UNIT – I

BINOMIAL QUEUE OPERATIONS:

- **Find – min:**

This is implemented by scanning the roots of the entire tree. Since there are at most log n different trees, the minimum can be found in O(log n) time.

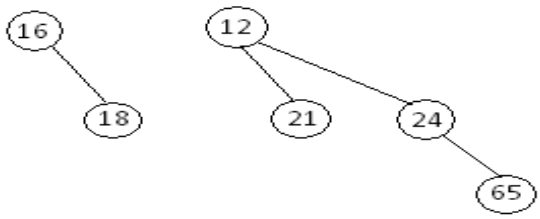
Alternatively, one can keep track of the current minimum and perform find – min in O(1) time. If we remember to update the minimum if it changes during other operations.

- **Merge:**

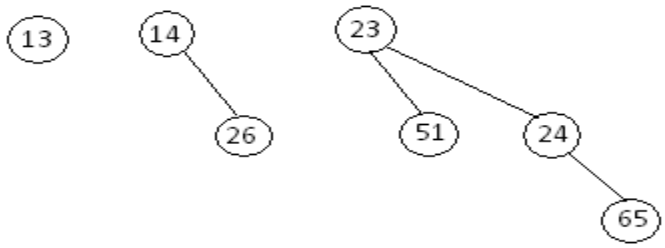
Merging two binomial queues is a conceptually easy operation, which we will describe by example.

Consider the two binomial queues H1 and H2 with six and seven elements respectively as shown below.

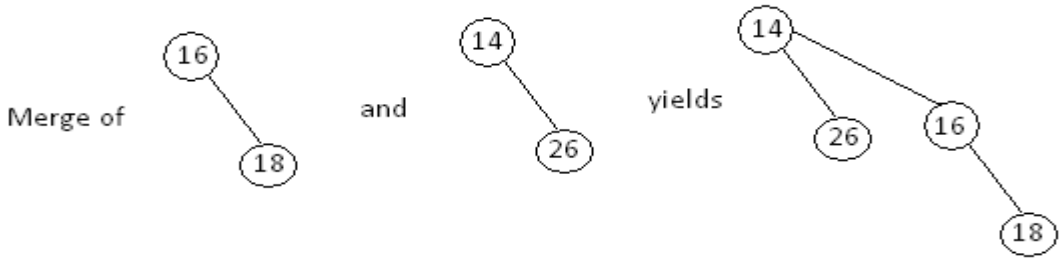
H1: with 6 elements



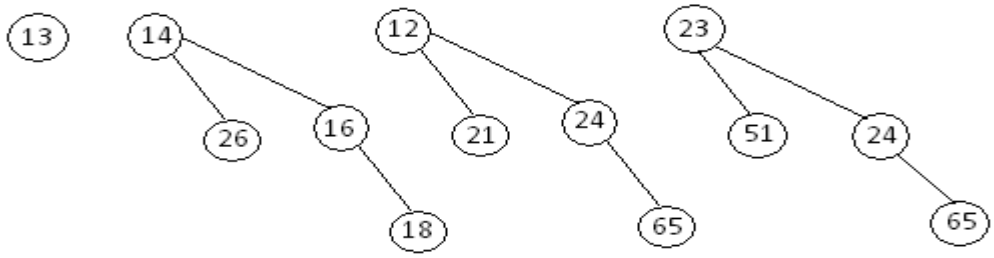
H2: with 7 elements



Merge of two B1 trees (i.e. 21 = 2 nodes) in H1 and H2. i.e.

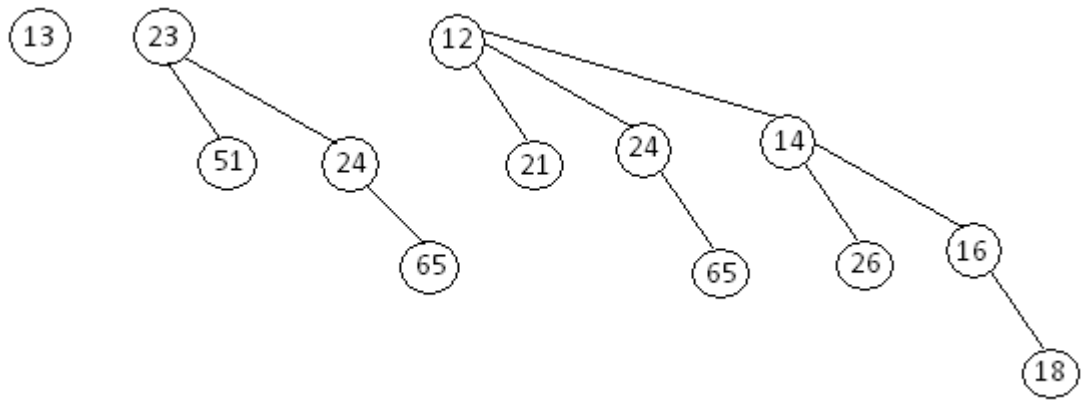


Now we left with 1 tree of height 0 and 3 trees of height 2



Binomial queue H3: the result of merging H1 and H2

H3: with 13 elements



The merging is performed by essentially adding the two queues together.
Let H3 be the new binomial queue.

Since H1 has no binomial tree of height 0, and H2 does, we can just use the binomial tree of height o in H2 as part of H3.

Next, we add binomial trees of height 1.

Since both H1 and H2 have binomial tree of height 1, we merge them by making the larger root a sub tree of the smaller, creating a binomial tree of height 2.

Thus, H3 will not have a binomial tree of height 1 as shown in the above diagrams.

There are now three binomial trees of height 2, namely, the original trees in both H1 and H2 plus the tree formed by adding of height 1 tree in both H1 and H2.

We keep one binomial tree of height 2 in H3 and merge the other two, creating a binomial tree of height 3.

Since H1 and H2 have no trees of height 3, this tree becomes part of H3 and we are finished. The resulting binomial queue is as shown in above figure.

Since merging two binomial trees takes constant time with almost any reasonable implementation, and there are O(log n) binomial tree, the merge takes O(log n) time in the worst case.

To make this operation efficient, we need to keep the trees in the binomial queue sorted by height, which is certainly a simple thing to do.

➢ Insertion:

Insertion is a special case of merging, since we merely create a one – node tree and perform a merge.

The worst – case time of this operation is likewise O(log n)

More precisely, if the priority queue into which the element is being inserted has the property that the smallest non existent binomial tree is Bi the running time is proportional to i+1.

For example:

In The previous example, H3 is missing a binomial tree of height 1, so the insertion will terminate in two steps. Since each tree in a binomial queue is present with probability ½.

If we define the random variable X as representing the no. of steps in an insert operation, then

X = 1 with probability 1/2 (B0 not present)

X = 2 with probability 1/2 (B0 and B1 not present)

X = 3 with probability 1/8

Thus average number of steps in an insert operation = 2.

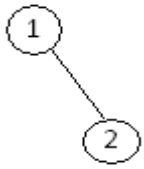
Thus we expect an insertion to terminate in two steps on the average. Further more performing n inserts on an initially empty binomial queue will take O(n) worst case time.

In deed it is possible to do this operation using only (n - 1) comparisons.

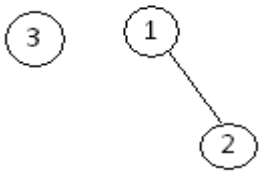
Consider an example, the binomial queue that are formed by inserting 1 through 7 in order.

After 1 is inserted:

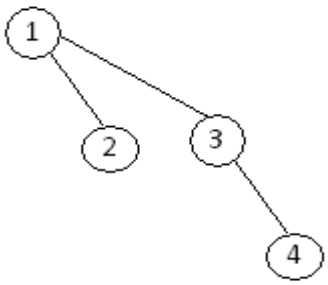
After 2 is inserted:



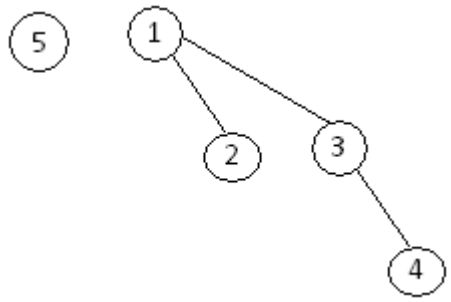
After 3 is inserted:



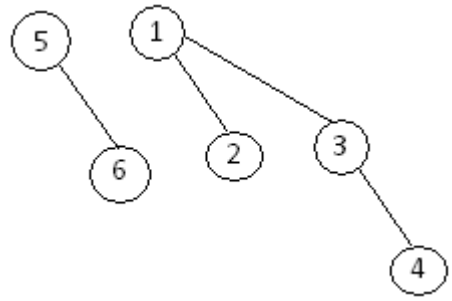
After 4 is inserted:



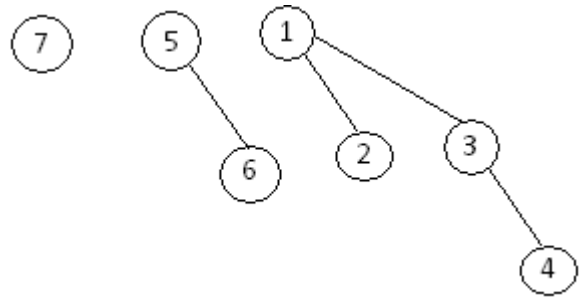
After 5 is inserted:



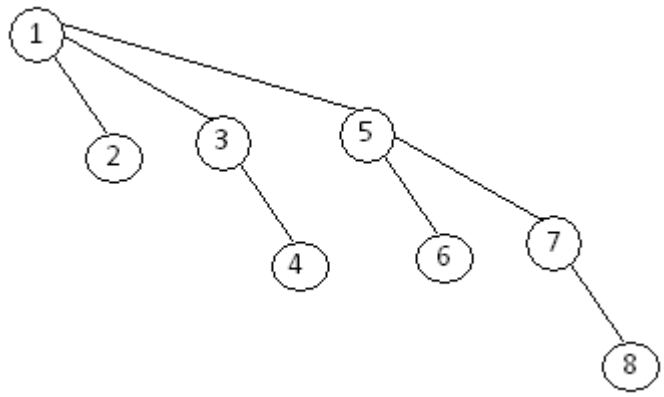
After 6 is inserted:



After 7 is inserted:



If we insert 8 then



Inserting 4 shows off a bad case, we merge 4 with B0 obtaining a new tree of height 1. We merge this tree with B1 obtaining a tree of height 2 which is the new priority queue.

The next insertion after 7 is another bad case and would require three merges.

Delete min:

A delete min can be performed by first finding the binomial tree with the smallest root.

Let this tree be Bk and let the original priority queue be H

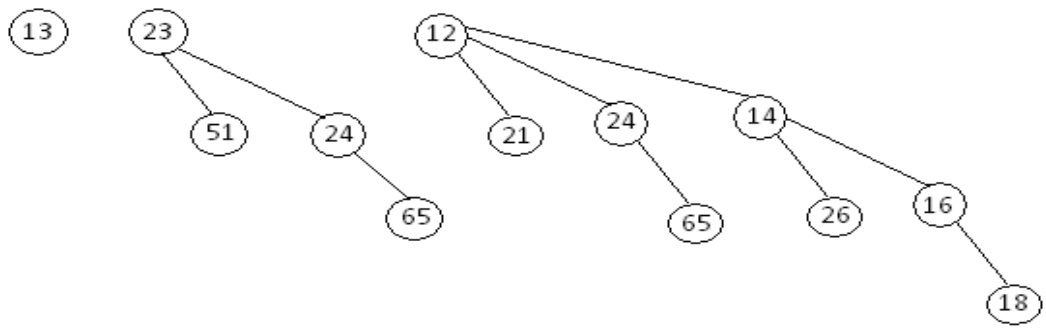
Remove the binomial tree Bk from the forest of trees in H forming the new binomial queue H′

Now remove the root of Bk creating binomial trees B0 B1 - - - Bk – 1 which collectively from priority queue H″.

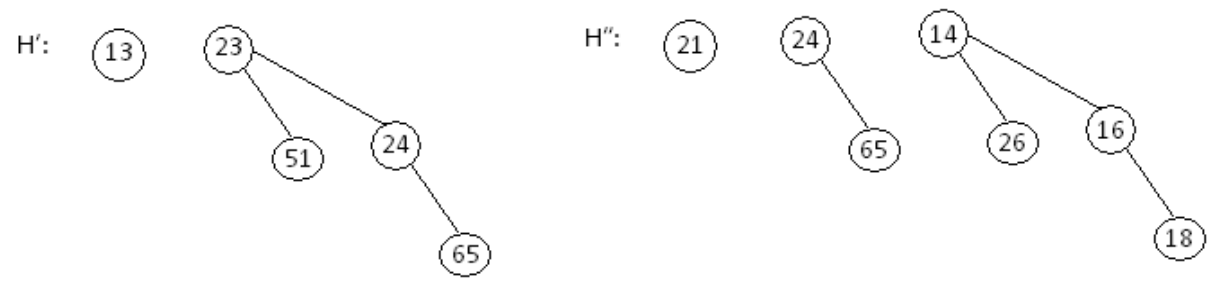
Finish the operation by merging H′ & H″

Consider the same example of merge operation which has H3.

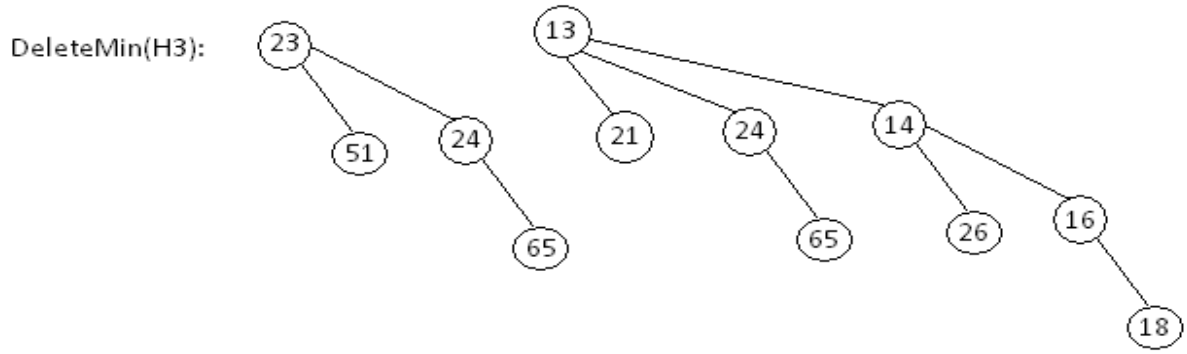
H3:



The minimum root is 12 so we obtain the two priority queues H′ & H″



The binomial queue that results from merging H′ & H″ is as shown below



NOTE: The entire delete min operation takes O(log n) worst case time

Binomial Amortized Analysis:

Reference Links

1. http://lcm.csa.iisc.ernet.in/dsa/node141.html
2. https://www.cs.princeton.edu/~appel/Binom.pdf
3. http://ac.informatik.unifreiburg.de/lak_teaching/ws07_08/algotheo/Slides/07_binomial_queues.pdf

Video Links

1. https://www.youtube.com/watch?v=HBZUHwDFPrE
2. https://www.youtube.com/watch?v=e_gh1aD4v-A

Questions

1. Explain Binomial queue deletion operation and construct the Binomial queue with the following elements and perform Delete Min operation.
   13, 23, 51, 24, 65, 12, 21, 24, 14, 65, 26, 16, 18
2. Construct Binomial Queue with the elements :
   18,3,37,6,8,30,45,55,32,23,24,29,31,48,50,10,17,44

Explain the implementation of a binomial heap and its operation with suitable example