# Computer Science Foundation Exam

## May 6, 2016

## Section I A

## 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 | DSN | 7 | |
| 2 | 10 | ANL | 7 | |
| 3 | 10 | ALG | 7 | |
| 4 | 10 | ALG | 7 | |
| 5 | 10 | ALG | 7 | |
| TOTAL | 50 | | 35 | |

**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 pts) DSN (Recursive Functions)

Complete the function below, to create a ***recursive*** function ***productDigits*** that takes in 1 parameter, a non-negative integer, *n*, and returns the product of *n*'s digits. For example *productDigits(232)* will return 12 (2 x 3 x 2), and *productDigits(13999019)* will return 0.

```
int productDigits(int number){

   if(number < 10)
     return number;
   return (number%10) * productDigits(number/10);
}
```

**Grading:**

**3 pts for base case**
**7 pts for recursive call - 1 pt return, 2 pts for getting last digit, 1 pt mult, 1 pt rec call**
**2 pts number/10**

**2)** (10 pts) ANL (Summations)

a) (5 pts) Determine the value of the following summation, in terms of *n*: $\sum_{i=1}^{2n}(4i+7)$. Express your final answer as a polynomial in the form $an^2 + bn$, where a and b are integers.

$$\sum_{i=1}^{2n}(4i+7) = \left(4\sum_{i=1}^{2n} i\right) + \sum_{i=1}^{2n} 7$$
$$= 4 \times \frac{2n(2n+1)}{2} + 7(2n)$$
$$= 2(2n)(2n+1) + 14n$$
$$= 8n^2 + 4n + 14n$$
$$= 8n^2 + 18n$$

**Grading: 1 pt for split, 2 pts for sum to i formula, 1 pt sum constant, 1 pt simplify**

b) (5 pts) Determine the value of the summation below:

$\sum_{i=21}^{100}(3i+1) = \sum_{i=1}^{100}(3i+1) - \sum_{i=1}^{20}(3i+1)$

$= \sum_{i=1}^{100}(3i) + \sum_{i=1}^{100} 1 - (\sum_{i=1}^{20}(3i) + \sum_{i=1}^{20} 1)$

$= 3 \times \frac{100\times101}{2} + 100 - (3 \times \frac{20\times21}{2} + 20)$

$= 3 \times 50 \times 101 + 100 - 3 \times 10 \times 21 - 20$

$= 3 \times 5050 + 100 - 30 \times 21 - 20$

$= 15150 + 100 - 630 - 20$

$= 14600$

**Grading (5 pts total): 1 pt for splitting sum, 3 pts for properly plugging into formulas for both sums, 1 pt for simplification**

**3)** (10 pts) ALG (Stacks)

A stack of ***positive integers*** is implemented using the struct shown below. Using this implementation of the stack write the ***push*** and ***peek*** functions. ***Assume that when a struct stack is empty, its top variable is equal to -1.***

```
#define MAX 12

struct stack{
   int top;   /* indicates index of top */
   int nodes[MAX] ;
};

// Attempts to push value onto the stack pointed to by s.
// If the stack is full 0 is returned and no action is taken.
// Otherwise, value is pushed onto the stack and 1 is returned.
int push(struct stack* s, int value){

    if(s->top >= MAX-1)
        return 0;

    s->nodes[s->top + 1] = value;
    s->top++;
    return 1;
}
```

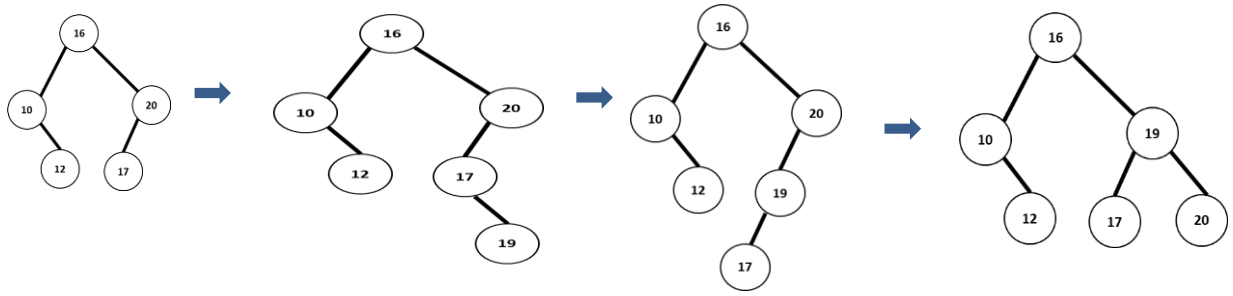
**Grading: 2 pts for full case, 2 pts for insertion, 1 pt update top, 1 pt return**

```
// Returns the value at the top of the stack. If the stack is
// empty, -1 is returned.
int peek(struct stack* s){

    if(s-> top == -1)
       return -1;

    return s->nodes[s->top];
}
```

**Grading: 2pt for empty case, 2 pts for return in regular case.**

**4)** (10 pts) ALG (Binary Search Trees and Hash Tables)

a) (5 pts) Show the AVL tree created when 19 is added to the AVL tree below




**Grading: 5 pts total - GIVE FULL CREDIT IF FINAL TREE IS CORRECT**
**1 for inserting 19 in the correct position**
**2 pt for left rotation**
**2 pt for right rotation (for rotation trace)**

b) (5 pts) In a binary heap of 100 elements, how many elements are at a depth of 6 (lowest level) from the root of the heap? (Note: the depth of an element is the number of links that have to be traversed from the root of the tree to reach it.)

A binary heap fills in each row before moving onto the next, since its structure is always a complete binary tree. Thus, the number of nodes at depths 0 through 5 are 1, 2, 4, 8, 16, and 32, respectively. This sums to 63. Thus, the next **37 nodes** will all be at a depth of 6 from the root.

**37**

**Grading: 2 pts for observation of heap structure, 2 pts for counting up the nodes at all the previous levels, 1 pt for calculating the final answer.**

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

a) Convert the hexadecimal number AF2E9 to binary without first converting to the base 10 equivalent

A F 2 E 9

10 15 2 14 9 1010 1111 0010 1110 1001

1010 1111 0010 1110 1001

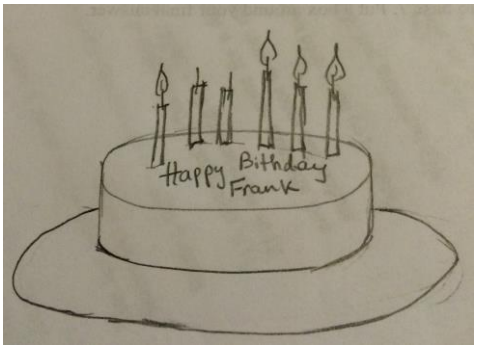
**Grading: 5 pts total - 1 pt for each group of 4 bits. All 4 bits in the group have to be correct to get the point.**

b) Frank is the team-lead for the software testing team at his job. He is celebrating his birthday. Some of his co-workers have baked a cake for the celebration and thought that it would be really cool to put candles on his cake to represent his age in binary. An unlit candle represents the 0 bit. From the pic of the cake below, how old is Max?

100111

32 4 2 1

= 32 + 4 + 2 + 1 = 39




**Grading: 5 pts total**
**4 pts : 1 for decimal for each digit**
**1 pt for final answer**

**Note: Also give full credit for 32 + 16 + 8 + 1 = 57, though most students will read left to right.**

# Computer Science Foundation Exam

## May 6, 2016

## 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 | ANL | 7 | |
| 3 | 10 | DSN | 7 | |
| 4 | 10 | DSN | 7 | |
| 5 | 10 | ALG | 7 | |
| TOTAL | 50 | | 35 | |

**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)** (10pts) ANL (Algorithm Analysis)

Determine the ***best case*** run time in terms *n* for each of the following functions/operations.

a) Finding the maximum value in an unsorted linked list of *n* elements O(n)

b) Inserting an item into a binary search tree of *n* elements O(1)

c) Inserting an item into a binary heap of *n* elements O(1)

d) Sorting an array of *n* elements using Merge Sort O(nlgn)

e) Deleting an element from a circular linked list of *n* elements O(1)

**Grading: 1 pt each, must be correct to earn the point.**

Determine the ***worst case*** run time in terms of *n* for each of the following functions/operations.

f) Deleting an item from an AVL tree of *n* elements O(lg n)

g) Deleting the minimum item from a binary min heap of *n* elements O(lg n)

h) Inserting an item into a binary search tree of *n* elements O(n)

i) Sorting an array of *n* elements using Heap Sort O(nlgn)

j) Deleting an element from a doubly linked list of *n* elements O(n)

**Grading: 1 pt each, must be correct to earn the point.**

**2)** (10 pts) ANL (Algorithm Analysis)

a) (5 pts) Given that a function has time complexity $O(n^2)$, if the function takes 338 ms for an input of size 13000, how long will the same function take for an input of size 8000?

$T(n) = cn^2$
$T(13000) = c13000^2 = 338ms$

$c = \frac{338}{13^2} \times \frac{1}{10^6} ms = \frac{338}{169} \times 10^{-6} ms = 2 \times 10^{-6} ms$

$T(8000) = c(8000)^2 = (2 \times 10^{-6} ms) \times 8^2 \times 10^6 = 128ms$

128 ms

**Grading: 2 pts for computing the constant c, 2 pts substituting 8 into general equation to find time for input size of 8000, 1 pt for simplifying the final answer.**

b) (5 pts) What is the run-time of the segment of code below, in terms of the variables *n* and *k*? Please provide a Big-Oh bound and briefly justify your answer. (Assume k has already been defined as set to a value prior to the code segment shown.)

```
int i, total = 0;
for (i=0; i<n; i+=2) {
    int start = k;
    while (start > 0) {
        total += ((k|i) & start);
        start /= 2;
    }
}
```

The outer loop runs n/2 times. The inner loop will always run O(lg k) times, since k never changes during the code segment and each iteration of the while loop divides start by 2. Since the two loops are independent of each other in terms of number of times they run, it follows that the run time of the code segment is O(nlg k).

O(nlg k)

**Grading: 2 pts outer loop analysis, 2 pts inner loop analysis, 1 pt final answer**

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

Write a function, `mode`, that takes in a pointer to the front of a linked list storing integers, and returns the mode of the list of integers. Recall that the mode of a list of values is the value that occurs most frequently. You may assume that all of the integers in the list are in between 0 and 999, inclusive. If there is more than one mode, your function must return the smallest of all of the modes. (For example, if the list contains the values 2, 4, 3, 2, 2, 4, 1, and 4, your function must return 2 and should NOT return 4, since both 2 and 4 occur three times in the list but 2 is smaller than 4.) Hint: declare an auxiliary array inside of the mode function. ***You may assume that the list pointed to by front is non-empty.***

Use the struct definition provided below.

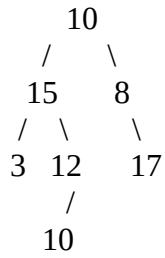
```
#include <stdlib.h>
#include <stdio.h>
#define MAX 1000

typedef struct node {
    int value;
    struct node* next;
} node;

int mode(node* front) {

    int freq[MAX], i;
    for (i=0; i<MAX; i++)
        freq[i] = 0;

    while (front != NULL) {
        freq[front->value]++;
        front = front->next;
    }

    int res = 0;
    for (i=1; i<MAX; i++)
        if (freq[i] > freq[res])
            res = i;

    return res;

}
```

**Grading: 3 pts initializing frequency array, 4 pts filling frequency array, 3 pts finding index of maximum value of frequency array. Only take off 1 pt if ties are broken incorrectly.**

**Note: Please readjust points for solution ideas different than this as necessary.**

**4)** (10 pts) DSN (Binary Trees)

For this problem you will write a modified inorder traversal of a binary tree. In a regular inorder traversal, you simply visit the nodes of the tree "in order". Consider the added task of adding up all of the numbers as you see them in the traversal, one by one, and printing out each intermediate sum. For example, if the input binary tree was:




the corresponding inorder traversal would visit 3, 15, 10, 12, 10(root), 8 and 17, in that order. For the added task, the traversal should print out 3, 18, 28, 40, 50, 58 and 75, respectively, the running sums after visiting each value.

Complete the function below, ***recursively***, so that is performs the given task. One way to accomplish this is to have the function take in a second value, `prevSum`, representing the previous sum of values prior to visiting the given node, and also to have the function return the sum of the nodes in its subtree.

```
typedef struct treenode {
    int value;
    struct treenode *left;
    struct treenode *right;
} treenode;

int inorderSum(treenode* root, int prevSum) {

    if (root == NULL) return 0;

    int sum = 0;
    sum += inorderSum(root->left, prevSum);

    sum += root->value;
    printf("%d ", sum+prevSum);

    sum += inorderSum(root->right, prevSum+sum);

    return sum;
}
```

**Grading: There are 10 blanks, give 1 pt for each blank. If two blanks are slightly wrong, you may award 1 pt for both of them together.**

**5)** (10 pts) ALG (Sorting)

For this question, implement the (very slow) sorting algorithm described below:

1. Randomly choose two distinct array indexes, i and j. (If i and j are equal, choose again.)
2. If array[min(i,j)] > array[max(i,j)], swap the two values.
3. Check if the array is sorted. If it's not, go back to step 1. If it is, return.

Recall that the function call `rand()` returns a random non-negative integer, so `rand()%n` will equal a random integer in between 0 and n-1.

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int min(int a, int b) {if (a < b) return a; return b;}
int max(int a, int b) {if (a > b) return a; return b;}

void randomSort(int* array, int length) {

    while (1) {

        int i = 0, j = 0;
        while (i == j) {
            i = rand()%length;
            j = rand()%length;
        }
        int minI = min(i,j);
        int maxI = max(i,j);

        if (array[minI] > array[maxI]) {
            int temp = array[minI];
            array[minI] = array[maxI];
            array[maxI] = temp;
        }

        int sorted = 1;
        for (i=0; i<length-1; i++)
            if (array[i] > array[i+1])
                sorted = 0;

        if (sorted) break;
    }

}
```

**Grading: 3 pts picking random indices, 3 pts performing swap, if necessary, 3 pts check at end, 1 pt stopping/breaking if the array is sorted**