CSCE 750
Fenner

https://cse.sc.edu/~fenner/csce750

Algorithms ⟷ Data Structures

Def: $f : \mathbb{N} \to \mathbb{R}$ is asymptotically positive (eventually positive) if $\exists n_0 \in \mathbb{N}\ \forall n \geq n_0,\ f(n) > 0$.

Ex: $2x - 10$ is asymp positive $n_0 := 6$ $(n_0 > 5)$.

Def: $f, g : \mathbb{N} \to \mathbb{R}$ asympt. pos. (1) Say that $f \in O(g)$ [$f(n) = O(g(n))$] if $\exists c > 0, \exists n_0\ \forall n \geq n_0,\ f(n) \leq c\,g(n)$.

Think: O —— $\leq$

(2) $f \in \Omega(g)$ [$f(n) = \Omega(g(n))$] if $\exists c > 0, \exists n_0 \forall n \geq n_0$, $f(n) \geq c\,g(n)$

Think Ω —— $\geq$

Fact: $f \in O(g)$ iff $g \in \Omega(f)$

Use asymptotic growth rates to measure resource (time, memory) usage of algorithms.

Def: $f \in \Theta(g)$ [$f(n) = \Theta(g(n))$] if $f \in O(g)$ and $f \in \Omega(g)$.

Θ —— $=$ (equals)

"f & g are asymptotically similar."

Don't care about constant factors.

Why? Want (machine / compiler / prog lang) independence

measure the algo itself, not a particular implementation.

Also, asymptotic analysis is just easier!

Don't care about small input size (small values of n) —— anything runs fast on small inputs — not enough to distinguish between good & bad algos.

Ex: $3n^2 + 10n - 20 = \Theta(n^2)$

Proof:

(1) $3n^2 + 10n - 20 = O(n^2)$

$3n^2 + 10n - 20 \leq 4n^2$

iff $n^2 \geq 10n - 20$.

True if $n^2 \geq 10n$,

iff $n \geq 10$ set $n_0 := 10$ // upper bound

(2) $3n^2 + 10n - 20 \geq 2n^2$

True iff $n^2 + 10n - 20 \geq 0$

$n^2 + 10n \geq 20$

if $n^2 \geq 20$

iff $n \geq \sqrt{20}$ $n_0 := 5$

$\therefore 2n^2 \leq 3n^2 + 10n - 20 \leq 4n^2$

for all $n \geq 10$.

$\therefore 3n^2 + 10n - 20 = \Theta(n^2)$

Two basic techniques:

Analizing sums
for algos that have loops.

Solving recurrences
for recursive algorithms.

```
sum = 0;
for (i=1; i<=n; i*=2)
    for (j=1; j<=i; j++)        } Θ(i)
        sum++;
```

Time $= \Theta\left(\#\text{times sum is incremented}\right)$

$$\text{Time} = \Theta\left(\sum_{k=0}^{\log_2 n} 2^k\right) = \Theta(n)$$

Versus

```
sum = 0;
for (i=1; i<=n; i*=2)
    for (j=i; j<=n; j++)        } n-i+1
        sum++;
```

$$\text{Time} = \Theta\left(\sum_{k=0}^{\log_2 n}(n-i+1)\right)$$

$$= \Theta\left(\sum_{k=0}^{\log_2 n} n - \sum_{k=0}^{\log_2 n} 2^k - \sum_{k=0}^{\log_2 n} 1\right)$$

$$= \Theta\left(n\log_2 n - \Theta(n) - \Theta(\log_2 n)\right)$$

$$= \Theta(n\log_2 n) \neq \Theta(n)$$

Sums:

Arithmetic sums:

$$\sum_{k=0}^{n-1} k^c \qquad (c \in \mathbb{R},\ c \geq 0 \text{ constant})$$

$$\sum_{k=0}^{n-1} k = \frac{n(n-1)}{2} = \Theta(n^2)$$

$$\sum_{k=0}^{n-1} k^c = \Theta\left(n^{c+1}\right) = \sum_{k=1}^{n} k^c$$

Proof; Upper bound: $(c \geq 0)$

$$\sum_{k=1}^{n} k^c \leq \sum_{k=1}^{n} n^c = n \cdot n^c = n^{c+1} \quad \therefore \Sigma = O(n^{c+1})$$

Lower bound:

$$\sum_{k=1}^{n} k^c \geq \sum_{k=\frac{n}{2}+1}^{n} k^c \geq \sum_{k=\frac{n}{2}+1}^{n} \left(\frac{n}{2}\right)^c$$

$$\geq \left(\frac{n}{2}\right)\left(\frac{n}{2}\right)^c = \frac{n^{c+1}}{2^{c+1}}$$

$$= \left(\frac{1}{2^{c+1}}\right) n^{c+1} = \Omega\left(n^{c+1}\right)$$

lower bound