Computer Science Foundation Exam

December 17, 2010

Section I B

COMPUTER SCIENCE

NO books, notes, or calculators may be used,
and you must work entirely on your own.

SOLUTION

Question #	Max Pts	Category	Passing	Score
1	10	ANL	7	
2	10	DSN	7	
3	10	DSN	7	
4	10	ALG	7	
5	10	ALG	7	
TOTAL	50			

You must do all 5 problems in this section of the exam.

Problems will be graded based on the completeness of the solution steps and <u>not</u> graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all <u>be neat</u>.

1) (10 x 1 pt): Order Analysis

Indicate the time complexity for each of the following operations in terms of Big-O notation, assuming that efficient implementations are used. Give the *worst case* complexities. Following notations are being used:

AINC is an array containing **`n`** integers arranged in increasing order.
AD is an array containing **`n`** integers arranged in decreasing order.
AR is an array containing **`n`** integers in random order.
Q is a queue implemented as a linked list and containing **`p`** elements.
LINK is a linked list containing **`n`** nodes.
CIRC is a circular linked list containing **`n`** elements, where **`C`** points to the last element.
T is a binary search tree containing **`n`** nodes.
AVL-T is an AVL tree containing **`n`** nodes.

a) Deleting a single item from AVL-T **O(lg n)**

b) Deleting a single item from T. **O(n)**

c) Executing x consecutive enqueue functions in Q **O(x)**

d) Inserting an element at the front of CIRC. **O(1)**

e) Using an insertion sort to sort AD in increasing order **$O(n^2)$**

f) Printing out all elements of T. **O(n)**

g) Using a Merge Sort to sort AR in increasing order **O(nlgn)**

h) Inserting an item to the back of LINK. **O(n)**

i) Inserting an item to the front of LINK. **O(1)**

j) Multiplying two n-digit numbers using the typical grade-school algorithm. **$O(n^2)$**

Grading: 1 pt each no partial credit.

2) (10 points) Binary Trees

Write a recursive function that will return the k^{th} smallest value stored in a binary **search** tree. You may assume that all the values stored in the tree are distinct and that there are at least k values stored in the tree pointed to by ptr. You are also given a helper function (which you should call) below. (Note: If k = 1, then the function should find the smallest value in the tree.)

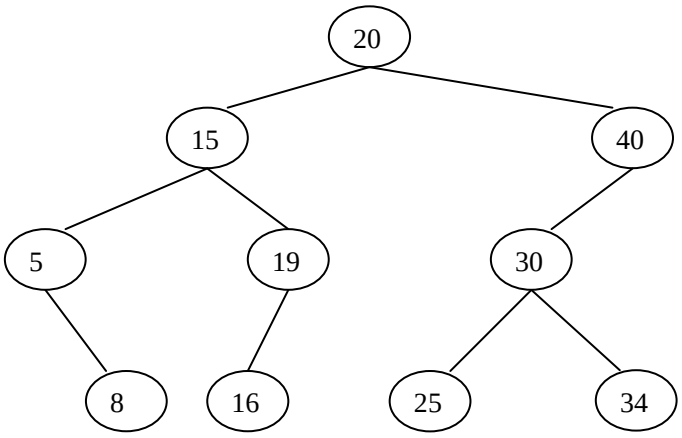
```
struct treeNode {
   int data;
   struct treeNode *left, *right:
};



int rank(struct treeNode* ptr, int k)  {

   int nodeRank = numnodes(ptr->left); // 3 pts

   if (nodeRank == k-1)                  // 1 pt
       return ptr->data;                 // 1 pt

   else if (nodeRank > k-1)              // 1 pt
       return rank(ptr->left, k);        // 2 pts

   else
       return rank(ptr->right, k-nodeRank-1); // 2 pts









}

int numnodes(struct treeNode* ptr) {
    if (ptr == NULL) return 0;
    return 1 + numnodes(ptr->left) + numnodes(ptr->right);
}
```

3) (10 points) Linked Lists
Write a function which accepts a linear linked list J and deletes the second node from the list. If the original list has fewer than two items, nothing should be done. The function prototype is provided for you below.

The node structure is as follows:

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

void delSecondNode(struct listNode* J) {

    if (J != NULL && J->next != NULL) {     // 2 pts

        struct listNode* temp = J->next;    // 3 pts

        J->next = J->next->next;            // 3 pts

        free(temp);                         // 2 pts
    }
}
```

4) (10 points) Binary Trees
Given the binary tree shown below, determine what gets printed when the function **A(root)** is called. Assume that the struct used is defined as shown below. Assume that **root** is of type struct treeNode* and is a pointing to the node containing 20 in the tree below. Place your answers in the boxes provided.




```
struct treeNode{
   int data;
   struct treeNode *left, *right:
}

void A(struct treeNode *node_ptr){
    if (node_ptr != NULL){
       printf("%d ,",node_ptr->data);
       B(node_ptr->left, 10);
       B(node_ptr->right, 20);
    }
}

void B(struct treeNode *node_ptr, int key){
    if (node_ptr != NULL) {
       A(node_ptr->left);
       printf("%d ,",key + node_ptr->data);
       A(node_ptr->right);
   }
}
```

ANSWER:

20	**5**	**28**	**25**	**19**	**26**	**30**	**35**	**54**	**60**

Grading: 1 pt per slot, no partial credit.

5) (10 points) Sorting In a Merge Sort of 8 items, the Merge function gets called 7 times. Show the contents of the array below after EACH of the calls to the Merge function. (The last answer is filled in for you, since it's just the sorted array.)

Index	0	1	2	3	4	5	6	7
Original	19	3	14	17	2	1	10	9
After 1st Merge	**3**	**19**	**14**	**17**	**2**	**1**	**10**	**9**
After 2nd Merge	**3**	**19**	**14**	**17**	**2**	**1**	**10**	**9**
After 3rd Merge	**3**	**14**	**17**	**19**	**2**	**1**	**10**	**9**
After 4th Merge	**3**	**14**	**17**	**19**	**1**	**2**	**10**	**9**
After 5th Merge	**3**	**14**	**17**	**19**	**1**	**2**	**9**	**10**
After 6th Merge	**3**	**14**	**17**	**19**	**1**	**2**	**9**	**10**
After 7th Merge	1	2	3	9	10	14	17	19

Grading: 1st and 2nd Merge: 1 pt each
3rd Merge: 3 pts
4th, 5th Merge: 1 pt each
6th Merge: 3 pts

If they do all 4 pairs first, then left 4, then right 4, give them 6/10.