ECE368 Mid-Term Exam 1
Fall 2005

Monday, October 3
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 11 pages (including this cover sheet and a grading summary sheet 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(a) (8 points) Design an algorithm to find both the maximum and minimum of n numbers using at most $\lceil 3n/2 \rceil$ comparisons and no swaps. (To be exact, you can do that using $\lceil 3n/2 \rceil - 2$ comparisons.) You can either describe your algorithm in words or in pseudo-code. For simplicity, assume that n is even.

1(b) (8 points) The following is the pseudo-code for the Selection-Sort algorithm, which uses $(n^2 - n)/2$ comparisons and $n - 1$ swaps to sort an array of n elements.

```
Selection_Sort(A[0..n-1]):
  for i ← n-1 downto 1 {
    max_idx ← 0
    for j ← 1 to i {
      if A[j] ≥ A[max_idx] {
        max_idx ← j
      }
    }
    A[i] ↔ A[max_idx]
  }
```

Modify the Selection-Sort algorithm to use the algorithm you have designed in **1(a)**. Assume that your algorithm in **1(a)** is called `Min_Max(A[lb..ub])`. It finds the indices of the minimum and maximum elements in the subarray `A[lb..ub]` and returns a pair of indices `(min_idx, max_idx)`:

`(min_idx, max_idx)` ← `Min_Max(A[lb..ub])`

1(c) (8 points) For simplicity, assume that the algorithm `Min_Max(`A`[lb..ub])` uses exactly $3(\texttt{ub} - \texttt{lb} + 1)/2$ comparisons to find the indices of the minimum and maximum elements in the subarray `Min_Max(`A`[lb..ub])`. What is the total number of comparisons taken by your Selection-Sort algorithm in **1(b)** to sort $A[0..n-1]$, an array of n elements? Assume that n is even.

2. (20 points) Show how to implement a queue using two stacks with the following primitives of the stack abstract data type: `EMPTY`(*S*), `STACK_TOP`(*S*), `PUSH`(*S*, *element*), and `POP`(*S*)? Assuming $O(1)$ time complexity for all these primitives, what are the run-time complexity of the queue operations: `EMPTY`(*Q*), `ENQUEUE`(*Q*, *element*), `DEQUEUE`(*Q*), `FRONT`(*Q*), `REAR`(*Q*). (Note: Not all queue operations can be perform in O(1) time complexity.)

3. (18 points) Consider an array-based implementation of singly linked list(s). The contents of the array are as follows:

index	key	next_index
9	A	0
8	B	-1
7	C	3
6	D	8
5	E	7
4	F	-1
3	G	-1
2	H	6
1	I	2
0	J	4

The array contains three lists:

- `Unused_List`: Maintains a list of unused objects. The list starts at index 9.
- `Stack_List`: Implements a stack. The list starts at index 5.
- `Queue_List`: Implements a queue. The list starts at index 1.

3(a) (3 points) Write down the keys (from the head to the tail) in the three lists.

- `Unused_List`:
- `Stack_List`:
- `Queue_List`:

3(b) (3 points) In order to implement all the stack and queue primitives in O(1) time complexity, what other indices, besides the indices pointing to the heads of the three lists, do you have to keep track? The goal here is to minimize the additional space requirement.

3(c) (12 points) Show the contents of the array after performing each of the following three operations in sequence:

1. `Enqueue(Queue_List, Stack_Top(Stack_List))`
2. `Push(Stack_List, K)`
3. `Dequeue(Queue_List)`

Update also the head indices of the lists, as well as the indices you have added in **3(b)**.

	Original		Enqueue		Push		Dequeue	
index	key	next_index	key	next_index	key	next_index	key	next_index
9	A	0						
8	B	-1						
7	C	3						
6	D	8						
5	E	7						
4	F	-1						
3	G	-1						
2	H	6						
1	I	2						
0	J	4						

Head index	Original	Enqueue	Push	Dequeue
Unused_List	9			
Stack_List	5			
Queue_List	1			

Other indices	Original	Enqueue	Push	Dequeue

4(a) (8 points) Consider the following depth-first traversal routine using a stack S:

```
Depth_First_Traversal(Tree_root):
   Push(S, Tree_root)
   while (S not empty) {
     node ← Pop(S)
     if (node != NULL) {
       print node
       Push(S, left(node))
       Push(S, right(node))
     }
   }
```

Given an n-node binary tree, what is the worst-case space complexity of the algorithm, i.e., the worst-case asymptotic growth rate of the space requirement of S? What is the best-case space complexity of the algorithm? (Hint: How would the topology of a binary tree affect the space requirement? For example, a complete binary tree has an $O(\log n)$ space complexity.)

4(b) (8 points) Consider the following breadth-first traversal routine using a queue Q:

```
Breadth_First_Traversal(Tree_root):
  Enqueue(Q, Tree_root)
  while (Q not empty) {
    node ← Dequeue(Q)
    if (node != NULL) {
      print node
      Enqueue(Q, left(node))
      Enqueue(Q, right(node))
    }
  }
```

Given an n-node binary tree, what is the worst-case space complexity of the algorithm, i.e., the worst-case asymptotic growth rate of the space requirement of Q? What is the best-case space complexity of the algorithm?

5. (22 points) Consider the following recursive tree traversal algorithms:

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

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

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

5(a) (8 points) Modify one of the three given algorithms to compute the level of each node. Assume that there is a field called `Level` for storing the computed value in your tree data structure.

5(b) (14 points) Modify one of the three given algorithms to compute the number of leaf nodes in the left subtree of each node. Assume that there is a field called `Left_Leaf_Descendents` for storing the computed value in your tree data structure. Note that the `Left_Leaf_Descendents` of a leaf node is 0.

Question	Max	Score
1	24	
2	20	
3	18	
4	16	
5	22	
Total	100	