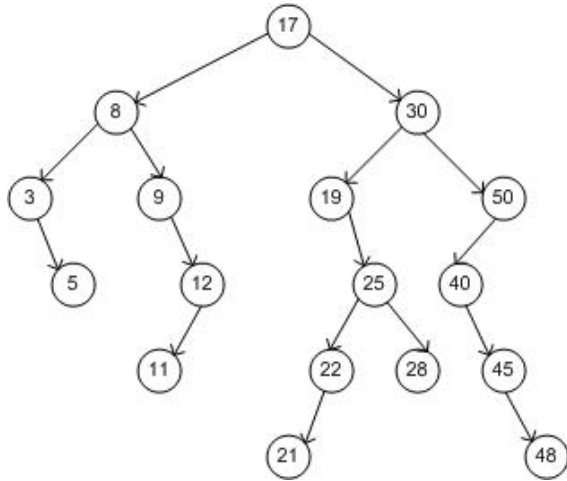
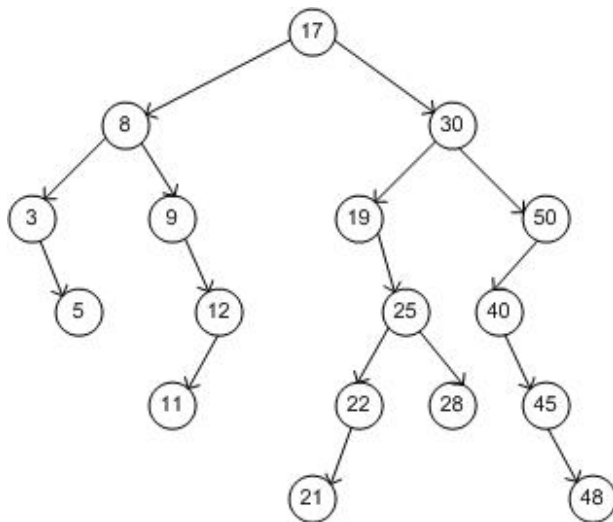


**CS 2420 – Written 3**  
**10 Points**  
**Due September 25, 2008**

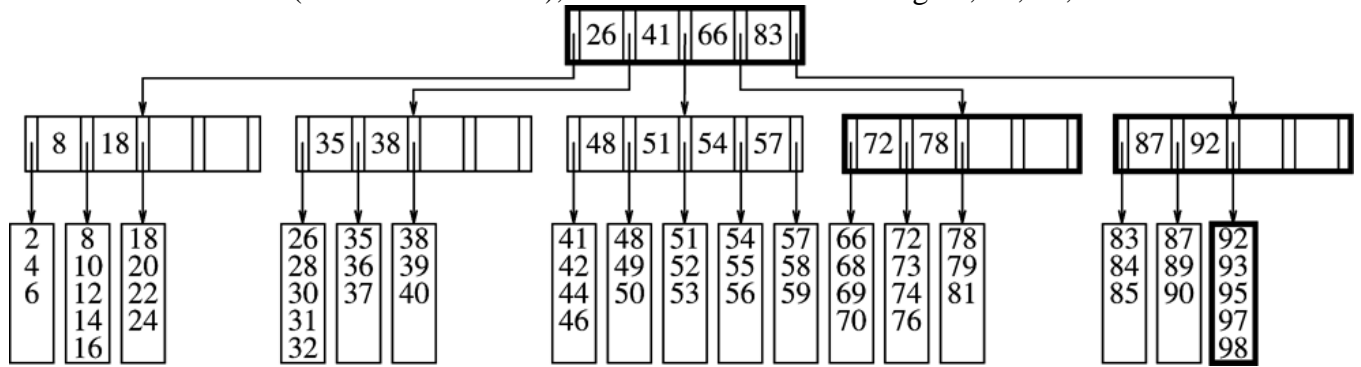
1. Show the result of inserting 10 into the top-down splay tree shown below.



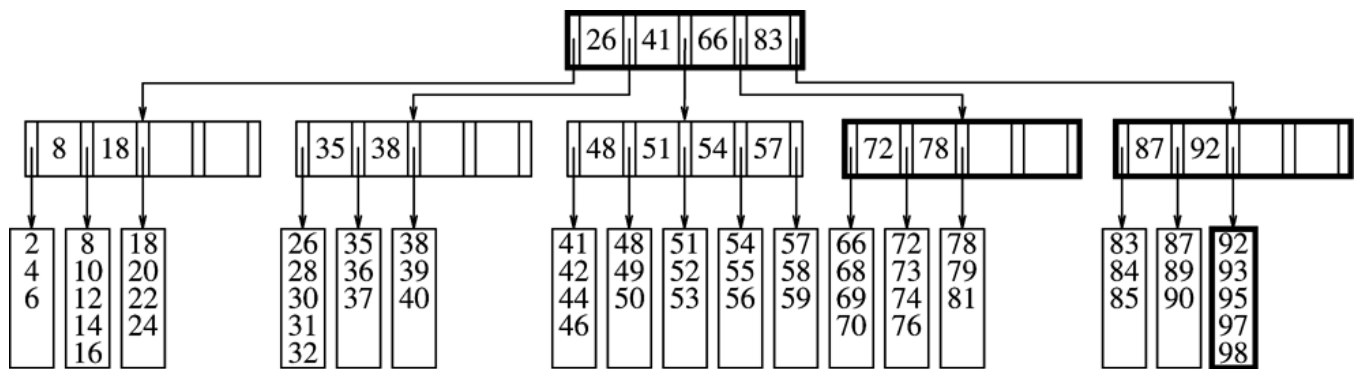
2. Show the result of deleteMax from the top-down splay tree show below:



3. For the B-tree below (with  $L=5$  and  $M=5$ ), show the tree after inserting 60, 61, 62, 100.



4. For the B-tree below (with  $L=5$  and  $M=5$ ), show the tree after deleting 98, 85, 83, 49, 50.



5. The definitions of primary and secondary clustering are sometimes difficult to get straight.

In primary clustering, not only do items collide because they hash to the same location, but one item may collide with the alternate location for another. This occurs in linear probing. When the item that hashes to  $x$  is placed in  $y$ , items that hash to either  $x$  or  $y$  both try the same next location.

In secondary clustering, two items that hash to the same location follow the same probe sequence for subsequent locations. This happens in quadratic probing. Only items that hash to location  $x$  continue to try the same sequence of alternate locations. If something has primary clustering, it will also have secondary clustering.

If a hash method does not exhibit primary or secondary clustering, it is termed non-clustering. This happens with double hashing. Even items which hash to the same location do not continue to try the same sequence of alternate locations because each is using a different step value.

Notice that non-clustering methods are the best, and methods that exhibit primary clustering are the worst.

Consider the following collision resolution schemes. Indicate whether they are non-clustering, secondary clustering, or primary clustering.

- You use linear probing, but always increment by 3 (rather than 1) in the event of a collision.

- b. You have 10 completely different hash functions. If the first method produces a collision, you try the second, and so on until you find a spot or until all hash functions are exhausted. At that point, the hash fails and you will need to rehash.
  - c. You have an overflow area at the bottom of the table. Instead of using open addressing (placing items that won't fit in the computed address in an open space), you put all items that won't fit in the computed address in this overflow area (sequentially).
6. A hash function is appropriate if it is (a) fast to compute, (b) minimizes collisions, (c) is repeatable, and (d) produces values between 0 and the size of the table. All of the following hash functions are inappropriate. Explain why. (Note, when we say a function minimizes collisions, we are saying, that for a reasonable number of typical data items, the number of collisions is not great. Anything can be forced to collide if you use too much data for the table or pick atypical data items.) All collisions are handled by double hashing.
- a. The hash table has a size of 2,047. The search keys are identifier names in C++. The hash function is  $h(\text{key}) = (\text{position of the first letter of key in alphabet}) \% \text{size}$ .
  - b. The hash table is 1000 entries long. The search keys are integers between 0 and 9999. The hash function is  $h(\text{key}) = (\text{key} * \text{random}) \% \text{tablesize}$  where random is a sophisticated random-number generator that returns a real value between 0 and .1.
  - c. The hash table is 10,007 entries long. The search keys are integers between 0 and 9999. The hash function is given by
 

```
const int TABLESIZE = 10007;
int Hash(int X)
{ for (int l = 1; l <= 1000; l++)
  X = X * X;
  return X % TABLESIZE;
}
```
  - d. The hash table is 10,007 entries long. The search keys are strings less than 10 characters in length.
 

```
const int TABLESIZE = 10007;
int Hash(string s )
{ tot = 0;
  for (int i = 1; i < s.length; i++)
    tot += s[i];
  return tot % TABLESIZE;
}
```

7. Using a hash table with eleven locations and hashing function  $h(i) = i \% 11$ , show the hash table that results when the following integers are inserted in the order given:  
 26, 42, 5, 44, 92, 59, 40, 36, 12.

Show the hash table using:

- a. Linear probing
- b. Quadratic probing
- c. Double hashing using the secondary hash function  $h_2 = x \% 9 + 1$  to compute the personalized step.
- d. Chaining

	Linear Probing	Quadratic Probing	Double Hashing	Chaining
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

