

Computer Science Foundation Exam

August 10, 2007

Computer Science

Section 1B

Name: Solution and Grading Criteria

SSN: _____

	Max Pts	Type	Passing Threshold	Student Score
Q1	10	ANL	7	
Q2	10	ANL	6	
Q3	10	DSN	7	
Q4	10	DSN	7	
Q5	10	ANL	6	
Total	50		33	

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 pts]

Indicate the *worst case* time complexity in terms of Big-O for each of the following operations:

- | | |
|--|-----------------|
| a) Searching for a target in an unsorted array of Q items | O(Q) |
| b) Determining the smallest integer in a sorted array of M integers. | O(1) |
| c) Searching for a target in an AVL tree of N elements | O(log N) |
| d) Printing out all items in a Binary Search Tree of K items | O(K) |
| e) Enqueuing an item into a queue of N items that is stored as a linked list with a pointer to the front of the list only. | O(N) |
| f) Inserting a node in a Binary Search Tree of P items. | O(P) |
| g) Popping an item from a stack of S items. | O(1) |
| h) running the following sorting algorithms on an array containing N integers already sorted in the correct order | |
| 1. Insertion sort | O(N) |
| 2. Merge sort | O(NlogN) |
| 3. Quick sort (middle element as the pivot) | O(NlogN) |

Grading: 1 pt each, no partial credit

2.(a) [5 pts] Use summations to express the final value of beta being returned by the following function. *You are not required to solve the summations.*

```
int compute (int n)
{
    int j, k;
    int beta = 0;
    int alpha = 10;
    for ( k= 1 ;    k <= 100 ;    k++)
        for ( j = n;    j < 2*n+1 ;    j++)
            beta += (alpha *k - 6*n + j) ;
    return beta;
}
```

$$\sum_{k=1}^{100} \sum_{j=n}^{2n} (10k - 6n + j)$$

Grading: 1 pt for outer summation, 1 pt for bounds on outer sum, 1 pt for inner summation, 1 pt for bounds on inner sum, 1 pt for expression inside the sum, give credit for equivalent expressions

b) **[5 pts]** Write the recurrence relation to indicate the total number of operations $T(n)$, needed to execute the following function in terms of n . Consider the worst case. *You are not required to solve the recurrence relation.*

```
int newfunction(int vals[ ], int n)
{
    if (n == 1) return 0;
    if (vals[n-1] > 50)
        return 2*newfunction(AA,n/2)+1;
    else
        return newfunction(AA, n/2)+1;
}
```

$T(n) = T(n/2) + c$, where c is a constant

Grading: 1 pt for defining a function, 1 pt for recursive definition, 1 pt for $n/2$, 1 pt for not multiplying $T(n/2)$ by 2, 1 pt for adding a constant (1, 2 or other)

3. **[10 pts]** Write a function which returns a pointer to a linked list which is the result of deleting the second node in the linked list pointed to by alpha. If alpha contains zero or one element, your function should simply return a pointer to the original linked list. The node structure is given below. You will ONLY receive full credit if you free the memory for the deleted node.

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

struct node* deleteSecond(struct node* alpha) {

    struct node* delnode;
    struct node* firstnode;

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

    delnode = alpha->next;
    alpha->next = delnode->next;
    free(delnode);
    return alpha;

}
```

Grading:

1 pt - zero node case
2 pts - one node case
2 pts - patching first node to third node
2 pts - freeing memory for deleted node
2 pts - returning ptr to front of new list
1 pt - floating point for any other issues

4. [10 pts] Write a function that searches for a node storing the value `val` in the binary search tree rooted at `p`. The struct `treenode` is defined below. Your function should return an integer indicating whether or not the value was found in the binary search tree. It should return 1 if the value was found, and 0 if it was NOT found.

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

int search(struct treenode* p, int val) {

    if (p == NULL) return 0;

    if (val == p->data) return 1;
    if (val < p->data)
        return search(p->left, val);
    return search(p->right, val);

}
```

Grading: 2 pts – null case

- 2 pts – found at root**
- 2 pts – found on the left**
- 2 pts – found on the left**
- 2 pts – floating points**

If everything is correct but they switch left and right, take off 2 points only

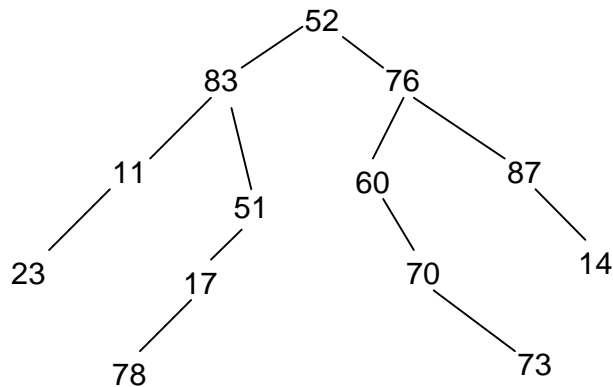
5. a) [5 pts] Indicate in few words, the purpose of the following function. The struct `treenode` is the same as the one defined in problem 4 on the previous page.

```
int f(struct treenode * p) {  
  
    int val;  
    if (p == NULL) return 0;  
    val = p->data;  
    if (val%2 == 0)  
        return val + f(p->left) + f(p->right);  
    else  
        return f(p->left) + f(p->right);  
  
}
```

Returns the sum of the nodes storing even values in the tree.

Grading: 2 pts for sum, 1 pt for node values, 2 pts for even

b) [5 pts] What does the function return, given the following tree?



Answer: $78+52+76+60+70+14 = 350$