Computer Science Foundation Exam

August 12, 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			

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) DSN (Recursive Functions)

Write a recursive function that will return the binary equivalent of its input parameter, `decimalNo`. You may assume that `decimalNo` is in between 0 and 1023, inclusive, thus the converted binary value will fit into an integer variable. For example, toBinary(46) should return the integer 101110 and toBinary(512) should return 1000000000.

```
int toBinary(int decimalNo) {

    if (decimalNo < 2)
        return decimalNo;

    return 10*toBinary(decimalNo/2) + decimalNo%2;

}
```

Grading: 2 pts base case (1 pt for 0 case, 1 pt for 1 case)
8 pts rest - 1 pt return
2 pts mult 10
1 pt toBinary call
2 pt decimalNo/2 in recursive call
2 pts adding decimalNo%2

2) (10 pts) ANL (Summations and Algorithm Analysis)

Find the closed form solution in terms of n for the following summation. Be sure to show all your work.

$$\sum_{i=n}^{3n}\sum_{j=1}^{n-2} j$$

$$= \sum_{i=n}^{3n} \frac{(n-2)(n-1)}{2}$$

$$= \frac{(3n-n+1)(n-2)(n-1)}{2}$$

$$= \frac{(2n+1)(n-2)(n-1)}{2}$$

In the second step, we are summing a constant with respect to the summation index i, thus we can simply multiply the item being summed by the number of times it's summed.

Note, also accepted is the polynomial multiplied out:

$$= \frac{2n^3 - 5n^2 + n + 2}{2}$$

Grading: 5 pts for solving the inner summation, 5 pts for then solving the outer summation, grader decides partial credit within each part.

3) (10 pts) ALG (Stacks and Queues)

Consider the process of merging two queues, `q1` and `q2`, into one queue. One way to manage this process fairly is to take the first item in `q1`, then the first item from `q2`, and continue alternating from the two queues until one of the queues run out, followed by taking all of the items from the queue that has yet to run out in the original order. For example, if `q1` contains 3, 8, 2, 7 and 5, and `q2` contains 6, 11, 9, 1, 4 and 10, then merging the two queues would create a queue with the following items in this order: 3, 6, 8, 11, 2, 9, 7, 1, 5, 4, and 10. Assume that the following struct definitions and functions with the signatures shown below already exist.

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

typedef struct queue {
    node* front;
    node* back;
} queue;


// Initializes the queue pointed to by myQ to be an empty queue.
void initialize(queue* myQ);

// Enqueues the node pointed to by item into the queue pointed
// to by myQ.
void enqueue(queue* myQ, node* item);

// Removes and returns the front node stored in the queue
// pointed to by myQ. Returns NULL if myQ is empty.
node* dequeue(queue* myQ);

// Returns the number of items in the queue pointed to by myQ.
int size(queue* myQ);
```

On the following page, write a function that takes in two queues, `q1` and `q2`, and merges these into a single queue, emptying out `q1` and `q2` in the process and returning a pointer to the resulting queue.

```
queue* merge(queue* q1, queue* q2) {

    queue* res = malloc(sizeof(queue));
    initialize(res);

    int list = 0;

    while (size(q1) > 0 || size(q2) > 0) {

        if (list == 0 && size(q1) > 0)
            enqueue(res, dequeue(q1));
        else if (list == 1 && size(q2) > 0)
            enqueue(res, dequeue(q2));

        list = (list+1)%2;
    }

    return res;
}
```

Grading: 2 pts for properly creating an empty list to return (1 pt malloc, 1 pt init)
2 pts for toggling mechanism between lists
2 pts for handling unequal list sizes somehow
1 pt dequeing
2 pts enqueing item into their created queue
1 pt return

4) (10 pts) ALG (Hash Tables)

Insert the following numbers (in the order that they are shown…..from left to right) into a hash table with an array of size 12, using the hash function, H(x) = x mod 12.

234, 344, 481, 567, 893, 178, 719, 686, 46, 84

Show the result of the insertions when hash collisions are resolved through

a) linear probing

b) quadratic probing

c) separate chaining

Index	**a** **Linear**	**b** **Quadratic**	**c** **Separate chaining**
0	46	84	84
1	481	481	481
2	686	686	
3	567	567	567
4	84		686
5	893	893	893
6	234	234	234
7		46	
8	344	344	344
9			
10	178	178	178->46
11	719	719	719

Grading: 3 pts total for Linear Probing, 3 pts for Separate Chaining Hashing, 4 pts for Quadratic Probing. Give full credit if all the answers are correct, 2/3 or 3/4 is most of the answers are correct, 1/3 or 1/4 if some answers but no more than half are correct, 0/3 or 0/4 if none of the answers in a column are correct.

5) (10 pts) ALG (Base Conversion)

(a) (5 pts) Convert $FAD8_{16}$ to octal.

Convert to binary
1111 1010 1101 1000

Realign bits (implied leading zeros for single 1 at beginning)

001 111 101 011 011 000

Translate to octal

$1\ 7\ 5\ 3\ 3\ 0_{8}$

Grading: 2 pts converting bits to binary, 2 pts realigning bits, 1 pt converting to octal

(b) (5 pts) Convert 2120_{10} to hexadecimal.

16 | 2120
16 | 132 R 8
16 | 8 R 4
16 | 0 R 8

$8\ 4\ 8_{16}$

Grading: 1 pt for each quotient and remainder (except 0), if solved differently, grade accordingly.

Computer Science Foundation Exam

August 12, 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	ALG	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 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) ALG (Analysis and Critical Thinking: AVL Trees, Hash Tables, and Heaps)

a) (1 pt) Using big-oh notation, what is the **best-case** runtime for inserting an integer into an AVL tree that contains *n* integers?

O(log n) Grading: All or nothing.

b) (1 pt) Using big-oh notation, what is the **worst-case** runtime for inserting an integer into an AVL tree that contains *n* integers?

O(log n) Grading: All or nothing.

c) (2 pts) What is the worst-case runtime for insertion into a hash table with *n* elements, assuming we use quadratic probing to resolve collisions? (You may assume that our hash table satisfies all conditions necessary to ensure that quadratic probing won't get stuck in an infinite loop.)

O(n) Grading: All or nothing.

d) (2 pts) Given the following hash table, suppose we know that no strings have been deleted, but we don't know the order in which these three strings were inserted into the hash table. If we used linear probing to resolve collisions, what are all the possible hash values for the string "of" (assuming those hash values are modded by the table size)?

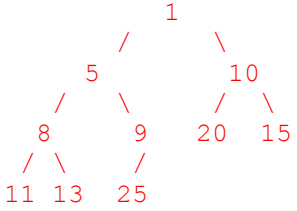
		pied	piper	of		Hamelin
0	1	2	3	4	5	6

Possible hash values: 2, 3, or 4
Grading: Give one point for each correct index given, and subtract half a point for each incorrect index given. Then take the floor.

e) (2 pts) Using big-oh notation, what is the **worst-case** runtime for deletion from a minheap that contains *n* elements?

O(log n) Grading: All or nothing.

f) (2 pts) Draw a minheap that contains 10 elements and which will incur the worst-case runtime if we call deleteMin() on it.



Note: Actual values may vary from answer to answer, but it's important that the percolate down operation takes the value to the left of the root, then left again, and then it can go left or right (doesn't matter).

Grading: 1 pt for a valid minheap, 1 pt for a minheap where the node in 25's position ends up where 11 or 13 are.

2) (10 pts) ANL (Summations and Algorithm Analysis)

a) (8 pts) Give a summation that represents the value returned by the following function, and then derive its closed form:

```
int something_to_ponder_over(unsigned int n)
{
   int i, retval = 0, pow = 1;
   for (i = 0; i < n; i++)
   {
      retval += pow;
      pow *= 14;
   }
   return retval;
}
```

The summation representing the return value is:

$$\sum_{i=0}^{n-1} 14^i$$

This is a geometric sum and can be solved as follows (or one might have the formula committed to memory, or get the formula from the formula sheet):

$$S = \sum_{i=0}^{n-1} 14^i = 14^0 + 14^1 + 14^2 + \cdots + 14^{n-1}$$

$$14S = 14^1 + 14^2 + 14^3 + \cdots + 14^n + 14^n$$

Now, subtracting S from 2S, most of the terms cancel out, leaving us with:

$$14S - S = 14^n - 14^0$$

Since 14S – S = 13S, we've solved for the summation:

$$S = (14^n - 1)/13$$

Grading: Award 5 points for the initial summation (2 pts for the correct bounds, 3 pts for the 14^i term). Award 3 points for deriving the closed form. Award partial credit as appropriate.

b) (2 pts) Using big-oh notation, what is the runtime of the function given in part (a)?

O(n) Grading: All or nothing

3) (10pts) DSN (Linked Lists)

Write a recursive function that takes the head of a linked list (possibly NULL) that contains positive integers only. The function must return -1 if the list contains any integer that is equal to the sum of all integers that come after it in the list. If not, the function can return whatever value you feel is appropriate other than -1. (Figuring out what to return is part of the fun for this problem.)

For example, the function should return -1 for the following linked list because 4 is the sum of all the nodes that follow it (1, 2, and 1):

```
20 -> 3 -> 1 -> 4 -> 1 -> 2 -> 1 -> NULL

^
head
```

The function signature and node struct are:

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

int listylist(node *head) {

   int sum;

   if (head == NULL)
      return 0;

   sum = listylist(head->next);

   if (sum == -1 || head->data == sum)
      return -1;

   return head->data + sum;
}
```

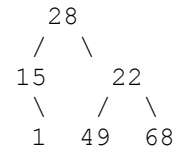
Grading:
2 pts for the base case (which should return 0 to work effectively)
2 pts for returning -1 if the recursive call itself returned -1
2 pts for returning -1 if head->data is equal to the sum from the recursive call
2 pts for returning a valid sum when not returning -1
2 pts for correct syntax and for avoiding segmentation faults

Note: There might be other solutions to this problem. Please award partial credit as necessary for alternate solutions.

4) (10 pts) DSN (Binary Trees)

Write a ***recursive*** function that takes the root of a binary tree (possibly NULL) and returns the sum of all the nodes that are left children in the tree. (See the example below, which returns 15 + 49 = 64, since the only nodes that are left children anywhere in the tree are 15 and 49.)

For this tree, the function should return 15 + 49 = 64:



The node struct and function signature are:

```
typedef struct node {
    int data;
    struct node *left;
    struct node *right;
} node;

int add_all_left_children(node *root) {

    if (root == NULL) return 0;

    int sum = 0;
    if (root->left != NULL)
        sum += root->left->data;

    return sum + add_all_left_children(root->left) +
                 add_all_left_children(root->right);

}
```

Grading: 2 pts base case, 1 pt NULL check left, 2 pts adding left node if it exists, 5 pts for the final return (1 pt for previous sum, 2 pts for each recursive call)

5) (10 pts) ALG (Sorting)

a) (3 pts) The following diagram shows an initial array, followed by what the array looks like after a single pass of some sorting algorithm. Indicate what sorting algorithm is being applied, and give that algorithm's worst-case runtime (using big-oh notation).

22	49	36	22	17	18	4

4	49	36	22	17	18	22

Sorting algorithm being applied: Selection Sort Grading: 2 pt, all or nothing

Worst-case runtime for algorithm: $O(n^2)$ Grading: 1 pt, all or nothing

b) (3 pts) For the following arrays, follow the same instructions from part (a):

84	19	23	66	91	44	42

19	23	66	84	44	42	91

Sorting algorithm being applied: Bubble Sort Grading: 2 pt, all or nothing

Worst-case runtime for algorithm: $O(n^2)$ Grading: 1 pt, all or nothing

c) (4 pts) Give a recurrence relation that represents the runtime for Merge Sort of n items. Let T(n) represent the runtime of Merge Sort of n items in setting up your recurrence relation.

$T(0) = T(1) = c_1$
$T(n) = 2T(n/2) + c_2*n + c_3$ (for $n > 1$)
Alternatively: $T(n) = 2T(n/2) + O(n)$ (for $n > 1$)

Grading: 2 pts for $2T(n/2)$
2 pts for $+ O(n)$ or similar