Asymptotic Analysis/Finding Time Complexity of Algorithms
Big-Oh vs. Theta

Commonly used Logarithms and Summations

Logarithms

$$\log x^y = y \log x \qquad \log n = \log_{10}^{n}$$

$$\log xy = \log x + \log y \qquad \log^k n = (\log n)^k$$

$$\log \log n = \log(\log n) \qquad \log \frac{x}{y} = \log x - \log y$$

$$a^{\log_b^x} = x^{\log_b^a} \qquad \log_b^x = \frac{\log_a^x}{\log_a^b}$$

Arithmetic series

$$\sum_{K=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Geometric series

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} (x \neq 1)$$

Harmonic series

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx log\, n$$

Other important formulae

$$\sum_{k=1}^{n} \log k \approx n \log n$$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + \cdots + n^p \approx \frac{1}{p+1} n^{p+1}$$

Problem Find the complexity of the below recurrence:

$$T(n) = \begin{cases} 3T(n-1), if\ n > 0, \\ 1, \qquad\qquad otherwise \end{cases}$$

Solution: Let us try solving this function with substitution.

$T(n) = 3T(n-1)$

$T(n) = 3(3T(n-2)) = 3^2T(n-2)$

$T(n) = 3^2(3T(n-3))$

.

.

$T(n) = 3^nT(n-n) = 3^nT(0) = 3^n$

Problem- Find the complexity of the below recurrence:

$$T(n) = \begin{cases} 2T(n-1) - 1, if\ n > 0, \\ 1, & otherwise \end{cases}$$

Solution: Let us try solving this function with substitution.

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

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

$T(n) = 2^2(2T(n - 3) - 2 - 1) - 1 = 2^3T(n - 4) - 2^2 - 2^1 - 2^0$

$T(n) = 2^nT(n - n) - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots. 2^2 - 2^1 - 2^0$

$T(n) = 2^n - 2^{n-1} - 2^{n-2} - 2^{n-3} \ldots. 2^2 - 2^1 - 2^0$

$T(n) = 2^n - (2^n - 1)$ [*note:* $2^{n-1} + 2^{n-2} + \cdots + 2^0 = 2^n$]

$T(n) = 1$

$\therefore$ Time Complexity is O(1). Note that while the recurrence relation looks exponential, the solution to the recurrence relation here gives a different result.

Problem What is the running time of the following function?

```
void Function (int n) {
   int i=1, s=1;
   // s is increasing not at rate 1 but i
   while( s <= n) {
      i++;
      s= s+i;
      printf("*");
   }
}
```

Solution: Consider the comments in the below function:

We can define the 's' terms according to the relation $s_i = s_{i-1} + i$. The value oft' increases by 1 for each iteration. The value contained in 's' at the i^{th} iteration is the sum of the first '('positive integers. If k is the total number of iterations taken by the program, then the *while* loop terminates if:

$$1 + 2 + \ldots + k = \frac{k(k+1)}{2} > n \Rightarrow k = O(\sqrt{n}).$$

Problem- Find the complexity of the function given below.

```
void function(int n) {
        int i, count =0;
        for(i=1; i*i<=n; i++)
                count++;

}
```

Solution:

In the above-mentioned function the loop will end, if $i^2 > n \Rightarrow T(n) = O(\sqrt{n})$.

Problem- What is the complexity of the program given below:

Solution: Consider the comments in the following function.

```
void function(int n) {
        int i, j, k , count =0;
          //outer loop execute n/2 times
        for(i=n/2; i<=n; i++)
                //middle loop executes n/2 times
                for(j=1; j + n/2<=n; j= j+1)
                        //inner loop execute logn times
                        for(k=1; k<=n; k= k * 2)
                                count++;
}
```

The complexity of the above function is $O(n^2 logn)$.

Problem- What is the complexity of the program given below:

Solution: Consider the comments in the following function.

```
void function(int n) {
	int i, j, k , count =0;
	//outer loop execute n/2 times
	for(i=n/2; i<=n; i++)
		//middle loop executes logn times
		for(j=1; j<=n; j= 2 * j)
			//inner loop execute logn times
			for(k=1; k<=n; k= k*2)
				count++;
}
```

The complexity of the above function is $O(nlog^2n)$.

Problem- What is the complexity of the program given below:

Solution: Consider the comments in the following function.

```
function( int n ) {
   //constant time
   if( n == 1 ) return;
   //outer loop execute n times
   for(int i = 1 ; i <= n ; i + + ) {
      // inner loop executes only time due to break statement.
      for(int j= 1 ; j <= n ; j + + ) {
         printf("*" );
         break;
      }
   }
}
```

The complexity of the above function is O(n). Even though the inner loop is bounded by n, due to the break statement it is executing only once.

Problem-. Running time of the following program?

```
function(int n) {
        for(int i = 1 ; i <= n ; i + + )
                for(int j = 1 ; j <= n ; j * = 2 )
                        printf( " * " );
}
```

Solution: Consider the comments in the below function:

```
function(int n) {
    for(int i = 1 ; i <= n ; i + + ) // this loops executes n times
        // this loops executes logn times from our logarithms guideline
        for(int j = 1 ; j <= n ; j * = 2 )
            printf( " * " );
}
```

Complexity of above program is: O(*nlogn*).

Problem Find the complexity of the below function:

```
function(int n) {
        int i=1;
        while (i < n) {
                int j=n;
                while(j > 0)
                    j = j/2;
                i=2*i;
        } // i
}
```

Solution:

```
function(int n) {
        int i=1;
        while (i < n) {
                int j=n;
                while(j > 0)
                    j = j/2;   //logn code
                i=2*i; //logn times
        } // i
}
```

Time Complexity: O($logn * logn$) = O(log^2n).

Problem- ` Solve the following recurrence.

$$T(n) = \begin{cases} 1, & if\ n = 1 \\ T(n-1) + n(n-1), & if\ n \geq 2 \end{cases}$$

Solution: By iteration:

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

$$\dots$$

$$T(n) = T(1) + \sum_{i=1}^{n} i(i-1)$$

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

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

$$T(n) = \Theta(n^3)$$

Problem- What is the complexity of $\sum_{i=1}^{n} log\, i$?

Solution: Using the logarithmic property, *logxy* = *logx* + *logy*, we can see that this problem is equivalent to

$$\sum_{i=1}^{n} logi = log\,1 + log\,2 + \cdots + log\,n = log(1 \times 2 \times \ldots \times n) = log(n!) \le log(n^n) \le nlogn$$

This shows that the time complexity = O(*nlogn*).

$$f(n) = n^2 \log n + n$$

$$1 * n^2 \log n <= f(n) <= 10 * n^2 \log n$$

$\Omega(n^2 \log n)$ $O(n^2 \log n)$

$$f(n) = \Theta(n^2 \log n)$$

Theta (Asymptotic Tight Bound is possible)

$f(n) = n! = n * (n-1) * (n-2) * \ldots *3*2*1$

$1* 1 * 1 \ldots. * 1 <= 1*2*3*\ldots*n <= n*n*n\ldots*n$

$1 <= n! <= n^n$

$\Omega(1)$ **$O(n^n)$**

$f(n) = O(n^n)$

Big-Oh (Asymptotic Upper Bound is possible)

Theta (Asymptotic Tight Bound is **not** possible)

$$f(n) = \log(n!) = n * (n-1) * (n-2) * \ldots *3*2*1$$

$$\log(1* 1 * 1 \ldots * 1) <= \log(1*2*3*\ldots*n) <= \log(n*n*n\ldots*n)$$

$$1 <= \log(n!) <= \log(n^n)$$

$\Omega(1)$ $\quad$ $O(\log n^n) = O(n \log n)$

$$f(n) = O(n \log n)$$

Big-Oh (Asymptotic Upper Bound is possible)

Theta (Asymptotic Tight Bound is **not** possible)