S. KIMMEL

Goals

- Describe why big-O is good for characterizing time complexity.
- Prove big-O bounds on functions.

Reflections

- Relations - will review, but not now
- Base case for stamps: anything larger than 12 is good!
- Graphs - we will do more practice on future Psets

S.KIMMEL

Intro to Algorithm Complexity

$\boxed{\text{Important function in C.S.}}$: worst case time complexity of an algorithm

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

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

(unless parallel computing, this tells you the time the computer will take to run the algorithm. Just multiply by time to do 1 operation)

Linear Search

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

1) $i = 1$
2) while $(i \leq n$ and $x \neq a_i)$
3) $\quad i = i+1$
4) if $i \leq n$:
   return i
5) else:
   return 0

Q: What is $T(n)$ for linear search? (Hint: n is not correct)

Report by group: $3n+1$, $n+3$, $2n+2$

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

S.KIMMEL

Issues:

- 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 $f, g: \mathbb{Z} \to \mathbb{R}^+$ or $f, g: \mathbb{Z} \to \mathbb{N}$,

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"

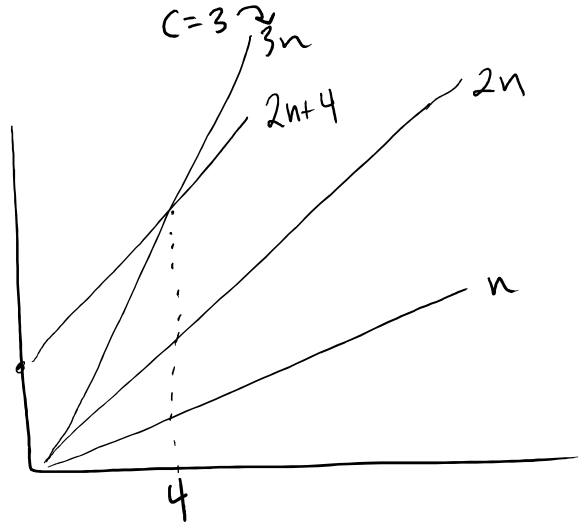
Note

- ✲ Just need to find a C, k pair that works. Doesn't need to be smallest
- ✲ Role of k: only care about large input sizes
- ✲ Role of C: only care about general scaling (not detailed)

S.KIMMEL

ex: $2n+4$ is $O(n) \equiv \exists\, k, C \in \mathbb{R}: 2n+4 \le Cn \quad \forall\, n \ge k$

(often use x, n as function inputs)



$\Longleftarrow$ Not a proof.

Proof:

- For $n \ge 1$, we have $4n \ge 4$. (Multiply both sides by 4). Thus for $n \ge 1$, $2n+4 \le 2n + 4n = 6n$, so $2n+4 = O(n)$ with $k=1$, $C=6$.

- For $n \ge 4$, we have $2n+4 \le 2n + n = 3n$, so $2n+4 = O(n)$ with $k=4$, $C=3$.

Infinitely many combos of k, C work. To prove, you need to find <u>ONE</u> combo.

General trick: try to get $f(x)$ to look like $g(x)$ by turning bad terms into good terms.

Above $4 \to 4n$ or $4 \to n$

S.KIMMEL

Back to C.S.:

All of our functions for linear search # of operations are $O(n)$.

Starting to see why big-O is good for algorithm time complexity:

- Small differences in how you calculate don't matter
- Not too fine grained
- Big-O only cares about large input sizes (larger than k)

However big-O is only upper bound:

ex: Prove: $7x+1$ is $O(x^2)$

Pf: We have $7x+1 \le 7x+x$ for all $x \ge 1$

Then $7x+x = 8x \le x^2$ for all $x \ge 8$

Thus with $k=8$, $C=1$, $7x+1 = O(x^2)$

S.KIMMEL

Q: Prove $10x^2$ is not $O(x)$. This means There do not exist constants k, C, such that $10x^2 \le Cx$ $\forall x \ge k$.

Pf: For contradiction, assume k, C exist. Then $\forall x \ge k$, we have

$$10x^2 \le Cx$$

When $x > 0$, we have $x \le \frac{C}{10}$. Thus, this inequality holds only when $0 < x \le \frac{C}{10}$, which contradicts that it should hold for all $x \ge k$.