COM S 311

Homework 4

Recitation 5, 1-2pm, Marios Tsekitsidis

Christian Shinkle

March 25, 2018

1. (a) $T(1) = 1$; $T(n)$
$= 3T(n/2) + n$
$= 3(3T(n/2^2) + n/2^1) + n = 3^2T(n/4) + 3/2n + n$
$= 3(3^2T(n/2^3) + 3/2n/2 + n/2) + n = 3^3T(n/8) + (3/2)^2n + 3/2n + n$
$= 3^kT(n/2^k) + n\sum_{i=0}^{k-1}(3/2)^i$ where $k = \log_2 n$
$= 3^{\log n}T(n/n) + n\sum_{i=0}^{\log n-1}(3/2)^i$
$= n^{\log 3} + n\sum_{i=0}^{\log n-1}(3/2)^i$
$= n^{\log 3} + n(\frac{1-(3/2)^{\log n}}{1-3/2})$
$= n^{\log 3} - 2n(1 - (3/2)^{\log n})$
$= n^{\log 3} - 2n(1 - n^{\log 3/2})$
$= n^{\log 3} - 2n(1 - n^{\log(3)-1})$
$= n^{\log 3} - 2n + 2n * n^{\log(3)-1}$
$= n^{\log 3} - 2n + 2n^{\log 3}$
$= 3n^{\log 3} - 2n$

   (b) $T(1) = 1$; $T(n)$
$= T(n/8) + n$
$= T(n/8^2) + n + n/8$
$= T(n/8^3) + n + n/8 + n/8^2$
$= T(n/8^k) + n\sum_{i=0}^{k}(1/8)^i$, where $k = \log_8 n$
$= 1 + n(\frac{1-(1/8)^{\log_8 n+1}}{7/8})$
$= 1 + \frac{8}{7}n(1 - \frac{1}{8n})$
$= \frac{8}{7}n + \frac{6}{7}$

2. BuildBST:
Input: Sorted Array arr
Integer $length$ = length of arr
if arr is empty
    return null
Node $root.data = arr[length/2]$
$root.left$ = BuildBST(subarray of arr from indices 0 to $length/2 - 1$)

$root.right$ = BuildBST(subarray of arr from indices
$length/2+1$, $length-1$)
return root

Note that when the indices for subarray are out of bounds(i.e. the starting index is greater than the end index), it will create an empty arr.

Recurrence relation: $T(n) = 2T(\frac{n-1}{2}) + c_1$, $T(1) = 1$.
Note $T(n) = 2T(\frac{n-1}{2}) + c_1 \leq 2T(\frac{n}{2}) + c_1$, so we will use the second function to analize the runtime.
Then, $T(n) = 2T(\frac{n}{2}) + c_1$
$= 2^1(2T(\frac{n}{2^2}) + c_1) + c_1) = 2^2T(\frac{n}{2^3}) + 3c_1$
$= 2^2(2T(\frac{n}{2^3}) + c_1) + 3c_1) = 2^3T(\frac{n}{2^3}) + 7c_1$
$= 2^3(2T(\frac{n}{2^4}) + c_1) + 7c_1) = 2^4T(\frac{n}{2^4}) + 15c_1$
$= 2^kT(\frac{n}{2^k}) + (2^k - 1)c_1$, where k = number of iterations.
Note $1 = n/2^k$ implies $k = \log n$, so
$2^{\log n}T(n/2^{\log n}) + (2^{\log n} - 1)c_1$
$= n + (n-1)c_1 \in O(n)$

3. I don't know how to solve this problem.

4. For G', E' will be an adjacency list composed of an Array of Lists. The size of the Array will be the number of vertices in G. In order to add to the adjacency list, the method add() will be used, which takes two arguments:Vertex u, Vertex v, where u is the location in the Array in the adjacency list and v is the Vertex that is added to E'.

   Algorithm:
   Input: Adjacency List G
   Create HashTable h which holds pairs of Vertices
   Create G' with V' being a copy of V and an empty adjacency list for E'
   for each Vertex a in the Array of G
       for each Vertex b in the List at a in the Array of G
           for each Vertex c in the List at b in the Array of G
           if h doesn't contain pair $< a, b >$
               add $< a, b >$ to h
               G'.add(u, v)

   return G'

   Runtime:
   Preparing the HashTable will take constant time and copying V into V' will take n time. The outer for-loop runs n times because it check every Vertex in the Array of G. The middle-loop and inner-loop runs m times

each because, in the worst case, the Array at a may contain m edges. The if-statement takes constant time because, in order to check if a HashTable contains an element, it requires a search of the HashTable, which takes average/expect constant time. Adding to a HashTable takes average/expect constant time and the add function for G' takes constant time because accessing an Array takes constant time and adding to a List also takes constant time. So, the inner most code takes a constant amount of time. The following expression describes the runtime:
$n + \sum_{i=0}^{n} \sum_{j=0}^{m} \sum_{k=0}^{m} c$
$= n + nm^2c \in O(nm^2)$