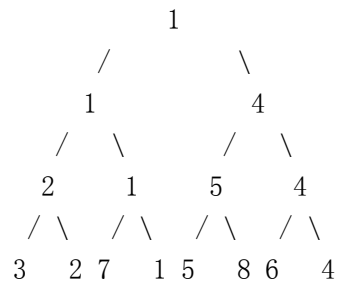
Exam 1 solution:

1.

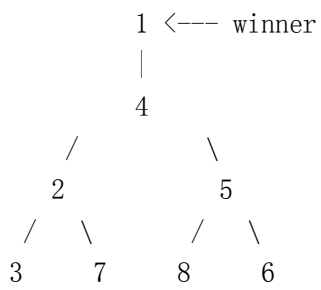
Note that the cost of relocating elements adds much less than a constant fraction of the cost of building the new array, so we can neglect the Theta(n) term and concentrate on the Theta(n^2) one.

Observe that the worst case will be when the last element inserted causes the array to square, as subsequent insertions will only increase the denominator without increasing the cost much. In this case,
the total cost of squarings is O(N^2) + O(N) + O(sqrt(N)) + O(N1/4) + ...,
and it is not hard to show that this sum is O(N^2)
(for example, bound each term O(N^(2/2i)) by O(N^2/2^i) and use the geometric series formula).
Dividing this cost over N insertions gives a cost of O(N) per insertion.

2. (a)



(b)

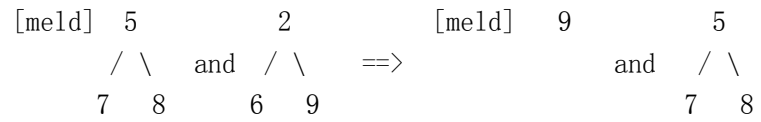


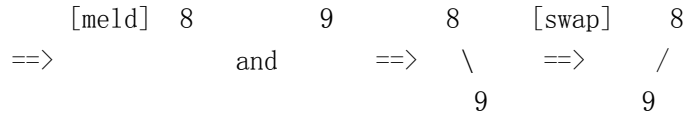
(c) We can use a loser tree to solve FirstFir Bin Packing heuristics as follows:

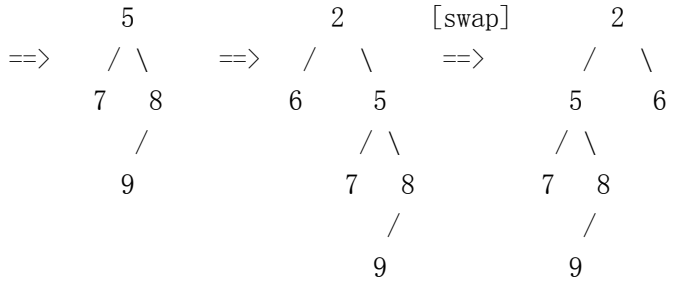
- Bins are arranged in left to right order
- Items are packed one at a time in given order.
- Construct a loser tree of these items. Winner (current item)

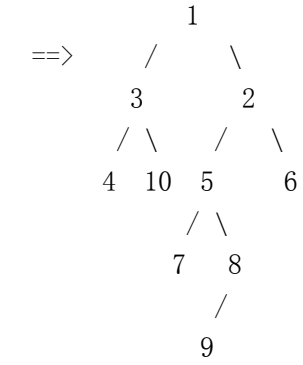
is packed into the leftmost bin into which it fits.
- If there is no bin into which current item fits, start a new bin.

3. (a) Please remember that when you meld two leftist trees, do meld right subtree of the tree with smaller root and all of the other tree.









(b)

```
int ConvertBinaryToLeftist(BinaryTreePtr t)
{
  BinaryTreePtr temp;
  int sh, lsh, rsh;
```

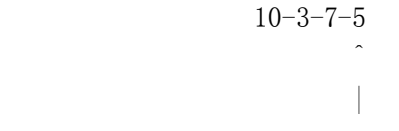
```
        if (t == NULL) sh = 0;
        else
        {
          lsh = ConvertBinaryToLeftist( t->LeftChild );
          rsh = ConvertBinaryToLeftist( t->RightChild );

          if ( lsh < rsh )
          {
            temp = t->LeftChild;
            t->LeftChild = t->RightChild;
            t->RightChild = temp;
            rsh = lsh;
          }
          t->Shortest = rsh+1;
          sh = t->Shortest;
        }
        return sh;
      }
```

Analysis : ConvertBinaryToLeftist traverses every node in the tree, doing constant work at each node. So the complexity is O(n).

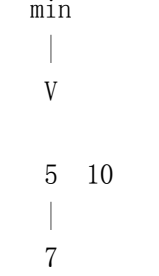
4.

Insert 10,3,7, and 5

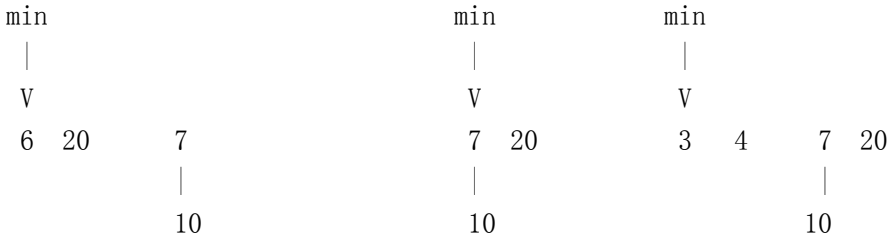


RemoveMin:

RemoveMinMin:



Insert 6 and 20 ---> RemoveMin: -----> Insert: 3 and 4



-5:: combine for Removemin
-2:: min value point
-5:: insert operation