Recursive Algorithm Analysis

By: Mohanapriya Singaravelu

19BPS1046

Methods

- Trial and error, logic
- Recurrence relation
  - Master’s theorem
  - Substitution method

Factorial of a number

```
Algorithm Factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n*Factorial(n-1);
}
```

For example, consider Factorial(4):

Return 1

Return 1*factorial(0)

Return 2*factorial(1)

Return 3*factorial(2)

Return 4*factorial(3)

Number of times the function is called is proportional to n.

O(n) – time complexity

Fibonacci series

```
Algorithm Fibo(int n) {
  if (n <= 1)
    return n;
  else
    return Fibo(n)*Fibo(n-1);
}
```

$T(n)= T(n-1) + T(n-2)+ c$ for $n>0$

$T(0)= 1$

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

$T(n-2) \approx T(n-1)$

$T(n)= 2*T(n-1)+c$

$= 4*T(n-2) + 3c$

$= 8*T(n-4) + 7c$

$= 2^k T(n-k) +(2^k -1)c$

N-k= 0. Therefore n=k

$T(n) = 2^n T(0) + (2^n -1)*c$

T(n)= (1+c)2^n -c

Time complexity = $O(2^n)$

Exponent of a number- method 1

```
Algorithm Pow(x,n){
if(n==0)
      return 1
Else
      return x*pow(x,n-1)
}
```

$$T(n) = T(n-1) + C \quad , \quad n > 0$$

$$T(0) = 1$$

$$T(n) = T(n-1) + c$$
$$= T(n-2) + 2c$$
$$= T(n-k) + kc$$

$$n-k=0 \Rightarrow k=n$$

$$T(n) = T(0) + nc$$
$$T(n) = nc + 1$$

$$O(n)$$

Exponent of a number – method 2

$$x^n = \begin{cases} x^{n/2} \times x^{n/2}, & n: \text{even} \\ x \times x^{n-1}, & n: \text{odd} \\ 1 & ; \; n = 0 \end{cases}$$

```
Pow(x, n)
{
  if n == 0
     return 1
  else if n%2 == 0
       y <- Pow(x, n/2)
       return y×y
  else
     return x × Pow(x, n-1)
}
```

$O(\log n)$

$$T(n) = \begin{cases} T(n/2) + c_1, & \text{if } n \text{ is even} \\ T(n-1) + c_2, & \text{if } n \text{ is odd} \end{cases}$$

$T(0) = 1$, $T(1) = 1 + c_2$

$T(n) = T\left(\frac{n-1}{2}\right) + c_1 + c_2$

$\Rightarrow T(n) = T(n/2) + c \quad , \quad c > c_1$

$= T(n/4) + 2c$

$= T(n/8) + 3c$

$= T(n/2^k) + kc$

$n/2^k = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$

$T(n) = 1 + c_2 + c \log n$