Computer Science Foundation Exam

May 4, 2012

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 (Recursion)

Write a **recursive** function to sum every other node in a linked list pointed to by head, which is a parameter to the function. Specifically, make sure you sum the first, third, fifth, etc. nodes and return the total sum. If the list has zero items in it, the sum should be 0. Your function should make use of the following `struct node` and function prototype:

```
struct node {
    int data;
    struct node *next;
};

int sumEveryOther(struct node *head) {

     if (head == NULL)                                  // 2 points
          return 0;

     if (head->next == NULL)                            // 2 points
          return head->data;

     return head->data +                                // 3 points
            semEveryOther(head->next->next);            // 3 points

}
```

2) (10 pts) ANL (Summations)

```
for (k = 0; k < 2n; k++) {
     for (j = 0; j < k; j++) {
          x++;
     }
}
```

(a) (5 pts) Determine a summation representing the number of times the statement: **`x++;`** is run in the above code segment.

$$\sum_{k=0}^{2n-1} \sum_{j=0}^{k-1} \mathbf{1}$$

OR **5 points**

$$\sum_{k=0}^{2n-1} \boldsymbol{k}$$

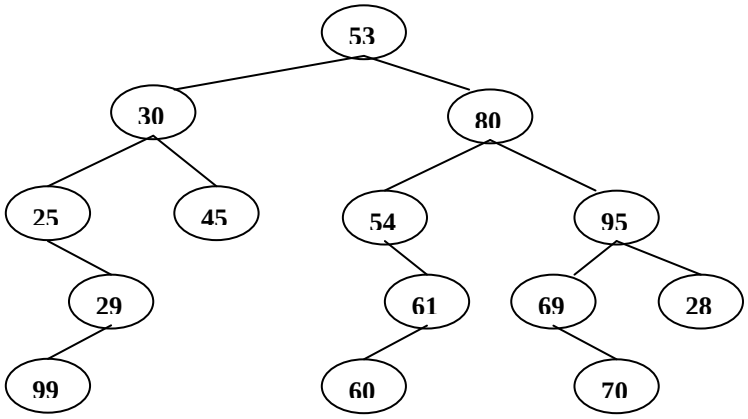
(b) (5 pts) Determine a **simplified**, closed-form solution for the summation from (a) in terms of n. Also provide the Big-O runtime of the above segment of code. **You MUST show your work.**

$\sum_{k=0}^{2n-1} \sum_{j=0}^{k-1} \mathbf{1} = \sum_{k=0}^{2n-1} \boldsymbol{k}$ **1 point**

$= \frac{2n(2n-1)}{2} = \boldsymbol{n(2n-1)}$ **2 points**

$= \boldsymbol{O(n^2)}$ **2 points**

3) (10 pts) **Stacks, Queues and Trees.**



(a) (2 pts) Is the tree above a valid Binary Search Tree? (circle one)

YES **NO** **// 2 points**

(b) (8 pts) Let `Q` be a queue and `S` be a stack, both initialized to be empty. Execute the algorithm shown below using the tree shown above. **Show the contents of both Q and S** when the initial call, **P3(root, Q, S)**, completes execution. Assume that the tree nodes and pointers are defined as shown and that attempting to remove an item from an empty stack or queue has no effect.

```
struct tree_node {
    int data;
    struct tree_node *left;
    struct tree_node *right;
}
void P3(struct tree_node *root, queue *Q, stack *S) {
     if (root != NULL){
          if (root->data %2 == 0){
               enqueue(Q, root->data);
               pop(S);
          }
          else {
               push(S, root->data);
               dequeue(Q);
          }
          P3(root->left, Q, S);
          P3(root->right, Q, S);
    }
}
```

// 2 points for each number in the correct spot.
// 1 point for any number in the incorrect spot.

front

Contents of Q:	**70**	**28**							

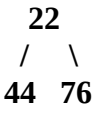
bottom

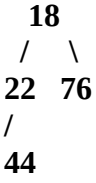
Contents of S:	**25**	**29**							

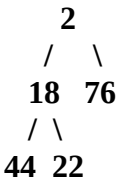
4) (10 pts) ALG (Heaps)

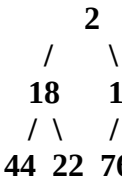
Consider inserting the following items in the order presented into an initially empty minimum heap: 22, 44, 76, 18, 2, 13, 55, 23. (The root node of a minimum heap stores the smallest item.) Draw the state of the heap after each insertion is completed.

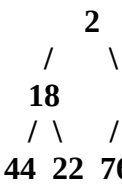
22 **1 point for each correct tree, except the final tree is worth 3 points**

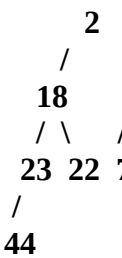






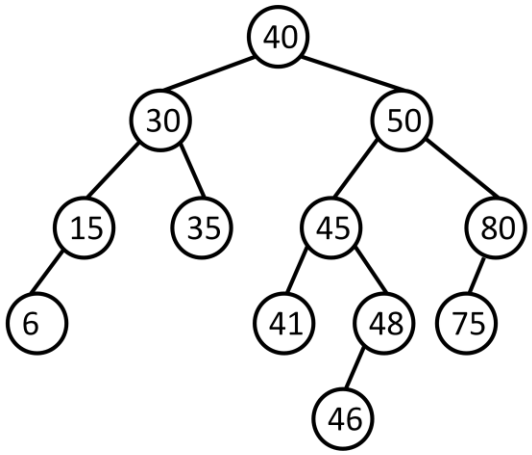






5) (10 pts) ALG (Binary Trees)

Write a function that would print the following traversal for the tree below:
80, 75, 50, 48, 46, 45, 41, 40, 35, 30, 15, 16
Your function should make use of the following `struct tree_node` and function prototype below.




```
struct tree_node {
     int data;
     struct tree_node *left;
     struct tree_node *right;
};

void backwards(struct tree_node *root) {

    if (root != NULL) {                  // 1 point

        backwards(root->right);          // 3 points
        printf("%d, ", root->data);      // 3 points
        backwards(root->left);           // 3 points

        // If any out of order, - 4 points
    }



}
```