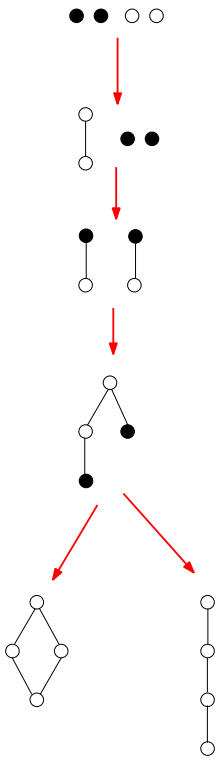
CSC 226: Spring 2016
Solution to Assignment 2

February 14, 2016

In the style of the sorting hand-out, write out the tree of posets for the min-max problem when $n = 4$.



Determine the smallest value of n for which mmA and mmB use a different number of comparisons.

For $n = 6$, mmA needs 8 comparisons and mmB should need 7 comparisons.

What is the minimum number of inversions of a permutation of $1, 2, , n$.

The minimum number of inversions is 0, when the numbers are in ascending order.

What is the maximum number of inversions of a permutation of $1, 2, , n$.

The maximum number of inversions is $1+2+\ldots+(n-1) = n(n-1)/2$, when the numbers are in descending order (i.e., for the permutation $n, n-1, \ldots, 2, 1$).

Explain carefully how to use red-black trees to compute the number of inversions in a permutation time $O(n \log n)$. Effectively, you may need to modify the code for Algorithm 3.4 on page 439. Explain in detail any changes that you would make to put.

Essentially, when you insert a build the red-black tree from a permutation, after inserting each node you count how many nodes are there in the tree that are bigger than that node. You add that count to some global variable and after inserting all the nodes, you have the total number of insertions.

One way of doing that would be adding the size of the left subtree plus 1, when you recurse on the right subtree. So your code would be like this:

```
private Node put(Node h, Key key, Value val)
{
if (h == null) // Do standard insert, with red link to parent.
return new Node(key, val, 1, RED);

int cmp = key.compareTo(h.key);
if (cmp < 0) h.left = put(h.left, key, val);
else if (cmp > 0){
inversions+=size(h.left)+1;
h.right = put(h.right, key, val);
}
else h.val = val;

if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
if (isRed(h.left) && isRed(h.right)) flipColors(h);

h.N = size(h.left) + size(h.right) + 1;
return h;
}
```

Since you are just adding a constant number of operations, your code would still be $O(n \log n)$ time.

Another way would be using the rank function directly. Like this:

```
public void put(Key key, Value val)
```

```
{ // Search for key. Update value if found; grow table if new.
root = put(root, key, val);
root.color = BLACK;
inversions+=size(root)-rank(key)-1;
}
```