CS21003 Algorithms–I, Spring 2018–2019

Class Test 2

09–March–2019 F116/F142, 06:00pm–07:00pm Maximum marks: 20

Roll no: ____________ **Name:** ______________________

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions. If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.*]

1. Let T be a binary search tree (BST) with n nodes and of height h. Let the keys stored in T be $k_1, k_2, k_3, \ldots, k_n$ in the ascending order. Assume that all these keys are positive integers. For $i = 1, 2, 3, \ldots, n$, denote the prefix sums of the keys by $p_i = k_1 + k_2 + k_3 + \cdots + k_i$. We want to update the keys at T such that it stores the prefix sums $p_1, p_2, p_3, \ldots, p_n$, and continues to remain a BST with respect to these updated key values. Propose an $O(n)$-time and $O(h)$-space algorithm to solve this problem. Notice that if you store the keys of T in an external array, you use $\Theta(n)$ space, but only $O(h)$ extra space is allowed, and we may have $h = o(n)$. Assume that each node in T contains a key and two child pointers (and nothing else). Do not add any extra field to the nodes. Your algorithm must not alter the structure of the input tree; only the key fields are to be updated in place. Write a proper pseudocode of your algorithm. **(6)**

Solution A modification of the standard inorder traversal of T does this job. Here, we maintain the prefix sums in a variable *sum*. This may be a global variable or passed via a pointer to the following function.

```
prefixsum ( BST T )
{
    if (T == NULL) return;
    prefixsum(T −> L);
    sum += T −> key;
    T −> key = sum;
    prefixsum(T −> R);
}

/* Outermost call */
sum = 0;
prefixsum(root);
```

2. Let T be a binary search tree with n nodes. Recall that T is called *height-balanced* if its height is $O(\log n)$. For any node v of T, let $|v|$ denote the size (number of nodes) of the subtree rooted at v (so $|T| = n$, for example). The tree T is called *count-balanced* if at every node v of T, the left and right subtrees satisfy $|\text{left}(v)| \geqslant \lfloor \frac{1}{3}|v| \rfloor$ and $|\text{right}(v)| \geqslant \lfloor \frac{1}{3}|v| \rfloor$. Prove/Disprove the following assertions.

(a) If T is count-balanced, then T must be height-balanced. **(4)**

Solution *True.* The maximum possible height $H(n)$ of a count-balanced tree on n nodes satisfies

$$H(n) \approx 1 + H\left(\frac{2}{3}n\right) \approx 2 + H\left(\frac{2^2}{3^2}n\right) \approx 3 + H\left(\frac{2^3}{3^3}n\right) \approx \cdots \approx k + H\left(\frac{2^k}{3^k}n\right).$$

If we take $k = \left\lfloor \log_{3/2} n \right\rfloor$, we have $H(n) \approx k = O(\log n)$.

(b) If T is height-balanced, then T must be count-balanced. **(4)**

Solution *False.* Construct a BST T on n nodes as follows. The left subtree of T is empty, whereas its right subtree is the complete binary tree of $n-1$ nodes. Then, the height of the left subtree is $\lceil \log_2 n \rceil - 1$, and so the height of T is $\lceil \log_2 n \rceil$. For $n \geqslant 3$, we have $\lfloor \frac{1}{3}n \rfloor \geqslant 1$, that is, the empty left subtree of T implies a violation of the count-balancing condition at the root.

Note: Your construction should be arbitrarily scalable. If you give an example of an eight-node tree of height five, it may fail to illustrate whether $h = O(\log n)$. For the constant n in your example, the question is rephrased as whether $5 = O(3)$ (which is technically true), but at the same time $5 = O(8)$, so $h = O(n)$ is also true.

3. A complex number $z = x + \mathrm{i}y$ is stored as a pair (x, y) of real numbers. The magnitude of z is $|z| = \sqrt{x^2 + y^2}$. You are given an array A of n complex numbers $z = x + \mathrm{i}y$ with each x, y integers in the range $[-n, +n]$. Propose a worst-case $O(n)$-time algorithm to sort A (in the ascending order) with respect to the magnitudes of the elements. **(6)**

Solution Notice that sorting with respect to the magnitude is the same as sorting with respect to the square of the magnitude, which in this case is always an integer in the range $[0, 2n^2]$. Let $R = \left\lceil \sqrt{2}\,n \right\rceil$. Express each magnitude square in radix R with two R-ary digits each in the range $[0, R-1]$. Now, apply radix sort on A. This involves two counting sort instances each running in $O(n+R) = O(n)$ time.

FOR ROUGH WORK