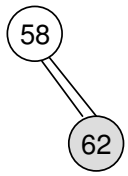
October 21, 2009 Name _____

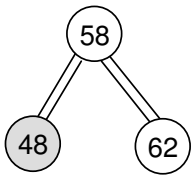
EECS 311 Data Structures
Midterm Exam
Don't Panic!

1. (10 pts) In the boxes below, show the **red-black** trees that result from the successive addition of the given values. Use doubled-lines for red links. Clearly indicate recoloring and rotations, if any, with intermediate trees and "left" or "right" for direction of rotation.

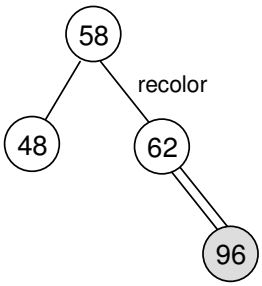
1. After adding 62 to a tree with 58.



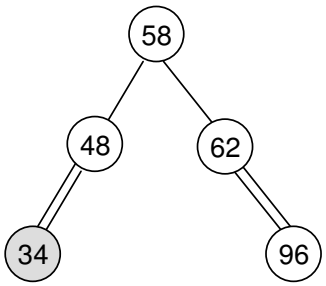
2. After adding 48 to the previous tree.



3. After adding 96 to the previous tree.

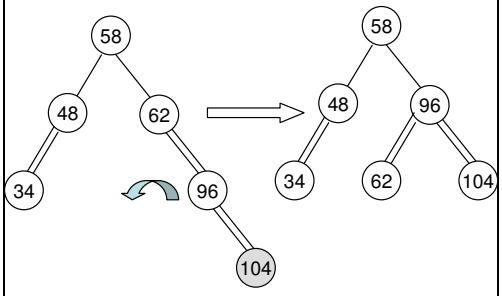


4. After adding 34 to the previous tree.

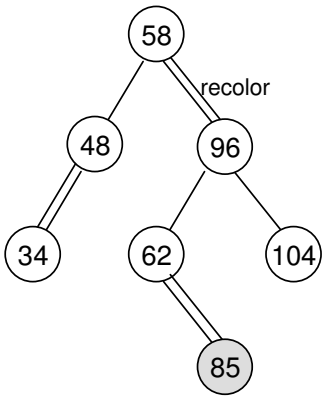


Comment [CKR1]: Common mistake: recoloring as soon as 2 red child links made, instead of when inserting through a node with 2 red child links.

5. After adding 104 to the previous tree.



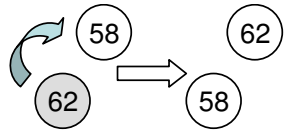
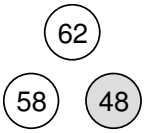
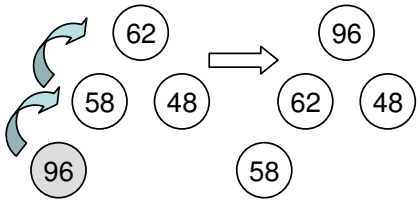
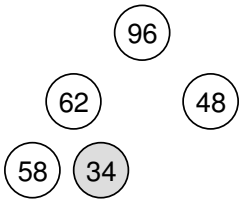
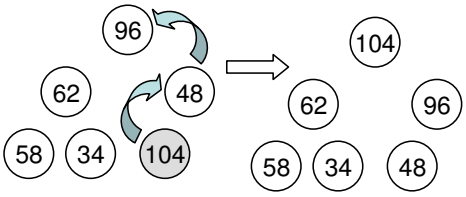
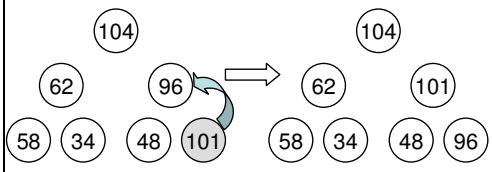
6. After adding 85 to the previous tree.



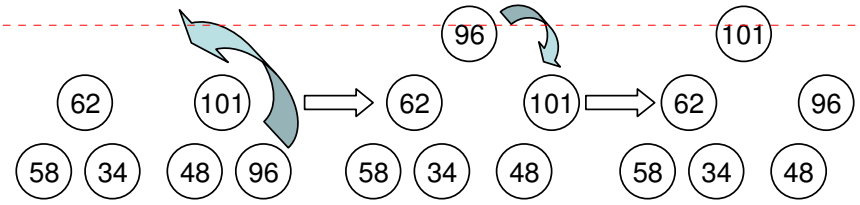
Comment [CKR2]: Mistake: forgetting to color the upper link red.

2. (10 pts) In the boxes below, show the **binary heaps** in tree form that result from the successive additions of the given values, where larger values beat lower values. Clearly indicate what swaps occur to maintain the heap.

Comment [CKR3]: A few students did binary search trees.

1. After adding 62 to a tree with 58. 	2. After adding 48 to the previous tree. 
3. After adding 96 to the previous tree. 	4. After adding 34 to the previous tree. 
5. After adding 104 to the previous tree. 	6. After adding 101 to the previous tree . 

3. (5 pts) Using the heap generated in question 2 as a priority queue, show the swaps that would occur after the first item in the queue is removed.



Comment [CKR4]: Most common mistake: filling gap directly with 101.

4. (20 pts) The function getWinner() is supposed to take a vector of names representing votes for candidates and return the name that appears strictly more than half the time, if any, or the empty string. Examples:

{ "A", "B", "C", "B", "A", "B", "B", "C", "B" } – winner is "B"
{ "A", "A", "A", "C", "C", "B", "B", "C", "C", "C", "B", "C", "C" } – winner is "C"
{ "A", "B", "C", "B", "A", "B", "C", "B" } – no winner ""

Three correct definitions are below. For each, give the computational complexity with a reasoned justification.

a)

```
string getWinner1( const vector<string> &ballots ) {
  int len = ballots.size();  this is O(1)
  for ( int i = 0; i < len; ++i )  this is O(N)
    if ( count( ballots.begin(), ballots.end(), ballots[i] )
         > len / 2 )  each count() is O(N), comparison is O(1)
      return ballots[i];  this is O(1)
  return "";  this is O(1)
}
```

Comment [CKR5]: Some people said this was O(N)

Comment [CKR6]: Common mistake: calling count() O(N) or saying the comparison was O(N).

It was required to identify count() as the O(N) component.

Because we have an O(N) operation done O(N) times, this is O(N²).

b)

```
string getWinner2( const vector<string> &ballots ) {
  int len = ballots.size();  this is O(1)
  map<string, int> votes;  this is O(1)
  for ( int i = 0; i < len; ++i ) ++votes[ ballots[i] ]; see below
  for ( map<string, int>::iterator iter = votes.begin();
        iter != votes.end();
        ++iter )  this is O(K)
    if ( iter->second > len / 2 ) return iter->first;  this is O(1)
  return "";
}
```

For N ballots and K candidates, the first FOR runs O(N) times. Each votes[] call is O(log K). Second loop runs K times. K is N in the worst case. So first loop is O(N log N), so the entire algorithm is O(N log N).

c)

```
string getWinner3( const vector<string> &ballots ) {
  int len = ballots.size();
  string winner = "";
  int tally = 0;  these are all O(1)
  for ( int i = 0; i < len; ++i ) {  this is O(N)
    if ( tally == 0 ) winner = ballots[i];  this is O(1)
    if ( winner == ballots[i] ) ++tally; else --tally;  this is O(1)
  }
  if ( count( ballots.begin(), ballots.end(), winner )
       > len / 2 )  the count() is O(N) and the comparison is O(1)
    return winner;  this is O(1)
  else
    return "";  this is O(1)
}
```

The FOR loop runs O(N) times and the body is O(1). So it plus the final count() call make this O(N).

d) Give an argument for the correctness of `getWinner3()`. Hint: a vote for one candidate cancels a vote for another candidate.

If X has a majority, i.e., more than half the votes, X must end up as the final winner, because it will have at least one vote not cancelled. The final count() is needed because cases with no majority have "winners" too, e.g., "AABCC" and "CCBAA".

Comment [CKR7]: Very common mistake: assuming this is O(1)

Comment [CKR8]: Common mistake saying this loop is O(N log N) without identifying the O(log N) part.

Comment [CKR9]: Common mistake: saying iterator access was O(log N).

Comment [CKR10]: Accepted just using N throughout.

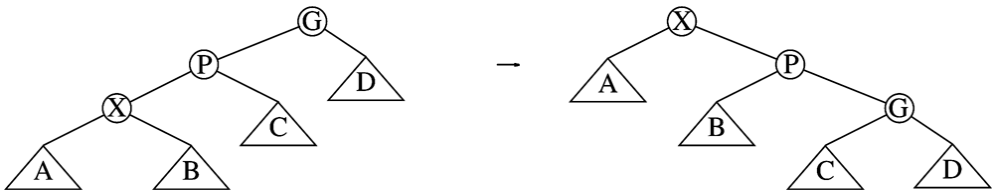
Comment [CKR11]: Mistake: saying this is O(N) worst case. It's O(N) in all cases.

Comment [CKR12]: Common mistake: repeating code in English, which doesn't argue for anything.

Comment [CKR13]: Some said not correct, overlooking the definition of majority and/or the 2nd FOR loop test. Note: this the real definition of majority, and why there's a runoff in Afghanistan.

Comment [CKR14]: Very common mistake: saying winner has most votes. Consider AAABBBC. C wins and has fewest votes.

5. (10 pts) Using the C++ tree class below, implement `zigzigRight( Node *&node )` so that `zigzigRight(node->left)` or `zigzigRight(node->right)` inside a `Tree` member function would do the rotation shown to the specified subtree:




```
template <typename T> class Tree {
  private:
    struct Node {
      Node *left, *right;
      T data;
      ...
    };
  public:
    Node * root;
    void zigzigRight(Node *&node)
    {
        Node *g = node;
        Node *p = g->left;
        Node *x = p->left;

        g->left = p->right;
        p->right = g;
        p->left = x->right;
        x->right = p;

        // updates old pointer to g because node is
        // passed by reference
        node = x;
    }
};
```

Comment [CKR15]: You can get by with fewer variables, but then you have to be extra careful about the order in which things are assigned.

Comment [CKR16]: Common mistake: not updating node

Comment [CKR17]: Common mistake: using names like left, right, or parent that are neither variables nor members of Tree.

Comment [CKR18]: No NULL checks needed or desired, since any null pointers need to be copied.

Comment [CKR19]: root is not relevant to anything here

Comment [CKR20]: Setting root to node is a very bad idea. You just reduced the tree to a subtree.

Comment [CKR21]: Using one variable temp, if correctly done, was accepted but that approach takes several times longer to understand