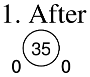
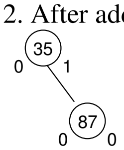
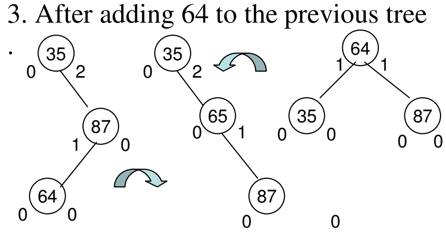
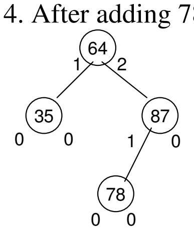
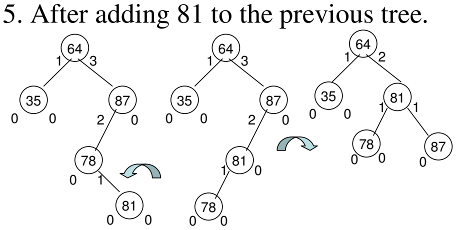
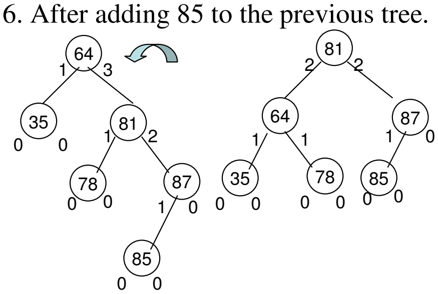
April 25, 2007 Name ______________________________

EECS 311 Data Structures
Midterm Exam
Don't Panic!

1. (10 pts) In each box below, show the AVL trees that result from the successive addition of the given elements. Show the nodes, links and balance factors. Draw intermediate trees and clearly indicate rotations, if any, and in what direction.

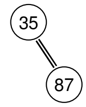
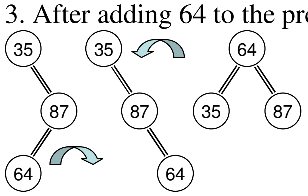
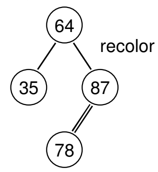
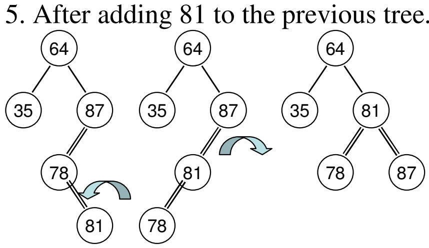
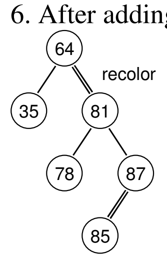
Comment [CKR1]: Common mistakes:
- neither balance factors nor heights
- balance factors not ±
- one balance factor for entire tree

Comment [CKR2]: Common mistakes:
- not indicating rotations clearly
- suggesting a single rotation when double rotations required

1. After adding 35 to an empty tree. 	2. After adding 87 to the previous tree. 
3. After adding 64 to the previous tree 	4. After adding 78 to the previous tree. 
5. After adding 81 to the previous tree. 	6. After adding 85 to the previous tree. 

2. (10 pts) In each box below, show the red-black trees that result from the successive additions of the given elements. Use doubled lines for red links Draw intermediate trees and clearly indicate recolorings and rotations, if any, and in what direction.

Comment [CKR3]: Common mistakes:
- not clearly indicating rotations
- suggesting single rotation when double rotations needed
- not indicating recolorings
- recoloring too soon (no points lost)
- doing an AVL rotation in step 6 instead of a red-black rotation
- building an invalid red-black tree (unbalanced black links)

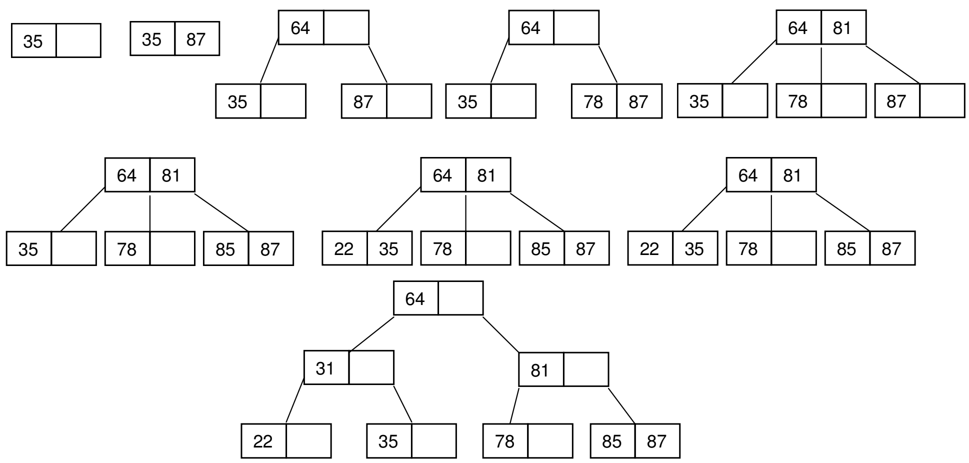
1. After adding 35 to an empty tree. 	2. After adding 87 to the previous tree. 
3. After adding 64 to the previous tree. 	4. After adding 78 to the previous tree. 
5. After adding 81 to the previous tree. 	6. After adding 85 to the previous tree. 

3. (10 pts) Draw the B-trees that result when adding the following values in succession, starting with an empty tree. Assume each node can only hold 2 keys. To save drawing time, you can choose to only draw a new tree when a split occurs, but <u>make it clear which value caused the split</u>.

Comment [CKR4]: I accepted either true B-trees, or B+trees (the book has B+trees mislabeled as B-trees – footnote p 161)

Comment [CKR5]: Common mistakes:
- more than 2 values in leaves
- nodes with # children <= # keys; should always be #keys + 1
- keys moving downwards
- completely empty nodes
- splits creating full nodes full, instead of half-full
- leaves at different depths

Values: 35, 87, 64, 78, 81, 85, 22, 31



4. (5 pts) Give the Big-Oh complexity *and a reasoned argument* for the following algorithm (in pseudo C++) for finding the position in s1 of a longest common substring of two strings s1 and s2, of lengths M and N, respectively. `string::compare()` returns 0 for equality, like C's `strcmp()`.

```
for i from 0 to M
  for len from 1 to M - i
    for j from 0 to N - len
      if s1.compare(i, len, s2, j, len) == 0
         if len > result_len
           result = i
           result_len = len
```

Comment [CKR6]: Most common mistake: treating compare() as O(1). It clearly depends on len, which is O(M), or O(min(M,N)) assuming it stops with the shorter string.

There are three nested loops with bounds $O(M)$, $O(M)$ and $O(N)$ respectively. The comparison will be up to $O(M)$ characters, or more accurately, $O(\min(M, N))$. The assignments inside the IF are $O(1)$. Therefore the worst case running time is $O(M^3N)$.

5. (10 pts total) a) Assume a 10-element hashtable, with hash(x) = x mod 10 and linear probing. Show what locations would be probed, in order, for each value in the table, and put the value in its final resting place, if any, in the array:

Comment [CKR7]: FYI: this is Exercise 1 in Chapter 5.

Value	Locations probed
4371	1
1323	3
6173	3, 4
4199	9
4344	4, 5
9679	9, 0
1989	9, 0, 1, 2

Array:

0	1	2	3	4	5	6	7	8	9
9679	4371	1989	1323	6173	4344				4199

b) Repeat, with the same hash(), but using double hashing with hash2(x) = 7 – (x mod 7).

Comment [CKR8]: Most common mistakes:
- Not using hash(x) + i * hash2(x)
- Using x mod 7, not 7 – (x mod 7)
- Linear probing (x, x + 1, x + 2, …) instead of x, x + hash2(x), x + 2 hash2(x), …

I only counted each mistake once, if other wrong answers were at least consistent.

4371 mod 7 = 3	1323 mod 7 = 0	6173 mod 7 = 6	4199 mod 7 = 6
4344 mod 7 = 4	9679 mod 7 = 5	1989 mod 7 = 1	

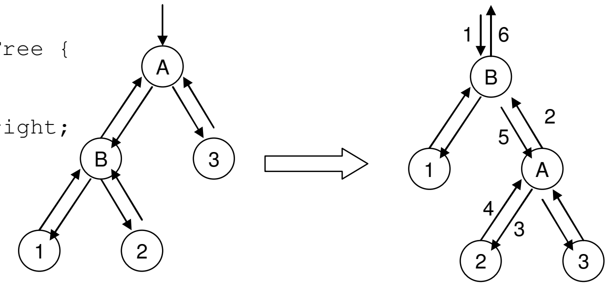
Value	Locations probed
4371	1
1323	3
6173	3, 4 (because 7 – 6173mod7 = 1)
4199	9
4344	4, 7 (because 7 – 4344mod7 = 3)
9679	9, 1, 3, 5 (because 7 –9679mod7 = 2)
1989	9, 5, 1, 7, 3, 9 (because 7 – 6173mod7 = 1) – no space found

Array:

0	1	2	3	4	5	6	7	8	9
	4371		1323	6173	9679		4344		4199

6. (10 pts) Using the (space-wasting) C++ tree and node classes below, implement `rotateRight()` so that `node.rotateRight()` rotates `node` clockwise (rightward) through its parent. Each node has a pointer to its parent and a flag indicating if it's a right child of the parent. <u>Drawing a picture first is not required but strongly recommended.</u> Be sure to update all affected fields of all affected nodes.

```
template <typename T> class Tree {
  private:
    struct Node {
      Node *parent, *left, *right;
      bool isRight;
      T data;
      void rotateRight();
    …};
    Node * root;
…};
```




```
template <typename T> void Tree::Node::rotateRight()
{
   Node * temp = right;
   right = parent; // link 5
   parent = right->parent; // link 6
   right->parent = right->left; // link 2
   if (isRight) parent->right = right->left; // link 1
   else parent->left = right->left; // link 1
   right->left = temp; // link 3
   if (right->left) {
     right->left->parent = right; // link 4
     right->left->isRight = false; // 2 isRight flag
   }
   isRight = right->isRight; // B isRight flag
   right->isRight = true; // A isRight flag
}
```

Comment [CKR9]: Lots of ways to do this. Most common mistakes:
- Updating only 3 links, missing the 3 backlinks
- Not updating the pointers to/from the top node
- Not updating the boolean isRight flags (there are 3)
- Testing isRight rather than parent->isRight; isRight for the nodecan only be false if it's going to be rotated right
- Not checking for null before updating link from node 2

Comment [CKR10]: No IF statement needed to do this.