Streaming video of lectures

http://video.sc.edu

ID: engineering

PW: APOGEE#2012

Two functions

$f, g : \mathbb{N} \to \mathbb{R}$

$[\mathbb{N} = \{0, 1, 2, \ldots\}$

$\mathbb{R}$ = reals

eventually positive (> 0 for all large enough n)

Time taken by an algorithm = number of primitive operations executed

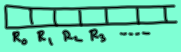
A primitive operation is something that take a constant amount of time (at most), regardless of the input.

- flip a bit
- follow a pointer
- increments a counter
- assign a simple type
- arithmetic on simple type
- comparison of "
- logical ops (AND, OR, NOT, XOR)

Formal (mathematical) model of computation:

- Turing Machine
- RAM model (Random Access Machine)
- Pointer model

RAM model



each R_i can hold an arbitrary integer.

Input resides in $R_1, \ldots, R_{n-1}$

rest of R_i: $R_n, R_{n+1}, \ldots$ are initially 0.

L: $R_6 := R_6 + 4$

Sequence of instructions

if $R_2 > 0$ goto L

goto L

halt

read R_i

write R_i

$R_6 := R_{R_4}$

$R_{R_4} := R_6$

not necessarily primitive

$R_6[2] := -5$

$R_4 := -5$

$R_6[R_4]$

An execution (of an algo) is a sequence of steps according to the semantics above.

First approx: time = # of steps. not realistic:

for $i := 1$ to n

$R_0 := R_0 * R_0$

$2, 4 = 2^{2^1}, 16 = 2^{2^2}$,

$256 = 2^{2^3} = 2^{2^3}$

$65536 = 2^{2^4}$

$\vdots$

2^{2^n}

We'll treat all these as primitive ops, provided contents of registers don't get too big.

Assume input of size n (first n registers)

"Not too big" means there is a constant k such that no reg exceeds $O(n^k)$ in absolute value during the execution.

Equivalently, each reg contents can be encoded in $O(\lg n)$ bits.

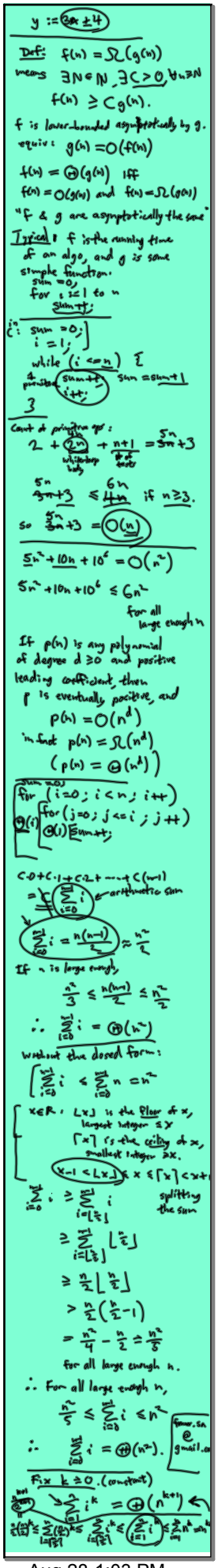
Def: $f(n) = O(g(n))$

means $\exists N \in \mathbb{N}, \exists C > 0, \forall n \geq N$

$f(n) \leq C g(n)$.

"f is asymptotically bounded by g (from above)

Aug 28-12:27 PM


y := 3x ± 4
Def: f(n) = Ω(g(n))
means ∃N∈ℕ, ∃C>0, ∀n≥N
f(n) ≥ Cg(n).
f is lower-bounded asymptotically by g.
equiv: g(n) = O(f(n))
f(n) = Θ(g(n)) iff
f(n) = O(g(n)) and f(n) = Ω(g(n))
"f & g are asymptotically the same"
Typical: f is the running time of an algo, and g is some simple function.
sum = 0;
for i := 1 to n
sum++;
i = 1;
while (i <= n) {
sum++;
i++;
sum = sum + 1
3
Count of primitive ops:
2 + 2n + n+1 = 3n+3
while loop
# of tests
3n+3 ≤ 4n if n≥3.
so 3n+3 = O(n)
5n² + 10n + 10⁶ = O(n²)
5n² + 10n + 10⁶ ≤ 6n²
for all large enough n
If p(n) is any polynomial of degree d ≥ 0 and positive leading coefficient, then
p is eventually positive, and
p(n) = O(n^d)
in fact p(n) = Ω(n^d)
(p(n) = Θ(n^d))
sum = 0;
for (i=0; i<n; i++)
Θ(i) for (j=0; j<=i; j++)
O(1) sum++;
c·0 + c·1 + c·2 + ... + c(n-1)
= c(Σ_{i=0}^{n-1} i)
arithmetic sum
Σ_{i=0}^{n-1} i = n(n-1)/2 ≈ n²/2
If n is large enough,
n²/3 ≤ n(n-1)/2 ≤ n²/2
∴ Σ_{i=0}^{n-1} i = Θ(n²)
without the closed form:
Σ_{i=0}^{n-1} i ≤ Σ_{i=0}^{n-1} n = n²
x∈ℝ, ⌊x⌋ is the floor of x, largest integer ≤ x
⌈x⌉ is the ceiling of x, smallest integer ≥ x.
x-1 < ⌊x⌋ ≤ x ≤ ⌈x⌉ < x+1
Σ_{i=0}^{n-1} i ≥ Σ_{i=⌈n/2⌉}^{n-1} i
splitting the sum
≥ Σ_{i=⌈n/2⌉}^{n-1} ⌊n/2⌋
≥ (n/2)⌊n/2⌋
> (n/2)(n/2 - 1)
= n²/4 - n/2 ≥ n²/8
for all large enough n.
∴ For all large enough n,
n²/8 ≤ Σ_{i=0}^{n-1} i ≤ n²
∴ Σ_{i=0}^{n-1} i = Θ(n²).
Fix k ≥ 0. (constant)
Σ_{i=0}^{n-1} i^k = Θ(n^{k+1})

Aug 28-1:03 PM