Θ notation, $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$

More on Asymptotic Notation

Proof: Let $f(n)$ and $g(n)$ be two asymptotic non-negative functions, then $\max(f(n), g(n)) = \Theta(f(n)+g(n))$

Let, $h(n) = \max(f(n), g(n))$

Then, $h(n) = \begin{cases} f(n), & \text{if } f(n) \ge g(n) \\ g(n), & \text{if } f(n) < g(n) \end{cases}$

Since $f(n)$ and $g(n)$ are asymptotically non-negative, there exists n_0 s.t. $f(n) \ge 0$ and $g(n) \ge 0, \forall n \ge n_0$

Thus, for $n \ge n_0$, $f(n) + g(n) \ge f(n) \ge 0$
and $f(n) + g(n) \ge g(n) \ge 0$

Since for any particular n, $h(n)$ is either $f(n)$ or $g(n)$, we have $f(n)+g(n) \ge h(n) \ge 0$, which shows that $h(n) = \max(f(n), g(n)) \le c_2 \cdot (f(n)+g(n)), \forall n \ge n_0$
$[c_2 = 1 \text{ here}]$

Similarly, since for any particular n, $h(n)$ is the larger of $f(n)$ and $g(n)$, we have $\forall n \ge n_0$,

$$\left.\begin{aligned} 0 \le f(n) \le h(n) \\ \text{and } 0 \le g(n) \le h(n) \end{aligned}\right\}$$

Adding these two inequalities,

$$0 \le f(n)+g(n) \le 2 \cdot h(n)$$

$$\text{or, } 0 \le (f(n)+g(n))/2 \le h(n)$$

which shows that $h(n) = \max(f(n), g(n))$

$\forall n \geq n_0$ $\quad \geq c_1(f(n)+g(n))$

$[c_1 = \frac{1}{2} \text{ here}]$

(Proved)

Proof: For two non-negative asymptotic functions, $f(n)$ and $g(n)$, $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$

- Let, $f(n) \leq g(n)$ ✓

From the basic definition of Big-oh, we can write,

$$O(f(n)) + O(g(n)) \leq c_1 f(n) + c_2 g(n), \quad c, c_1, c_2 \Rightarrow \text{two positive constants}$$

$$\leq c_1 g(n) + c_2 g(n) \quad [\text{As per our assumption}]$$

$$\leq (c_1+c_2) g(n)$$

$$\leq c \cdot g(n) \quad [\because c = c_1 + c_2]$$

So, $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$

(Proved)

Ex: For two non-negative asymptotic functions $f(n)$ and $g(n)$, $O(f(n)) * O(g(n)) = O(f(n) * g(n))$

- Prove that, $n^K = O(a^n)$ for $a > 1$

To prove, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ $\quad f(n) = n^K, \; g(n) = a^n$ (here)

$$\lim_{n\to\infty} \frac{n^K}{a^n} = \frac{\infty}{\infty}$$

So, applying L' Hospital's rule,

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{K.n^{K-1}}{a^n . \ln a} = \lim_{n\to\infty} \frac{K(K-1)\, n^{K-2}}{a^n . \ln^2 a}$$

$$\cdots = \lim_{n\to\infty} \frac{K(K-1)\cdots 1}{a^n \ln^K a} = 0$$

So, $n^K = O(a^n)$ (Proved)

- Show that $\log n = O(\sqrt{n})$ but $\sqrt{n} \neq O(\log n)$

First case, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \frac{\infty}{\infty}$ $\quad \left[f(n) = \log n, \; g(n) = \sqrt{n} \right]$

Applying L' Hospital's rule,

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{1/n}{1/2 \cdot n^{-1/2}} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

So, $\log n = O(\sqrt{n})$ (Proved)

Second case, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log n} = \frac{\infty}{\infty}$ $\quad \left[f(n) = \sqrt{n}, \; g(n) = \log n \right]$

Applying L' Hospital's rule,

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim_{n\to\infty} \frac{1/2 \cdot n^{-1/2}}{1/n} = \lim_{n\to\infty} \frac{\sqrt{n}}{2} = \infty$$

So, $n \neq O(\log n)$ (Proved)

Proof of correctness for finding maximum of all elements in an array

```
Algorithm Maximum(A)
{ // A is an array
   max := A[0];
   for i := 1 to A.length - 1 step 1 do
        if A[i] > max then
            max := A[i];
   return max;
}
```

Case 1: A is having only one element

Case 2: A is having more than one element

→ The only element present in A, will be returned as max.

→ Loop invariant: It is a condition which must hold true during entire execution of the loop.

→ For this algorithm, For any given value i, max will contain maximum of elements whose index is smaller than i.

Method of induction: For the first time at loop, $i = 1$ and max will have the first element of the array, which is the greatest one till now.

Now, we have to prove that loop invariant will hold at the end of the loop iteration.

For any given value i, the "if" inside "for" loop is executed. If value at position i, i.e. A[i] > max, then max will have new value, making maximum of all elements till now.

Otherwise, value of max will remain same.

So, either way, max will reflect the max$^{\underline{m}}$ value from 0 to i, both inclusive.

So, by induction, when we traverse the entire length of the array, max will reflect maximum of array A. (Proved)

Calculating time complexity of recursive algorithm:

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

$1+2+\dots+n$

$T(n) \Rightarrow$ Time required to compute $f(n)$

$$T(n) = \begin{cases} t_1, & \text{when } n=0 \\ T(n-1)+t_2, & \text{when } n\geq 1 \end{cases}$$

$t_1, t_2 \Rightarrow$ constants.

Recurrence relation.

Solving recurrence relation:

1. Method of substitution
2. Recursion tree
3. Master theorem

(1)

$$T(n) = T(n-1)+t_2$$
$$= T(n-2)+t_2+t_2 \quad [\text{As } T(n-1) = T(n-2)+t_2]$$
$$= T(n-2)+2t_2$$
$$= T(n-3)+3t_2$$
$$\vdots$$

After $(K-1)^{th}$ substitution, $T(n) = T(n-K)+Kt_2$

Putting $K=n$, we get, $T(n) = T(0)+nt_2$

$[T(n)=t_1, \text{ when } n=0] \quad = t_1+nt_2$

$O(n)$

- Find the time complexity of the following function.

```
int f(int n)
{
  if(n==1)
    return 1;
  else
    return (f(n/2)+1);
}
```

$$T(n) = \begin{cases} t_1, & \text{when } n=1 \\ T(n/2)+t_2, & \text{when } n\geq 2 \end{cases}$$

$$T(n) = T(n/2) + t_2$$

Putting $n = 2^K$ we have,

$$T(2^K) = T(2^{K-1}) + t_2$$

$$T(2^K) = T(2^{K-2}) + 2t_2 \quad \text{[Applying method of substitution]}$$

$$= T(2^{K-3}) + 3t_2$$

$$\vdots$$

After $(m-1)^{th}$ substitution,

$$T(2^K) = T(2^{K-m}) + mt_2$$

Putting $K=m$, $= T(2^0) + Kt_2$

$$= T(1) + Kt_2$$

$$T(2^K) = t_1 + Kt_2 \quad [\because T(1) = t_1]$$

Now, $n = 2^K \Rightarrow K = \log_2 n$

$$T(n) = t_1 + (\log_2 n)\cdot t_2$$

$$\underline{O(\log n)}$$

Solve the following recurrence relation by method of substitution.

$$T(n) = \begin{cases} T(n-1) + \log n, & \text{when } n \geq 1 \\ C, & \text{when } n = 0 \end{cases}$$

$$T(n) = T(n-1) + \log n$$
$$= T(n-2) + \log(n-1) + \log n$$
$$= T(n-3) + \log(n-2) + \log(n-1) + \log n$$
$$\vdots$$
$$= T(n-n) + \log(n-n+1) + \cdots + \log(n-2) + \log(n-1) + \log n$$

[After $(n-1)^{th}$ substitution]

$$= T(0) + \log n + \log(n-1) + \log(n-2) + \cdots + \log 1$$

$$T(n) = C + \log \lfloor n$$

$$\underline{\underline{O(n \log n)}}$$

[$\lfloor n = 1 \cdot 2 \cdot \ldots \cdot n \leq n \cdot n \cdots n = n^n$

$\lfloor n \leq n^n$

Taking log on both sides,

$$\log \lfloor n \leq \log n^n$$
$$\log \lfloor n \leq n \log n \,]$$
$$O(n \log n)$$

- Solve the following recurrence relation by method of substitution.

$$T(n) = \sqrt{n}\, T(\sqrt{n}) + n$$

Dividing both sides by n,

$$\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 1$$

Putting $\underline{n = 2^m}$,

$$\frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 1$$

Let, $\frac{T(2^m)}{2^m} = S(m)$

So, $S(m) = S(m/2) + 1$

$\Downarrow$

From the previous example, we got the time complexity of this type of recurrence relation as $O(\log n)$

$\rightarrow O(\log m)$

$$\frac{T(2^m)}{2^m} = O(\log m)$$

$$T(2^m) = O(2^m \log m)$$

$$T(n) = O(n \log \underline{\log n})$$ $\left[\text{As } n = 2^m\right]$ $\left[m = \log_2 n\right]$

(Ans.)