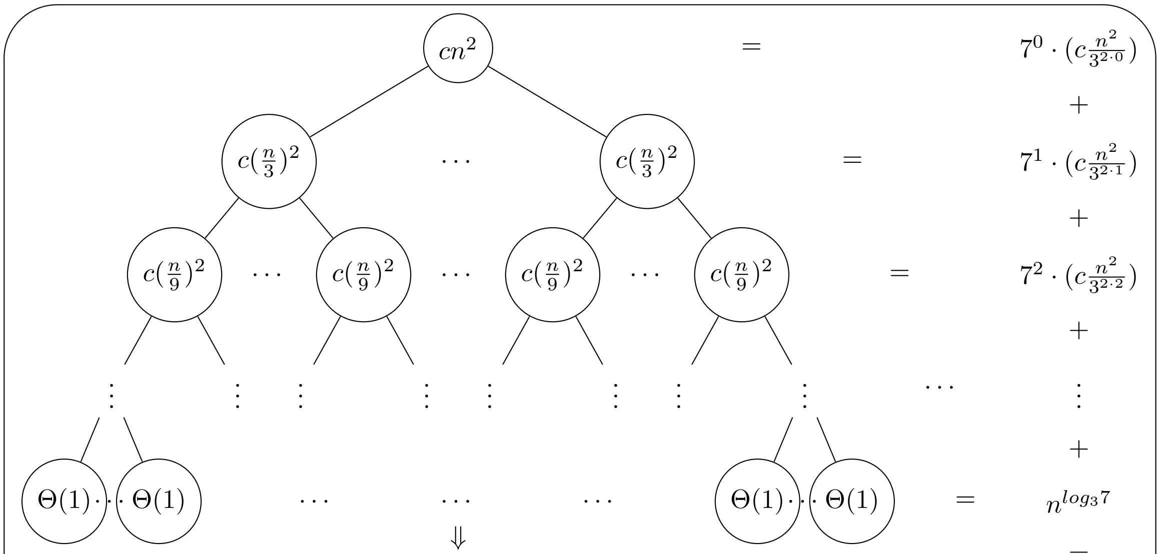
Analysis of Algorithms Final

Jay R Bolton

June 8, 2012

1 ***(a)*** *height* $= log_3 n$



$$n^{log_3 7} + \sum_{i=0}^{h} 7^i \cdot \frac{cn^2}{3^{2i}} = n^{log_3 7} + \sum_{i=0}^{h} cn^2 \cdot \frac{7^i}{9^i} \Leftrightarrow O(n^2)$$

(b)

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

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

$$\begin{aligned}
&\text{Inductive Hypothesis: } T(n) \geq cn^2 \\
&\text{Induction: } T(n) \geq 7c\frac{n^2}{3^2} + dn^2 \\
&= cn^2 \frac{7}{9} + dn^2 \\
&\geq cn^2 \quad \text{ with } c \leq 4d \text{ and } n \geq 0
\end{aligned}$$

Induction for Ω

$$\begin{aligned}
&\text{Inductive Hypothesis: } T(n) \leq cn^2 \\
&\text{Induction: } T(n) \leq 7c\frac{n^2}{3^2} + dn^2 \\
&= cn^2\frac{7}{9} + dn^2 \\
&\leq cn^2 \quad \text{with } c \geq 5d \text{ and } n \geq 0
\end{aligned}$$

(d)

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

2 ***(a)*** I'll use the four conditions listed in our book on page 379. It took me a bit to realize how 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:

```haskell
-- 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
-- Map our change function over the matrix
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

-- Get the cell in the matrix for which we used n coins on c sum
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 factor out memoization in haskell that looked really 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 it isn't optimal. Might as well leave it here.

Let C be our goal sum and $v(n)$ be the coin of n^{th} value.

$$NCoins(C, 1) = C$$

$$NCoins(C, n) = NCoins(C \;\%\; v(n), n-1) + \lfloor C/v(n) \rfloor$$

3 ***(a)***

(b)