Test 2: CPS 100

Owen Astrachan and Dee Ramm

April 9, 1997

Name: ______________________________

Honor code acknowledgment (signature) ______________________________

	value	grade
Problem 1	15 pts.	
Problem 2	12 pts.	
Problem 3	17 pts.	
Problem 4	13 pts.	
Problem 5	12 pts.	
Problem 6	25 pts.	
TOTAL:	92 pts.	

This test has 9 pages, be sure your test has them all. Do NOT spend too much time on one question — remember that this class lasts 50 minutes.

In writing code you do not need to worry about specifying the proper `#include` header files. Assume that all the header files we've discussed are included in any code you write.

The declaration for Tree nodes used on this test is provided below:

```
template <class Type>
struct Tree
{
   Type info;
   Tree * left;
   Tree * right;
   Tnode(Type val, Tree * lbranch = 0, Tree * rbranch = 0)
     : info(val),
       left(lbranch),
       right(rbranch)
   {}
};
```

PROBLEM 1 : (*Stuff* (15 points))

Part A: (3 points)

When a priority queue is implemented using a min heap, the heap is stored in an array. As discussed in class, if a node of the heap has two children, and the index of the node is k, what are the indexes of the left and right children of the node, in terms of k.

Part B: (3 points)

If a priority queue is implemented using a sorted vector, the complexity of *FindMin* is $O(1)$. Explain why the worst-case complexity of *Insert* is $O(n)$ for an n-element priority queue implemented using a sorted vector.

Part C: (3 points)

Of inorder, preorder, postorder, and levelorder tree traversals, which is best for deleting all the nodes of a tree and which is best for writing a tree to a file so that the same tree can be read back in.

Part D: (6 points)

Consider the following method for finding the largest value stored in a vector.

1. Divide the vector in two, find the largest value in each half (e.g., recursively)
2. Compare the two values, the largest of these is the overall largest value.

What is the complexity (use big-Oh) for this method? Briefly justify your answer. Answer the question (give complexity) if the same method is used, but with linked lists instead of vectors.

PROBLEM 2 : (*Big Oh'ks* (16 points))

For each operation below give the asymptotic complexity (using big-Oh) for the operation as defined on binary *search* trees storing strings. Assume that, in the average case, binary trees are balanced. Express answers in terms of N, where there are N nodes in a tree. **You must provide answers for both average case and worst case.**

Part A:

Find the alphabetically last string in the tree.

average case: worst case:

Part B:

Compute the average word length of all the strings in the tree

average case: worst case:

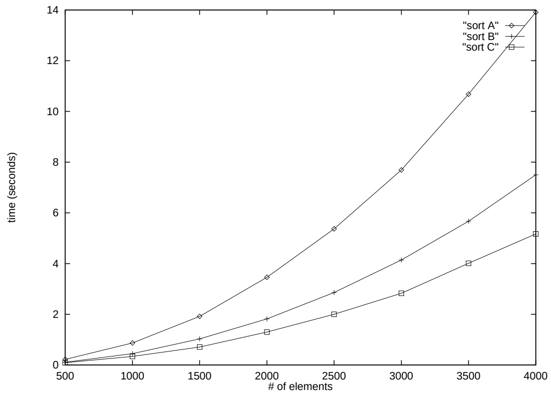
Part C:

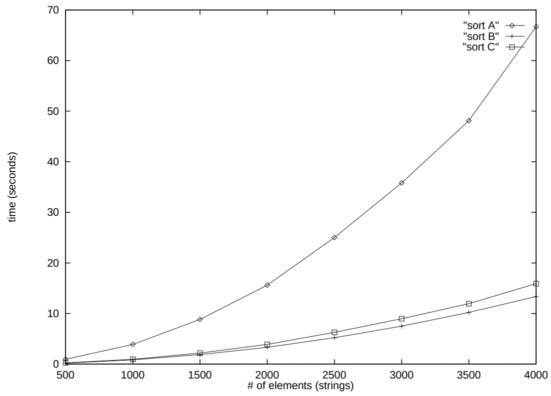
Create a new search tree with the same strings, but in which the tree is organized according to the sorted version of the string (e.g., as in anagram). For example: the string "guava" would be sorted as "aaguv".

average case: worst case:

PROBLEM 3 : (*Sort* (17 points))

Three $O(n^2)$ sorts of integer vectors as plotted by sortall are shown below: integers on the left, strings on the right. On the left, sort A is the slowest and sort C is the fastest, on the right sort A is the slowest and sort B is the fastest.





The data that generated the plot for integers on the left is shown in the table below:

	number of integers							
	500	1000	1500	2000	2500	3000	3500	4000
sort A	0.22	0.87	1.92	3.46	5.37	7.69	10.68	13.91
sort B	0.11	0.45	1.03	1.82	2.86	4.14	5.67	7.50
sort C	0.09	0.34	0.71	1.30	2.00	2.83	4.01	5.17

Part A: *6 points*

Identify sorts A, B, and C as either bubblesort, insertion sort, and selection sort.

Part B: *3 points*

True or False: For sorting 10,000 strings mergesort is much faster than the sorts graphed above

Part C: *8 points*

Based on the data and graphs above, how long will it take sorts A and C to sort a vector of 16,000 integers. Your answer doesn't need to be exact, approximate but close is fine. Your math/reasoning counts for 4 out of the 8 points (so two numbers for times with no reasoning/arithmetic gets 4/8 points).

PROBLEM 4 : (*Trees* (13 points))

Part A: *6 points*

Complete the **recursive** function *Insert* whose header is given below. `Insert` inserts `item` into the binary search tree `t` so that `t` remains a search tree (the declaration for type `Tree` is on the first page of the test).

```
template <class Type>
void Insert(Tree<Type> * & t, const Type & item)
// pre:  t is a binary search tree
// post: item stored in t so that t is a binary search tree
{
    if (t == 0)
    {



    }
    else
    {
        if (t->info < item) Insert(                    , item);
        else                Insert(                    , item);
    }
}
```

Part B: *4 points*

A naive algorithm for constructing a binary search tree from a **sorted** linked list or a sorted vector can be described as:

```
tree = NULL;   // initially empty
for each item in SORTED vector/linked list
    Insert(tree,item)
```

The complexity of this method is $O(n^2)$ because a sorted list/vector is used. Explain why

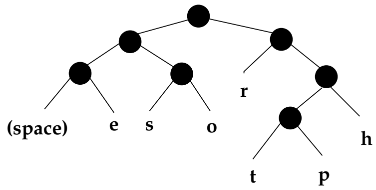
Part C: *3 points*

Consider the complexity of the following alternative method for turning sorted linked lists/vectors into binary search trees. What is the complexity of this method (average case)? Justify briefly, assume that shuffling a vector/list of N elements is an $O(N)$ operation.

```
Shuffle the elements in the vector/list so they are NOT sorted
Now apply the method above to the shuffled vector/list
```

PROBLEM 5 : (*If u cn rd ths (*12 points))

Assume the Huffman tree below is used for compressing/uncompressing a file.



Part A: (4 points)

Which letter occurred at least as often (or more often) than any other letter to make the tree diagrammed above?

Which two letters occurred less often than all other letters?

Part B: (4 points)

Uncompress the stream of bits below using the tree.

`1 1 0 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1`

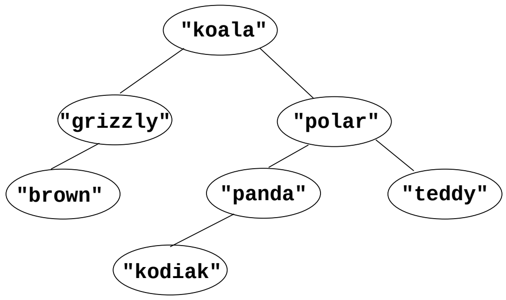
Part C: (4 points)

Write the bit sequence that encodes the characters:

```
the sort
```

PROBLEM 6 : (*TreesII* (25 points))

Consider the tree diagrammed below:



Part A: *3 points*

Write the postorder traversal of tree above whose root is `"koala"`.

Part B: *3 points*

The height of the tree diagrammed above is three (3). Draw a **search tree** containing the same seven strings, but whose height is two (2).

Part C: *3 points*

It is possible for an n-node tree to be shaped so that every node in the tree has either two children or zero children, but for the height of the tree to be $O(n)$ (hint: one way is for every right subtree to be either empty or to contain exactly one node). Diagram how to construct such a tree and give a brief reason justifying that it's height is $O(n)$. You might, for example, draw 3, 5, and 7 node trees and then generalize.

(continued)

Part D: *points 8*

The function `CountNodes` below is correct, it returns the number of nodes in the tree pointed to by t.

```
template <class Type>
int CountNodes(Tree<Type> * t)
// postcondition: returns # of nodes in t
{
    if (t == 0)
    {
        return 0;
    }
    else
    {
        return 1 + CountNodes(t->left) + CountNodes(t->right);
    }
}
```

Write the function *CharCount* whose header is given below. `CharCount` returns the number of times the character `key` occurs in all the strings stored in a binary tree of strings. For example, given the tree of bears on the previous page, the call `CharCount(tree,'a')` should evaluate to 6 and the call `CharCount(tree,'f')` should evaluate to 0.

```
int CharCount(Tree<string> * t, char key)
// postcondition: returns # times key occurs in all strings of t
```

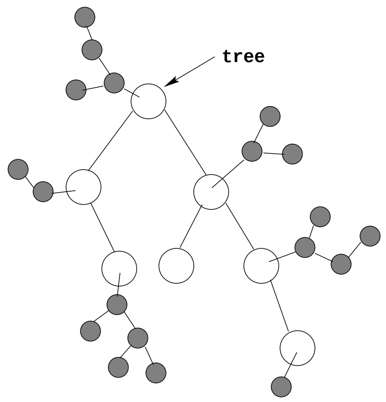
(continued)

Part E (8 points)

Suppose a tree in which each info field stores a tree of strings is declared as follows:

```
Tree<Tree<string> *> tree;
```

A diagram of a seven-node tree of string trees is shown below, the shaded nodes are part of each string tree. Note that one string tree is empty.



Write the function *MetaCount* whose header is given below. `MetaCount` returns the number of string tree nodes, i.e., the number of shaded nodes in the diagram above. In the diagram above the call `MetaCount(tree)` should evaluate to 19 since there are 19 string nodes (and 7 nodes whose info fields are of type `Tree<Tree<string> *>`. In writing `MetaCount` you may call the templated function `CountNodes` given on the previous page.

```
int MetaCount(Tree<Tree<string> *> tree)
// post: returns # nodes containing strings
```