Analysis of Algorithms

January 7

- What do we look for in an algorithm?
  1. Correctness: In the context of this class, this means it gets the exact right answer every time
  2. Speed: Harder to define

- How do we define the speed of an algorithm?

  - Definition: The time complexity of an algorithm is ~~the~~ a function $T(n)$ [where $n \in \mathbb{Z}^+$] where $T(n)$ is the number of steps executed alh run on a problem of size n, in the worst case

    Note: key decisions made here:
    - Worst case: We look at the longest running time for any input of size n because it gives a guarantee of performance
    - Function of input size: Input size tends to drive run time
    - "Number of steps": We assume each line of pseudocode takes constant time, for simplicity / generality

- <u>Example</u>: Consider the following algorithm:

```
A(list, x):
  1.  for i = 1 to n:
  2.      if list[i] = x, return TRUE
  3.  return FALSE
```

Then $T(n) = n(c_1 + c_2) + c_3$
(in the worst case, x is not in the list)

We usually make simplifying abstractions to ease the analysis. We <u>drop constant factors and lower order terms</u> from time complexity to get "asymptotic growth rate".

In this case, we say
$$T(n) = n(\cancel{c_1 + c_2}) + \cancel{c_3} = O(\underline{n})$$

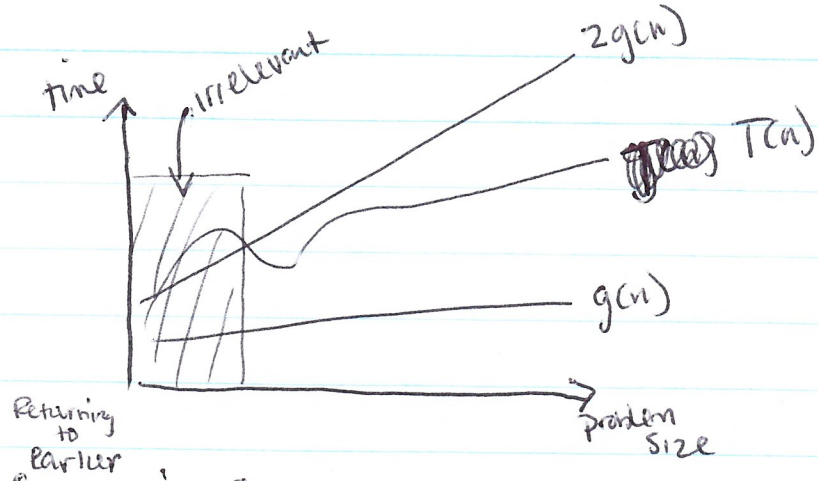
- "Big-O" notation:

More formally, we define:

another function

- <u>Definition</u>: A function $T(n)$ is $O(\boxed{g(n)})$ iff there exists a constant $c > 0$ so that $T(n) \leq c \cdot g(n)$ for all n above some threshold

→ Practically, this is math notation for an upper bound on the asymptotic growth of a function, and for our purposes the function $T(n)$ is usually the time complexity of some algorithm



- <u>Example</u>: Show $n(c_1+c_2)+c_3$ is $O(n)$

  Let $c = c_1+c_2+c_3$

  Then $cn = (c_1+c_2)n + c_3 n$

  $\geq (c_1+c_2)n + c_3$ when $n > 1$

  $\therefore O(n)$

- <u>Example</u>: Show all polynomials $p(n) = a_0 + a_1 n + \dots + a_d n^d$ are $O(n^d)$:

  Let $c = |a_0| + |a_1| + \dots + |a_d|$

  Then $cn^d = |a_0|n^d + \dots + |a_d|n^d$

  $\geq |a_0| + |a_1|n + |a_2|n^2 + \dots |a_d|n^d$ when $n > 1$

  $\geq a_0 + a_1 n + \dots + a_d n^d$

  $\therefore O(n^d)$

- <u>Example</u>: Some functions that are $O(n^2)$:

  n^2, n^2+n, $1000n^2+1000n$, $n^{1.999}$, n

- More useful rules:
  - For all $x > 0$: $\log n = O(n^x)$
  - For all $r > 1$, $d > 0$: $n^d = O(r^n)$

    → Polynomials are always preferred over exponentials, regardless of degree or constants

- Related Notation:

  Definitions:

  - $T(n)$ is $\Omega(g(n))$ iff $\exists\, c>0$ s.t. $T(n) \geq c\,g(n)$ eventually
  - $T(n)$ is $\Theta(g(n))$ iff $\exists\, c_1, c_2 > 0$ s.t.

    $c_1 g(n) \leq T(n) \leq c_2 g(n)$ eventually

- Theorem: $T(n)$ is $\Theta(g(n))$ iff $T(n)$ is $O(g(n))$ and $\Omega(g(n))$

  ↳ Example: Show $\sum_{1 \leq i \leq n} i = \Theta(n^2)$:

$$O: \sum_{1 \leq i \leq n} i \leq \sum_{1 \leq i \leq n} n = n^2 = O(n^2)$$

$$\Omega: \sum_{1 \leq i \leq n} i \geq \sum_{\frac{n}{2} \leq i \leq n} i \geq \sum_{\frac{n}{2} \leq i \leq n} \frac{n}{2} \geq \left(\frac{n}{2}\right)^2 = \Omega(n^2)$$

- Properties:
  - Transitivity: $f = O(g),\ g = O(h) \Rightarrow f = O(h)$

    Same w/ Ω, Θ
  - Reflexivity: $f(n) = O(f(n))$, same w/ Ω, Θ
  - Symmetry: $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$, for Θ only
  - Transpose Symmetry: $f = O(g) \Leftrightarrow g = \Omega(f)$, for O, Ω only