CS2210a: Data Structures and Algorithms
Winter 2012: Midterm Examination Solution
110 minutes

Instructions:

- Write your name in the space provided.
- Please check that your exam is complete. It should have 12 pages in total
- The examination has a total of 100 marks.
- This is an open textbook/lecture notes exam. You can consult your notes/textbook but you cannot talk/communicate with other students. No assignment/sample midterm solutions are allowed.
- When you are done, raise your hand and we will pick up your exam.

Name:_____

Part 1	
9	
10	
11	
12	
13	
14	
15	
TOTAL	

PART1: Short Answers

Instructions: *Show all the work you do. Each question is worth 4 marks.*

1. Give the best asymptotic ("big-O") characterization of function

$$f(n) = n^2\ log(n^4) + 10^6\ n^{1.5} + 0.01\ n^3$$

   Solution: $O(n^3)$. previous typo correction.

2. Let $f(n) = 15n^3 + 6n^2$ and $g(n) = 15n^3 + 5n^2$. Write down the best asymptotic ("big-O") characterization of the following functions:

   - $f(n) + g(n)$
   - $f(n) - g(n)$
   - $f(n) \cdot g(n)$

   Solution:

   - $f(n) + g(n) = 30n^3 + 11n^2$is $O(n^3)$
   - $f(n) - g(n) = n^2$ is $O(n^2)$
   - $f(n) \cdot g(n) = 15^2n^6$+ lower order terms is $O(n^6)$

3. Give the best asymptotic ("big-Oh") characterization of the worst case *and* the best case time complexities of the algorithm $Add(A, n)$.

Algorithm *Add(A,n)*
Input: Array *A* of size n. *A* stores integer values.
    $sum \leftarrow 0$
    if $A[0] < 0$ **then**
        for $c \leftarrow 0$ to $n - 1$ **do**
            $sum \leftarrow sum + A[c]$
            for $k \leftarrow c$ **to** $c + 1000$ **do**
                $sum \leftarrow sum + k \cdot A[c]$

   Solution: Best case is $O(1)$, happens when $A[0] < 0$, in which case the **for** loop is never executed. Worst case is when both **for** loops are executed, but the second nested one is always run for 1000 times, independent of the value of n. So the worst case is $O(n)$.

4. Suppose we have a hash table with N buckets, currently containing n entries. Suppose that instead of a linked list, each bucket is implemented as a binary search tree. Give the best asymptotic ("big-Oh") characterization of the worst and best case time complexity of adding an entry to this hash table.

   Solution: Best case happens when we insert into an empty bucket, and this is $O(1)$ time. Worst case is when we insert in a bucket that has all n entries. Worst case insertion into a BST is $O(n)$ so worst case is $O(n)$.

5. Let T be a full binary tree with 111 nodes. How many external nodes does T have?

   Solution: For a full binary tree we have property: $i = e - 1$, where i is number of internal nodes and e is number of external nodes. The total number of nodes is $111 = e + i$, therefore we have $111 = 2e - 1$, and $e = 56$.

6. Let T be a full binary tree with root r where each node stores an integer key. Consider the algorithm below. What does the algorithm return when called on the root r? Here $v.left$ and $v.right$ give the left and the right children of v, respectively, and $v.key$ is the key stored at node v.

```
Algorithm TreeAlg(v)
    if v is external
        return 2 * v.key
    else
        return TreeAlg(v.left)+TreeAlg(v.right) − v.key
```

   Solution: The algorithm returns twice the sum of external keys minus the sum of internal keys.

7. Let H be a heap storing more than 3 entries. Assume it stores unique keys, i.e. no repeated keys allowed. As usual, smaller key corresponds to higher priority. Mark the following statements as true or false.

   - Preorder traversal always lists keys in increasing order
   - Preorder traversal sometimes lists keys in increasing order
   - Inorder traversal always lists keys in increasing order
   - Inorder traversal sometimes lists keys in increasing order

   Solution: F, T, F, F.

8. Let H be a heap storing 30 unique integer keys. List all the levels of the heap H that can have the 7th smallest key. Assume that smaller keys have higher priorities.

   Solution: Since at level 0 there is 1 entry, level 1 there are 2 entries, level 2 there are 4 entries, level 3 there are 8 entries, level 4 there are the rest, that is 15 entries, the height of this heap is 4. The 7th smallest key can be anywhere except level 0. Therefore it can be at levels 1,2,3,4.

9. Let T be an AVL tree of height 6. What is the smallest number of entries it can store? Remember that the leafs do not store any entries.

   Solution: Let $n(h)$ be the smallest number of entries for an AVL tree of height h. In class we have shown that $n(1) = 1$, $n(2) = 2$, and $n(h) = 1+n(h-1)+n(h-2)$. Therefore, we get:

   $n(3) = 1 + 2 + 1 = 4$

   $n(4) = 1 + 2 + 4 = 7$

   $n(5) = 1 + 4 + 7 = 12$

   $n(6) = 1 + 7 + 12 = 20$

Part 2: Written Answers

Instructions: *Write your answers directly in these sheets. Show all the work. Use the back of the sheets if necessary.*

10. [8 marks] Prove that $f(n) = 7^5 + 5n^5\ log(n) + 3n^5$ is $O(n^5 \log(n))$ using the definition of "big-Oh".

    Solution:

    $7^5+5n^5\ log(n)+3n^5 \leq 7^5 n^5 \log(n)+5n^5 \log(n)+3n^5 \log(n) = (7^5+5+3)n^5 \log(n)$ for all $n \geq 2$.

    Take $c = 7^5 + 8$ and $n_0 = 2$. We have that $f(n) \leq cn^5 \log(n)$ for all $n_0 \geq 2$, which means that $f(n)$ is $O(n^5 \log(n))$ by the definition of "big-Oh".

11. [12 marks] Consider a hash table of size 7 storing entries with integer keys. Suppose the hash function is $h(k) = k \bmod 7$. Insert, in the given order, entries with keys 2,4,12,19,20 into the hash table using:

(a) [5 marks] Linear probing to resolve collisions. Show all the work.

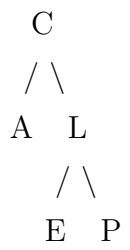
0	1	2	3	4	5	6
20		2		4	12	19

(b) [7 marks] Double hashing to resolve collisions with secondary hash function $h'(k) = 5 - (k \bmod 5)$. Show all the work.

0	1	2	3	4	5	6
20		2		4	12	19

12. [10 marks].
(a) [6 marks] Draw a Binary Search Tree tree containing keys P, L, A, C, E, such that the preorder traversal visits nodes in this order: C, A, L, E, P.

Solution:

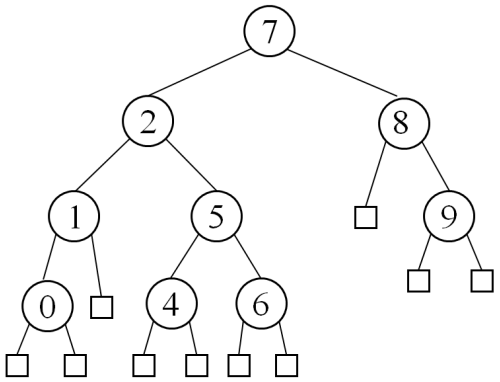


(b) [4 marks] Give the array representation of the tree you got in part (a).

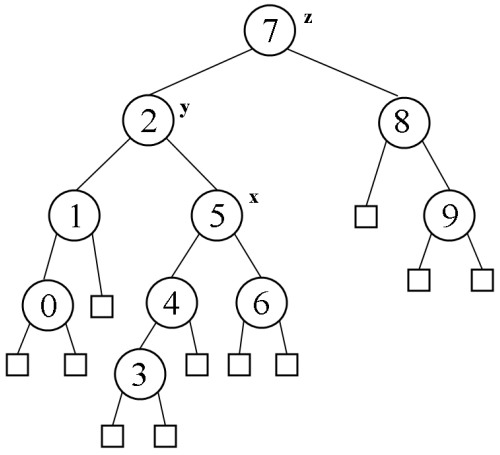
Solution:

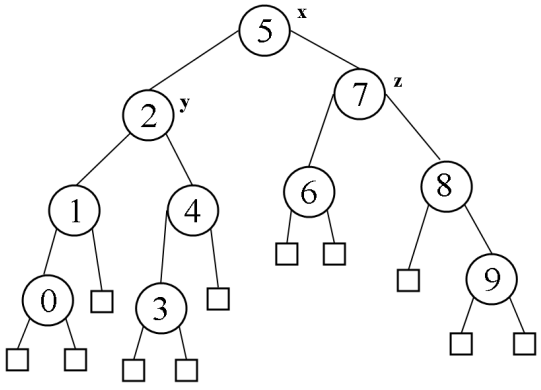
0	1	2	3	4	5	6	7
	C	A	L			E	P

13. [8 marks] Consider the following AVL tree. Perform operation insert(3) and rebalance the tree if necessary. *You have to use exactly the same algorithm as we studied in class.* Show all the intermediate trees.



Solution:





14. [13 marks] Let A be an array of size n storing positive integers.

(a) [8 marks] Write, in pseudocode, an algorithm that takes as input array A storing positive integers and its size n, and replaces each unique element of A by -1. For example, if $A = \{5, 4, 7, 5, 9, 6, 6\}$ then after you run the algorithm, $A = \{5, -1, -1, 5, -1, 6, 6\}$. You are allowed to use only O(1) of additional memory. This means that is you can use a few variables, but you cannot declare arrays or any other data structures whose size depends on n.

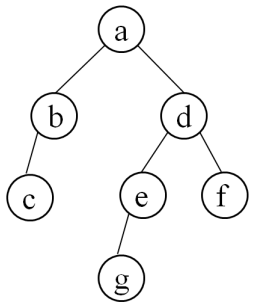
```
Algorithm Unique(A,n)
    for i = 0 to n-1                          n
        temp = A[i]                           n
        unique = true                         n
        for j = 0 to n-1                      n^2
            if A[j] == temp and i ≠ j         n^2
                unique = false                n^2
        if ( unique ) A[i] = -1               n
```

(b) [5 marks] Give the best "big-Oh" characterization of the worst case time complexity of your algorithm above. Express the complexity as a function of the array size n. Explain how you computed the time complexity of your algorithm.

Solution: Adding up all primitive operations listed in the pseudo-code above we get $O(n^2)$.

15. [13 marks]

(a) [9 marks] Let T be a binary tree. Write an algorithm that receives as input a node n and an integer l and outputs the number of nodes that are at level l of the tree rooted at n. For example, in the picture below, if the inputs are node a and $l = 1$, then the algorithm should return 2, and if the input is node a and $l = 2$, the algorithm should return 3. If there are no nodes on level l, then the algorithm should return 0. For example, for the tree below, if the input is node a and $l = 4$, the algorithm should return 0. For node v, use $v.left$ and $v.right$ to denote the left child and the right child of v, respectively, and $v.parent$ to denote the parent of v. You cannot assume that each node knows on which level of the tree it is located, but you can store an additional variable at each node, namely $v.var$. Hint: use $v.var$ for computing and storing the levels.



Solution:

My solution uses an extra variable at each node, but there is a version without using an extra variable at each node.

```
Algorithm NodesAtLevel(v,l)
    if v.parent == null
        v.var = 0 // l is the root note in this case, level is 0
    else
        v.var = v.parent.var + 1 //  sets the correct level for v
    if v.var == l
        return(1)
    else
        s = 0
        if v.left ≠ null
            s = s+NodesAtLevel(v.left,l)
        if v.right ≠ null
            s = s+NodesAtLevel(v.right,l)
        return(s)
```

(b) [4 marks] Give the best "big-Oh" characterization of the time complexity of your algorithm above. Express the complexity as a function n, the number of nodes. Explain how you computed the time complexity of the algorithm.

Solution: The algorithm performs pre-order traversal, and so it visits every node exactly once (in the worst case). At each visitation, it performs a constant number of operations. So the time complexity is constant multiplied by the number of nodes n, which is $O(n)$.