

Homework 12 Solutions

#11: The solution in your book contains an error. The results of the recursive calls should be multiplied by 2, not by $2x$.

#28: The recurrence for $P_{m,n}$ is slightly simpler if you include cases where m or n is zero, with the understanding that there is one partition of zero, namely the empty partition. Here's a recursive algorithm based on that modification.

```

procedure partitions( $m$ )
return partitions1( $m, m$ )

procedure partitions1( $m, n$ )
if  $m = 0$  return 1
else if  $n = 0$  return 0
else if  $n > m$  return partitions( $m$ )
else return partitions1( $m - n, n$ ) + partitions1( $m, n - 1$ )
    
```

This algorithm is not efficient. An iterative algorithm would be much faster, using the recursive definition of $P_{m,n}$ to compute a table of $P_{j,k}$ for all $0 \leq k \leq j, j \leq m, k \leq n$.

#38: Here are the partially sorted sublists at each level of recurrence:

```

          357819246
        12 3 578946
       1 2 3 4 5 7896
      1 2 3 4 5 6 7 89
     1 2 3 4 5 6 7 8 9
    
```

#43: The worst case occurs when the list is already in order! Since it is $O(n^2)$, quicksort is not an efficient algorithm in the worst case. However, it can be shown to have average case running time $O(n \log n)$ on random input. The algorithm itself can also be randomized by choosing a random a_i as the threshold to split the list into sublists, instead of a_1 . The randomized version almost always runs in time $O(n \log n)$ on any input. One reason quicksort is popular is that it can be implemented to sort a list in place, without using extra memory.

Extra problem: If $n = 1$, there is nothing to do.

If $n > 1$, divide your coins into two equal piles A and B of size $\lceil n/3 \rceil$ and a third pile C of size $n - 2\lceil n/3 \rceil$. Weigh A against B . If they balance, the bad coin is in pile C . If they don't, the bad coin is in the lighter of piles A and B . In every case, proceed recursively with the pile that contains the bad coin.

Now we prove that the number of weighings required is at most $\lceil \log_3 n \rceil$. Equivalently, we are to show that that if $n \leq 3^N$, then we use at most N weighings. If $n = 1$, we use $N = 0$ weighings, so this case is correct. If $n > 1$, we use $1 + M$ weighings, where M is the number of weighings for the recursive call. In every case, the number of coins for the recursive call is $m \leq \lceil n/3 \rceil$, since $n - 2\lceil n/3 \rceil \leq n - 2(n/3) = n/3 \leq \lceil n/3 \rceil$. Note that $\lceil 3^N/3 \rceil = 3^N/3 = 3^{N-1}$, and $\lceil n/3 \rceil$ is an increasing function of n , so if $n \leq 3^N$, then

$\lceil n/3 \rceil \leq 3^{N-1}$. It follows by induction that the recursive call takes at most $M = N - 1$ weighings, and the original problem therefore takes at most N weighings.

Finally, we prove that it is impossible to do better than this in general. There are n possibilities for which coin is counterfeit. Each weighing has one of three outcomes (pile 1 heavy, pile 2 heavy, or balance). A sequence of N weighings has 3^N possible outcomes. Therefore any algorithm that can identify the counterfeit coin using at most N weighings must have $3^N \geq n$, or equivalently $N \geq \lceil \log_3 n \rceil$.