ECE368 Exam 1
Spring 2011

Thursday, February 24, 2010
6:30-7:30pm
PHYS 203

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 10 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 exercise and will be subject to possible disciplinary action."

`Printed Name:`

`login:`

`Signature:`

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO ...

1. (30 points) Consider the following recursive function that is defined based on two non-negative parameters m and n:

$$A(m,n) = \begin{cases} m & \text{if } n = 0, \\ n & \text{if } m = 0, \\ A(m-1,n) + A(m,n-1) & \text{otherwise.} \end{cases}$$

a. (10 points) Write a recursive algorithm (like the recursive algorithm to compute Fibonacci numbers) based on the above definition to compute $A(m,n)$. Do not attempt to derive a closed-form formula for $A(m,n)$. Assume that the algorithm will be invoked with only non-negative m and n.

b. (10 points) Draw a tree to represent the computation of $A(4,2)$ using the algorithm in **(a)**.

c. (5 points) What is the space complexity, in the big-*O* notation in terms of *m* and *n*, of your recursive algorithm in **(a)**? Justify your answer.

d. (5 points) What is the worst-case time complexity, in the big-*O* notation in terms of *m* and *n*, of your recursive algorithm in **(a)**? Justify your answer.

2. (30 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 {3,2,1} 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. (10 points) Consider the array $r = [3,7,9,0,5,1,6,8,4,2,0,6,1,5,7]$. Show the array r right after the iteration of $k = 3$ is completed. Show also the array r right after the iteration of $k = 2$ is completed.

After the iteration of $k = 3$:

After the iteration of $k = 2$:

b. (5 points) Consider $k = 3$. For any j in line 2, let t_j denote the number of times the while-loop test in line 5 is executed for that value of j. Express in terms of i, j, k, n, and t_j, the number of times instruction 5 is executed. Note that not all of i, j, k, n, and t_j may appear in your expression. (This question is similar to question 1 of Homework 1.)

Instruction 5:

c. (8 points) Compared to the conventional Insertion Sort algorithm, how much faster, on the average, is the code from **lines 2 through 10** for $k = 3$? Justify your answer. Note that the conventional Insertion Sort algorithm takes about $n^2/4$ comparisons on the average to sort n integers. Assume that the run-times are dominated by comparisons.

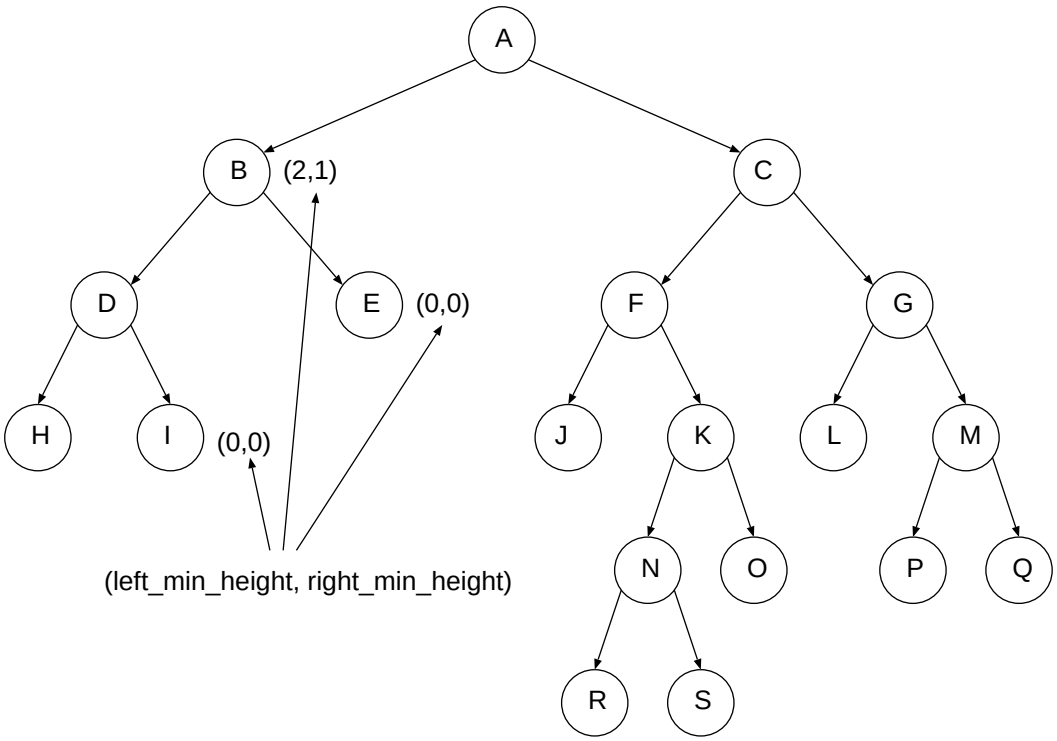
d. (7 points) Now, consider $k = 1$ (assume that you have performed 3-sorting and 2-sorting). Compared to the conventional Insertion Sort algorithm, how much faster, on the average, is the code from **lines 2 through 10** for $k = 1$? Justify your answer.

3. (30 points) Consider a strictly binary tree T implemented with the following data structure:

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

The fields `left_min_height` and `right_min_height` of a node store the minimum numbers of edges from the node to a leaf node in the left subtree and a leaf node in the right subtree, respectively.

Suppose you have a binary tree that looks as follows:



The fields `left_min_height` and `right_min_height` (shown in parentheses) of leaf nodes *E* and *I* are all assigned the value 0. The `right_min_height` of node *B* is 1, because *B* is only 1 edge away from *E*. The `left_min_height` of node *B* is 2 because *B* is two edges from a leaf node in the left subtree.

a. (5 points) Fill in the `left_min_height` and `right_min_height` fields for the rest of the nodes in the preceding figure.

b. (10 points) Modify one or more of the following tree traversal algorithms to compute the `left_min_height` and `right_min_height` fields of a given binary tree. (Hint: Determine the relationship between the `left_min_height` and `right_min_height` fields of a parent node and its child nodes. That relationship will reveal the correct traversal order.)

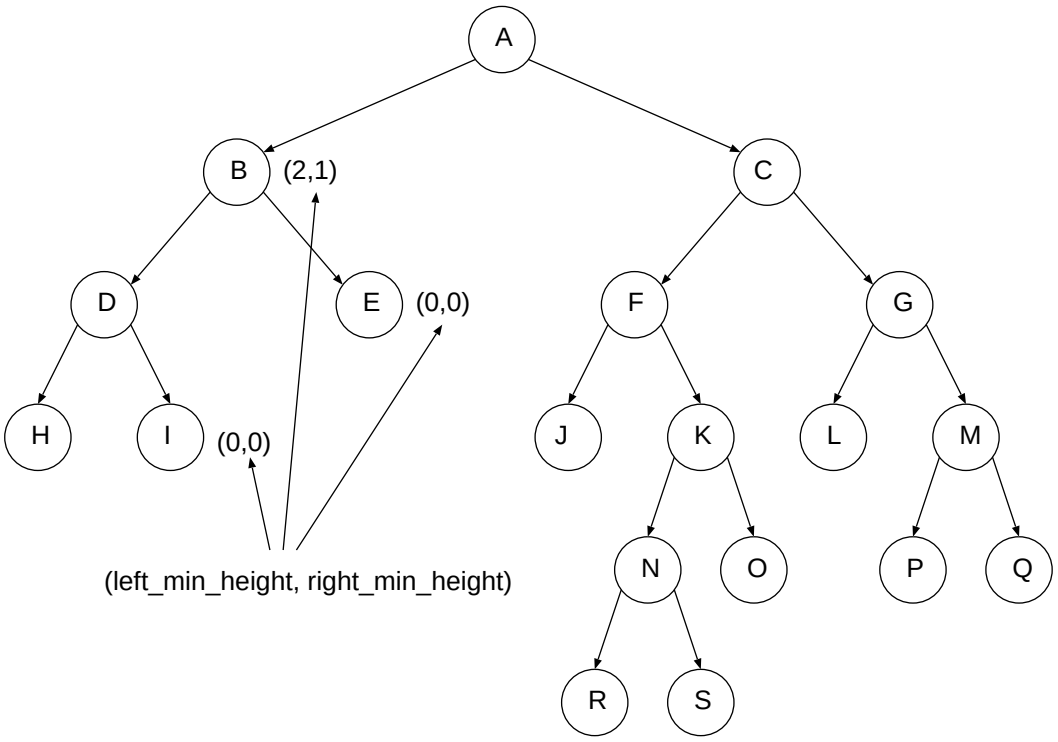
```
Preorder_Traversal(tree_node):
  if (tree_node == NULL)
    return;
  print tree_node->label;
  Preorder_Traversal(tree_node->left);
  Preorder_Traversal(tree_node->right);


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) Now, print the labels of the nodes in the binary tree given earlier in a preorder traversal fashion. (The binary tree is reproduced here for your convenience.) However, you **DO NOT** always visit the left subtree first, as given in the pseudo code in **(b)**.

Instead, you visit the subtree that corresponds to the smaller of the two fields `left_min_height` and `right_min_height` of a node first. If there is a tie, you visit the left 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*.



d. (10 points) Write down an $O(n)$ time-complexity algorithm to perform a traveral and printing of the strictly binary tree according to the order given in question **(c)**. You will get full credit only if the space complexity of your algorithm is

$$O(\max_{\text{all nodes in Tree}} \{\min(\texttt{left\_min\_height}, \texttt{right\_min\_height})\}).$$

The function in the big-O notation means for each node in the tree, we find the smaller of the two fields `left_min_height` and `right_min_height` of the node. Among these minimums of all nodes in the tree, we find the largest.

(Hint: Write an algorithm with recursions and then remove suitable recursion(s).)

Question	Max	Score
1	30	
2	30	
3	30	
Total	90	