Computer Science Foundation Exam

August 9, 2013

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 (Recursion)

a) (7 pts) Write an **recursive** function that takes a **positive** decimal integer, *n*, and prints it in base *b*. (*b* is also an argument to the function, and you are guaranteed your function will only be called with $1 < b < 10$.) The function signature is:

```
void basePrint(unsigned int n, int b) {

     // 2 points for a base case. Note that the original
     // call is guaranteed not to have n == 0.
     if (n <= 0)
          return;

     // 3 points
     basePrint(n / b, b);

     // 2 points
     printf("%d", n % b);

}


// Note: Another valid base case is n < b.
```

b) (3 pts) What is the big-oh runtime of your function? (Give a closed-form answer, not a recurrence relation.) *Briefly* justify your answer.

Two possible answers:

O(log n), because the base *b* representation of *n* has approximately $\log_b(n)$ digits to be printed. – *or* – O(1), because we will print the most digits with the smallest base (base 2), and we will print at most 32 bits (a constant) in that case with this function.

1 point for a correct answer, 2 points for the explanation.

2) (10 pts) ANL (Summations)

a) (2 pts) Write a summation that represent the value of *x* returned by this function:

```
int foo(int n)
{
   int i, x = 0;

   for (i = n - 10; i <= n; i++)
      x = x + i;

   return x;
}
```

Solution: $\sum_{i=n-10}^{n} i$

// 1 point for the limits
// 1 point for taking the sum of i (instead of the sum of 1)

b) (6 pts) Determine a **simplified**, closed-form solution for your summation from part (a), in terms of *n*. **You MUST show your work.**

Solution: $\sum_{i=n-10}^{n} i = \sum_{i=1}^{n} i - \sum_{i=1}^{n-11} i$

// notice (n – 10 – 1) = (n – 11)
// 2 points for breaking the summation into two parts correctly

= [n (n + 1) / 2] - [(n – 11)(n – 10) / 2] **// 3 points for applying the rule for**
// $\sum_{i=1}^{n} i = n(n+1)/2$ to get this
// closed form.

= [(n^2 + n) / 2] - [(n^2 -21n +110) / 2]
= [(22n - 110) / 2]
= 11n – 55 **// 1 point for simplification**

Note: Try to give as much partial credit as possible for the steps they can do, even if they start with the wrong sum and/or the limits of their sums are off by one. (Note: Their solution can also just sum the corresponding arithmetic series.)

c) (2 pts) Using big-oh notation, what is the runtime of the *foo()* function from part (a)? *Briefly* justify your answer.

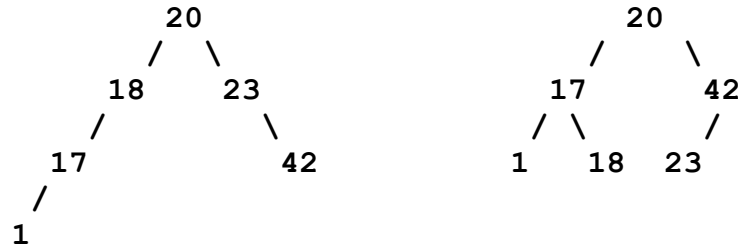
O(1), because we enter into the for-loop exactly 10 times (*i* goes from (*n* – 10) to *n*) no matter what *n* is. (1 point for O(1), and 1 point for attempting to justify whatever order they gave.)

3) (10 pts) ALG (Binary Search Trees)

a) (5 pts) Draw a binary **search** tree that will yield the following **in-order** traversal:

1, 17, 18, 20, 23, 42

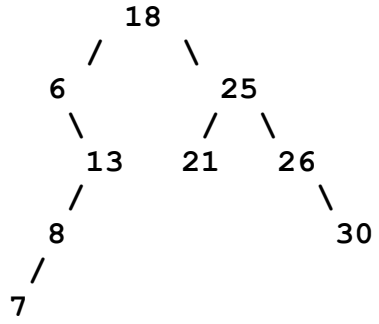
There are many equally fantastic solutions. Here are two of them:



+2 if it's a BST
+3 if it has the desired in-order traversal
Award partial credit if their traversal is close but not perfect.

b) (5 pts) Draw a binary **search** tree whose **post-order** traversal will be:

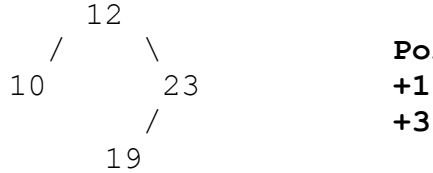
7, 8, 13, 6, 21, 30, 26, 25, 18



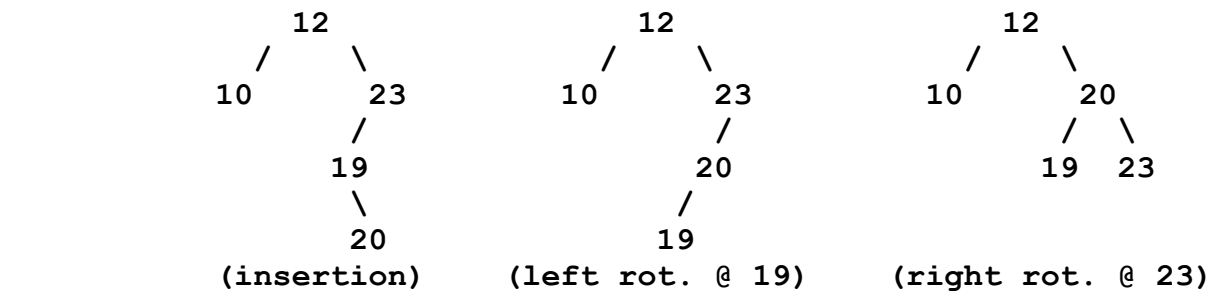
+2 if it's a BST
+3 if it has the desired in-order traversal
Award partial credit if the traversal is close but not perfect.

4) (10 pts) ALG (AVL Trees)

a) (4 pts) Insert the value 20 into the following AVL tree. Clearly show where the node is inserted initially and what the tree looks like after **each** rotation that takes place (if any).

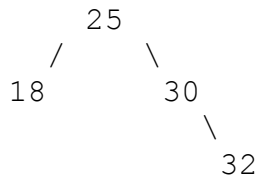


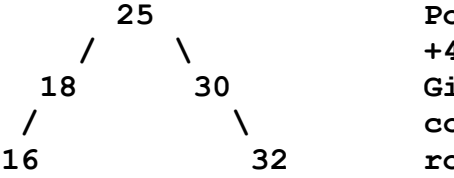
Points:
+1 for inserting in correct position
+3 for correct double rotation (give partial credit if applicable)



Note: Please assign full credit if the final tree is correct.

b) (4 pts) Insert the value 16 into the following AVL tree. Clearly show where the node is inserted initially and what the tree looks like after **each** rotation that takes place (if any).





Points:
+4 for a correct answer
Give 1 point if they insert in the correct position but then perform a rotation.

c) (2 pts) What is the benefit of keeping our AVL trees balanced? (I.e., what advantage do AVL trees have over binary search trees?)

We guarantee *worst-case* insertion, delete, and search (retrieval) runtimes of O(log n), compared to the worst-case runtimes of O(n) for the same operations with BSTs. BSTs only sport *average-case* O(log n) operations.

5) (10 pts) ALG (Hash Tables)

Insert the following elements into the hash tables below using the two collision resolution algorithms specified. The tables are both of size 10, so assume that to determine the hash function for a given input *key*, we calculate hash(*key, 10)*.

```
int hash(int key, int size) {
   return ((key - 31) * 2)%size;
}
```

Keys to insert: 37, 61, 39, 46, 87, 92

a) (5 pts) Insert the elements into this hash table using **linear probing** to resolve collisions:

0	1	2	3	4	5	6	7	8	9
61	46	37	87	92		39			

```
hash(37) = 12; 12 % 10 = 2; insert at position 2
hash(61) = 60; 60 % 10 = 0; insert at position 0
hash(39) = 16; 16 % 10 = 6; insert at position 6
hash(46) = 30; 30 % 10 = 0; insert at position ~~0~~ 1
hash(87) = 112; 112 % 10 = 2; insert at position ~~2~~ 3
hash(92) = 122; 122 % 10 = 2; insert at position ~~2~~ ~~3~~ 4
```

Points: **+2 points for 37, 61, and 39 getting into the correct positions, and +1 point each for 46, 87, and 92 getting into the correct positions.**

b) (5 pts) Insert the elements into this hash table using **quadratic probing** to resolve collisions:

0	1	2	3	4	5	6	7	8	9
61	46	37	87			39		92	

```
hash(37) = 12; 12 % 10 = 2; insert at position 2
hash(61) = 60; 60 % 10 = 0; insert at position 0
hash(39) = 16; 16 % 10 = 6; insert at position 6
hash(46) = 30; 30 % 10 = 0; insert at position ~~0~~ 1
hash(87) = 112; 112 % 10 = 2; insert at position ~~2~~ 3
hash(92) = 122; 122 % 10 = 2; insert at position ~~2~~ ~~3~~ ~~6~~ ~~1~~ 8
```

Points: **Same as part (a).**