ECE368 Mid-Term Exam 1
Fall 2006

Monday, October 2
7:00-8:00pm
EE 270

READ THIS BEFORE YOU BEGIN

This is a *closed book* exam. Calculators are not needed nor are they allowed. Since time allotted for this exam is *exactly* 60 minutes, you should work as quickly and efficiently as possible. *Note the allocation of points and do 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. *An answer without justification would not receive full credit.* Be *concise*. It is fine to use the blank page opposite each question for your work.

This exam consists of 10 pages (including this cover sheet and a grading summary at the end); 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. (24 points) Suppose you are given two disjoint sets of elements, i.e., S_1 and S_2. Your task is to implement the set union operator: $S = S_1 \cup S_2$. Assume that you do not have to keep a copy of S_1 or S_2.

1(a) (8 points) Assume that you have chosen the stack abstract data type to represent the sets. The primitive operations associated with the stack abstract data type S are `EMPTY`(S), `STACK_TOP`(S), `PUSH`(S, *element*), and `POP`(S), each with $O(1)$ time complexity and $O(1)$ space complexity.

Write down the most efficient algorithm to implement the set union operator. What are the run-time complexity and space complexity of your algorithm?

1(b) (8 points) Assume that you have chosen the queue abstract data type to represent the sets. The primitive operations associated with the queue abstract data type Q are `EMPTY`(Q), `ENQUEUE`(Q, *element*), `DEQUEUE`(Q), `FRONT`(Q), `REAR`(Q), each with $O(1)$ time complexity and $O(1)$ space complexity.

Write down the most efficient algorithm to implement the set union operator. What are the run-time complexity and space complexity of your algorithm?

1(c) (8 points) Assume that you have chosen the list abstract data type to represent the sets. The primitive operations associated with the list abstract data type L are `EMPTY`(L), `FIRST`(L), `LAST`(L), `INFO`(L, *node*), `NEXT`(L, *node*), `REMOVE_AFTER`(L, *node*), `INSERT_AFTER`(L, *node*, *new_node*), each with $O(1)$ time complexity and $O(1)$ space complexity. Note that the primitive operation `REMOVE_AFTER`(L, *node*) removes the item that comes immediately after *node* in L and the primitive operation `INSERT_AFTER`(L, *node*, *new_node*) inserts *new_node* immediately after *node* in L.

Write down the most efficient algorithm to implement the set union operator. What are the run-time complexity and space complexity of your algorithm?

2. (25 points) Consider the following recursive routine that computes the Fibonacci sequence and stores the computed sequence in array $Fib[0..N]$, which is a global variable initialized with $Fib[i] = -1$ for $0 \leq i \leq N$:

```
Fibonacci($n$)
    if ($Fib[n]$ == $-1$) {
        if ($n$ == 0) or ($n$ == 1) {
            $Fib[n] = n$
        } else {
            $Fib[n]$ = Fibonacci($n-1$) + Fibonacci($n-2$)
        }
    }
    return $Fib[n]$
```

2(a) (8 points) Draw a binary tree to illustrate the recursion calls involved in the execution of the routine `Fibonacci`(5). Assume that this is the first time the recursive routine is called and that $5 \leq N$. Every node in this binary tree should be labeled `Fibonacci`(i) to represent a recursion call with parameter i. Hence, the root node of this binary tree should be labeled `Fibonacci`(5).

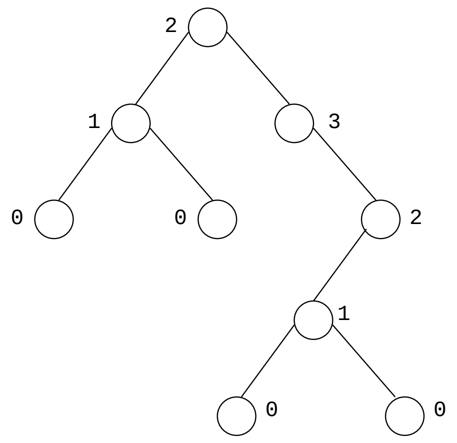
2(b) (5 points) Suppose the recursive routine is invoked again with input n being 7, i.e., `Fibonacci`(7). Assume that the array $Fib[n]$ stays unchanged since the function call `Fibonacci`(5) in **2(a)** and that $7 \le N$. Draw a binary tree to illustrate the recursion calls involved in the execution of the routine `Fibonacci`(7).

2(c) (12 points) Disregard the space occupied by the array $Fib[0..N]$. Also, disregard the time taken to initialize $Fib[0..N]$. What are the worst-case and best-case run-time complexity of `Fibonacci`(n)? What are the worst-case and best-case space complexity of `Fibonacci`(n)?

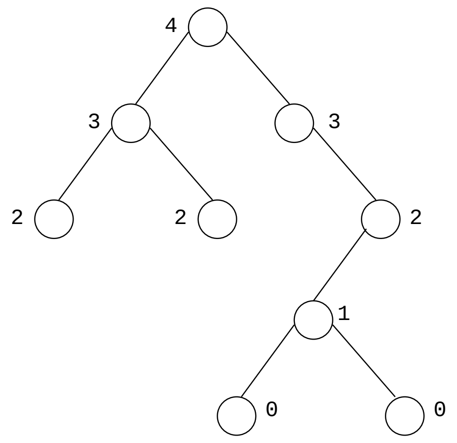
3. (25 points) Consider the following recursive tree traversal algorithms for binary trees:

```
Preorder_Traversal(node):
  if (node != NULL) {
    print node
    Preorder_Traversal(left(node))
    Preorder_Traversal(right(node))
  }


Inorder_Traversal(node):
  if (node != NULL) {
    Inorder_Traversal(left(node))
    print node
    Inorder_Traversal(right(node))
  }


Postorder_Traversal(node):
  if (node != NULL) {
    Postorder_Traversal(left(node))
    Postorder_Traversal(right(node))
    print node
  }
```

3(a) (10 points) Modify one (or more) of the three given algorithms to compute the shortest path length to a descendant leaf node of each node. The shortest path length to a descendant leaf node of a leaf node is 0. The shortest path length to a descendant leaf node of a non-leaf node is greater than the minimum shortest path length of its child nodes by exactly one. The following figure shows the shortest path length to a descendant leaf node beside each node. Assume that there is a field called `Shortest_Path_Length` for storing the computed value in your tree data structure.



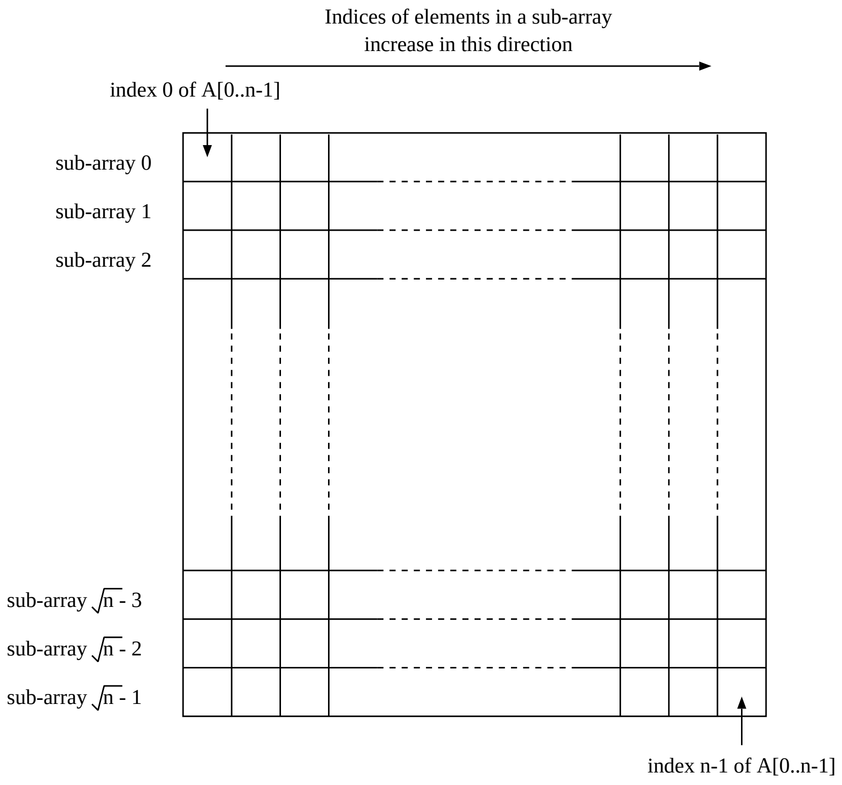
3(b) (15 points) Modify one (or more) of the three given algorithms to compute the height of each node. The node height is defined to be the difference between the depth of the tree and the level of the node. The following figure shows the node height beside each node. Assume that there is a field called `Height` for storing the computed value in your tree data structure.



4 (26 points) Consider the following sorting algorithm, applied on an array $A[0..n-1]$, with n being the square of an integer:

```
Step 1:          Divide the $n$ elements of the array $A[0..n-1]$ into $\sqrt{n}$ sub-arrays
                 (labeled 0 through $\sqrt{n}-1$), each with $\sqrt{n}$ elements;
Step 2:          Sort in ascending order each of the $\sqrt{n}$ sub-arrays using Insertion Sort;
Step 3:          for $i = \sqrt{n}-1$ downto 1 do {
Step 3(a):           Find (as in Selection Sort) max_index, the index of the maximum element
                     among the sub-arrays labeled 0 through $(i-1)$;
Step 3(b):           While $A[i*\sqrt{n}] < A[\texttt{max\_index}]$ do {
Step 3(b)(i):            $A[i*\sqrt{n}] \leftrightarrow A[\texttt{max\_index}]$;
Step 3(b)(ii):           Insert (as in Insertion Sort) $A[i*\sqrt{n}]$ into sub-array $i$;
Step 3(b)(iii):          Insert (as in Insertion Sort) $A[\texttt{max\_index}]$ into the sub-array from which
                         the old $A[\texttt{max\_index}]$ came;
Step 3(b)(iv):           Find (as in Selection Sort) max_index, the index of the maximum element
                         among the sub-arrays labeled 0 through $(i-1)$;
                     }
                 }
```

4(a) (6 points) The following diagram is an illustration of the division of $A[0..n-1]$ into $\sqrt{n}$ sub-arrays. Consider the case when $i = \sqrt{n}-1$ (the first iteration of *Step 3*). Indicate on the diagram (i) the location of $A[i*\sqrt{n}]$ (*Step 3(b)*), and (ii) the region where you could find $A[\texttt{max\_index}]$ (*Step 3(a)* or *Step 3(b)(iv)*) in $O(\sqrt{n})$ time complexity.



4(b) (6 points) There are two approaches for inserting an item into a sorted sub-array:

- **Method 1:** The item to be inserted is to the right of the sorted sub-array. The item iteratively swaps with the neighbor to its left until it is in the correct position.
- **Method 2:** The item to be inserted is to the left of the sorted sub-array, and the item iteratively swaps with the neighbor to its right until it is in the correct position.

Identify a suitable method for performing the insertion of

1. **(3 points)** $A[i * \sqrt{n}]$ into sub-array i (in *Step 3(b)(ii)*).
2. **(3 points)** $A[\texttt{max\_index}]$ into the sub-array from which the old $A[\texttt{max\_index}]$ came (in *Step 3(b)(iii)*).

4(c) (3 points) What is the worst-case time complexity of *Step 2*?

4(d) (4 points) In the worst case, how many times would the body (*Steps 3(b)(i)–(iv)*) of the while-loop in *Step 3(b)* be executed for a particular i? Justify your answer.

4(e) (4 points) What is the worst-case time complexity of the sorting algorithm?

4(f) (3 points) What is the space complexity of the sorting algorithm?

Question	Max	Score
1	24	
2	25	
3	25	
4	26	
Total	100	