Test 2: CPS 103

Owen Astrachan

November 19, 1993

Name: ____________________________

Honor code acknowledgement (signature) ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 4</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 5</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 6</td>
<td>9 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 7</td>
<td>5 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 8</td>
<td>6 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>65 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1:  (Vocabulary: 9 points)

For each of the words/phrases below, circle the definition that is the best description as it pertains in the context of computer science, programming, and C/C++.

1. templates
   (a) The method employed in C/C++ that prevents recursion from using all available memory
   (b) The method employed in C++ to parameterize classes
   (c) A software-engineering practice in which programs are designed using a bottom-up, scaffold-building approach

2. hashing
   (a) A method of maintaining a table that supports $O(1)$ search operations in the average case.
   (b) A method of generating random numbers based on maintaining a seed and performing a series of exclusive-or operations on the seed.
   (c) A method of turning left-over food into inedible glop.

3. bubble sort
   (a) An $O(n^2)$ worst-case sort that, on large arrays of C++ Strings, performs better than selection sort
   (b) An $O(n^2)$ worst-case sort that can be coded so that already sorted arrays can be recognized in $O(\log n)$ time
   (c) A sort whose usefulness is comparable to that of a flea-infested dog in a pet-grooming business

PROBLEM 2:  (Complexity (12 points))

Part I:
An AVL tree is used to store $n$ strings so that an inorder traversal of the tree would print the strings in alphabetical order. For each of the operations listed below, give a big-O expression for the complexity of the operation and a brief explanation justifying the expression.

A. find the alphabetically first string in the tree

B. find the average string length of strings in the tree
C. delete a string from the tree

D. find the smallest (# of characters) string in the tree

Part II:
Which of the operations A–D above has a different worst-case complexity if a “normal” search tree is used rather than an AVL tree. Briefly justify your answers.

PROBLEM 3: (Trees and functions (6 points))

Consider the following functions on binary trees of integers:

```c
int Sum(Tree * t)
{
    if (t == NULL) return 0;
    else return t->info + Sum(t->left) + Sum(t->right);
}

int Count(Tree * t)
{
    if (t == NULL) return 0;
    else return 1 + Count(t->left) + Count(t->right);
}
```

For a tree \( t \), what is the value of the expression

\[
\text{double(Sum(t))/Count(t)}
\]

and what is the complexity of computing this value (using big-O notation) for a tree of \( n \) nodes? Briefly justify your answer.
PROBLEM 4: *(Meta Functions)*

Consider the following functions:

```c
int TreeStat(int (*Func)(Tree *), Tree * t)
{
    if (t == NULL) return 0;
    else return (*Func)(t) + TreeStat(t->left) + TreeStat(t->right);
}

int One(Tree * t)
{
    return 1;
}

int Value(Tree * t)
{
    return t->info;
}
```

The expression

```
double(TreeStat(Value,t))/TreeStat(One,t)
```

has the same value as the expression \( \frac{\text{double}(\text{Sum}(t))}{\text{Count}(t)} \) of the previous problem because \( \text{TreeStat(Value,t)} = \text{Sum}(t) \) and \( \text{TreeStat(One,t)} = \text{Count}(t) \).

Write the function *LeafIt* whose header is given below so that the expression \( \text{TreeStat(LeafIt,t)} \) evaluates to the number of leaves in the binary tree \( t \).

```c
int LeafIt(Tree * t)
```
PROBLEM 5:  (Priority Queues (12 points))

If a sorted linked list is used to implement a priority queue, what is the complexity of the \( \text{DeleteMin} \) operation and the \( \text{Insert} \) operation on a queue of \( n \) elements. Briefly justify your answer.

If a hash table is used to implement a priority queue, what is the complexity of the \( \text{DeleteMin} \) operation and the \( \text{Insert} \) operation on a queue of \( n \) elements. Briefly justify your answer.

If a heap is used to implement a priority queue, what is the complexity of the \( \text{DeleteMin} \) operation and the \( \text{Insert} \) operation on a queue of \( n \) elements. Briefly justify your answer.

PROBLEM 6:  (General Trees (9 points))

A tree in which each node can have an arbitrary number of children, and in which each node stores an integer, can be implemented using the following declarations:

```c
struct Tree {
    int info;
    Tree * firstChild;   // first child of node
    Tree * sibling;      // sibling of node
}
```

Write the routine \( \text{TreeProduct} \) whose header is given below. \( \text{TreeProduct} \) returns the product of all the values stored in the nodes of \( t \). For example, it would return \( 12! = 479,001,600 \) for the tree diagrammed below. In the tree below, the solid arrows \( \rightarrow \) represent the connections using the declaration for \( \text{Tree} \) given above, the dotted arrows represent the "conceptual" children for each node of the tree.
By definition, the value returned for an empty tree is zero.

```
int TreeProduct(Tree * t)
// postcondition: returns product of all nodes in t
//                  returns 0 if t is empty
```
PROBLEM 7:  (Profound Leaves (5 points))

Part I
The code for function DeepLeaf given below returns a pointer to a leaf in the tree that is farthest from the root, i.e., the deepest leaf in the tree.

```c
Tree * DeepLeaf(Tree * t)
{
    if (t == NULL)
        return NULL;
    else if (IsLeaf(t))
        return t;
    else if (height(t->left) >= height(t->right))
        return DeepLeaf(t->left);
    else
        return DeepLeaf(t->right);
}
```

For a tree of $n$ nodes, the average-case complexity of the function DeepLeaf shown above is NOT $O(n)$, what is it? (Briefly justify your answer, you may assume that, on the average, trees are “balanced”.)

Part II
It is possible to write DeepLeaf so that it works in time $O(n)$ for a tree of $n$ nodes. This is shown below. Fill in the recursive calls of DeepWork so that DeepLeaf works correctly in $O(n)$ time.

```c
Tree * DeepLeaf(Tree * t)
{
    Tree * deep = NULL; int max = 0;
    DoDeep(t,0,max,deep);
    return deep;
}
void DoDeep(Tree * t,int depth, int & maxDepth, Tree * & deepTree)
// precondition:
// postcondition:
{
    if (t == NULL){
        if (IsLeaf(t)){
            if (depth > maxDepth){
                maxDepth = depth;
                deepTree = t;
            }
        }
    }
    else{
        DeepWork(t->left,depth + 1,maxDepth,deepTree);
        DeepWork(t->right,depth + 1,maxDepth,deepTree);
    }
}
```
PROBLEM 8: *(Lazy Dog Sorts (6 points) — EXTRA CREDIT)*

Consider sorting a linked list using quicksort. Assume the following declarations have been made:

```c
struct Dlist{
    double info;
    Dlist * next;
};

void Pivot(Dlist * & first, Dlist * & second)
// precondition: first is non-empty list of n nodes
// postcondition: second is non-empty list
//   all nodes in first have info values strictly less than
//   second->info. Also, second->info is the smallest value
//   of all nodes in second
//
// all nodes originally in first are in either first
// or second
```

The idea is to use the following code to implement quicksort for linked lists, it is complete except for after the comment: *"connect things back so that list is sorted".*

You are to complete the code.

```c
void Quicksort(Dlist * & list)
{
    if (list != NULL){
        //partition list into two parts
        Dlist * hold;
        Pivot(list,hold);

        // note: all nodes in list are less than hold->info
        // and hold->info is smallest value in hold
        // (see spec. for Pivot)

        Dlist * anchor = hold;
        hold = hold->next;
        anchor->next = NULL;

        Quicksort(list);
        Quicksort(hold);

        // connect things back so that list is sorted>
    }
}
```

---

1 Why is this question called “lazy dog sorts” — it has nothing to do with bubble sort and dogs.