CS 580 HW 1

Ankur Dhoot

January 20, 2017

Q1

T(n) = $\theta(n^{log_5 7}$lgn). Using the notation of the master method, a = 7, b = 5 so f(n) = $\theta(n^{log_5 7})$ putting us in case 2 of the master method.

Thus, $n^{log_5 7}$lgn is an asymptotic upper and lower bound on T(n) (i.e $c_1 n^{log_5 7}$lgn $\leq$ T(n) $\leq$ $c_2 n^{log_5 7}$lgn for n > n_0, it's trivial to see c_1 = 1/2, c_2 = 2 and n_0 = 5 suffice as constants)

Q2

Claim: T(n) $\leq$ 45/8 cn

Proof: By induction on n

Base case:

T(n) = c $\leq$ 45/8 cn for n < 5

Induction Step: Assume the statement is true for n' < n.

We must show the statement is true for n.

T(n) =

$$\begin{aligned} & T(\left\lfloor \frac{2n}{9} \right\rfloor) + 3T(\left\lfloor \frac{n}{5} \right\rfloor) + cn \\ & \leq a\left\lfloor \frac{2n}{9} \right\rfloor + 3a\left\lfloor \frac{n}{5} \right\rfloor + cn \\ & \leq a2n/9 + 3an/5 + cn \\ & = an(2/9 + 3/5) + cn \\ & = an(37/45) + cn = \\ & 37/8cn + cn \\ & = 45/8cn \\ & = an \end{aligned} \tag{1}$$

where the second step follows by the induction hypothesis. Thus, by the property of mathematical induction, the statement holds.

Q3

(a) Base case: a = 1

$$\begin{aligned} H(3)H(n-1) - H(1)H(n-2) - H(2)H(n-3) \\ = 2H(n-1) - H(n-3) \\ = H(n) \end{aligned} \tag{2}$$

so the base case holds.

Induction step: Suppose true for a' < a < n - 2

We want to show the claim holds for a:

$$\begin{aligned} H(n) = H(a+2)H(n-a) - H(a)H(n-a-1) - H(a+1)H(n-a-2) \\ = [2H(a+1) - H(a-1)]H(n-a) - H(a)H(n-a-1) - H(a+1)H(n-a-2) \\ = H(a+1)[2H(n-a) - H(n-a-2)] - H(a-1)H(n-a) - H(a)H(n-a-1) \\ = H(a+1)H(n-a) - H(a-1)H(n-a) - H(a)H(n-a-1) \\ = H(n) \end{aligned} \tag{3}$$

where the last step follows with the induction hypothesis applied with a - 1.
Thus, by the principle of mathematical induction the claim holds.

(b) Using a = k-1:

$$\begin{aligned} H(2k) = H(k+1)H(2k-(k-1)) - H(k-1)H(2k-(k-1)-1) - H(k)H(2k-(k-1)-2) \\ = H(k+1)H(k+1) - H(k-1)H(k) - H(k)H(k-1) \\ = H(k+1)H(k+1) - 2H(k)H(k-1) \end{aligned} \tag{4}$$

$$\begin{aligned} H(2k-1) = H(k+1)H(2k-1-(k-1) - H(k-1)H(2k-1-(k-1)-1) \\ -H(k-1+1)H(2k-1-(k-1)-2) \\ = H(k+1)H(k) - H(k-1)H(k-1) - H(k)H(k-2) \\ = H(k+1)H(k) - H(k-1)H(k-1) - H(k)[2H(k) - H(k+1)] \\ = 2H(k+1)H(k) - 2H(k)H(k) - H(k-1)H(k-1) \end{aligned} \tag{5}$$

(c) Using a = k-1:

$$\begin{aligned} H(2k+1) = H(k-1+2)H(2k+1-(k-1)) - H(k-1)H(2k+1-(k-1)-1) \\ -H(k-1+1)H(2k+1-(k-1)-2) \\ = H(k+1)H(k+2) - H(k-1)H(k+1) - H(k)H(k) \\ = H(k+1)[2H(k+1) - H(k-1)] - H(k-1)H(k+1) - H(k)H(k) \\ = 2H(k+1)H(k+1) - 2H(k+1)H(k-1) - H(k)H(k) \end{aligned} \tag{6}$$

$$
\begin{aligned}
H(2k-2) &= H(k-1+2)H(2k-2-(k-1)) - H(k-1)H(2k-2-(k-1)-1) \\
&\quad -H(k-1+1)H(2k-2-(k-1)-2) \\
&= H(k+1)H(k-1) - H(k-1)H(k-2) - H(k)H(k-3) \\
&= H(k+1)H(k-1) - H(k-1)[2H(k) - H(k+1)] - H(k)[2H(k-1) - H(k)] \\
&= 2H(k+1)H(k-1) - 4H(k)H(k-1) + H(k)H(k)
\end{aligned} \tag{7}
$$

(d) The algorithm is fairly straightforward given the recurrence relations in parts (b) and (c). If n is even, we use the recurrence relation for H(n) = H(2k) and call Hopskotchy on k. Else, n is odd, and we use the recurrence relation for H(n) = H(2k-1) and call Hopskotchy on k. Either call will return H(k+1), H(k), H(k-1). We then use the values returned to compute H(n+1), H(n), H(n-1) using the relations in (b) and (c).

Algorithm 1 Solving Hopskotchy quickly

```
1:  function HOPSKOTCHY(n)
2:      if n = 1 then
3:          return (1, 0, 0)
4:      end if
5:      if n = 2 then
6:          return (2, 1, 0)
7:      end if
8:      if n is even then
9:          a, b, c = Hopskotchy(n/2)
10:         H(n-1) = H(2k-1) = 2 * a * b - 2 * b * b
11:         H(n) = H(2k) = a * a - 2 * b * c
12:         H(n+1) = H(2k + 1) = 2 * a * a - 2 * a * c - b * b
13:     else
14:         a, b, c = Hopskotchy((n+1)/2)
15:         H(n-1) = H(2k - 2) = 2 * a * c - 4 * b * c + b * b
16:         H(n) = H(2k - 1) = 2 * a * b - 2 * b * b
17:         H(n+1) = H(2k) = a * a - 2 * b * c
18:     end if
19:     return (H(n+1), H(n), H(n-1))
20: end function
```

(e) It's clear that for a given value of n, we call Hopskotchy on an argument of at most $\lceil n/2 \rceil$. After getting, the return value we do a constant amount of multiplications and additions and then return the three values. Thus, the recurrence is

$T(n) \leq = T(\lceil n/2 \rceil) + c$

which makes T(n) = O(logn) under the uniform cost criterion by the master method.

If we were to use the obvious method, the recurrence would be T(n) = T(n-1) + c which gives us an O(n) bound under the uniform cost criterion. Thus, the smart method is significantly faster (logarithmic vs linear).

We can bound the size of the nth Hopskotchy number by 2^n (Note that this isn't necessarily a tight bound, but the point is that the size is exponential).

Proof: By induction on n

Base case: $H(n) < 2^n$ for $n \leq 3$

Induction Step: Suppose true for all n' < n

$H(n) = 2H(n-1) - H(n-3) \leq 2H(n-1) \leq 2 * 2^{n-1} = 2^n$

where the second to last step follows by the induction hypothesis.

Since $\log(2^n) = n$, we find that the number of bits required to represent the nth Hopskotchy number is O(n). Since we need the n/2th Hopskotchy numbers to compute the nth Hopskotchy number, we get that the cost for fast algorithm under the logarithmic cost criterion is $c(n/2 + n/4 + n/8 + ...) \leq 2cn = O(n)$. If we were to use the obvious method, we use the n-1 and n-3rd number to compute the nth, meaning the cost under the logarithmic cost criterion is $c(n-1 + n-2 + n-3 + ...) = O(n^2)$. Thus, the fast method is significantly better under the logarithmic cost criterion as well (linear vs quadratic).