# Computer Science Foundation Exam

**May 4, 2012**

**Section I B**

**SOLUTION**

**COMPUTER SCIENCE**

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

| Question # | Max Pts | Category | Passing | Score |
|---|---|---|---|---|
| 1 | 10 | ALS | 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 not 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 be neat.**

**1)** (10 pts) ALS (Order Notation)

Give the Big-O running times of the following operations in terms of *n*. Assume an efficient implementation of each data structure and algorithm listed.

| | |
|---|---|
| (a) (1 pt) Insertion (or deletion) into (or from) a Binary Heap of *n* elements: | **O(lg n)** |
| (b) (1 pt) Sorting *n* elements using Insertion Sort (worst case): | **$O(n^2)$** |
| (c) (1 pt) Sorting *n* elements using Quick Sort (worst case): | **$O(n^2)$** |
| (d) (1 pt) Finding the maximum value in an unsorted array of *n* items: | **O(n)** |
| (e) (1 pt) Finding the maximum value in a Binary Search Tree of *n* elements: | **O(n)** |
| (f) (1 pt) Deleting the 17th node of a linked-list containing *n* items: | **O(1)** |
| (g) (1 pt) Finding the *k*th smallest value in a sorted array of *n distinct* integers: | **O(1)** |
| (h) (1 pt) Building a heap from *n* arbitrary values (worst case): | **O(n)** |
| (i) (1 pt) Inserting an item at the end of a doubly-linked linked list with *n* items: | **O(n)** |
| (j) (1 pt) Inserting a new node into an AVL tree containing *n* nodes: | **O(lg n)** |

**Grading: 1 pt each, all or nothing. Technically, if they write something like O(17) or O(2lg n), these are equivalent to O(1) and O(lg n), and should be counted correctly.**

**2)** (10 pts) DSN (Recursive Algorithms)

Suppose you are building an N node binary search tree with the values 1…N. How many structurally different binary search trees are there that store those values? Write a recursive function, **`count_trees`**, that, given the number of distinct values, computes the number of structurally unique binary search trees that store those values.

For example, **`count_trees(4)`** should return 14, since there are 14 structurally unique binary search trees that store 1, 2, 3, and 4. Your code should not construct any actual trees; it's just a counting problem. (*Hint: consider that each value could be the root. Find the size of the left and right subtrees, and recursively count the number of structures each side could have.*)

Make use of the function header below:

```
int count_trees(int numKeys) {

     if (numKeys <=1) {            // 1 pt
          return 1 ;               // 1 pt
     }

     else {
          int sum = 0;
          int left, right, root;

          for (root=1; root<=numKeys; root++) {       // 1 pt
               left = count_trees(root - 1);           // 2 pts
               right = count_trees(numKeys - root);    // 2 pts
               sum += left*right;                      // 2 pts
          }

          return(sum);                                 // 1 pt
     }

}
```

**3)** (10 pts) DSN (Linked Lists)

Write a function that operates on an existing linked list of integers. The function will have two parameters passed in: the head of the list and an integer value (**`num`**). Your function should find (if it exists) and delete the first node with the value "**`num`**" in the list and should then return a pointer to the updated head of the linked list. If the listed pointed to by front does NOT contain num, do not make any changes to the list and return the original front of the list.

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

struct node* delete(struct node *front, int num) {

     struct node *temp, *del;
     temp = front;

     // Only need to delete if the list is not null.
     if (temp != NULL) {

          // Take care of the case where first node needs to be deleted.
          if (temp->data == num) {
               del = temp;
               temp = temp -> next;
               free(del);
               return temp;
          }

          // Otherwise, loop until you find the node to delete and do so.
          while (temp->next != NULL) {
               if (temp ->next->data == num) {
                    del = temp -> next;
                    temp->next = temp ->next->next;
                    free(del);
                    return front;
               }
               temp = temp -> next;
          }
     }
     return front;
}
```

```
Grading: 2 pts for no NULL ptr errors
         2 pts for handling the case where num is the 1st node
         2 pts for handling the case where num isn't in the list
         4 pts for handling the typical case (num is in the list
              but isn't the first node.)
```

**4)** (10 pts) ALG (Tracing)

What is printed out by running the following program? Fill in the result in the boxes below:

```
#include <stdio.h>

int foo(int n);
int foo2(int n);

int main() {

    int i;
    for(i = 3; i <= 7; i++)
        printf("%d ", foo(i));
    return 0;
}

int foo(int n) {
    if (n < 2)
        return n;
    else
        return foo(n-1) + foo2(n-2);
}

int foo2(int n) {
    if (n <= 1)
        return n;
    else
        return (2 * foo(n-2) + 1);
}
```

| | | | | |
|---|---|---|---|---|
| **2** | **3** | **6** | **9** | **14** |

**Grading: 2 pts each, no partial credit**

**5)** (10 pts) ALG (Base Conversion)

(a) (5 pts) Convert 7923 in base 10 to hexadecimal (base 16) using the deterministic algorithm taught in class. (Note: An ad hoc method using guessing and checking will NOT be given credit!!!)

```
16   |   7923     R 3
16   |    495     R 15
16   |     30     R 14
16   |      1     R 1
16   |      0
```

**Reading the remainders from bottom to top, we get $1EF3_{16}$ as the converted value in hexadecimal.**

**Grading: 1 pt for each step and 1 pt for the final answer.**

(b) (5 pts) Convert $3A76_{16}$ to octal (base 8). (Note: Use an intermediate base for conversion. Either base 2 or base 10 will be accepted as this intermediate base.)

**Converting the value given to base 2, we get:**

**0011101001110110 (2 pts)**

**Regrouping by groups of 3 to convert into base 8 from the right, we get**

**011 101 001 110 110 (2 pts)**

**Converted to base 8, we arrive at:**

**$35166_8$ (1 pt)**

**Note: The converted base 10 value is $14966_{10}$. If they take this route, 2 pts for this base 10 value and 3 pts for the final base 8 value.**