3.1) O-notation, Ω-notation and θ-notation

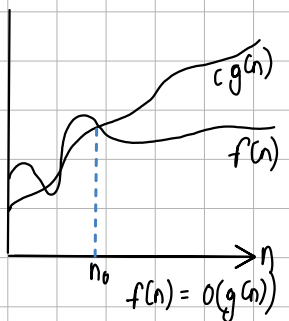
(i) O-notation → characterizes an upper bound.

Eg: $f(n) = 7n^3 + 3n^2 + 2n + 7$

$\therefore O(n^3)$ and $c \geq n^3$

Big Oh: A function $f(n)$ is said to be O of $g(n)$ iff (if and only if) there exist a constant 'c' and $n_0$ s.t:

$$0 \leq f(n) \leq c\,g(n) \quad \forall \; n \geq n_0$$




* f(n) must be non-negative for sufficiently large values of 'n'

* Consequently, g(n) must be asymptotically positive or else the set O(g(n)) is empty.

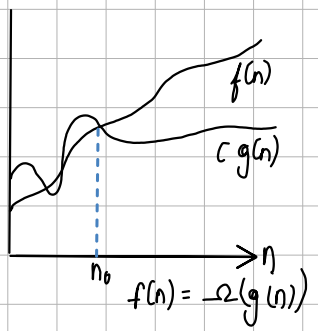
(ii) Ω-notation → characterizes a lower bound

Eg: $f_n = 7n^3 + 3n^2 + 2n + 7$

$\therefore O(n^3)$ and $c \leq n^3$

Big omega: A function $f(n)$ is said to be Ω of $g(n)$ iff there exist a constant 'c' s.t:

$$0 \leq c\,g(n) \leq f(n) \quad \forall \; n \geq n_0$$




(iii) θ-notation → characterizes a tight bound, or order of growth is certain and precise

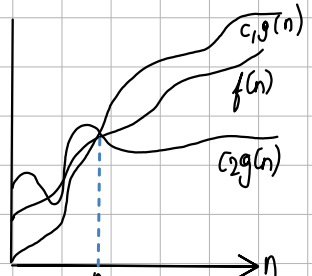
If $O(n^3)$ and $\Omega(n^3)$ then $\theta(n^3)$

Big theta: A function $f(n)$ is said to be Ω of $g(n)$ iff there exist a constant 'c' s.t:

$$0 \leq f(n) \leq c_1 g(n) \quad \forall \; n \geq n_0 \text{ ------- (i)}$$

$$0 \leq c_2 g(n) \leq f(n) \quad \forall\ n \geq n_0 \text{ ------ ii)}$$

$$\therefore \quad c_2 g(n) \leq f(n) \leq c_1 g(n) \quad \forall\ n \geq n_0$$




$f(n) = \theta(g(n))$

# Best case, Worst case and Expected case $\Longrightarrow$ Code with Harry

Given a sorted array called arr = [1, 3, 9, 17, 31] (indices 0, 1, 2, 3, 4)

To find : an element $a = 9$

We have, Best case : if $a$ exists at position 0, then $O(1)$
Worst case : if $a$ exists at any random position, then 'a' has to iterate the whole array and hence $O(n)$

Expected case : $\Sigma \left( \frac{\text{total no. of possible runtimes}}{\text{total no. of possibilities}} \right)$

$$= \frac{[k + 2k + 3k + \cdots + nk] + nk}{n+1}$$

$$= \frac{k\left([1 + 2 + 3 + \cdots + n] + n\right)}{n+1}$$

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

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

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

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

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

# Time complexity: Tricks and questions ⟹ Code with Harry

Rule: (i) Remove the non-dominant term
(ii) Remove the constant term
(iii) Break the code into fragments

E.g:
```
int n;
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        printf("Hello world!");
    }
}
```

$T = k_1 n (k_2 n) = k_1 k_2 (n^2)$ — from rule (i) and (ii), we have $n^2$

$\therefore T = O(n^2)$

We can analyse the code in detail:

For $i=0$: $j = 0, 1, 2, \dots, n-1$ ; $i=1$: $j = 0, 1, 2, \dots, n-1$ ; $\cdots$ ; $i=n-1$: $j = 0, 1, 2, \dots, n-1$

We have,

$i = 0 + 1 + 2 + \cdots + n-1$

$j = n-1 + n-1 + n-1 + \cdots + n-1$

$\Rightarrow j = (n-1)(1 + 1 + 1 + \cdots + n-1)$

$\Rightarrow j = (n-1)(n-1)$

$\Rightarrow j = n^2 - n - n - 1 = n^2 - 2n - 1 = n^2 - 2n$ [∵ Remove the constant]

$\Rightarrow j = n^2$ [∵ Remove the non-dominant term]