01 D&C

Friday, 15 November 2024 07:14

Quick recap for analyzing time complexity.

- count the number of basic operations
- ignore constant factors

Input: n

```
for i=1....n
    for j=1...n
        print("Hello World")
```

$n \cdot n \cdot O(1) = O(n^2)$

← constant $O(1)$

$10 = O(1)$

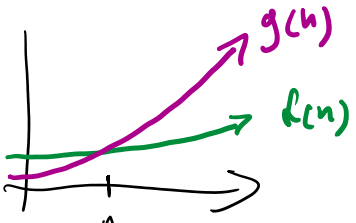
$5 = O(1)$

Formal definition

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

if there exist constants $n_0, c > 0$

$f(n) \leq c \cdot g(n)$ for all $n \geq n_0$



Lemma: $f(n) = O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

Intuition for why the Lemma should be true

$f(n) \leq c \cdot g(n)$

$\Leftrightarrow \frac{f(n)}{g(n)} \leq c \quad \forall n \geq n_0$

$n = O(n^2)$ Proof $\lim_{n\to\infty} \frac{n}{n^2} = \lim_{n\to\infty} \frac{1}{n} = 0 < \infty$

$$O(n \cdot \log(\log(n)))$$

Intuition: $f(n) = O(g(n))$ $f(n)$ "$\leq$" $g(n)$ (Lem) $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$

$f(n) = o(g(n))$ $f(n)$ "$<$" $g(n)$ Def: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$n = o(n^2)$

$5n \neq o(n)$ $5n = O(n)$

$f(n) = \Omega(g(n))$ $f(n)$ "$\geq$" $g(n)$ Def: a) $\lim \frac{g(n)}{f(n)} < \infty$

b) $g(n) = O(f(n))$

$f(n) = \omega(g(n))$ $f(n)$ "$>$" $g(n)$ Def: a) $\lim \frac{g(n)}{f(n)} = 0$

b) $g(n) = o(f(n))$

$\Theta(\ldots)$ $f(n)$ "$=$" $g(n)$ Def: if $f(n) = O(g(n))$

b) $g(n) = o(f(n))$

$f(n) = \Theta(g(n))$ **"f(n) = g(n)"** Def: if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

In practice
- ignore constants
- keep only the largest term

$5n^2 + 10n = O(n^2)$

This extends to the case with more than 1 variable:

$nk + nk^2 = O(nk^2)$ ← keep largest term

$n^2k + nk^2 = O(n^2k + nk^2)$ ← no term is strictly larger than the other, keep both terms

$n^a = O(n^b)$ if $b \geq a$ $\quad n^2 = O(n^3)$

$(\log(n))^a = O(n^b)$ for all $a, b > 0$ $\quad (\log(n))^{1000000} = O(n^{0.00001})$

$n^a = O(b^n)$ for all $a \geq 0$, $b > 1$ $\quad n^{100000} = O(2^n)$

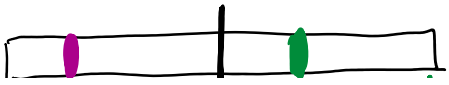
---

5 min break 2:52 pm

Divide & Conquer
- split problem into smaller pieces
- solve smaller pieces via recursion
- combine solutions into answer for the original problem.

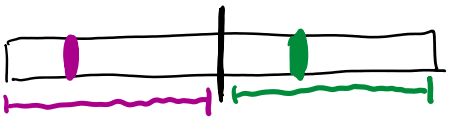
Array A[0... n-1], find $\max_i A[i]$

FindMax(A[0... n-1]) if n==2



```
FindMax (A[0... n-1])
    if n==1
        return A[0]
    left = FindMax( A[0... n/2 -1])
    right = FindMax ( A[n/2 ... n])
    if left > right
        return left
    else
        return right
```

```
if n==2
    if A[0] > A[1]
        return A[0]
    else
        return A[1]
```



if left > max A[i]
$i > \frac{n}{2}$

$\Rightarrow$ left = max A[i]
i

$O(1)$

$T(n)$ = time of FindMax on arrays of length n

$T(n) = 2 \cdot T(\frac{n}{2}) + O(1)$
$O(n^0)$

time to merge answers

Master-Theorem
Theorem: $T(n) = a \cdot T(\frac{n}{b}) + O(n^c)$

b = size decrease

recursive calls

then $T(n) = \begin{cases} O(n^c) & \text{if } c > \log_b(a) \\ O(n^c \log n) & \text{if } c = \log_b(a) \\ O(n^{\log_b(a)}) & \text{if } c < \log_b(a) \end{cases}$

Complexity of FindMax

$\log_b(a) = \log_2(2) = 1 > 0 = c$

$\Rightarrow$ case 3 $T(n) = O(n^{\log_b(a)}) = O(n^1) = O(n)$

$\log_b(n) = \frac{\log_2 n}{\log_2 b}$

depth
a #recursion
$= (2^{\log_2(a)})^{\#\text{rec.}}$
$= 2^{\log_2(a) \cdot \log_b(n)}$
$= n^{(\log_2(a)/\log_2 b)}$