CSC 236 Thursday Lecture

Jan 25, 2018

Slide:

2, 3 - review of running-time algorithm complexity

3 - examples of constant time algorithm: hash tables, array access

3 - examples of log(n) alogorithm: insert, remove, search BST, mergeable BST

- algorithms of degree ≥ 2 are generally too inefficient to use

4 - getting running time of linear search function:

① Chunkify by assigning constants

i=0	$\boxed{=a}$
while i < len(A)	
[--]	$\boxed{=b}$
return -1	$\boxed{=c}$

② $T(n) = a + nb + c$ where $n \leq len(A)$

↳ this is linear!

5 - getting running time of recursive algorithm

① [{if n==1}=a_1 ; {return 1}=a_2; else {return}=a_3 {n×}=a_4 {fact(n-1)}]

② $T(n) = \begin{cases} a_1 + a_2 & \text{if } n = 1 \\ a_1 + a_2 + a_3 + a_4 + T(n-1) & n > 1 \end{cases}$

6 - goal to arrive at closed form running time, when that is not possible we use asymptotic approximation

7 - $\left.\begin{array}{l} T(1) = a \\ T(2) = b + a \\ T(3) = 2b + a \ldots \end{array}\right\} T(n) = (n-1)b + a = nb - b + a$

- Prove by induction claim that closed form $T(n) = (n-1)b + a \ \forall n \in \mathbb{N} \geq 1$

① $P(n)$: $T(n) \stackrel{?}{=} (n-1)b + a$

② Base case $n = 1$: $T(n) = a = (1-1)b + a = (n-1)b + a \ \therefore P(1)$ holds.

③ Assume $P(k)$ for arbitrary $k \in \mathbb{N} \geq 1$, ie $T(k) = (k-1)b + a$

From definition, $T(k+1) = b + T(k) = b + kb - b + a = kb + a = ((k+1)-1)b + a$.

Thus $P(k) \rightarrow P(k+1)$. ④ By PSI, $P(n)$ holds $\forall n \in \mathbb{N} \geq 1$.

Conclude that $T(n) = (n-1)b + a$ is closed form of given $T(n)$

- Now we can state that $T(n) \in O(n)$.

Slide:

8 method 1

- $f(1) = 2 - 1 = 1 = 1^2$
  $f(2) = 1 + 4 - 1 = 4 = 2^2 \quad .. \; f(n) = n^2$
  $f(3) = 4 + 6 - 1 = 9 = 3^2$

method 2

$f(n) = f(n-1) + 2n - 1$
$= f(n-2) + 2(n-1) - 1 + 2n - 1 = f(n-2) + 2[n + n-1] - 2$
$= f(n-3) + 2(n-2) - 1 + 2[$ ... continue unwinding
$= f(0) + 2 - 1 + 2[n + (n-1) + \ldots + 2] - (n-1)$
$= 2[n + (n-1) + \ldots + 2 + 1] - n = 2(n(n+1)/2) - n$
$= n^2 + n - n = n^2$

- do inductive proof to show that n^2 is indeed the closed form

9

- finding running time for divide & conquer recursive algorithm
- $T(n=1) = a_1 + a_2$; $T(n>1) = a_1 + a_3 + a_4 + T(\lfloor n/2 \rfloor)$;
  ↳ note a_i $i \in \{1,2,3,4\}$ assigned to blocks of code on slide
- summarize contents from slide 10, 11

12 m1 – $T(n) = a + \log_2 n \cdot b$; $T(2) = a + b$; $T(4) = a + 2b = a + (\log_2 4) b$

m2 – $T(n) = T(n/2) + b = T(n/2^2) + 2b = T(n/2^i) + ib = T(1) + b\log_2 n + a$

- again: inductive proof that $T(n) = a + b\log_2 n \;\; \forall n \in \mathbb{N} \geq 1$

13

- finding running time of recursive D&C algo w/ two functions
- looking first at merge; done in linear time merge $\in O(n)$
- $T_{merge}(n) = c + dn$; $T_{ms}(n) = \begin{cases} a & n=1, \text{ else:} \\ a_1 + a_2 + T_{ms}(\lfloor n/2 \rfloor) + T_{ms}(\lceil n/2 \rceil) + T_{merge}(\lfloor n/2 \rfloor) \end{cases}$
- $T_{ms}(n) = \begin{cases} a \\ b + 2T_{ms}(n/2) + c + d(n/2) = g + 2T_{ms}(n/2) + e \cdot n \end{cases}$
- can say input size of merge is max (size A, size B) or size A + size B, but it really doesn't matter because both are linear

↳ $T(n) = 2 \cdot T(n/2) + en + g = 2^2 T(n/2^2) + 2en + 2g = \ldots$

$=$