

2/7 Asymptotic Notation

(formal way to model growth of functions)

Three Notations:

1. Big-Oh: $O()$
2. Big-Omega: $\Omega()$
3. Theta: $\Theta()$

Big-Oh

***Represents the upper bound of an algorithm's running time**

- Formal Definition: The function $f(n) = O(g(n))$ if
 - \exists (there exists) constants c and n_0 such that..
 - $f(n) \leq c * g(n)$
 - For all $n \geq n_0$
 - Then it will be true
 - Usually we think $n \geq 1$ since we don't worry abt empty/neg sized arrays/data (works if nothing if specified)

Ex: math example

- $f(n) = 10n + 100$
- A $c * g(n)$ combo that is $\geq f(n)$ for $n \geq 1$
 - $10n + 100 \leq 110n$ for all $n \geq 1$
 - $10n + 100 = O(n)$ [$c = 110$, $g(n) = n$]
 - Trick to quickly find function that satisfies:
 - Take summation of highest degree terms by looking at the $f(n)$
 - Eg: $4n + 7 \leq 4n + 7n \rightarrow 4n + 7 \leq 11n$ for all $n \geq 1$
 - $4n + 7 = O(n)$ [$c = 11$, $g(n) = n$]
- ~~**but we can't~~ we can say that for $f(n) = 10n + 100$
 - $10n + 100 \leq 110n^2$, for all $n \geq 1$ i.e $f(n) = O(n^2)$
 - Works for higher order like $O(n^3)$
 - Yes it can
- But we try to use a $g(n)$ that as close as possible, b/c its not helpful to say i am shorter than an giraffe
 - We need a frame of reference that is close

Big-Omega

*represents the lower bound of an algorithms running time

- Formal Def: the function $f(n) = \Omega g(n)$ if
 - There exists constants c and n_0
 - $c \cdot g(n) \leq f(n)$
 - For all $n \geq n_0$

Ex: math example

- $f(n) = 10n + 100$
- Can you think of a $c \cdot g(n)$ combo that is less than/ equal to $f(n)$ for $n \geq 1$
 - Possible solution: $n \leq 10n + 100$, for all $n \geq 1$
 - $10n + 100 = \Omega(n)$
 - Similar to $O(n)$ the following are also true
 - $10n + 100 = \Omega(\sqrt{n})$
 - $10n + 100 = \Omega(\log n)$
 - $10n + 100 = \Omega(1)$
 - But we want to get as close of a bound for it to be helpful, there is no point saying i am definitely bigger than a mouse

Theta

*represents BOTH upper and lower bounds of an algorithms running time

Formal Definition:

- The function $f(n) = \theta g(n)$ if..
 - $C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$
 - For all $n \geq n_0$

Ex: $10n + 100$

Alr know lower and upper bounds of this fn

Possible sol.

- $n \leq 10n + 100 \leq 110n$, for all $n \geq 1$
- $C_1 = 1, C_2 = 110$
- Then $10n + 100 = \theta(n)$
 - We can't say that $10n + 100 = \theta(n^2)$ or $\theta(\log n)$ (for this def)

Visual Summary

Check slides

Ok to cross the $c \cdot g(n)$, if its before the n_0 , for $O(n)$, shows the upper bound

- Opposite of omega
- For θ , we're trying to trap $f(n)$ between the two boundaries, if it isn't true before n_0 it is fine, but after n_0 everything is true
- $N_0 = 1$, since its p much useless for us to do an empty array, which is special case

Practice:

$$f(n) = n^2 + 2n + 1, n_0 = 1$$

Big-o : factor out the largest term?

$$- : 4n^2, c = 4, g(n) = n^2, \rightarrow O(n^2)$$

big-omega: $\frac{1}{4}$, \rightarrow we want the closest one which is $\rightarrow \omega(n^2)$

- **Take the biggest term on its own.,**
- $n^2 \leq n^2 + 2n + 1$, we want the closest one so, $o(n)$ still works but we want the closest so $\omega(n^2)$

$$\text{theta} : \frac{1}{4} \leq n^2 + 2n + 1 \leq 4n^2, \rightarrow$$

- Since $f(n) = O(n^2)$ and $f(n) = \omega(n^2)$, we can conclude that $f(n) = \theta(n^2)$

Ex2:

$$f(n) = n^2 \log n + 2n; (n_0 = 1)$$

big -O: just take the highest order term

- $O(n)$: $n^2 \log n + 2n \leq n^2 \log n + 2n^2 \log n$
- **$O(n^2 \log n)$**

big-Omega:

- $\omega(n^2 \log n)$

Theta:

- $\theta(n)$: since both big=0 and big-omega are $n^2 \log n$, we can conclude that
- Theta is $n^2 \log n$

Ex3: practice finding out $O(n)$, $\Omega(n)$, Theta

$$f(n) = n!, n_0 = 1$$

$$O: n! = n \cdot (n-1) \cdot (n-2) \rightarrow \leq n \cdot n \cdot n \cdot n \rightarrow f(n) = O(n^4)$$

- $O(n^4)$

$$\Omega: 1 \cdot 1 \cdot 1 \cdot 1 \leq n \cdot (n-1) \cdot (n-2) \dots \rightarrow f(n) = \Omega(1)$$

- $\Omega(1)$, ** just because **its a constant** it is $\omega(1)$, not b/c it is 1

Theta:

- Since **upper(big o) and lower(big omega) bounds** are different we **cannot define theta** for this $f(n)$
- We can only say $f(n) = O(n^4)$ and $f(n) = \Omega(1)$

Desmos might help on hw*

*Asymptotic Notation in Code

- We can say that the order of growth for the linear function(defined above) is $O(n)$

Best, worst, and average cases

- Linear searching (goes thru every value)
 - **Best case** if key is in first value (1 comparison)
 - In any size, consider it all , as array grows, comparisons don't
 - In asymptotic notation , the best case time complexity is $O(1)$, $\Omega(1)$, $\Theta(1)$, **can use all three do not confuse best and worse case with upper and lower bound**
 - **Worst Case** if key is last or not in the array
 - As size of array grow, the # of comparisons grow
 - In asymptotic notation, worst case time complexity $O(n)$... works for ohm and they also since is the same
 - **Average case** if key is somewhere in the middle
 - Comparisons do increase as size of array increase
 - Avg comparisons $(1+2+3...n)/2 = n(n+1)/n$
 - Is also $O(n)$ though we probably would expect it to be $n/2$, but in big o , you drop constants