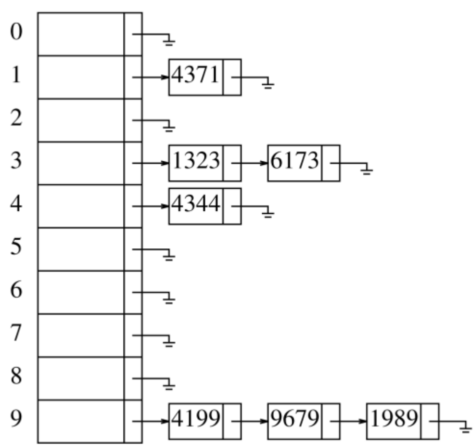
Assignment 5 -Key

1. **[5 Points]** Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and a hash function h(x) = x mod 10, show the resulting:
    a. Separate Chaining hash table
    b. Hash Table using linear probing
    c. Hash table using quadratic probing
    d. Hash table with second hash function $h_2(x) = 7 – (x \bmod 7)$

(A) On the assumption that we add collisions to the end of the list (which is the easier way if a hash table is being built by hand), the separate chaining hash table that results is shown here.



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

(b)

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

(c)

(d) 1989 cannot be inserted into the table because $hash_2(1989) = 6$, and the alternative locations 5, 1, 7, and 3 are already taken. The table at this point is as follows:

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

2. **[10 Points]** A min-max heap is a data Structure that supports both deleteMin and deleteMax in O(logN) per operation. The structure is identical to a binary heap, but the heap-order property is that for any node, X, at even depth, the element stored at X is smaller than the parent but larger than the grandparent (where this makes sense), and for any node X at odd depth, the element stored at X is larger than the parent but smaller than grandparent. See Fig.

a. How do we find the minimum and maximum element?

b. Give an algorithm to insert a new node into the min-max heap.

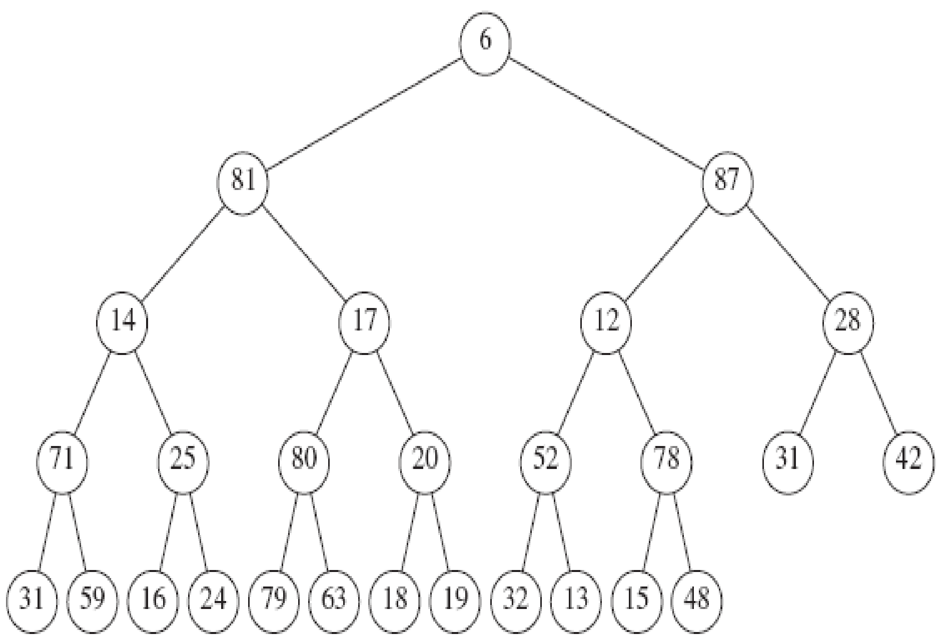


Figure 6.57 Min-max heap

Finding minimum is easy it is at the root. The maximum is max(2nd and 3rd) nodes, i.e. array[1] is minimum and max(array[2], array[3]) is maximum.

To insert element X we create a hole in next available slot if X can inserted into the hole with no violation of order property then we are done. If not we percolate the hole up.

```
public void insert(int x){

      int hole = ++currentSize;

int height = Math.log(currentSize);

boolean min = true;

if (height%2 ) min = false;


      while(hole > 1){

            //this boolean should help swap between whether we want to swap
for the            smallest element or for the largest element

            if (min){

                  if(x > array[hole/2] ){

                  array[hole] = array[hole/2];

                  hole = hole/2}

                  else if( x < array[hole/4]){

                        array[hole] = array[hole/4];

                  hole =  hole/4}  // check if x is less than the grand
            parent

            else { break ; }


            }

            else {

                  if(x < array[hole/2] ){

                  array[hole] = array[hole/2];

                  hole = hole/2}

                  else if( x > array[hole/4]){

                        array[hole] = array[hole/4];

                  hole =  hole/4}  // check if x is greater than the grand
            parent
```

```
            else { break ; }
        }
        array[hole] = x;
}
```

3. **[5 Points]** Merge the two binomial queues

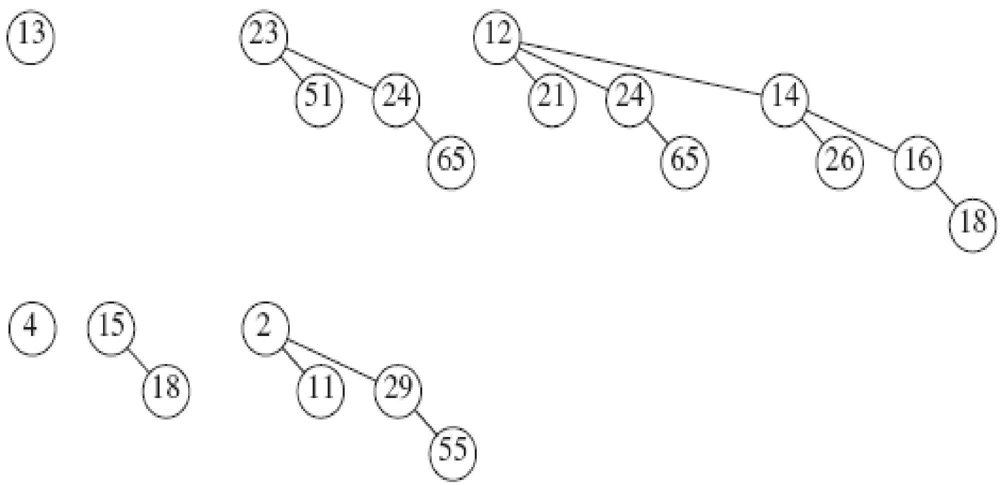
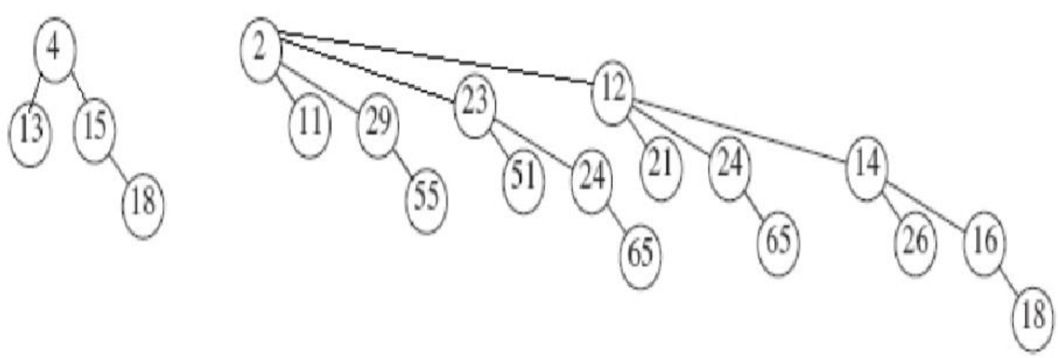
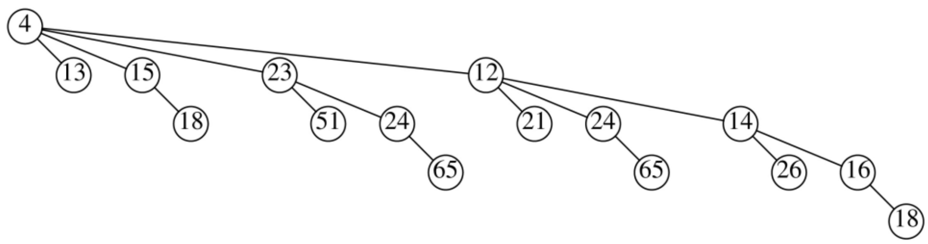
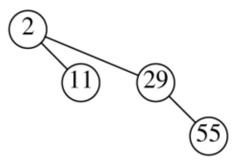


Figure 6.59 Input for Exercise 6.32



or a different possibility is





23 as one of the roots is another possibility.

4. **[10 Points]** Give an algorithm to find all nodes less than some value X, in a binary heap. Your algorithm should run in O(K), where K is number of nodes output.

   Perform a preorder traversal of the heap.