Scott Corcoran
CS 411

Assignment

Questions:L: 2.2 #7, 8a 2.3 #5, 6, 9, 11 2.4 #1, 3, 4, 5, 10
2.2
7.a. $t(n) \in O(g(n))$
$t(n) <= cg(n)$ for all $n >= n_0$, where $c > 0$
$(1/c)t(n) <= g(n)$ for all $n >= n_0$

7.b. $f(n) <= c\alpha(n)$ for all $n >= n_0$ where c>0
$f(n) <= c\alpha(n)$ for all $n >= n_0$ where c_1 = cα >0
let f(n) $\epsilon\Theta(g(n))$
$f(n) <= cg(n)$ for all $n >= n_0$ where c>0
$f(n) <= (c/\alpha)\alpha g(n) = c_1\alpha g(n)$ for all $n >= n_0$ where $c_1 = (c/\alpha) > 0$

7.c. $\Theta(g(n)) \subseteq O(g(n)) \cap \Omega(g(n))$
$O(g(n)) \cap \Omega(g(n)) \subseteq \Theta(g(n))$

7.d. This is false
$t(n) = \begin{matrix} n \text{ if n is even} \\ n^2 \text{ if n is odd} \end{matrix}$
and
$g(n) = \begin{matrix} n \text{ if n is odd} \\ n^2 \text{ if n is even} \end{matrix}$

8.a. Ω notation
If there exists some positive constant c an a nonnegative integer n_0 such that $t(n) >= cg(n)$ for all $n >= n_0$.
2.3
5.a. what does it compute? It finds the minimum and the maximum value passing through the array

b. what is the basic op? Comparisons

c. how many times is that executed? two for checking if it is the lowest or highest. So it has one time through n elements with 2 comparisons per. 2n

d. what is the efficiency? $\Theta(n)$

e. suggest an improvement. You could make it so it checks just one then the other. So If you have a new lowest it obviously wont be your new highest

6.a. It should move through checking if the pairs match up to indicate that the two arrays are symmetric

b. Comparison between array elements

c. $\sum_{n-2}^{i=0} \sum_{n-1}^{j=i+1} 1$ ((n-1)n/2)

d.this moves through at a quadratic rate

e. I believe there really is not a good way to increase efficiency

9. S= $\sum_{n}^{i=1}$ = 1+2+3+4+...+n
2S=(1+n)+(2+(n-2))+...+(n+1)
2S=n(n+1)
S=n(n+1)/2

11.a. $2\sum_{i=1}^{n}(2i+1) = 2n^2 + 2n + 1$
b. This triple loop really seems silly at first glance. I think this can re-factored simpler

2.4
1.a. $x(n) = x(n-1) + 5$ for $n > 1$, $x(1) = 0$
$x(n) = x(n-i) + 5i$
$\sum_{i=0}^{k}(x(k) + 5k)$
Let K = n
x(n)=x(n-(n)+5(n-1)
x(1)+5(n-1) x(n)=5n-5

b.x(n)=3x(n-1)
x(n)=3^ix(n-i)
$\sum_{i=1}^{k} 3^k x(k)$
multiply with some c
x(n)= 4 * 3^{n-1}

c. x(n)=x(n-1)+n for n>0, x(0)=0
x(1)=[x(n-2)+(n-1)+n]
x(n)=x(n-i)+(n-i+1)+x(n-i+2)+....+n
so it is adding like factorial
=x(0)+1+2+...+n
x(n)=(n(n+1))/2

d. x(n)=x(n/2)+n for n >1, x(1)=1 (solve for n = 2^k)
$x(2^k) = x(2^{k-1}) + 2^k$
=$[x(2^{k-2}) + 2^{k-1}] + 2^k$
=$[x(2^{k-3}) + 2^{k-2}] + 2^{k-1} + 2^k$
=$[x(2^{k-i}) + 2^{k-i+1}] + ... + 2^k$
=$x(2^{k-k}) + 2^1 + 2^2 + 2^k$
=$1 + 2^1 + 2^2 + ... + 2^k$
=$2^{k+1} - 1 = 2 * 2^k - 1 = 2n - 1$

e. x(n)=x(n/3)+1 for n>1, x(1)=1 (solve for n = 3^k)
$x(3^k) = x(3^{k-1} + 1)$
$= [x(3^{k-2} + 1) + 1] = [x(3^{k-2}) + 2]$
$= [x(3^{k-2} + 1) + 2] = [x(3^{k-2}) + 3]$
$= x(3^{k-i} + i)$
$= x(3^{k-k} + k) = x(1) + k$
$= 1 + \log_3 n$

3.a. M(n)=M(n-1)+2, M(1)=0
M(n) = M(n-1)+2

= M(n-2)+2 +2
= M(n-3)+2 +2 +2
= M(n-i) + 2i
=M(n-n) +2(n-1) = 2(n-1) b. It should take the same amount of compares.

However this one will prove faster because it wont have the bulky overhead formed by a bunch of recursive calls
4.a. Q(n-1)+2n-1 for n>1
Q(2)=Q(1)+2*2-1 = 1+2*2-1 = 4
Q(3)=Q(2)+2*3-1 = 4+2*3-1 = 9
Q(4)=Q(3)+2*4-1 = 9+2*4-1 = 16
Q(5)=Q(4)+2*5-1 = 16+2*5-1 = 25
=n^2

b. M(n) = M(n-1)+1,for n>1, M(1)=0
M(n) = M(n-1)+1
= M(n-2)+1 +1
= M(n-3)+1 +1 +1
= M(n-i) + i
=M(n-n) +(n-1) = (n-1) c.Let M(n) be the number of additions and

subtractions made
M(n) = M(n-1) + 3 for n>1, M*(1)=0

5.a. The tower of hanoi have M(n)=2^n-1 for an algorithm
($2^6$4 - 1) / 60mins * 24hours * 365days = 3.5096545e13 years

b. i th disk (1 <= i <= n)
M(i+1) = 2M(i) (1 <= i <= n), m(1)=1
M(i)=2^{i-1}
$\sum_{i=1}^{n} 2^{i-1} = 1 + 2 + ... + 2^{n-1} = 2^n - 1$

c. 3 stacks
towers ¡0,n-1¿ vector //disks
while not done
select highest value at top of stacks
select largest disk
dont undue prior move //if it will skip
MOVESTACK(disk selected)
else if select second largest
dont undue prior move //if it will skip
MOVESTACK(disk selected)
else if select 3rd largest dont undue prior move //if it will skip
MOVESTACK(disk selected)
store last move if stack 3 has n items return done
MOVESTACK(disk)// move disk item to 3 if on stack 1
move disk item to 2 if on stack 3
move disk item to 1 if on stack 2
return

10. in best case we have constant time.
We have M(n) comparisons though in worst case
It is moving through a set of two arrays giving it the look of a quadratic time algorithm.