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
Functions

Intro to Algorithm Complexity

Important function: worst case time complexity of an algorithm

\[ T_A : D \to \mathbb{N}, \text{ for } D \subseteq \mathbb{N} \]

\[ T_A(n) = \# \text{ of operations performed by algorithm A in worst case on input size n.} \]

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

1. \(i = 1\)
2. while \((i \leq n \text{ and } x \neq a_i)\)
3. \(i = i + 1\)
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.
Issues:
- too fine-grained/detailed
  - different computers might do operations differently
  - when n gets large, don't care about 10,000 vs 10,0000
- too difficult to count every operation

Big-O to Rescue!

↑
special notation to describe how functions grow

**def**: Let $f, g : \mathbb{Z} \rightarrow \mathbb{R}^+$ or $f, g : \mathbb{Z} \rightarrow \mathbb{N}$,

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

$$f(x) \leq Cg(x).$$

"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)
ex: \(2n + 4\) is \(O(n)\) \(\equiv\) \(\exists K, C \in \mathbb{R}: 2n + 4 \leq Cn\) \(\forall n \geq k\) (often use \(x, n\) as function inputs)

\[\begin{array}{c}
\text{not a proof.}
\end{array}\]

Proof:

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

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

Infinitely many combos of \(k, C\) work. To prove, you need to find ONE combo.

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

Above \(4 \rightarrow 4n\) or \(4 \rightarrow n\)
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 $7x + 1 \leq 7x + x$ for all $x \geq 1$

Then $7x + x = 8x \leq x^2$ for all $x \geq 8$

Thus with $k = 8$, $C = 1$, $7x + 1 = O(x^2)$
Q: Prove $10x^2$ is not $O(x)$. This means there do not exist constants $k, c$, such that $10x^2 < cx$ for all $x \geq k$.

Pf: For contradiction, assume $k, c$ exist. Then for all $x \geq k$, we have

$$10x^2 < cx$$

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