Final Exam

Eng 4892 Data Structures. © Theo Norvell.

Aug 8, 1998

Total marks: 145

Name:

Student #:

Answer each question in the space provided or on the last page with a clear indication of what question you are answering.

Answer questions of complexity using the big-Oh notation. Indicate the smallest complexity class that is correct. E.g. don't answer $O(N^3)$ if $O(\log N)$ is also correct.

Linked lists are represented by pointers to nodes, with the 0 (NULL) pointer marking the end of the list. Nodes are defined as:

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

Trees are represented by pointers to TreeNodes, with the empty tree being represented by the 0 (NULL) pointer. TreeNodes are defined as:

```
struct TreeNode { int label ; TreeNode *left, *right; }
```

Linked list and tree inputs may be empty unless otherwise stated.

Q0 [5]

Why are large software systems (and even small ones) best designed in terms of independent components?

Q1 [45] *(Answer* **3** *of the following* **4** *parts a, b, c, and d.)*

In this question, you may define 0, 1, or more auxiliary subroutines, if it helps you. Answer with C++ code.

(a) Count distinct items

(i) Write a subroutine that counts the number of distinct items in a list of integers. For example $\langle 9, 8, 9, 1000000, 8, 5, 3, 9, 9, 8\rangle$ has 5 distinct members. Make your algorithm as efficient as possible. Use the ADTs presented in the course.

(Hint: The Set ADT does not contain a accessor for size.)

Complete this subroutine:

```
int countDistinct( const List<int> &inList )
{




}
```

(ii) Indicate which implementations for the ADTs would be best for this algorithm, and the resulting worst-case time complexity of your function.

(b) Write a subroutine that makes a copy of a tree.

Trees are represented by pointers to TreeNodes, with the empty tree being represented by the 0 (NULL) pointer.

Parameter inTree is a pointer to a tree. Parameter outTree is to be set to a copy of the tree.

Set ok to false, if space could not be allocated, and to true otherwise.

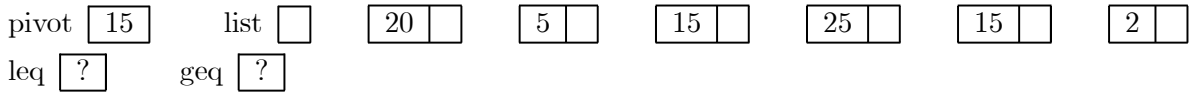
Complete this subroutine:

```
void treeCopy( TreeNode *inTree, TreeNode *&outTree, bool &ok ) {




}
```

(c) Write an in-place list 'partition' subroutine.

Given a linked list of numbers 'list' and a number 'pivot', your function will split the list in two so that all the nodes containing numbers less than 'pivot' are one list and all the nodes containing numbers greater than 'pivot' are in another (nodes containing values equal to 'pivot' may end up in either output list). Note that this is to be done 'in-place'; *do not allocate any new nodes, instead rearrange the links of the input list so that it becomes two lists.* Remember that 'list' may be empty.

Here is an example input:



and a possible output:

(*Hint: Use recursion.*)

Complete this subroutine:

```
void partition(int pivot, Node *list, Node *&leq, Node *&geq)
{

}
```

(d) Cumulative Sum *(Remember only to do 3 of 4 parts of* **Q1***)*

The cumulative sum of the sequence of numbers $\langle 4, 5, 7, 2, 3\rangle$ is $\langle 4, 9, 16, 18, 21\rangle$. Item i of the cumulative sum is the sum of the first $i + 1$ items of the input list. **Write a C++ subroutine to compute the cumulative sum of a sequence represented by a linked list.** The parameters are as follows:

- inputList — (input) a linked list representing the input sequence (may be empty)
- sumList — (output) to be set to a *new* linked list representing the cumulative sum.
- ok — (output) to be set to false, if heap storage runs out, and to true, otherwise.

You may assume that the integers in question are small enough or few enough that overflow is not a worry.

(Hint: Use recursion.)

Complete this subroutine.

```
void cumulativeSum( Node *inputList, Node *&sumList, bool &ok )
{




}
```

Q2 [5]

Consider this algorithm for computing x^i for a natural number i.

```
y ← 1
// Invariant: y × x^i = the desired result
while i > 0
    if i is odd
        y ← y × x
        i ← i − 1
    else
        x ← x × x
        i ← i/2
```

All arithmetic, assignment, and comparison operations are constant time ($O(1)$).

What is its worst-case time complexity? (Use big-Oh notation.)

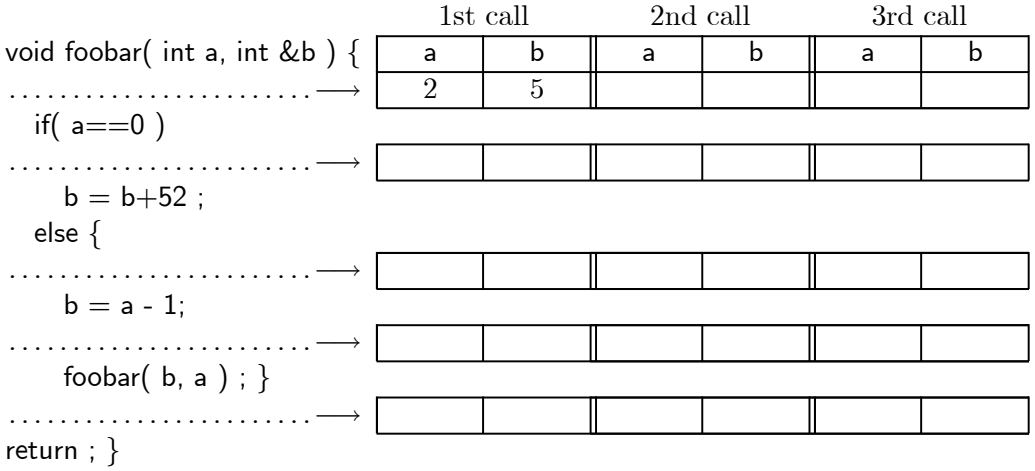
Q3 [5]

Indicate the value of each variable at each point (indicated with a dotted line) in following subroutine as it executes the call foobar(2,z). The initial value of z is 5.

This 1st call to foobar will cause 2 recursive calls to be made. Indicate the values of a and b in each of these 3 subroutine calls.

Leave squares blank if the corresponding point in the subroutine is not reached.

(Note that this subroutine is not intended to have a useful purpose.)



Q4 [10]

Consider the following sorting algorithm. It is a variation on the insertion sort. The input is in an array A of size N and the output in an array B of size N.

```
for( int i=0 ; i <N ; ++i ) {
    int k ;
    use binary search to set k to the smallest number k such that 0 ≤ k ≤ i + 1 and

                    for any l, such that k ≤ l ≤ i, B[l] > A[i]

    move all items in B at or after location k, one space to the right.
    B[k] = A[i] ; }
```

All arithmetic, assignment, and comparison operations are constant time ($O(1)$).

Note that binary search is order $O(\log i)$ for both worst-case and average-case.

(a) **What is the worst-case time complexity** of this algorithm. (Use big-Oh notation).

(b) **What is the average-case time complexity** of this algorithm? (Use big-Oh notation.)

Q5 [6]

Suppose p is a variable of type 'pointer to int'. **Give 3 reasons why the statement**

```
*p = 10 ;
```

may be in error.

0.

1.

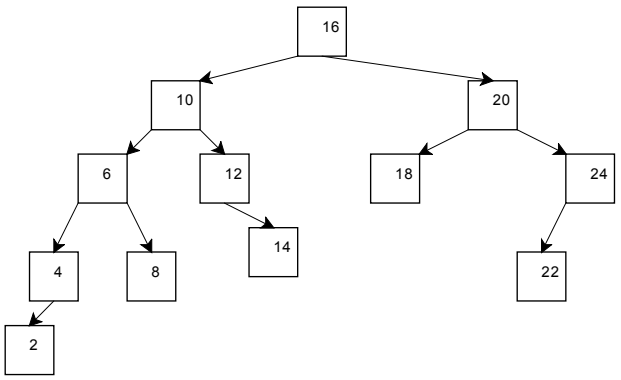
2.

Q6 [10]

Consider this AVL search tree.

(a) Write the height and balance factor next to each node.

(b) Suppose the 18 node is deleted. According to the algorithms presented in class, this will result in a sequence of 0 or more rotations. **Draw the tree after each of these rotations** (one diagram per rotation).



Q7 [10]

A list (sequence) can be represented by an AVL tree in which each node contains a value and the size of its left subtree.

(a) What are the complexities of retrieve, insert, and delete? (Use big-Oh

notation.)

(b) Describe an algorithm for looking up item n in the list.

Q8 [10]

You need to represent a function whose domain consists of roughly one million integers evenly distributed in the range $0,..,2^{31}-1$

(a) Suppose an AVL tree is used. **Estimate the number integer comparisons needed to search for a typical index**. Briefly justify your answer.

(b) Suppose a hash table of 1000 AVL trees is used. **Estimate the number of integer comparisons needed to search for a typical integer.** Briefly justify your answer.

Q9 [10]

(a) Briefly explain procedural abstraction.

(b) Briefly explain data abstraction.

Q10 [7]

(a) **Evaluate the following expressions:**

(i) $(\{'a','b','c','d'\} \cap \{'e','f'\}) \cup (\{'e','f'\} \cup \{'f','g','h'\}) =$

(ii) $((\{1,3,5,7,9,11\} \cup \{2,3,5,7,11\}) \cap \{1,2,3,4,5\}) - \{4,5,6,7\} =$

(iii) $\{x \in int \mid 4x = x^2\} =$

(iv) $|\{1,2,3,4,5,6,7,8,9\} \times \{'a','b','c','d'\}| =$

(b) **Indicate if the following statements are 'always true', 'always false', or 'sometimes true'** (i.e. true or false depending on the values of the variables).

(i) $|(S \cup S) \cap (S \cap S) \cap (S \cup \{\})| = |S|$

(ii) $S \cap (T \cup U) = (S \cap T) \cup (S \cap U)$

(iii) $(S - T) \cup U = (S \cup U) - (T \cup U)$

Q11 [7]

Let f be the function $\{(0,1),(1,1),(2,3),(4,3)\}$

What are:

(a) $dom(f) =$

(b) $f(4) =$

(c) $f(3) =$

(d) $f_0 =$

(e) $f_{3:=4} =$

(f) $f_{0:=3} =$

(g) $\{x \mid (x,1) \in f\} =$

Q12 [5]

An AVL tree has a height of 8. **What is the smallest number of nodes it could have? What is the largest number of nodes it could have?**

Q13 [10] (Optional)

In the tower of Hanoi puzzle there are 7 disks (numbered 0,1,..,6 from smallest to largest) stacked on 3 pegs (numbered 0, 1, 2). Each disk is on one peg. A larger disk must never be placed atop a smaller disk. Each move transfers the topmost disk on one peg to the top of the stack on another peg. The goal is to place all 7 disks on peg 2.

In the standard puzzle, the initial state has all disks on peg 0. For this question, however, you may not assume that. You can only assume that initially every disk is on one peg and no disk is atop a smaller one.

Compose and explain an algorithm that will move all disks to peg 2 regardless of where they are initially. Your algorithm may interact with the puzzle's state with 2 subroutines

- **move**(x, y) — moves the topmost disk on peg x to peg y.
- **find**(d) — returns the number of the peg on which disk number d is found.

(You may answer this on the back.)
