Test 2: CPS 100

Owen Astrachan

November 19, 1997

Name: ____

Honor code acknowledgment (signature) ______________________________________

	value	grade
Problem 1	9 pts.	
Problem 2	12 pts.	
Problem 3	10 pts.	
Problem 4	15 pts.	
Problem 5	10 pts.	
Problem 6	9 pts.	
TOTAL:	65 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.

Declarations for Tree and List nodes, the function *height* for trees, and some recurrence relations, are found on the last page.

PROBLEM 1 : (*ghiiknnt abotu orsst (*9 points*)*)

1. In the Anagram program at the beginning of the semester it was suggested that you use selection sort to sort the letters of a word in forming the canonical form of a word, e.g., in converting the word `"smile"` to `"eilms"`. Briefly, why was this a better choice than using Mergesort which is an $O(n \log n)$ sort whereas selection sort is $O(n^2)$.

2. Briefly, why does using smart string pointers (rather than strings) reduce the time to sort more for bubble sort and insertion sort than it does to reduce the time for selection sort?

3. Briefly, why does a Quicksort of 10,000 ints, all in the range 0..99 take <u>much</u> longer than quicksort of 10,000 ints that range over all int values [as implemented in `sortall.cc`].

PROBLEM 2 : (*Bonsai Problems (*12 points*)*)

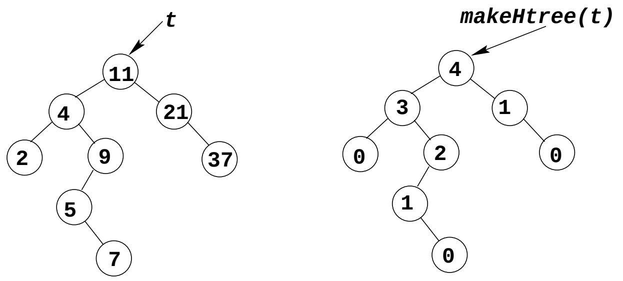
Part A: 4 points Write the function *twoChildCount* that returns the number of nodes in tree *t* that have exactly two children.

```
int twoChildCount(Tree<string> * t)
// post: returns number of nodes in t with exactly two children
{



}
```

Part B: 8 points Write the function *makeHtree* whose header is given below. *makeHtree* creates a new tree with the same shape as the tree *t*, but in which each node stores the height of the node. Recall that the height is the number of edges on the longest root-to-leaf path, that the height of any node is one more than the maximum of the left/right subtree heights, and the height of a leaf node is zero. (the function *height* we discussed in class is on the last page of the test).

For full credit your function should work in *O(n)* time for an n-node tree.

For example, in the diagram below, the tree t on the left would generate the tree shown on the right as a result of `makeHtree(t)`:




```
Tree<int> * makeHTree(Tree<int> * t)
// postcondition: returns a new tree with the same shape as t
//                in which each node contains a value that is the height
//                of the node
{




}
```

PROBLEM 3 : (*Hash, hash, and more hash (*10 points))

Part A: 8 points You are inserting the strings below into a hash table. Assume the hash value for each string is as shown.

string	**hash value**
`"pea"`	12
`"carrot"`	25
`"bean"`	17
`"broccoli"`	37
`"corn"`	27
`"kale"`	8
`"okra"`	19

If these values are inserted in this order (i.e., `"pea"` first and `"okra"` last) into a hash table with 11 entries draw the table assuming that:

hashing is resolved by chaining

0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***

hashing is resolved by linear probing

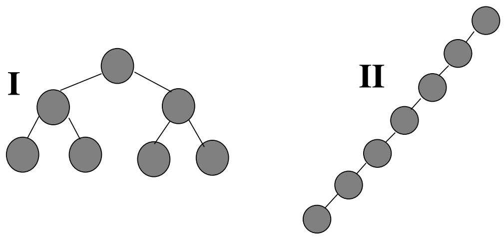
0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***

Part B: 2 points Once the entries are in the hash table, describe a method for printing the entries in alphabetical order in *O(n log n)* time, where n is the number of elements in the hash table. **Be brief, a few words suffice.**

PROBLEM 4 : (*Sometimes a branch is just a branch (*15 points*)*)

For each of the questions below, give a big-Oh expression that best describes the quantity described, and draw a picture of a 7 node tree that "justifies" your expression. For example: in a binary tree of n nodes, what are the smallest and largest values for the height?

Answer: smallest height is $O(\log n)$, largest height is $O(n)$. Pictures showing this are on the left (I) and the right (II). If you want to use one of these pictures, refer to them as I and II.



1. In a binary tree of n nodes what are the minimal and maximal number of leaves (give big-Oh expressions)?

2. In a binary **search** tree of n nodes what are the minimal, maximal, and average case number of nodes with values less than the root value (give big-Oh expressions)?

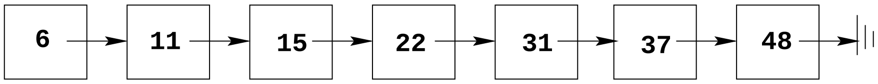
3. What are the minimal and maximal height of a binary **search** tree of n nodes in which the number of values less than the root is equal to the number of values greater than the root (give big-Oh expressions)?

PROBLEM 5 : (*Trees to starboard* (10 points))

Consider the problem of inserting the values in a *sorted* linked-list into a binary search tree. Two methods are suggested: *list2Tree* which uses the standard *insert* function for search trees, and a recursive function *recList2Tree*. You don't need to look at these functions in detail, they both succeed in constructing a binary search tree, they both have a complexity that's bigger than *O(n)* for an n-node list.

Part A: 5 points What is the big-Oh complexity of *list2Tree*? Briefly justify your answer.

Draw the tree that results from using *list2Tree* when the list below is converted into a search tree.




```
void insert(Tree<int> * & t, int key)
// pre:  t is a search tree
// post: key is inserted into t, t is a search tree
{
    if (t == 0)                 t = new Tree<int>(key);

    else if (key < t->info)     insert(t->left,key);

    else                        insert(t->right,key);
}

Tree<int> * list2Tree(Node<int> * list)
// pre:  list is sorted
// post: returns a search tree whose elements are those in list
{
    Tree<int> * root = 0;
    while (list != 0)
    {
        insert(root,list->info);
        list = list->next;
    }
    return root;
}
```

Part B: 5 points What is the big-Oh complexity of *recList2Tree*? Briefly justify your answer.

Draw the tree that results from using *recList2Tree* when the list above is converted into a search tree.

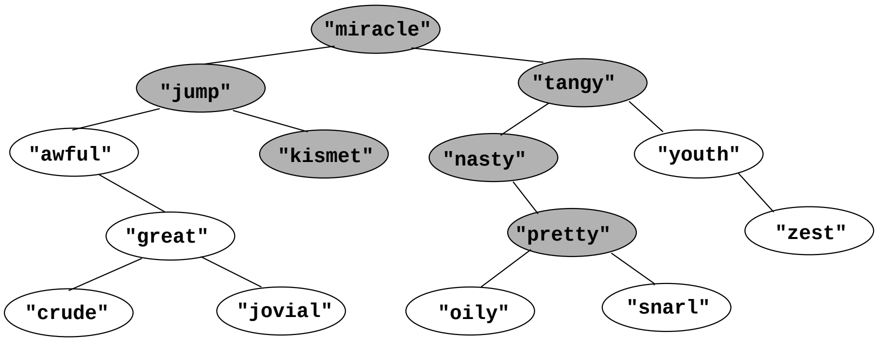
```
Tree<int> * recList2Tree(Node<int> * list)
// pre:  list is sorted
// post: returns a search tree whose elements are those in list
{
    Node<int> * middle = 0;
    Node<int> * temp = list;

    // treat 0 and 1 node lists separately
    if (list == 0)      return 0;
    if (list->next == 0) return new Tree<int>(list->info);

    while (temp != 0)
    {
        temp = temp->next;
        if (temp != 0)
        {
            if (middle == 0) middle = list;
            else             middle = middle->next;
            temp = temp->next;
        }
    }
    temp = middle->next;   // unlink middle node from front sublist
    middle->next = 0;

    Tree<int> * t =  new Tree<int>(temp->info,
                                  recList2Tree(list),
                                  recList2Tree(temp->next));

    middle->next = temp;   // link middle node back to front sublist
    return t;
}
```

PROBLEM 6 : (*Home on the Range (* 9 points*)*)

Write the function *range* whose header is given below. The tree *t* is a search tree, and *range* stores all values from t between (and including) *first* and *last* in the vector *words*. The values stored in *words* should be stored in alphabetical order. For full credit your code should visit the minimal number nodes in *t*. For example, in the tree *t* diagrammed below, the call

```
range(t,"kangaroo","pancake",words);
```

should store the values `"kismet"`, `"miracle"`, `"nasty"`, `"oily"` in *words*, and visit only the shaded nodes shown in the diagram.




```
void range(Tree<string> * t, const string & first, const string & last,
           Vector<string> & words)
// postcondition: all strings s in t, such that first <= s <= last,
//                are stored in words,in alphabetical order
```

```
template <class Type>
struct Tree
{
    Type info;
    Tree * left;
    Tree * right;
    Tree(const Type & s, Tree * lchild = 0, Tree * rchild = 0)
        : info(s), left(lchild), right(rchild)
    {}
};

template <class Type>
struct Node
{
    Type info;
    Node * next;
    Node(const Type & s, Node * link = 0)
       : info(s), next(link)
    { }
};


int max(int x, int y)
{
   if (x > y) return x;
   else return y;
}

template <class Type>
int height(Tree<Type> * t)
{
    if (t == 0) return -1;
    else return 1 + max(height(t->left),height(t->right));
}
```

recurrence	**solution**
$T(n) = T(\frac{n}{2}) + O(1)$	$O(\log n)$
$T(n) = 2T(\frac{n}{2}) + O(1)$	$O(n)$
$T(n) = 2T(\frac{n}{2}) + O(n)$	$O(n \log n)$
$T(n) = T(n-1) + O(n)$	$O(n^2)$