Computer Science Foundation Exam

May 2, 2014

Section I A

COMPUTER SCIENCE

**NO books, notes, or calculators may be used,
and you must work entirely on your own.**

SOLUTION

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	10	ANL	7	
3	10	ALG	7	
4	10	ALG	7	
5	10	ALG	7	
TOTAL	50			

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and not graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all be neat.

1) (10 pts) DSN (Binary Trees)

Write a function, `int isMaxHeight(treenode *root)`, that takes the root of a binary tree and determines whether or not the tree has the maximum height for the number of nodes it stores. If the tree is of maximum possible height, return 1, otherwise, return 0. You may write any helper functions as you deem necessary. The node struct definition is:

```
typedef struct treenode {
    int data;
    struct treenode *left;
    struct treenode *right;
} treenode;

int isMaxHeight(treenode *root) {
```

Solution #1. Key observation: If any node has more than one child, the nodes can be rearranged into a taller binary tree.

```
int isMaxHeight(node *root) {
    if (root == NULL)                    // 2 pts
        return 1;
    else if (root->left != NULL && root->right != NULL) // 4 pts
        return 0;
    else if (root->left != NULL)                    // 2 pts
        return isMaxHeight(root->left);
    else
        return isMaxHeight(root->right);          // 2 pts
}
```

Solution #2. Key observation: If the tree has n nodes and height n-1, the nodes cannot be rearranged into a taller binary tree.

```
int isMaxHeight(node *root) {
    if (height(root) == countNodes(root) - 1)     // 1 pt
        return 1;                                 // 1 pt
    return 0;
}

int countNodes(node *root) {              // 4 pts
    if (root == NULL) return 0;
    return 1 + countNodes(root->left) + countNodes(root->right);
}
int height(node *root) {                  // 4 pts
    if (root == NULL) return -1;
    return 1 + max(height(root->left), height(root->right));
}
int max(int a, int b) {
    return (a > b) ? a : b;
}
```

2) (10 pts) ANL (Recurrence Relations)

(a) (9 pts) Write a recurrence relation that represents the runtime of the following function, then solve it (i.e., derive its closed form) using iterative substitution:

```
int foo(int n)
{
   if (n == 0 || n == 1)
      return 18;

   else
      return foo(n-2) + foo(n-2);
}
```

Recurrence Relation:

T(0) = T(1) = c (some constant)
T(n) = 2 * T(n-2) for n > 1, 3 pts

Iterative Substitution:

T(n) = 2*T(n-2)
= 2*2*T(n-4), since T(n-2) = 2*T((n-2)-2), 1 pt
= 2*2*2*T(n-6), since T(n-4) = 2*T((n-4)-2), 1 pt
= 2*2*2*2*T(n-8), since T(n-6) = 2*T((n-6)-2) = 2*T(n-8)

= 2^kT(n-(2*k)), after k iterations. 2 pts

Let (n – 2k) = 0. Then n = 2k, which implies k = n/2. (1 pt) This yields:

= $2^{n/2} \times T(0)$, since k = n/2
= $c2^{n/2}$, since T(0) = c
= $c(\sqrt{2}^n)$, 1 pt

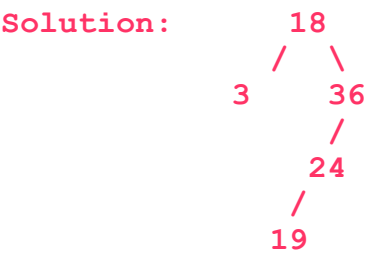
(b) (1 pt) Express the solution from part (a) using theta notation: $O(\sqrt{2}^n)$ (1 pt)

3) (10 pts) ALG (Binary Search Trees)

(a) (4 pts) Draw a binary search tree that has the following post-order traversal:

3 19 24 36 18

Solution:



Grading: 4 pts correct, 2 pts correct root incorrect tree, 0 pts otherwise

(b) (3 pts) Following are three traversals produced by the exact same binary search tree. Using your powers of inference, determine which one is which. (Fill in each of the blanks with "in-order," "pre-order," or "post-order.")

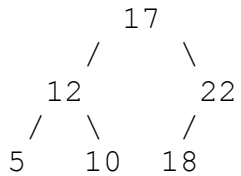
pre-order traversal: 6 3 2 -7 4 10 25 23 11 18

post-order traversal: -7 2 4 3 18 11 23 25 10 6

inorder traversal: -7 2 3 4 6 10 11 18 23 25

The sorted traversal must be the in-order traversal. In pre-order, the root is always printed first. In post-order, the root is always printed last. Using that knowledge, we can infer that 6 must be the root; it's the only number that appears in both the first and last position among those traversals. (1 pt each)

(c) (3 pts) Consider the data structure below, then answer the following questions:



Is it a binary tree? **yes** (yes / no)

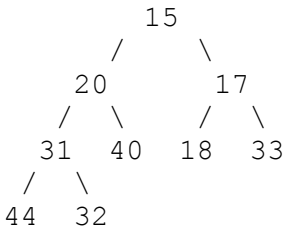
Is it a binary search tree? **no** (yes / no)

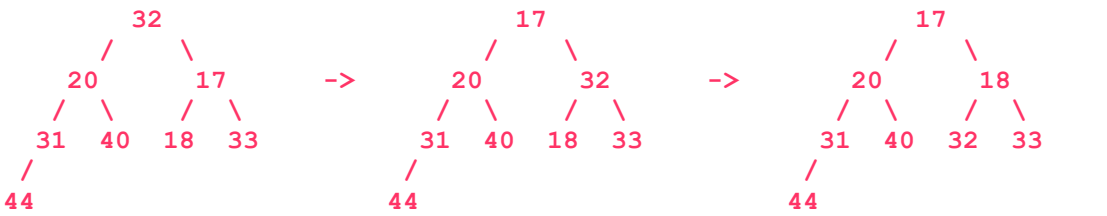
Is it a minheap? **no** (yes / no)

Grading: 1 pt each

4) (10 pts) ALG (Heaps)

(a) (3 pts) Show what happens when 15 is deleted from the following minheap. For full credit, clearly show each step of the process:





Last node moves up to root position. 1 pt

32 percolates down: swaps with min(20,17). 1 pt

32 percolates down: Swap with min(18,33) 1 pt

(b) (5 pts) Using theta notation, give the best- and worst-case runtimes for the following minheap operations (assuming we use a standard array implementation for the heap):

Insertion (best-case): **O(1)**

Insertion (worst-case): **O(log n)**

Deletion (best-case): **O(1)**

Deletion (worst-case): **O(log n)**

Finding the minimum element in a minheap (best- and worst-case): **O(1)**

Grading = 1 pt each

(c) (2 pts) Describe what situation leads to the **worst-case** runtime for insertion into a minheap.

Inserting a key that is less than the current min will cause worst-case insertion, because it has to percolate all the way up to the root position. (2 pts)

5) (10 pts) ALG (Base Conversion)

(a) (5 pts) Convert 2772 from decimal (base 10) to octal (base 8).

```
2772 / 8 = 346 R 4
346 / 8 = 43 R 2
43 / 8 = 5 R 3
5 / 8 = 0 R 5

2772 (base 10) = 5324 (base 8), 5 pts, 1 pt per step.
```

(b) (5 pts) Convert 12012 from base 3 to base 10.

$$
\begin{aligned}
12012\ (\text{base } 3) &= (1*3^4)+(2*3^3)+(0*3^2)+(1*3^1)+(2*3^0) \\
&= (1*81)+(2*27)+(0*9)+(1*3)+(2*1) \\
&= 81 + 54 + 3 + 2 \\
&= 140
\end{aligned}
$$

Grading = 5 pts, 1 pt per term added, 1 pt off for adding error