Review Session: 30 June 2016

Thursday, June 30, 2016

Practice with Asymptotic Anaylsis

What are the asymptotic runtimes of the following algorithms?

Example 1

```
public static void method(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum++;
    }
}
```

executes once

executes once

loop executes n times

$1 + n(1) \rightarrow \boxed{O(n)}$

Example 2

```
public static void method(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            sum++;
        }
    }
}
```

n times

n times

$n \times n \rightarrow \boxed{O(n^2)}$

Example 3

```
public static void method(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= 4; j++) {
            sum++;
        }
    }
}
```

4 times

n times

$4 \times n \longrightarrow \boxed{O(n)}$

Example 4

```
public static void method(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            sum++;
        }
    }
}
```

i times

# of times sum++ executes

$$= 1 + 2 + 3 + 4 + \dots + n$$

$$= \sum_{i=1}^{n} a_i$$

arithmetic series!

$$\left(\sum_{i=1}^{n} a_i = \frac{n}{2}(a_1 + a_n) \right)$$

$$= \frac{n}{2}(1 + n)$$

$$= \frac{1}{2}n + \frac{1}{2}n^2$$

$$\Rightarrow \boxed{O(n^2)}$$

Example 5

```
// Pre-Condtion: Current is not null
// We'll only consider balanced binary trees for this problem
public static int countNodes(Node current) {
    if (current.left == null && current.right == null) {
        return 1;
    } else {
        int leftCount = countNodes(current.left);
        int rightCount = countNodes(current.right);
        return leftCount + rightCount + 1;
    }
}
```

Recursion!
$\Rightarrow$ Let's use recurrence relations.

Let n = # nodes in tree

If statement executes only when $n=1$
executions in body of if statement = 1

$\Rightarrow$ We have a <u>base case</u>

$$T(1) = 1$$

Else statement executions:

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 1$$

nodes in left subtree

nodes in right subtree

$$\Rightarrow T(n) = 1 + 2T\left(\frac{n}{2}\right)$$

Expanding the recurrence relation by plugging $1+2T(\frac{n}{2})$ for $T(n)$:

1st expansion $\quad T(n) = 1 + 2T(\frac{n}{2})$

2nd $\quad = 1 + 2 + 4T(\frac{n}{4})$

3rd $\quad = 1 + 2 + 4 + 8T(\frac{n}{8})$

$\vdots$

kth $\quad = \underbrace{2^0 + 2^1 + 2^2 + \cdots + 2^{k-1}} + 2^k T(\frac{n}{2^k})$

$$T(n) = 2^k - 1 + 2^k T(\frac{n}{2^k})$$

We know $T(1) = 1$ (base case)

Let's solve $\frac{n}{2^k} = 1$

$\rightarrow n = 2^k$

$\rightarrow k = \log_2 n$

Plug it into

$$T(n) = 2^k - 1 + 2^k T\left(\frac{n}{2^k}\right)$$

$$\Rightarrow \quad = 2^{\log_2 n} - 1 + 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right)$$

$$= n - 1 + nT\left(\frac{n}{n}\right)$$

$$= n - 1 + nT(1)$$

$$= n - 1 + n$$

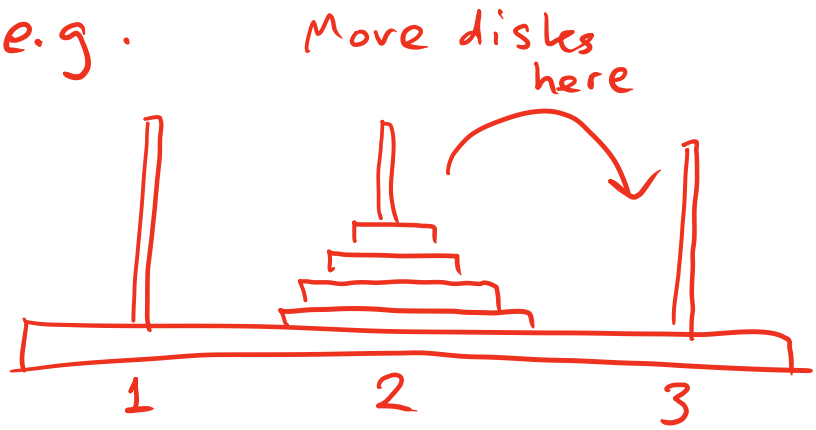
$$T(n) = 2n - 1$$

$$\Rightarrow \boxed{O(n)}$$

Example 6 -- Towers of Hanoi

(see https://en.wikipedia.org/wiki/Tower_of_Hanoi)

e.g.



from #2
to #3
(other = #1)

```
// Prints instructions for moving disks from one
// pole to another, where the three poles are
// labeled with integers "from", "to", and "other".
// Code from rosettacode.org
public void move(int n, int from, int to, int other) {
  if (n == 1) {
    System.out.println("Move disk from pole " + from +
                       " to pole " + to);
  } else {
    move(n - 1, from, other, to);
    move(1, from, to, other);
    move(n - 1, other, to, from);
  }
}
```

Recursive function!

Let's use recurrence relations.

Let $H(n)$ = # executions to run alg. on n.

Base case @ $n = 1$

```
if (n == 1) {
  System.out.println("Move disk from pole " + from +
                     " to pole " + to);
```

1 execution $\Rightarrow H(1) = 1$

All other $H(n)$:

```
} else {
  move(n - 1, from, other, to);
  move(1, from, to, other);
  move(n - 1, other, to, from);
}
```

$H(n-1)$ executions

$H(1) = 1$

$H(n-1)$

All together:

$$H(n) = H(n-1) + 1 + H(n-1)$$
$$= 1 + 2H(n-1)$$

Expanding (plug in for $H(n)$):

1st $H(n) = 1 + 2H(n-1)$

2nd $= 1 + 2 + 4H(n-2)$

3rd $= 1 + 2 + 4 + 8H(n-3)$

$\vdots$

kth $= 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} + 2^k H(n-k)$

$$= 2^k - 1 + 2^k H(n-k)$$

Base case is at $H(1)$, so let's solve $n-k=1$

$$\Rightarrow k = n-1$$

Plug it in:

$$H(n) = 2^k - 1 + 2^k H(n-k)$$

$$= 2^{n-1} - 1 + 2^{n-1} H(n-[n-1])$$

$$= 2^{n-1} - 1 + 2^{n-1} H(1)$$

$$= 2^{n-1} + 1 + 2^{n-1}(1)$$

$$= 2(2^{n-1}) + 1$$

$$H(n) = 2^n + 1$$

$$\Rightarrow \boxed{O(2^n)}$$