# Computer Science Foundation Exam

## December 16, 2011

## Section I A

## COMPUTER SCIENCE

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

**Name:**____________________________________________

**PID:** ____________________________________________

| 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 <u>not</u> 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 <u>be neat</u>.**

**1)** (10 pts) **Recursion.** Write a **recursive** function that deletes every other node in a linked list pointed to by head, which is a parameter to the function. Specifically, make sure you delete the second, fourth, sixth, etc. nodes and return a pointer to the front of the new list. If the list has zero or one item in it, the list should be unchanged and a pointer to its front should be returned. Your function should make use of the following `struct node` and function prototype:

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

struct node* delEveryOther(struct node *head) {

     // Base case
     // Grading:  1 point
     if (head == NULL || head->next == NULL)
          return NULL;

     // Assign a temp pointer to the node to be deleted
     struct node* temp = head->next;    // Grading:  1 points

     // Bypass the node to be deleted.
     head->next = temp->next;           // Grading:  2 points
     // Can also do:  head->next = head->next->next;

     // Free the temp node
     free(temp);                        // Grading:  1 points

     // Recursively call the function on the rest of the list
     head->next = delEveryOther(head->next); // Grading 4 points
     // Can also do:  delEveryOther(head->next);

     // Return the original front of the lis
     return head;                       // Grading 1 point

}
```

**2)** (10 pts) **Summations.** Determine a **simplified**, closed-form solution for the following summation in terms of n. **You MUST show your work.**

$$\sum_{j=n}^{2n}\left(2\sum_{i=1}^{n}2\,ij\right)$$

Grading: 4 points for inner summation:

$$=\sum_{j=n}^{2n}\left(4j\sum_{i=1}^{n}i\right)=\sum_{j=n}^{2n}(2j(n)(n+1)=$$

Grading: 2 points for changing limits of outer summation:

$$=2n(n+1)\sum_{j=n}^{2n}j\;=2n(n+1)(\sum_{j=1}^{2n}j-\sum_{j=1}^{n-1}j\,)=$$

Grading: 2 points for applying sum formula:

$$=2n(n+1)(\frac{2n(2n+1)}{2}-\frac{n(n-1)}{2})$$

Grading: 2 points for simplifying:

$$=n(n+1)((4n^2\;+\;2n)-\;(n^2-n))$$

$$=n(n+1)(3n^2\;+\;3n)$$

$$=\;3n^2(n+1)$$

$$=\;3n^4+6n^3+3n^2$$

**3)** (10 pts) **Stacks and Queues.** Let `Q` be a queue and `S` be a stack. The functions `dequeue` and `pop` obey the convention that they return whatever they remove. Assume that `Q` and `S` are initially empty and that `print` is a function that prints the value of its argument. Execute, in top-to-bottom order, the operations below and answer the following questions.

```
push(S, 'O');
enqueue(Q, '-');
push(S, 'G');
enqueue(Q, 'F');
print(pop(S));
enqueue(Q, 'C');
print(pop(S));
print(dequeue(Q));
enqueue(Q, 'U');
push(S, 'G');
push(S, 'H');
push(S, dequeue(Q));
enqueue(Q, 'K');
push(S, dequeue(Q));
enqueue(Q, 'N');
push(S, dequeue(Q));
print(pop(S));
enqueue(Q, 'I');
print(pop(S));
enqueue(Q, 'T');
print(pop(S));
```

**a)** Show the output from the print statements:

Grading: (6pts) 1 point per letter:

| G | O | - | U | C | F |
|---|---|---|---|---|---|
| first output | second output | third output | fourth output | fifth output | sixth output |

**b)** After the above operations are completed, how many items are left in stack S?

Grading: 2 points

2

**c)** After the above operations are completed, how many items are left in queue Q?

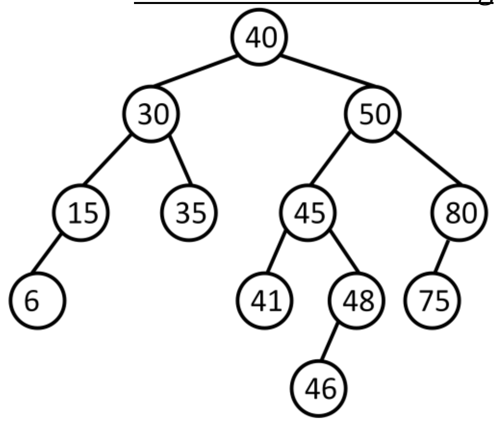
Grading: 2 points

4

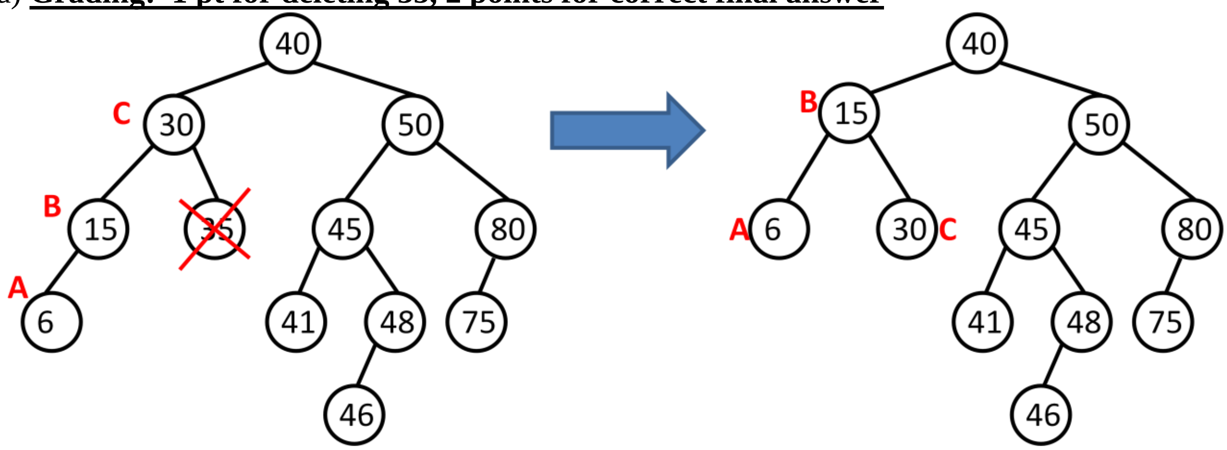
**4)** (10 pts) **AVL Trees.** The tree shown below is a valid AVL tree. You must **delete** the node that has 35 as a data value and rebalance the AVL tree as needed, which will require two restructures.

(a) Show the state of the AVL tree after the first rebalancing.

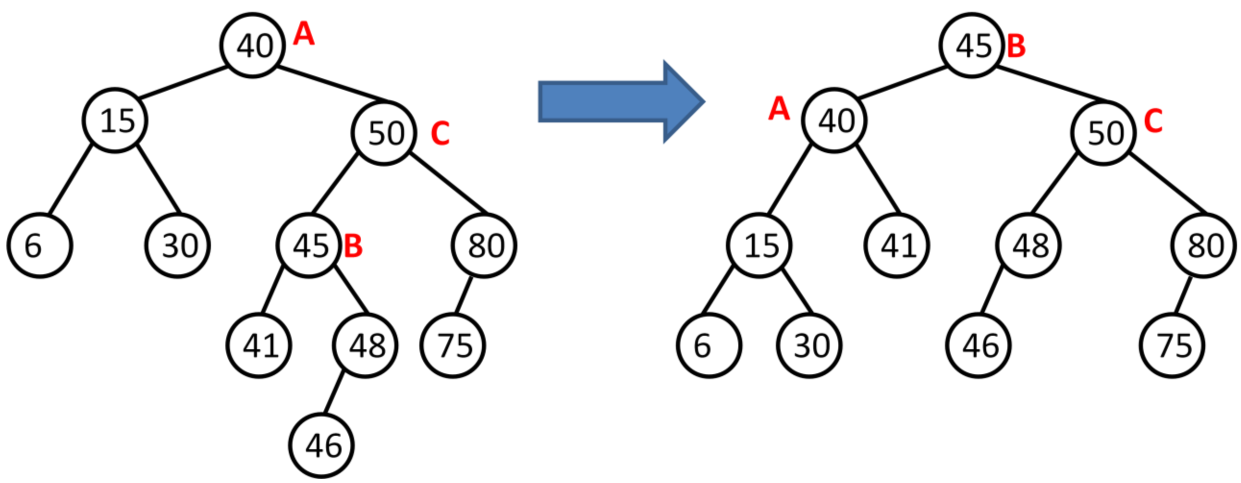
(b) Show the state of the AVL tree after the second rebalancing.




(a) **Grading: 1 pt for deleting 35, 2 points for correct final answer**




(b) Grading**: 1 pt for 45 root, 1 pt for 40, 1 pt for 50, 1 pt for each sub-tree correct.**



**5)** (10 pts) **Binary Trees**

**a)** Write a function that frees the memory for each node in a Binary Tree. In particular, the memory should be **freed in the right subtree first, then the left subtree, and finally the root will be freed.** Your function should make use of the following `struct tree_node` and function prototype:

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

void freeBinTree(struct tree_node *head) {

     if (head != NULL)                    // Grading:  1 pt
     {
          freeBinTree(head->right);       // Grading:  2 pt
          freeBinTree(head->left);        // Grading:  2 pt
          free(head);                     // Grading:  1 pt
     }

}
```

**b)** List the order that the nodes would be freed if the above function is executed on the following tree:

**Grading: (4pts) 1 pt per mistake**

**37, 92, 21, 62, 77, 56, 24, 12, 48**