CSCE 750
8/21/2024

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

Analysis of Algorithms [& design] ↔ Data Structures

Topics: Growth ~~and~~ rates of functions (asymptotic)

- Summations (algos with loops)
- Recurrences (" " recursive calls)
- Basic algos: sorting, searching, selection
  " data structs: binary trees, heaps, hash tables, graphs
- NP-completeness & polynomial reductions

```
~~for i:=1 to n~~
sum:=0
i:=1
while i ≤ n do:
    ~~j:=1~~
    for j:=1 to i do        — O(n)
        sum++
    i:=2*i
```

$O(n \log_2 n)$

versus

```
sum:=0
i:=1
while i ≤ n do:
    for j:=i to n do        — O(n)
        sum++
    i:=2*i
```

$O(n \log_2 n)$

Finer analysis: Inner loop takes time ②

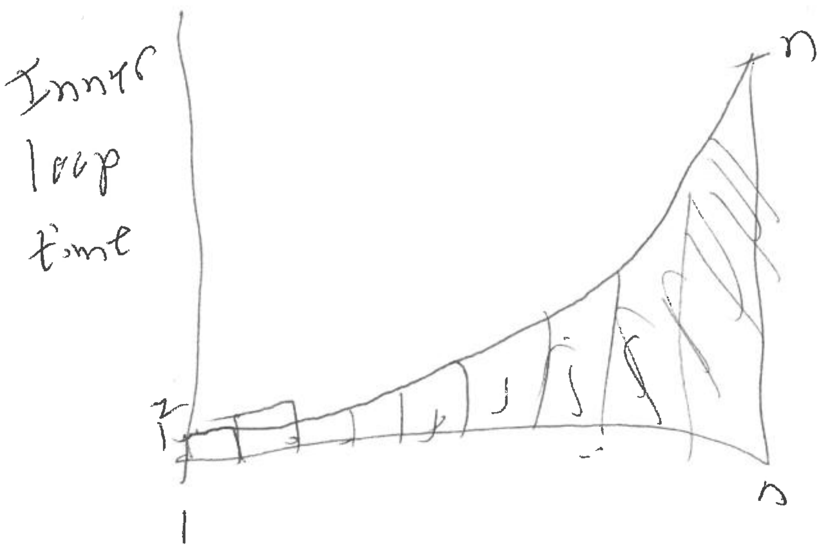
$O(i)$

$1 + 2 + 4 + 8 + \cdots + \text{approx } n$

$\log_2 n$ many terms

$\sim (n + (n-1))$

$\simeq 2n$



$O(n-i)$

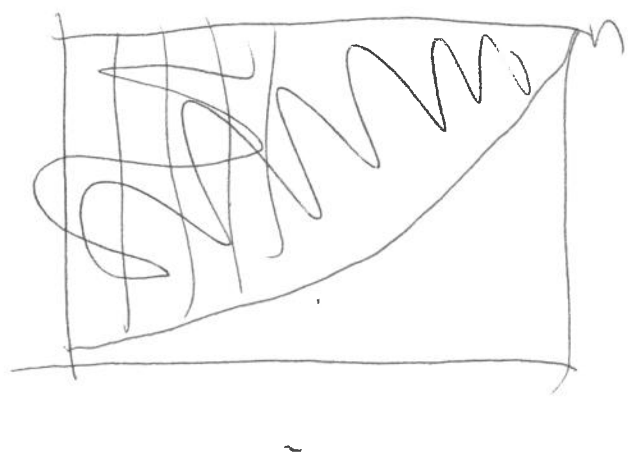
$(n-1) + (n-2) + (n-4) + \cdots (0)$

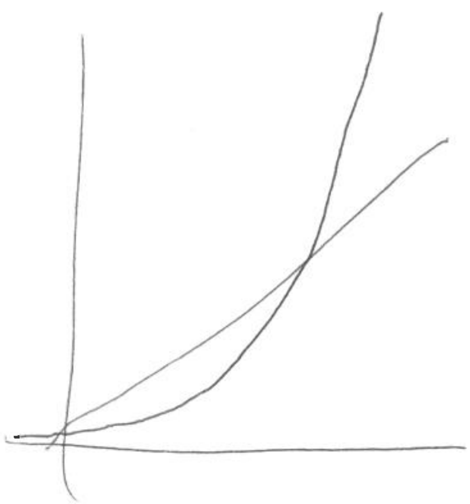
$\log_2 n$ many terms

$n \log_2 n - 2n$

$> 4n \quad (n > 16)$

$> 4n - 2n = 2n$





First loop runs much faster!

Multiply 2 (unsigned) integers in binary (n-bit)

Recursive algorithms (divide & conquer)

Two integers

$$A = A_h \cdot 2^{n/2} + A_\ell$$

$$B = B_h \cdot 2^{n/2} + B_\ell$$

$$AB = (A_h \cdot 2^{n/2} + A_\ell)(B_h \cdot 2^{n/2} + B_\ell)$$

$$= \underbrace{A_h B_h}\cdot 2^n + (\underbrace{A_h B_\ell} + \underbrace{A_\ell B_h})2^{n/2} + \underbrace{A_\ell B_\ell}$$

recursively find these products

1. Arith shift of $A_h B_h$ by n positions left

2. Add $\underbrace{A_h B_\ell + A_\ell B_h}_{O(n)}$ then arith shift left by $\frac{n}{2}$

Add $\underbrace{(1.) \ \& \ (2.) \text{ to } A_\ell B_\ell}_{O(n) \text{ time}}$ → return this

Analysis: this algo takes quadratic time $O(n^2)$ (tight).

Another divide/conquer algo for the same thing:

$$A = A_h \cdot 2^{n/2} + A_\ell$$

$$B = B_h \cdot 2^{n/2} + B_\ell$$

3 recursive calls:

$$A_h B_h, \quad A_\ell B_\ell, \quad \underbrace{(A_h + A_\ell)(B_h + B_\ell)}_{P}$$

$(A_h + A_\ell)(B_h + B_\ell)$

$$= A_h B_h + A_h B_\ell + A_\ell B_h + A_\ell B_\ell = P$$

Return: $2^n \cdot A_h B_h + 2^{n/2}(P - A_h B_h - A_\ell B_\ell) + A_\ell B_\ell$

$= AB$ $\quad O(n)$

Takes time $O(n^{\log_2 3})$ $\quad n^2 = n^{\log_2 4}$

Asymptotic growth rates

$f, g : \mathbb{N} \rightarrow \mathbb{R}$ eventually positive

[$f(n) > 0$, $g(n) > 0$ for all sufficiently large n]

Def: (1) $f(n) = O(g(n))$ [$f \in O(g)$]

means $\exists C, n_0, \forall n \geq n_0,$ $f(n) \leq C \cdot g(n)$

(C: [pos constant]; n_0: threshold)

"f(n) grows asymptotically no faster than g(n)"

(2) $f(n) = \Omega(g(n))$ means $g(n) = O(f(n))$

i.e., $\exists C > 0, \exists n_0, \forall n \geq n_0,$ $f(n) \geq C \cdot g(n)$

"f(n) grows asymptotically at least as fast as g(n)"

(3) $f(n) = \Theta(g(n))$ means $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

"f & g are asymptotically the same"

In analyzing algorithms, we only care about the asymptotic growth rate of the time (or space) required by the algo.

n generally refers to the size of the input.

Why no care const factors? Need machine independence

Why n_0 threshold? Small inputs don't distinguish algorithms well.

Claim: $3n^2 + 10n - 2 = O(n^2)$

Proof: Show that $3n^2 + 10n - 2 \leq 4n^2$

$$\Updownarrow$$

$$10n - 2 \leq n^2$$

$$\Uparrow$$

$$10n \leq n^2$$

true if $n \geq 10$ ← n_0

QED.