Computer Science Foundation Exam

December 19, 2008

Computer Science

Section 1B

Name: Solution and Grading Criteria

SSN: _____

	Max Pts	Type	Passing Threshold	Student Score
Q1	10	ANL	7	
Q2	10	DSN	7	
Q3	10	ANL	7	
Q4	10	DSN	7	
Q5	10	ANL	7	
Total	50		35	

You must do all 5 problems in this section of the exam.
Partial credit cannot be given unless all work is shown and is readable.

Be complete, yet concise, and above all be neat. Do your rough work on the last page.

1) (10 points) **Order Notation**

Indicate the time complexity in terms of Big-O and the appropriate variables for each of the following operations:

a) Using mergesort to sort *n* integers. **O(nlgn)**

b) Pushing *n* items on to an initially empty stack. **O(n)**

c) Searching for an element in a sorted array of *n* elements using a binary search (*worst case)* **O(lgn)**

d) Searching for an element in a sorted array of *n* elements using a binary search (*best case*) **O(1)**

e) Searching for an element in a sorted array of *n* elements using a linear search (*worst case*) **O(n)**

f) Converting an infix expression with *n* symbols to a postfix expression using the standard algorithm **O(n)**

g) Inserting *n* elements into an initially empty AVL tree (*worst case*) **O(nlgn)**

h) Inserting *n* elements into an initially empty BST that does not enforce balance properties (*worst case*) **$O(n^2)$**

i) Adding two *n* digit numbers together using the standard addition algorithm you were taught in elementary school **O(n)**

j) Performing a postfix traversal of a binary tree containing *n* elements **O(n)**

Grading: 1 pt each, all or nothing. Accept answers that are correct within a constant factor.

2) (10 points) **Binary Trees**

Write a function that operates on a binary tree of integers. Your function should sum up the all of the odd numbers in the tree EXCEPT for the numbers in leaf nodes, which should instead be ignored. Make use of the binary tree node structure and function prototype shown below.

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

int sum_nonleaf_odd(struct treenode* p)
{

  if (p == NULL) return 0; // 1 pt

  // 2 pts
  if (p->left == NULL && p->right == NULL)
    return 0;

  // 2 pts
  int sum = 0;
  if (p->data%2 == 1)
    sum += p->data;

  // 2 pts
  sum += sum_nonleaf_odd(p->left);

  // 2pts
  sum += sum_nonleaf_odd(p->right);

  // 1 pt
  return sum;
}
```

3) (10 points) **Recursion**

Consider the following recursive function:

```
void mystery(int x) {
     printf("%d\n", x);
     if(x == 0)
          return;
     else if(x % 2 == 1)
          mystery(2 * x);
     else
          mystery(x/2 - 1);
}
```

a) What would be printed by the call to `mystery(5)`?

5
10
4
1
2
0

Grading: 1 pt off for each mistake, capped at 5 off.

b) What would be printed by the call to `mystery(18)`?

18
8
3
6
2
0

Grading: 1 pt off for each mistake capped at 5 off.

4) (10 points) **Linked Lists**

Write a function which moves the last element of a non-empty linked list to the front and returns a pointer to the new head of the list. For example, if the list contains the numbers 3, 2, 8, 1, and 6, then the order after the function is called should be 6, 3, 2, 8, 1, and the function should return a pointer to the node containing the value 6. Utilize the struct and function prototype provided below.

```
struct node {
     int data;
     struct node* next;
};

struct node* move_back_to_front(struct node* alpha) {

  // unnecessary since the list won't be empty.
  if (alpha == NULL) return NULL;

  // No changes are made for a one node list. (1 pt)
  if (alpha->next == NULL) return alpha;

  struct node* tmp = alpha;

  // Move tmp to second to last node. (2 pts)
  while (tmp->next->next != NULL)
    tmp = tmp->next;

  // Set up pointer to new front node. (2 pt)
  struct node* newFront = tmp->next;

  // Link back to front. (2 pt)
  newFront->next = alpha;

  // End the list here. (2 pt)
  tmp->next = NULL;

  // Now this is the front of the list. (1 pt)
  return newFront;




}
```

5) (10 points) **Linked Lists**

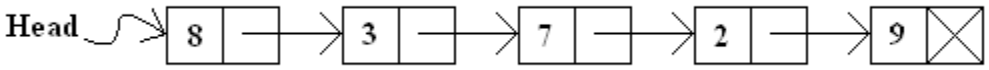
Consider the following code that makes use of the node structure from the previous question:

```
void mysterious(struct node* alpha)
{
     struct node* temp;

     if(alpha == NULL || alpha->next == NULL)
          return;

     temp = alpha->next;
     alpha->next = alpha->next->next;
     free(temp);

     mysterious(alpha->next);
}
```

a) Explain concisely what the `mysterious` function does.

It deletes every other node (2nd, 4th, 6th, etc.) from the list pointed to by alpha. (5 pts – give partial credit in a reasonable and consistent manner.)

b) Show the state of the following linked list after a call to `mysterious(head)`.



Head -> 8 -> 7 -> 9

Grading: 5 pts – 2 pts for having some subsequence of the items listed in the list pointed to by head.
3 more points for having the correct ones.
Give 3 points out of 5 for having the even items instead of the odd ones.