CS21003 - Tutorial 1

Solution Sketch

1. Put the following functions in order from lowest to highest in terms of their Θ classes. (Some of the functions may be in the same Θ class. Indicate that on your list also.)
   (a) $f_1(n) = nlogn$
   (b) $f_2(n) = n^{3/2}$
   (c) $f_3(n) = 1000$
   (d) $f_4(n) = \sqrt{n}(n + logn)$
   (e) $f_5(n) = 3^n$
   (f) $f_6(n) = 2^{n+2}$
   (g) $f_7(n) = 0.00001$

   $\Rightarrow \{f_7, f_3\} < f_1 < \{f_2, f_4\} < f_6 < f_5$

2. Give an example of two positive real valued functions $f(n)$ and $g(n)$ of natural numbers that satisfy the property that $f(n)$ is not $O(g(n))$ and $g(n)$ is also not $O(f(n))$.

   $\Rightarrow$ $f(n) = n^2$ for odd n and $= 1$ for even n, $g(n) = n$

3. Assume that each node in a binary tree T contains only a positive integer value and two child pointers (left and right). No parent pointers or additional values can be stored in the nodes. Let r be the root of the tree, and v any node in the tree T. The weight of v is defined as the sum of all the values stored on the unique $r - v$ path. Your task is to locate the maximum of the weights of all the nodes in T in $O(n)$ time. Assume that all the values are positive.

   $\Rightarrow$ The following is a pseudo code for the same.

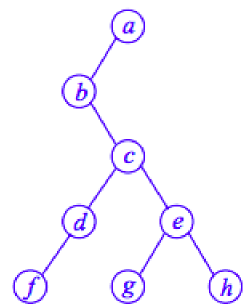
```
maxweight (T, wt)

if (T == NULL) return wt
wt += T -> val /* Add the value stored at the current node */
lwt = maxweight(T->L,wt) /* Recurse on the left subtree */
rwt = maxweight(T->R,wt) /* Recurse on the right subtree */
/* Take the larger of the weights returned by the two recursive calls */
wt = max(lwt,rwt)
return wt
```

   Outermost call $maxweight(T, 0)$

4. The vertex set of a binary tree T on eight nodes is $\{a, b, c, d, e, f, g, h\}$. The inorder listing of the vertices of T is *bfdcgeha*, and the postorder listing is *fdghecba*. Reconstruct the tree T. Explain the relevant steps.

   $\Rightarrow$ The last element in the postorder listing is the root, so T has root a. Since nothing follows a in the inorder listing, the right subtree of a is empty. We need to construct the left subtree of T from the inorder listing



bfdcgeh and postorder listing fdghecb. The root is b. Since nothing precedes b in the inorder listing, the left subtree of b is empty. So we recursively construct the right subtree of b from the inorder and postorder listings fdcgeh and fdghec. The final tree is shown below.

5. How many distinct binary search trees can be created out of 4 distinct keys? [**Note:** Try solving it for the general case of n keys and use that to find the solution for $n = 4$.]

   ⇒ Consider all possible binary search trees with each element at the root. If there are n nodes, then for each choice of root node, there are n-1 non-root nodes and these non-root nodes must be partitioned into those that are less than a chosen root and those that are greater than the chosen root. Lets say node i is chosen to be the root. Then there are i-1 nodes smaller than i and n-i nodes bigger than i.

$$t(n) = \sum_{i=1}^{n} t(i-1)t(n-i)$$

   Solving, this gives $t(4) = 14$

6. **Prove or disprove:** You are given a sequence of integers $a_1, a_2, \ldots, a_n$ in an array. This can lead to a BST having the maximum possible height only if the integers are in sorted order.

   ⇒ Disprove using counterexample – take the sequence 2, 8, 6 and insert in BST in that order. It has a height of 2 (maximum) but the sequence is not sorted.

7. You are given a sequence of integers $a_1, a_2, \ldots, a_n$ in an array. You need to decide whether inserting these integers in that sequence leads to a height of $n-1$ of the binary search tree. Propose a worst-case $O(n)$-time algorithm to solve the problem. Note that if you actually build the tree, you end up in a $\Theta(n^2)$ running time in the worst case.

   ⇒ In order that we get a chain of length $n-1$, we need each non-leaf node to have only one child. This limits the allowed values of a_i given that $a_1, a_2, \ldots, a_{i-1}$ have already resulted in a binary search tree of height $i-2$. For example, if $a_2 < a_1$, then a_2 is inserted as the left child of the root a_1. But then, the root cannot have a right child, that is, none of $a_3, a_4, \ldots, a_n$ can be larger than a_1. The following pseudocode implements these observations.

   - 1. If $(n \leq 2)$, return true.
   - 2. Initialize lower limit $= -\infty$ and upper limit $= +\infty$.
   - 3. For $i = 2, 3, 4, \ldots, n$, repeat:
   - 4. If $(a_i <$ lower-limit$)$ or $(a_i >$ upper limit$)$, return false.
   - 5. If $(a_i < a_{i-1})$, set upper limit $= a_{i-1}$.
   - 6. If $(a_i > a_{i-1})$, set lower limit $= a_{i-1}$.
   - 7.

- 8. Return true.

8. Let us define a *relaxed red-black tree* as a binary search tree that satisfies red-black properties 1, 3, 4 and 5. Thus, the root can be either *red* or *black*. Consider a relaxed red-black tree T whose root is red. If we color the root of T black but make no other changes to T, is the resulting tree a *red-black* tree?

   $\Rightarrow$ Yes, the resulting tree will be a red-black tree. There will be no changes even in the black height.