ECE368 Exam 1
Spring 2012

Wednesday, February 22, 2012
6:30-7:30pm
EE 170

READ THIS BEFORE YOU BEGIN

This is an *open-book, open-notes* exam. Electronic devices are not allowed. The time allotted for this exam is *exactly* 60 minutes. *It is in your best interest to not spend too much time on any problem*.

Always show as much of your work as practical–partial credit is largely a function of the *clarity and quality* of the work shown. Be *concise*. It is fine to use the blank page opposite each question (or at the back of each question) for your work. Do draw an arrow to indicate that if you do so.

This exam consists of 8 pages; please check to make sure that *all* of these pages are present *before* you begin. Credit will not be awarded for pages that are missing – it is *your responsibility* to make sure that you have a complete copy of the exam.

IMPORTANT: Write your login at the TOP of EACH page. Also, be sure to *read and sign* the *Academic Honesty Statement* that follows:

"In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action."

`Printed Name:`

`login:`

`Signature:`

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO ...

1. (15 points) Consider the following procedure that performs multiplication of two lower triangular matrices $A[1..n][1..n]$ and $B[1..n][1..n]$.

```
MATRIX_MULTIPLY(A[1..n][1..n], B[1..n][1..n])
1.      for i ← 1 to n {
2.          for j ← 1 to i {
3.              c_ij ← 0
4.              for k ← j to i {
5.                  c_ij ← c_ij + a_ik · b_kj
6.              }
7.          }
8.      }
9.      return C
```

As the product $C[1..n][1..n]$ is also lower triangular, we assume that we do not have to be concerned with the strictly upper triangular entries of the matrix.

For each instruction, write down the expression for the number of times the instruction is executed in terms of i, j, k, and n (you may not need all of these terms). Do not attempt to evaluate/expand your expressions.

Line 1:

Line 2:

Line 3:

Line 4:

Line 5:

Line 9:

2. (15 points) Consider the following Shell Sort algorithm for the sorting of an array $r[0..n-1]$ of n integers:

```
1.      for each k in {7,3,1} in descending order {
2.          for j ← k to n−1 {
3.              temp_r ← r[j];
4.              i ← j;
5.              while i ≥ k and r[i−k] > temp_r {
6.                  r[i] ← r[i−k];
7.                  i ← i−k;
8.              }
9.              r[i] ← temp_r;
10.         }
11.     }
```

(a) (8 points) Consider the array $r = [4,9,8,2,3,7,9,0,4,2,0,6,1,5,7,3,5,1,6,8]$. Show the array r right after the iteration of $k = 7$ is completed.

After the iteration of $k = 7$:

(b) (7 points) Show also the array r when $k = 3$ and $j = 15$ (right before j becomes 16). Of course, you have already performed the sorting for $k = 7$ in **(a)**.

After the inner iteration of $j = 15$ (and when k = 3 for the outer iteration):

3. (30 points) Consider an array-based implementation of singly linked list(s). Assume that each node in the linked list has the following structure:

```
typedef struct _node {
  int info;
  int next;
} node;
```

The contents of the array implementing the linked list(s) are as follows:

index	info	next
9	109	-1
8	108	0
7	107	9
6	106	-1
5	105	4
4	104	-1
3	103	7
2	102	3
1	101	2
0	100	6

Among the linked lists in the array, one maintains a list of unused nodes. We refer to that list as `Unused_List`, which at the moment starts at index 1. Whenever other functions require a new node to be inserted into a linked list (other than `Unused_List`), the node is obtained from `Unused_List`. Whenever a node is deleted from a linked list and is no longer required, it is returned to `Unused_List`.

(a) (4 points) Draw the linked list `Unused_List` stored in the array.

There are two essential operations related to `Unused_List`, namely, removing a node from it and returning a node to it. Assuming $O(1)$ time complexity for each of these two operations, what is the abstract data type that best describes `Unused_List`: Stack, Queue, or List?

(b) (3 points) Draw the other linked list(s) stored in the array.

(c) (4 points) You are given the array in the previous page. Consider the linked list that is longest in **(b)**. Suppose you delete from the linked list the last node, i.e., the node that does not point to a valid node. Show in the following table, the contents of the array.

index	info	next
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		

What is the index of the first item of `Unused_list`?

(d) (4 points) You are given the array in the previous page. Consider the linked list that is shortest in **(b)**. Suppose this linked list is an implementation of a queue, and that each of the primitive operations associated with the queue has $O(1)$ time complexity. You may assume that you are allowed to maintain $O(1)$ additional indices so that $O(1)$ time complexity is possible.

Suppose you first dequeue the front element of the queue. Then, you enqueue the integer '1000'. Show in the following table, the contents of the array after the operations.

index	info	next
9		
8		
7		
6		
5		
4		
3		
2		
1		
0		

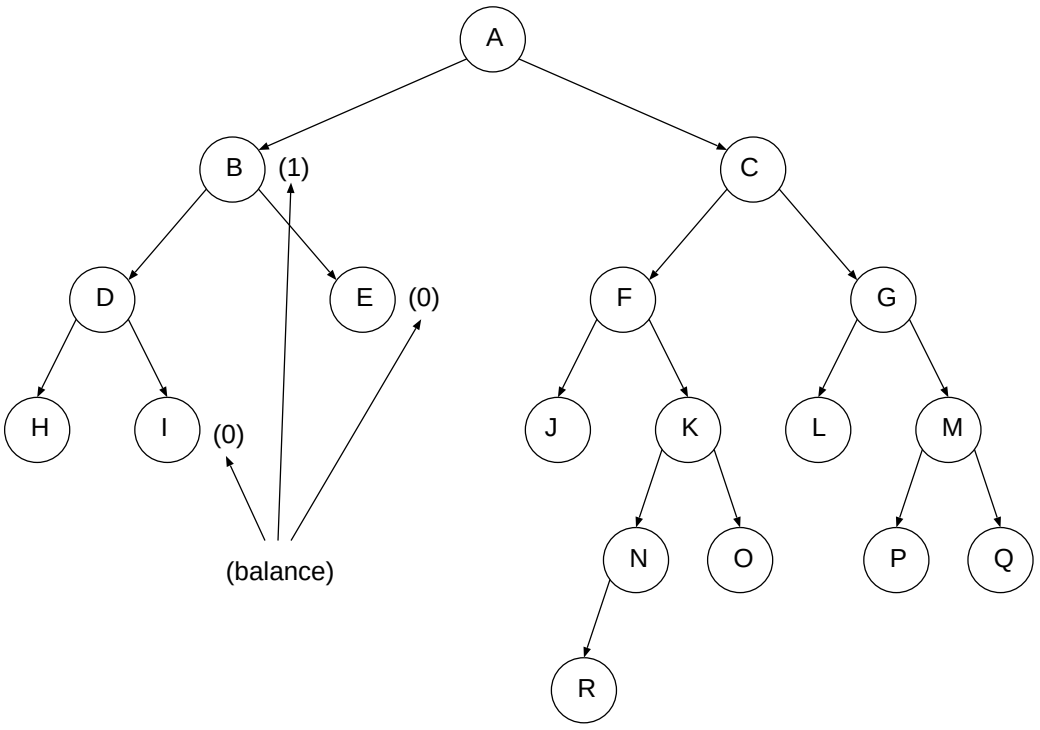
What is the index of the first item of `Unused_list`?

4. (15 points) Consider a binary tree T implemented with the following data structure:

```
typedef struct _Node {
  char label;
  int balance;
  struct _Node *left;
  struct _Node *right;
} Node;
```

The field `balance` of a node stores the difference of the height of the left subtree and the height of the right subtree, i.e., the height of the left subtree − height of the right subtree.

Suppose you have a binary tree that looks as follows:



The fields `balance` (shown in parentheses) of leaf nodes E and I are all assigned the value 0. The `balance` of node B is 1 because the left subtree is taller than the right subtree by 1 level.

(a) (5 points) For each of the remaining nodes in the preceding figure, fill in the `balance` field.

(b) (5 points) Modify one or more of the following tree traversal algorithms to compute the `balance` fields of all nodes in a given binary tree of n nodes.

```
Preorder_Traversal(tree_node):
  if (tree_node == NULL)
    return;
  print tree_node->label;
  Preorder_Traversal(tree_node->left);
  Preorder_Traversal(tree_node->right);


Inorder_Traversal(tree_node):
  if (tree_node == NULL)
    return;
  Inorder_Traversal(tree_node->left);
  print tree_node->label;
  Inorder_Traversal(tree_node->right);


Postorder_Traversal(tree_node):
  if (tree_node == NULL)
    return;
  Postorder_Traversal(tree_node->left);
  Postorder_Traversal(tree_node->right);
  print tree_node->label;
```

(c) (5 points) Write down an $O(n)$ time-complexity algorithm to perform a pre-order traversal of a given binary tree of n nodes in the following fashion: You **DO NOT** always visit the left subtree first. Instead, you visit the left subtree of a node first if the `balance` field of the node is negative (< 0). If the `balance` field of the node is non-negative (≥ 0), you visit the right subtree first. For example, you would print leaf node E before printing leaf node I (or H) and you would print leaf node H before printing leaf node I.

You will get full credit only if you remove all tail recursions.

Question	Max	Score
1	15	
2	15	
3	15	
4	15	
Total	60	