CS 101, Midterm

Christopher Huynh

May 2016

Q1

A

Answer A is incorrect because if $n^{1.1} = O(nlog_2 n)$, then $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \lim_{n\to\infty} \frac{n^{1.1}}{nlog_2 n} \leq c$, where c is a positive non-zero integer. If you find $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \lim_{n\to\infty} \frac{n^{1.1}}{nlog_2 n} = \lim_{n\to\infty} \frac{n^{0.1}}{log_2 n} \overset{L'H}{=} \lim_{n\to\infty} \frac{\frac{0.1}{n^{0.9}}}{\frac{1}{nln(2)}} = \lim_{n\to\infty} \frac{0.1nln(2)}{n^{0.9}} = \lim_{n\to\infty} 0.1n^{0.1}ln(2) = \infty$. There is no possible positive non-zero integer, c, that is greater than ∞, therefore, $n^{1.1} \neq O(nlog_2 n)$

C

Answer C is incorrect because if $nlog_2 n = \theta(n^{1.1})$, then $nlog_2 n = O(n^{1.1})$ and $nlog_2 n = \Omega(n^{1.1})$. If $nlog_2 n = \Omega(n^{1.1})$, then $\lim_{n\to\infty} \frac{T(n)}{f(n)} = \lim_{n\to\infty} \frac{nlog_2 n}{n^{1.1}} \geq c$ where c is a positive non-zero integer. If you find $\lim_{n\to\infty} \frac{nlog_2 n}{n^{1.1}} = \lim_{n\to\infty} \frac{log_2 n}{n^{0.1}} \overset{L'H}{=} \lim_{n\to\infty} \frac{\frac{1}{nln(2)}}{\frac{0.1}{n^{0.9}}} = \lim_{n\to\infty} \frac{n^{0.9}}{0.1nln(2)} = \lim_{n\to\infty} \frac{1}{0.1n^{0.1}ln(2)} = 0$. There is no possible positive non-zero integer, c, that is less than 0, therefore $nlog_2 n \neq \Omega(n^{1.1})$ so $nlog_2 n \neq \theta(n^{1.1})$

Q2

B

In order for the sorting algorithm to be in $\theta(n^2)$, the sorting algorithm must be in $O(n^2)$ and $\Omega(n^2)$. Because the sorting algorithm runs Mergesort on input lengths larger than 10^9 , the sorting algorithm runs in $O(nlog_2 n)$ and $\Omega(nlog_2 n)$i for the largest of input lengths. Therefore, B is incorrect.

C

$T(n) = \Omega(nlog_2 n)$ because for the largest of largest inputs, $T(n) = \theta(nlog_2 n)$ because it runs mergesort and mergesort is in $\theta(nlog_2 n)$. Therefore C is incorrect because $T(n) = O(nlog_2n)$ and $T(n) = \Omega(nlog_2n)$

Q4

Merge-Sort(A)

$n = A.length$
if $n == 1$, **return** A
$mid = n/2$
$L = A[1..mid]$
$R = A[(mid + 1)..n]$
$sortL =$ Merge-Sort(L)
$sortR =$ Merge-Sort(R)
return merge$(sortL, sortR)$

merge(B, C)

$B[m + 1] = C[m + 1] = \infty$
#Initialize array D
$i = j = k = 0$
$while(B[i] < \infty)$ or $(C[j] < \infty)$:
  if$(B[i] \leq C[j])$: #modification is $\leq$ instead of just $<$
    $D[k] = B[i]$
    $i = i + 1$
  else :
    $D[k] = C[j]$
    $j = j + 1$
  $k = k + 1$
return D

Proof Modified Merge Sort is Stable
Base case: n = 1: Merge-Sort returns A, which is trivially sorted
Induction: Assume n > 1 and Merge-Sort works. By (strong) induction hypothesis, Merge-Sort correctly sorts any input of size $\leq$ n - 1. Specificaly, Merge-Sort works on L and R. Thus, $sortL$ and $sortR$ are sorted version of L and R respectively. Since Merge works correctly and ensures that elements of same value stay in stabe order, ouput is a sorted stable version of A.
Therefore by proof by induction, this modified Merge-Sort is stable. ■

Proof that Quicksort is not stable
Let $A = [3_1.6_1, 6_2, 4_1]$ and A is inputed into Quicksort.
First step: Chooses 6_1 as pivot.
Second step: Starts Partitioning A
Third step: $i = 0$, A[0] ¡ 6_1, i = i + 1 = 1, which is the pivot so $i = 1$
Fourth step: $j = 3$, A[3] ¿ 6_1, so nothing happens and $j = 3$
Fifth step: $A[i]$ swaps with $A[j]$, array A is now $[3_1, 4_1, 6_2, 6_1]$
A is sorted but is not in stable order.

Therefore by proof by contradiction, Quicksort is not stable.

Q5

Q6

Pseudocode for Algorithm that returns index of sorted array
1 #Create Array B that is Array with tuples where the second value is the position of the element, e.g if A = [5,4,3,2,1] then B = [(5,1),(4,2),(3,3),(2,4),(1,5)]. This runs in O(n).
2 B = Merge-Sort(B) #This runs in $O(nlog_2n)$ #The mergesort sorts B so that from the above example B = [(1,5),(2,4),(3,3),(4,2),(5,1)]
3 #Return an array, C , that is only the second part of tuple, e.g. from the above example C = [5,4,3,2,1]

The above pseudocode would run in $O(O(n) + O(nlog_2n) + O(n)) = O(nlog_2n)$