Algorithm Analysis ~ 13

CSE/IT 122 ~ Algorithms & Data Structures

Another Pattern

- $T(1) = a$
- $T(n) = bn+c+2T(n/2)$
  - Unroll
  - $T(n/2) = b(n/2)+c+2T(n/4)$
  - $T(n) = bn+c+2(b(n/2))+c+2T(n/4)) = 2bn+(c+2c)+2^2T(n/4)$
  - $T(n/4) = b(n/4)+c+2T(n/8)$
  - $T(n) = 2bn+(c+2c)+2^2(b(n/4)+c+2T(n/8) = 3bn+(c+2c+2^2c)+2^3T(n/8)$
  - So in general we get
    - $T(n)=kbn+(c+2c+...+2^{k-1}c)+2^kT(n/2^k)$
  - In terms of n and k, when $n/2^k = 1$, can remove k terms with $lg(n) = k$
  - $T(n) = b*n*log(n)+(2^k-1)c+log(n)T(1) = b*n*log(n)+(n-1)c+a*log(n)$
  - $T(n) = O(n*log(n))$

Factorial Function

```
factorial(int n) : int
    if n <= 1
        return 1;
    else
        return n * factorial(n - 1)
```

➔ What is the running time T(n) for this function?

Factorial Function

```
factorial(int n) : int
    if n <= 1               // O(1)
        return 1;           // O(1)
    else
        return n * factorial(n - 1)    // O(1) + T(n-1)
```

➔ If you add these up you will get
  - $T(n) = c + T(n-1)$ if $n \geq 1$ and $T(n) = d$ if $n \leq 1$
  - How do we solve this recurrence?
    - We *unroll* it, either forwards or backwards.

Factorial Function: Forward

- Forward
  - T(2) = c+T(1) = c+d
  - T(3) = c+T(2) = c+c+d = 2c+d
  - T(4) = c+T(3) = c+2c+d = 3c+d
- Generalize
  - T(n) = c+T(n-1) = c+(n-1)c+d = nc+d
  - So T(n) = O(n)

Factorial Function: Backward

- ➔ Backwards
  - T(n) = c+(c+T(n-2)) = 2c+T(n-2), n > 2
  - T(n) = 2c+(c+T(n-3)) = 3c+T(n-3), n > 3
- ➔ Generalize after i times
  - T(n) = (i-1)c+(c+T(n-i)) = ic+T(n-i), n > i
  - And when i = n-1
  - T(n) = (n-1)c+T(n-(n-1)) = (n-1)c+T(1) = (n-1)c+d
  - Running time is O(n)

In Class: Fibonacci Sequence

```
int fib(int n){
    if ((n==1) || (n==0){
        return 1;
    }
    else{
        return fib(n-1) + fib(n-2)
```

➔ T(n) is what?

In Class: Fibonacci Sequence

```
int fib(int n){
    if ((n==1) || (n==0){
        return 1;
    }
    else{
        return fib(n-1) + fib(n-2)
```

- T(n) is what?
- Answer: $O(2^n)$
  - Actually that is a loose upper bound … a better upper bound would be $O(1.6180)^n = \varphi^n$ which is the golden ratio
  - Please look forward to this proof in 344 (:

Towers of Hanoi

- 1883 Lucas invented the Towers of Hanoi puzzle
  - 8 discs of wood with holes in the center, which were piled in order of decreasing size on one pole in a row of three poles
  - Players are supposed to move all the discs one by one from one pole to another, never placing a larger disc on top of another
  - There was a prize for someone to move 64 discs
- Goal:
  - Goal: Move the discs to the third peg such that the discs are back in order.
- Instructions:
  - Three pegs with n discs, with smaller discs on top of larger discs.
  - Can only move one disc at a time and you can’t place a larger disc on top of a smaller disc

Towers of Hanoi

http://towersofhanoi.info/Animate.aspx

Towers of Hanoi

- Key to Solution
  - Step 1: Transfer the top k-1 discs from pole A to pole B. If k > 2 this requires a number of moves
  - Step 2: Move the bottom disc from pole A to C
  - Step 3: Transfer the top k-1 discs from pole B to C.

```
void towers(int n, int start, int finish, int spare){
    if (n==1) // O(1)
        printf("move disc from peg %ld to %ld\n", start, finish); // O(1)
    else{
        towers(n-1, start, spare, finish);    //T(n-1)
        printf("move disc from peg %ld to %ld", start, finish); // O(1)
        towers(n-1, spare, finish, start);    //T(n-1)
```

Towers of Hanoi

- Adding these up, you get
  - $T(n) = 2T(n_1)+b$, for $n > 1$ and if $n = 1$, $T(1) = a$. (a and b are constants)
- Lets unroll the recurrence
  - $T(n) = b + 2T(n-1)$
  - $T(n) = b+2(b+2T(n-2)) = b+2b+2^2T(n-2)$
  - $T(n) = b+2b+2^2(b+2(T(n-3)))$
  - $T(n) = b+2b+2^2b+2^3(T(n-3))$
  - For the i^{th} term
    - $T(n) = b+2b+2^2b+...+2^{i-1}b+2^iT(n-i)$
  - And when $i = n-1$
    - $T(n) = 2^0b+2^1b+2^2b+...+2^{n-2}b+2^{n-1}T(1)$

Towers of Hanoi

- Continuing with our unrolling
  - $T(n) = 2^0b+2^1b+2^2b+...+2^{n-2}b+2^{n-1}T(1)$
  - $T(n) = 2^0b+2^1b+2^2b+...+2^{n-2}b+2^{n-1}a$
  - Writing as a sum

$$T(n) = \sum_{i=0}^{n-2}(2^i b) + 2^{n-1}a = b\sum_{i=0}^{n-2}(2^i) + 2^{n-1}a$$

  - And can be simplified to
  - Test
  - Test
  - So the sum becomes
    - $T(n) = b(2^{n-1}-1)+2^{n-1}a = (a+b)2^{n-1}-b$
    - $T(n) = O(2^n)$

Towers of Hanoi

- Did anyone ever win the prize for moving 64 discs?
  - Number of moves required: $2^{64}-1$
  - Even if you only took one second to move each disc, it would take around 590 BILLION YEARS to complete all the required moves

HW 3 Problem #4

```
for i:= 1 to n
    for j:= 1 to i
        for k:= 1 to j
            x:= i * j * k
        next k
    next j
next i
```

- ➔ How to solve?

Different way of Counting 2 for Loops

i	1	2		3		
j	1	1	2	1	2	3

- Can think of ordered pairs
  - (1,1),(1,2),(2,2),(1,3),(2,3),(3,3)
- How many combinations of ordered pairs are there with repetition?
- Theorem. The number of r-combinations with repetition allowed that can be selected from a set of n elements is $\binom{r+n-1}{r}$
- Or put another way, number of ways r objects can be selected from n categories of objects with repetition allowed $\binom{2+3-1}{2} = \frac{(4)!}{2!(4-2)!} = \frac{4!}{2!2!} = \frac{4 \cdot 3 \cdot 2!}{2!2!} = 6$

Different way of Counting 2 for Loops

➔ So in general for two for loops and n elements you get:

$$\binom{r+n-1}{r} = \binom{2+n-1}{2} = \binom{n+1}{2} = \frac{(n+1)!}{2!(n+1-2)!} = \frac{(n+1)n(n-1)!}{2!(n-1)!} = \frac{n(n+1)}{2}$$

Different Way of Counting 3 For Loops

i	1	2			3					
j	1	1	2		1	2		3		
k	1	1	1	2	1	1	2	1	2	3

- Have ordered triplets
  - (1,1,1),(1,1,2),(1,2,2),(2,2,2),(1,1,3),(1,2,3),(2,2,3),(1,3,3),(2,3,3),(3,3,3)
  - So for our loop that only goes to n=3

$$\binom{3+3-1}{3} = \frac{(5)!}{3!(5-3)!} = \frac{5 \cdot 43!}{3!2!} = \frac{5 \cdot 4!}{2!} = 10$$

  - And in general for three for loops and n elements we get:

$$\binom{r+n-1}{r} = \binom{3+n-1}{3} = \binom{n+2}{3} = \frac{(n+2)!}{3!(n+2-3)!}$$

$$= \frac{(n+2)!}{3!(n-1)!} = \frac{(n+2)(n+1)n(n+1)!}{6(n-1)!} = \frac{(n+2)(n+1)n}{6}$$