Homework Set 1, Quarter 2

Jay R Bolton

March 31, 2012

Note: The following is incomplete, but I'm submitting it as reassurance that it is being worked on. The complete version is coming soon.

Problems: 6.16, 6.17, 6.22, 6.23, 6.24, 6.27, 6.28, 6.29

6.16 Prove that the number of comparisons used in the worst case by mergesort is $N\lceil logN\rceil - 2^{\lceil log(N)\rceil} + 1$.

I guess I have to figure out what constants he left out. Look at the code?

6.17 *Quick sort 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 with a cutoff of 3.* I'll do the simple version with list copying. When doing median on duplicates, I'll choose the last duplicate.

A right arrow represents the application of insertion sort when we're at the cutoff.

```
Pivot: 5
Left: [3, 1, 4, 1, 5, 2, 5, 3]
  Pivot: 3
  Left: [3, 1, 1, 2]
    Pivot: 2
    Left: [1, 1] -> [1, 1] (cutoff)
    Right: [3]
  Right: [4, 5, 5] -> [4, 5, 5] (cutoff)
Right: [9, 6] -> [6, 9] (cutoff)

Result: [1, 1, 2, 3, 3, 4, 5, 5, 5, 9, 6]
```

6.22 The worst case for quicksort will be an execution tree that is completely skewed, where the smallest or largest element is always chosen as the pivot. The list with all duplicates would then be a worst case:

```
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

6.23

```
-- quickselect algorithm implemented from the guidelines in: Data Structures
-- and Algorithm Analysis in C

-- J Bolton

-- find the kth smallest element in xs
-- won't check for out of bounds
quickselect [x] _ = x
quickselect (x:xs) k
 | k <= length s1 = quickselect s1 k
 | k == 1 + (length s1) = x
 | otherwise = quickselect s2 (k - (length s1) - 1)
 where
 ys op = [y | y <- xs, y `op` x]
 s1 = ys (<=)
 s2 = ys (>)
```

```
-- i also wanted to see if i could quickly write quicksort in haskell.
-- i'll use the quick/inefficient one-liner version of quicksort.
-- i threw in a nested higher order function for bonus. i don't think people
-- usually write it that way. I think it looks kind of cool.
quicksort [] = []
quicksort (x:xs) = (ys (<=)) ++ [x] ++ (ys (>))
 where ys op = quicksort [y | y <- xs, y `op` x]

-- tests
-- this should select each element from 1st to 8th in proper order
xs = [8, 3, 6, 3, 1, 0, 4, 6]
test = quicksort xs == map (quickselect xs) [1..8]
-- returns True
```

6.24 Solve the recurrence (I assume he means get the closed form):

$$T(0) = 0$$
$$T(N) = (1/N)\left(\sum_{i=0}^{N-1} T(i)\right) + cN$$

I think I need to read up more on how to solve this stuff.

6.27 Relate it to binary sort?

6.28 WAT? Find the section on this shit.

6.29 *a. In how many ways can two sorted arrays of N elements be merged?*

I will guess that he's asking how many ways can two lists of elements be interspersed maintaining original order.

If we don't compare the two each corresponding elements, resulting in an unordered list, then it's $2N^2 - 2$. This is because we distribute N elements from the first list N times throughout the second list. We then do this for the inverse. We subtract 2 because we will produce 2 duplicates this way: one duplicate each for the arrays concatenated.

If we're talking about merging two sorted lists into one ordered list using merge sort, then there is only one way to do it: whatever results in the ordered lists.

If we count duplicates as "different ways to merge to sorted lists", since you can add either duplicate arbitrarily, then it's going to be $D^2 - 1$, where D is the number of corresponding duplicates. It's just like binary numbers. For the upper bound just replace D with N. This probably isn't what he's asking though.

b. Give a nontrivial lower bound on the number of comparisons required to merge two sorted lists of N elements.

We're talking lower bound, so best case. It seems that in merge sort you do a comparison at every step, so the lowest possible would be the same as the most possible: $N - 1$ comparisons? There must be something more to this.

I wish he'd give more description in his problems.