311 Home Work 1

Ben purdy

February 15, 2020

1

a. f(n) = Θ g(n)
b. f(n) = Og(n)
c. f(n) = Θ g(n)
d. f(n) = Og(n)
e. f(n) = Ω g(n)
f. f(n) = Ω g(n)
g. f(n) = Ω g(n)
h. f(n) = Og(n)
i. f(n) = Ω g(n)
j. f(n) = Og(n)
k. f(n) = Og(n)
l. f(n) = Ω g(n)
m. f(n) = Og(n)

2

$$p(n) = \sum_{i=0}^{d} = a_i n^i$$

where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.
(a) If d $\geqslant$, then p(n) = O(nk).
(b) If k $\leqslant$ d, then p(n) = Ω (nk).
(c) If k = d, then p(n) = θ(nk).

Proof. (a)
let c = 1, n = 0
since d $\leqslant$ k the $n^d \leqslant c \cdot n^k$
$\forall n_0 < n, p(n^d) \leqslant c \cdot g(n^k)$
then $p(n) = O(n^k)$

□

Proof. (b)

let c = 1, n = 0 since $d \geqslant k$
$n^k \leqslant n^d \Rightarrow g(n) \leqslant p(n)$
$\forall n_0 < n, c \cdot g(n^k) \leqslant p(n^d)$
then $p(n) = \Omega(n^k)$

□

Proof. (c)

$k = d$ where $g(n) = n^k$ and $f(n) = n^d$
let $c_1 = 2$ and n = 0
then $p(n) \leqslant c \cdot g(n)$ since $n^d \leqslant 2n^k$
this shows us that $p(n) = \Omega g(n)$
now show $c_2 \cdot g(n) \leq p(n)$ for sum $c_2 > 0$
let $c_2 = 1$
$n^d = 1 \cdot n^k$
$p(n) = g(n)$
$p(n) = \theta g(n)$ □

3

Give Big-O bounds, in terms of n, for the worst-case running times of each of the code fragments below. In each case, assume that A and B are arrays of integers. You should formally derive the running times of each algorithm and then give the Big-O result (it does not suffice to merely state running times or give high-level explanations of running times).

(a)

algorithm A
running time is dictated by line 3,5,6
$t_0 = \sum_{i=0}^{n-1} t_1$, $t_1 = \sum_{j=0}^{n-1} t_2$, and $t_2 = \sum_{k=0}^{j} 1$
from A we get $t_1 = T_5$, $t_2 = T_6$

$$\begin{aligned} T(n) &= \sum_{i=0}^{n-1} t_1 \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} t_2 \\ &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{j} 1 \\ &\textit{since } t_2 = j \end{aligned}$$

$$\begin{aligned} t_1 &= \sum_{j=0}^{n-1} j = \frac{n(n+1)}{2} \\ t_0 &= \sum_{i=0}^{n-1} t_1 = n \cdot \frac{n(n+1)}{2} \\ T(n) &= \frac{n^3+n^2}{2} \\ \textit{so } T(n) &= O(n^3) \ \square \end{aligned}$$

(b)
Algorithm B
B contains two loops, so $T(n) = \sum_{i=0}^{n} t_1$ and $t_1 = \sum_{j=1}^{i} 2^i$

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n} t_1 \\
&= \sum_{i=0}^{n} \sum_{j=1}^{i} 2^i \\
&\Rightarrow \sum_{j=1}^{i} 2^i = log_2(n) \\
&\sum_{i=0}^{n} log_2(n) = n \cdot log_2(n) \\
&hence\ T(n) = O(nlog(n)) \ \square
\end{aligned}$$

(c)
Are the big-O bounds of parts (a) and (b) also Θ bounds?
(a) Theta for part A
Let $c_1 = 2, c_2 = \frac{1}{2}$, and $n_0 = 0$
f(n) = O(A) there A is the run time of the algorithm in part A $Og(n) = n^3$
then $f(n) \leq c_1 \cdot g(n)$
hence g(n) is a upper bound to f(n), f(n) = O(g(n))
now to find $\Omega g(n)$
since $c_2 = \frac{1}{2}$,
then $f(n) \geq c_2 \cdot g(n)$
showing that $\Theta(n^3)$ holds $\square$

4

Proof. 4
Initialization: It is true before the first iteration of the loop. shown by left equaling the first index of the array. The right holds because it equals the last index of the array
left = 0, right = A.length-1
so $left \leq right$

Maintenance: since A is sorted the Initialization implies that at no point throughout the algorithm will right > left

Without loss of generality let i ≤ j

now $\forall\ entrys\ left, i \in A.\ left \leq i$

and $\forall\ entrys\ right, j \in A.\ j \leq right$

left ≤ i and j ≤ right ⇒ left ≤ i ≤ j ≤ right
for some given T if there exist indices i, j in A such that A[i] + A[j] = T. Becuase the array is sorted indices left, right will behave based on whether A[left] + A[right] > T or A[left] + A[right] < T.

case 1: A[left] + A[right] > T
then by moving the right to the left by one
A[left] + A[right] ≥ A[left] + A[right-1] holds

case 2: A[left] + A[right] < T.
then by moving the left to the right A[left] + A[right] ≥ A[left] + A[right-1] holds

case 3: A[i] + A[j] = T
the algorithum terminates because we have found indices i,j

Theses three cases will insure that the left, right will found indices i, j if indices i, j exist

Termination: At termination, left > right
since left = 0 and right = n-1 in array A[0,..,n-1] for left > right the whole array will have been iterated through. Because A is a finite set this algorithm will terminate □

5

A full binary tree is a binary tree where each node is either a leaf or has exactly 2 children. There are no nodes with only one child. The internal path length of a full binary tree is the sum, taken over all internal nodes of the tree, of the depth of each node. Likewise, the external path length is the sum, taken over all leaves of the tree, of the depth of each leaf. Consider a full binary tree with n internal nodes, internal path length i, and external path length e. Prove that e = i + 2n.

Proof. By Induction

Base: let n = 1 then the graph has exactly 2 leaf attached to the root.
so i = 0 and e = 2 since the path to each leaf is one.
e = i + 2(n) = 0 + 2(1) = 2

Step: Assume theorem holds for the hole binary tree with n-2 nodes

let s be the hight of a node with two leafs removing the leafs from that node we get

now I' = I-s and N' = N-1

$$\begin{aligned} E &= E' + 2(s+1) - s = E' + s + 2 \\ or\ E' &= E - s - 2 = I + 2N - s - 2 \\ plug\ in \\ I &= I' + s \\ N &= N' + 1 \\ so\ E' &= I' + s + 2(N'+1) - s - 2 \\ &= I' + 2N' \end{aligned}$$

□