋮	
100	101
101	2
102	2
103	2

S. KIMMEL

Assume for contradiction that

$$2x^2+10 = O(x)$$

This means $\exists\, k, C \in \mathbb{R}^+$ s.t. $\forall\, x \geq k$, $2x^2+10 \leq Cx$

Since we are only considering $x \geq k$, we are only considering positive x, so if we divide both sides by x, we get $2x + \frac{10}{x} \leq C$. Now consider a value of x that is larger than both k and C.

Then $2x + \frac{10}{x} \geq 2C + 10$ this contradicts that

$$2x + \frac{10}{x} < C.$$