Analysis of Algorithms Final

Jay R Bolton

June 8, 2012

1 ***(a)*** (Each inner node has 7 children)

```
                                cn^2                       ...7^0 * (cn^2/3^(2*0))
h=log_3 7                      /    \
                              /      \
                     c(n/3)^2   ... c(n/3)^2              ...7^1 * (cn^2/3^(2*1)
                      /       \        /       \
                     /         \      /         \
           c(n/9)^2  ... c(n/9)^2  c(n/9)^2 ... c(n/9)^2     ...7^2 * (cn^2/3^(2*2))
               ...          ...      ...      ...        ...leaves=n^(log_3 7)
```

Width of leaves, each with $T(1) = \Theta(1)$, is $n^{log_3 7}$. Height of the tree is $log_3 n$.
The summation form is $T(n) = \sum_{i=0}^{h} 7^i \frac{n}{3^i}^2$.
(I should learn how to make prettier trees)

(b)

$$\sum_{i=0}^{log_3 n} \frac{7^i cn^2}{3^{i2}} + n^{log_3 7}$$
$$< cn^2 \sum_{i=0}^{\infty} \frac{7^i}{9^i} + n^{log_3 7}$$
$$= cn^2 \frac{1}{1 - 7/9} + n^{log_3 7}$$
$$= 3cn^2 + n^{log_3 7}$$
$$< 4cn^2$$
$$= \mathcal{O}(n^2)$$

(c) Induction for $\mathcal{O}$

Inductive Hypothesis: $T(n) \geq cn^2$

Induction: $T(n) \geq 7c\frac{n^2}{3^2} + dn^2$

$$= cn^2 \frac{7}{9} + dn^2$$
$$\geq cn^2 \quad \text{with } c \leq 4 \text{ and } n \geq 0$$

Induction for Ω

Inductive Hypothesis: $T(n) \leq cn^2$

Induction: $T(n) \leq 7c\frac{n^2}{3^2} + dn^2$

$$= cn^2 \frac{7}{9} + dn^2 \qquad \leq cn^2 \quad \text{with } c \geq \frac{9}{3}d \text{ and } n \geq 0$$

(d)

$$a = 7,\ b = 3$$
$$f(n) = n^2 = \Omega(n^{log_3 7+\epsilon})$$
$$lim_{n\to\infty}\frac{n^2}{n^{log_3 7+\epsilon}} < \infty \text{ (it is polynomially larger)}$$
Regularity condition:
$$7\frac{n^2}{3^2} \leq cn^2$$
$$\frac{7}{9}n^2 \leq cn^2 \qquad \text{for } c \geq 7/9 \text{ and } c \leq 1 \text{ and } n \geq 0$$
(Passes)

Case 3: $\Theta(n^2)$

2 ***(a)*** I'll use the four conditions listed in our book on page 379. It took me a bit to realize this was not a greedy choice problem. Tricky!

i. Our inital choice can be a division that we make of C, where $C_{choice} < C$ is optimal and $C - C_{choice}$ is our subproblem.
ii. The optimal choice given to us would be the sum C_{choice}, which is less than C so divides it at some point.
iii. The subproblem that ensues is $C - C_{choice}$ which is the remaining sum we have yet to find an optimum on.
iv. Suppose we have come to an optimal solution with suboptimal C_{choice} and $C - C_{choice}$. If we "cut" away our two suboptimal subproblems and replace them with more optimal ones, then we would supposedly increase the optimality of the whole problem. But this is a contradiction of our suppostion that we had an optimal solution.

(b) In haskell:

```
vs = [1,5,10,25,100,200]

change 0 _ = 0
change c [1] = c
change c v
 | last v <= c = min (change c (init v)) (1 + change (c-(last v)) v)
 | last v > c  = change c (init v)

-- >> change 7 vs
-- >> 3
-- >> change 42 vs
-- >> 5
```

With memoization:

```
-- Generate our memoization matrix.
-- There's probably a prettier way to do it, perhaps with list comp.
matrix c v = map (\(n,cs) -> (map (\c -> (n,c)) cs)) (zip [1..n] (replicate n [c,c-1..1]))
 where n = length v
change_matrix c v = map (map (\(x,y) -> ch y (take x v))) (matrix c v)
 where
 ch 0 _ = 0
 ch c [1] = c
 ch c v
  | last v <= c = min (mchange c (init v)) (1 + mchange (c-(last v)) v)
  | last v > c  = mchange c (init v)
  where n = length v
```

```
mchange c v = (change_matrix c v) !! (length v - 1) !! 0

-- >> mchange 7 vs
-- >> 3
-- >> mchange 42 vs
-- >> 5
```

That was fun. It works correctly but I'm not entirely sure it memoizes rather than recomputes the matrix every time. I've read before about using a fixed point function to do memoization in haskell that looked real neat but I probably don't have time to figure it out.

(c) For each coin $v(n)$ we have to consider $min(f(C-v(n),n), f(C-v(n-1),n), \ldots, f(C-v(1),n))$ where f is our choice function. That amounts to n total choices per coin.

(d) Since for each coin we must consider n subproblems, and the worst case is C total coins, our complexity would then be $\Theta(Cn)$ total choices overall.

Here is a greedy version of the algorithm I made before realizing they weren't optimal. Might as well leave them here.

Let C be our goal sum and $v(n)$ be the coin of n^{th} value.
$NCoins(C, v(1)) = C$
$NCoins(C, v(n)) = NCoins(C \; \% \; v(n), v(n-1)) + \lfloor C/v(n) \rfloor$

3 ***(a)***

(b)