Computer Science Foundation Exam

August 26, 2017

Section I A

DATA STRUCTURES

SOLUTIONS

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

Question #	Max Pts	Category	Passing	Score
1	5	DSN	3	
2	10	DSN	7	
3	10	ALG	7	
TOTAL	25		17	

You must do all 3 problems in this section of the exam.

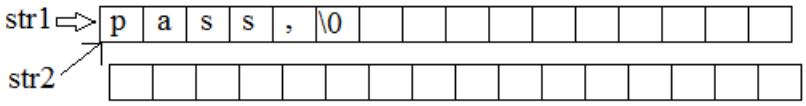
Problems will be graded based on the completeness of the solution steps and <u>not</u> graded based on the answer alone. Credit cannot be given unless all work is shown and is readable. Be complete, yet concise, and above all <u>be neat</u>. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (5 pts) DSN (Dynamic Memory Management in C)

There is something terribly wrong with the code given below: it has two memory leaks. After carefully inspecting the code, answer the questions below.

```
1:    int main(void)
2:    {
3:      char *str1 = malloc(sizeof(char) * 16);
4:      char *str2 = malloc(sizeof(char) * 16);
5:
6:      str1[0] = 'p';
7:      str1[1] = 'a';
8:      str1[2] = 's';
9:      str1[3] = 's';
10:     str1[4] = ',';
11:     str1[5] = '\0';
12:
13:     printf("%s ", str1);
14:     str2 = str1;
15:     printf("%s ", str2);
16:     str2 = NULL;
17:     strcpy(str1, "pass the exam!");
18:     printf("%s\n", str1);
19:
20:     free(str1);
21:     free(str2);
22:
23:     return 0;
24:   }
```

(a) (3 pts) Draw a picture that indicates the relevant state of memory ***after*** line 14 has completed. (Draw a rectangular box to indicate dynamically allocated memory.)



Note: All cells left empty represent uninitialized character variables.

Grading: 1 pt for having two boxes drawn indicating allocated memory. 1 pt for having str1 point to the box that stores "pass," (this must be indicated), 1 pt for having str2 point to this same box.

(b) (1 pt) Explain why line 14 causes a memory leak.

When the pointer str2 moves, nothing is pointing to the memory that it used to be pointing to originally.

(c) (1 pt) Why is it possible for the code to crash on line 21?

str2 is pointing to NULL (nothing), so it's not pointing to dynamically allocated memory. Attempting to free memory that isn't dynamically allocated may crash a program.

Grading parts (b) and (c): Give the point for each if the answer is reasonably close or shows that the student understands the key issue at hand. **No half points! Only award an integer number of points.**

2) (10 pts) DSN (Linked Lists)

Write a **recursive** function that takes in the head of a linked list and frees all dynamically allocated memory associated with that list. You may assume that **all** the nodes in any linked list passed to your function (including the head node) have been dynamically allocated. It's possible that your function might receive an empty linked list (i.e., a NULL pointer), and you should handle that case appropriately.

Note that your function must be recursive in order to be eligible for credit.

The linked list node struct and the function signature are as follows:

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

void destroy_list(node *head) {
     if (head == NULL)
          return;

     destroy_list(head->next);
     free(head);
}
```

Grading: Award 3 pts for correctly handling the base case. Award 3 points for a correct recursive call. Award 4 points for freeing *head* after the recursive call.

Note: Some students might set head->next to NULL before freeing *head*., which causes some unnecessary computation, but doesn't render their solution incorrect. You can ignore that.

However, if they set *head* itself to NULL before freeing it, they should not be eligible to receive the 4 points for freeing *head* after the recursive call.

3) (10 pts) ALG (Stacks) Suppose we pass the string "cupcake" to the following function. What will the function's output be, and what will the stacks *s1* and *s2* look like when the function terminates? You may assume the stack functions are written correctly and that the stacks are designed for holding characters.

```
void string_shenanigans(char *str)
{
   int i, len = strlen(str);
   char *new_string = malloc(sizeof(char) * (len + 1));
   Stack s1, s2;
   init(&s1);
   init(&s2);

   for (i = 0; i < len; i++) {
      push(&s1, str[i]);  // this pushes onto stack s1
      push(&s2, str[i]);  // this pushes onto stack s2
   }

   for (i = 0; i < len; i++) {
      if (i % 2 == 0) {
         // Note: pop() returns the character being removed from the stack.
         if (!isEmpty(&s1))
            new_string[i] = pop(&s1);
         if (!isEmpty(&s1))
            push(&s2, pop(&s1));
      }
      else {
         pop(&s2);
         new_string[i] = pop(&s2);
      }
   }

   new_string[len] = '\0';
   printf("%s\n", new_string);
   free(new_string);
}
```

eeakpac	**(the stack is empty)**	**c <- top** **p** **u** **c**
printf() output	final contents of *s1* (please label 'top' for clarity)	final contents of *s2* (please label 'top' for clarity)

Grading: Award 7 points for the correct output ("eeakpac"), 1 pt per letter, 1 pt for the correct contents of *s1*, and 2 pts for the correct contents of *s2*. (Give 1 pt if the stack is flipped.) If the output looks reasonably close and has a minor error, feel free to award partial credit.

Computer Science Foundation Exam

August 26, 2017

Section I B

DATA STRUCTURES

SOLUTIONS

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

Question #	Max Pts	Category	Passing	Score
1	10	ALG	7	
2	5	ALG	3	
3	10	DSN	7	
TOTAL	25		17	

You must do all 3 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. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (10 pts) ALG (Binary Search Trees)

(a) (6 pts) Following are three traversals produced by the exact same **binary search tree**. Using your powers of inference, determine which one is which. (Fill in each blank with "in-order," "pre-order," "post-order.") ***(Note: All sets of answers which don't contain each traversal exactly once will automatically be awarded 0 points.)***

preorder traversal: 18 14 12 9 31 24 19 22 23 36

post-order traversal: 9 12 14 23 22 19 24 36 31 18

in-order traversal: 9 12 14 18 19 22 23 24 31 36

Grading: Each answer above is worth two points. However, if they try to game the system by putting the same answer in all three blanks, award zero points.

(b) (1 pt) What value must be at the root of the BST that produced the traversals listed in part (a)?

18

(c) Using big-oh notation, what is the best-case runtime for searching for a particular value in a BST with *n* nodes?

O(1)

(d) Using big-oh notation, what is the worst-case runtime for searching for a particular value in a binary tree (a regular old binary tree, not necessarily a BST) with *n* nodes?

O(n)

(e) Using big-oh notation, what is the worst-case runtime for searching for a particular value in a BST with *n* nodes?

O(n)

Grading: For parts (b) through (e), each answer is worth 1 pt and is either right or wrong.

2) (5 pts) ALG (Hash Tables)

Use the following hash function to insert the given elements into the hash table below. Use **quadratic probing** to resolve any collisions. You may assume that the correct table size (in this case, 10) is always passed to the function with the key that is being hashed.

```
int hash(int key, int table_size)
{
      int a = (key % 100) / 10;
      int b = key % 10;

      return (a + b) % table_size;
}
```

Keys to insert (one by one, in the following order): 2555, 1523, 5893, 800, 956

0	1	2	3	4	5	6	7	8	9
2555	**800***	**5893**			**1523**		**956***		

Grading: Award 1 pt each for each value being in the correct location.

Note: An asterisk indicates that a value encountered at least one collision before arriving at that position. Those asterisks are included simply for the grader's convenience and are not part of the actual solution.

3) (10 pts) DSN (Tries)

Write a recursive function that takes the root of a trie, *root*, and a single character, *alpha*, and returns the number of strings in the trie that contain that letter. You may assume that letter is a valid lowercase alphabetic character ('a' through 'z'). At some point, you will probably find it enormously helpful to write a helper function to assist with part of this task.

Note that we are *not* simply counting how many times a particular letter is represented in the trie. For example, if the trie contains only the strings "apple," "avocado," and "persimmon," then the following function calls should return the values indicated:

```
countStringsWithLetter(root, 'p') = 2
countStringsWithLetter(root, 'm') = 1
```

The TrieNode struct and function signature are as follows:

```
typedef struct TrieNode {
    struct TrieNode *children[26];
    int numwords;
    int flag; // 1 if the string is in the trie, 0 otherwise
} TrieNode;

int countStringsWithLetter(TrieNode *root, char alpha) {
     int i, total = 0;
     if (root == NULL) return 0;

     for (i = 0; i < 26; i++) {
          if (i == alpha - 'a') {
              if (root->children[i] != NULL)
                  total += (root->children[i])->numwords;
          }
          else
              total += countStringsWithLetter(root->children[i], alpha);
     }
     return total;
}
```

Grading:

+1 for the base case
+1 for initializing a counter (such as *total*) to 0
+2 for the loop to go through all 26 letters.
+1 for using (alpha – 'a') or (alpha – 97) in their comparison
+2 for adding numwords to total in the case of the letter match ***(be nice and don't take off if they miss the NULL check for root->children[i])***
+3 for adding in the recursive call in the case that the letters don't match. (1 pt for recursive call, 1 pt for total +=, 1 pt for parameter root->children[i], don't take off if they forget the parameter alpha)

Computer Science Foundation Exam

August 26, 2017

Section II A

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Passing	Score
1	10	ANL	7	
2	5	ANL	3	
3	10	ANL	7	
TOTAL	25		17	

You must do all 3 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. For each coding question, assume that all of the necessary includes (stdlib, stdio, math, string) for that particular question have been made.

1) (10 pts) ANL (Algorithm Analysis)

Consider the problem of taking n sorted lists of n integers each, and combining those lists into a single sorted list. For ease of analysis, assume that n is a perfect power of 2. Here are two potential algorithms to solve the problem:

Algorithm A: Run the Merge Algorithm, defined between two lists, on lists 1 and 2 to create a single merged list. Then rerun the algorithm on this merged list and list 3, creating a merged list of all items from lists 1, 2 and 3. Continue in this fashion, running the Merge Algorithm n-1 times, always between the currently "growing" list and the next list to be merged into it, until list n is merged in, creating a single sorted list.

Algorithm B: Pair up the lists into $\frac{n}{2}$ pairs of lists of size n. Run the Merge Algorithm on each of these pairs. Once this phase finishes, there will be $\frac{n}{2}$ lists with 2n integers. With the new lists, repeat the process until we are left with a single sorted list.

With sufficient work and proof, determine the Big-Oh run time, in terms of n, of both of these algorithms. Clearly put a box around your final answer for both algorithms.

Analysis of Algorithm A

In this algorithm, we run the Merge Algorithm n-1 times. The first time, the list sizes are n and n. The second time the list sizes are 2n and n. In general, on the k^{th} time the list sizes are kn and n. When merging lists of these two sizes, the run-time is O(kn+n) = O(n(k+1)) ≤ cn(k+1) for some constant c. Now, we must sum up these run-times in the range of k = 1 to k = n-1. Mathematically, we have

$$\sum_{k=1}^{n-1} cn(k+1) = cn\sum_{k=1}^{n-1}(k+1) = cn\sum_{k=2}^{n} k = cn\left(\frac{n(n+1)}{2} - 1\right) = \frac{cn^3 + cn^2 - 2cn}{2} = \boldsymbol{O(n^3)}$$

Analysis for Algorithm B

In the first iteration, we run n/2 merges on lists of size n. The total run time for this iteration is $O(n^2)$, since each of the elements in all lists is involved in a single merge. (Alternatively, we have n/2 merges, each of which take O(n) time to run, for a total of $O(n^2)$.)

In the next iteration, we run n/4 merges on lists of size 2n. For the same reason as before, this step also takes $O(n^2)$ time.

The question becomes: How many iterations do we have to run before we get a single list? Notice that after the k^{th} iteration of merging lists, we have a total of $\frac{n}{2^k}$ lists left. We stop when we have a single list, so to calculate the number of iterations, set $\frac{n}{2^k} = 1$. When we solve this equation for k, we get $k = log_2 n$.

It follows that the total run time of Algorithm A is **$O(n^2 \lg n)$.**

Grading: Alg A - 1 pt for set up of summation, 3 pts for doing the summation (be lax here), 1 pt for the final Big-Oh result. Alg B - 2 pts for the initial analysis, 2 pts for calculating the number of iterations, 1 pt for the final Big-Oh result.

2) (5 pts) ANL (Algorithm Analysis)

An algorithm processing an array of size n runs in $O(n\sqrt{n})$ time. For an array of size 10,000 the algorithm processes the array in 16 ms. How long would it be expected for the algorithm to take when processing an array of size 160,000? Please express your answer in ***seconds,*** writing out exactly three digits past the decimal.

Let the algorithm with input size n have a runtime of $T(n) = cn\sqrt{n}$, for some constant c. Using the given information we have:

$$T(10000) = c(10000)\sqrt{10000} = 16ms$$
$$c(10000)(100) = 16ms$$
$$c = \frac{16}{10^6}ms$$

Now, we must find T(160000);

$$T(160000) = c(160000)\sqrt{160000} = \frac{16\ ms}{10^6} \times 16 \times 10^4 \times 4 \times 10^2 = 16 \times 16 \times 4ms = 1024ms$$

Converted to seconds, our final answer is **1.024 seconds.**

Grading: 1 pt to set up the initial equation for c, 1 pt to solve for c, 2 pts to get answer in ms, 1 pt to convert to seconds. Give partial credit for the 2 pts if the setup is correct but some algebra issue occurred. (For example, an answer of 1.536 seconds would get 4 pts probably, since a conversion was done to seconds but some arithmetic error occurred. An answer of 1536 ms would get 3 pts, 1 off for an arithmetic error and 1 off for no conversion to seconds.)

3) (10 pts) ANL (Summations and Recurrence Relations)

Let *a*, *b*, *c*, and *d*, be positive integer constants with $a < b$. ***Without using the arithmetic sum formula***, prove that

$$\sum_{i=a}^{b}(ci+d) = \frac{(c(a+b)+2d)(b-a+1)}{2}$$

$$\sum_{i=a}^{b}(ci+d) = \sum_{i=1}^{b} ci - \sum_{i=1}^{a-1} ci + \sum_{i=a}^{b} d$$
$$= \frac{cb(b+1)}{2} - \frac{c(a-1)a}{2} + (b-a+1)d$$
$$= \frac{c(b^2+b-(a^2-a))}{2} + \frac{2d(b-a+1)}{2}$$
$$= \frac{c(b^2+b-a^2+a)}{2} + \frac{2d(b-a+1)}{2}$$
$$= \frac{c(b^2-a^2+b+a)}{2} + \frac{2d(b-a+1)}{2}$$
$$= \frac{c((b-a)(b+a)+(b+a))}{2} + \frac{2d(b-a+1)}{2}$$
$$= \frac{c(b+a)(b-a+1)}{2} + \frac{2d(b-a+1)}{2}$$
$$= \frac{(c(a+b)+2d)(b-a+1)}{2}$$

The key to this proof is setting up the usual summation formulas and then recognizing that b^2 - a^2 factors. Once this term is factorized, it can be recognized that (a+b) is a factor in the first term that can be pulled out. This, in turn, reveals (b - a + 1) to be a factor between the two larger terms, something that could have been anticipated since (b - a + 1) is a factor in the result on the RHS and also an immediate factor in the last term after the initial algebra.

Grading: 2 pts to split up summations, 2 pts for evaluating sums to i, 1 pt for evaluating all constant sums, 5 pts for simplification to the RHS. Note - the grader must be very careful to give partial credit on the last five points. Indeed, a bulk of the work of this problem is in this algebra that can be done in many different ways. Give partial based on how accurate and what percentage of the total algebra was carried out. Any correct answer gets full credit. Any response with a single isolated error should just get 1 point off. An answer that is about halfway there (factors out c, tries to do something with the squares) should get 2 or 3 of the 5 points based on discretion.

Computer Science Foundation Exam

August 26, 2017

Section II B

ALGORITHMS AND ANALYSIS TOOLS

SOLUTION

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

Question #	Max Pts	Category	Passing	Score
1	10	DSN	7	
2	5	ALG	3	
3	10	DSN	7	
TOTAL	25		17	

You must do all 3 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 Coding)

Define the weighted sum of an integer array a[0], a[1], ..., a[n-1] to be $\sum_{i=1}^{n}(ia[i-1])$. For example, the weighted sum of the array [7, 5, 8] would be $1 \times 7 + 2 \times 5 + 3 \times 8 = 41$. Write a ***recursive*** function that takes in an array `numbers` and its length `n`, and returns its weighted sum. You may assume that there will be no issues with integer overflow and that **`n` is non-negative.**

```
int weightedSum(int numbers[], int n) {

    if (n == 0) return 0;

    return n*numbers[n-1] + weightedSum(numbers, n-1);

}
```

Grading: 3 pts for base case, give 2 pts if they use n = 1,
1 pt return
3 pts for calculating term n*numbers[n-1]
1 pt rec call
1 pt param numbers
1 pt param n-1

2) (5 pts) ALG (Sorting)

Show the contents of the following array after each iteration of Insertion Sort. The result after the last iteration has been included. (Note: due to the nature of this question, relatively little partial credit will be awarded for incorrect answers.)

index	0	1	2	3	4	5	6
Initial	17	22	16	5	18	14	2
1st iter	17	22	16	5	18	14	2
2nd iter	16	17	22	5	18	14	2
3rd iter	5	16	17	22	18	14	2
4th iter	5	16	17	18	22	14	2
5th iter	5	14	16	17	18	22	2
6th iter	2	5	14	16	17	18	22

Grading: 1 pt per row, row has to be perfect to earn the point.

3) (10 pts) DSN (Bitwise operators)

Two useful utility functions when dealing with integers in their binary representation are

(a) `int lowestOneBit(int n)` - returns the value of the lowest bit set to 1 in the binary representation of n. (eg. `lowestOneBit(12)` returns 4, `lowestOneBit(80)` returns 16.)

(b) `int highestOneBit(int n)` - returns the value of the highest bit set to 1 in the binary representation of n. (eg. `highestOneBit(12)` returns 8, `highestOneBit(80)` returns 64.) **Note: You may assume that the input is less than 10^9. The largest positive bit value in an integer is equal to $2^{30} > 10^9$.**

The pre-condition for the first function is that n must be a positive integer. The pre-condition for the second function is that n must be a positive integer less than 10^9. Write both of these functions in the space below. To earn full credit, you ***must*** use bitwise operators when appropriate. (Namely, there are ways to solve this question without using bitwise operators, but these solutions will NOT receive full credit.)

```
int lowestOneBit(int n) {

    int res = 1;
    while ((res & n) == 0)
        res = res << 1;
    return res;

}

int highestOneBit(int n) {

    int res = 1;
    while (((res<<1) <= n) && (res<<1) > 0)
        res = res << 1;
    return res;

}
```

Grading:

lowestOneBit - 2 pts for selecting a single bit at a time somehow, 2 pts for going in order from lowest to highest, 1 pt for returning the correct answer.

highestOneBit - 2 pts for selecting a single bit, 2 pts for a valid method to find the most significant one, 1 pt for returning the correct answer. Note - there is no need to check if (res<<1) > 0, this is unnecessary if $n < 10^9$.

Note: There are other ways of writing these functions. Please carefully trace through all student solutions and map points from the criteria above as best as possible.