ASYMPTOTIC NOTATION

1- Introduction : Remember we wish to determine mathematically the quantity of resources needed by an algorithm as a function of the size of the instances. Because there is no standard computer to which all measurements of computing time might refer, we shall be content to express the time taken to within a multiplicative constant.

We introduce the asymptotic notation, which permits substantial simplifications, even when we are interested in measuring something more tangible, like number of times a given instruction is executed in a program. This notation deals with functions in the limit.

2- Notation for "the order of":

Let $t(n) : \mathbb{N} \longrightarrow \mathbb{R}^+$, where $\mathbb{R}^+$ is the set of all nonnegative real numbers, and you may think of n as a size of an instance on which a given algorithm is required to perform, and $t(n)$ as representing the quantity of a given resource spent on that instance by a particular implementation of this algorithm.

For example: it could be that the implementation spends $t(n)$ µs in the worst case on an instance of size n, or perhaps $t(n)$ represents the amount of storage.

$$t(n) = 27\,n^2 + \frac{355}{113}\,n + 12$$

and let also $f : \mathbb{N} \to \mathbb{R}^+$ such that $f(n) = n^2$

We say that $t(n)$ is <u>in the order of</u> $f(n)$ if $t(n)$ is bounded above by a positive real multiple of $f(n)$ for all sufficiently large n. $\quad t(n) \leqslant c\,f(n) \quad \forall n \geqslant n_0$

For instance, it is clear that $n \leq n^2 \quad \forall n \geqslant 1$, therefore: $n \geqslant 1$

$$t(n) = 27n^2 + \frac{355}{113}n + 12$$
$$\leqslant 27n^2 + \frac{355}{113}n^2 + 12n^2$$
$$= 42\frac{16}{113}n^2 = 42\frac{16}{113}f(n)$$

Taking $c = 42\frac{16}{113}$ (or anything larger) and $n_0 \geqslant 1$, we conclude that $t(n)$ is in the order of $f(n)$.

We could have chosen: $c = 28$ and $n_0 = 6$

$$t(n) \leqslant 28 f(n) \quad \forall n \geqslant 6$$

This shows a trade off between c and n_0 that is quite common.

So if an instance takes $t(n)$ μs in the worst case to solve we simplify by saying that the time is in the order of n^2. Naturally, there is no point stating that we are talking about the order of n^2 μs, since this differs from n^2 years by only a constant factor. Therefore we are entitled to assert that the algorithm itself takes a time of the order of n^2 or more simply it takes "quadratic time".

Notation for the order of: $O(f(n))$ big Oh of $f(n)$

$$O(f(n)) = \left\{ t: \mathbb{N} \to \mathbb{R}^+ \mid \exists c \in \mathbb{R}^+, \overset{\infty}{\forall} n \in \mathbb{N}, t(n) \leqslant c f(n) \right\}$$

So $t(n) \in O(f(n))$ and the traditional $t(n) = O(f(n))$ (one way equalities, as they used to be called). A lot of misuse happen with this notation, like when $t(n)$ is negative or when $t(n)$ is not defined for some values of n.

In other terms $t(n) \in O(f(n))$ if $0 \leq t(n) \leq cf(n)$ $\forall n \geq n_0$ regardless of what happens to t & f when $n \leq n_0$

Exple:

$$n^3 - 3n^2 - n - 8 \in O(n^3)$$

even though $n^3 - 3n^2 - n - 8 < 0$ when $n \leq 3$

The threshold is often useful to simplify arguments, but it is never necessary when we consider strictly positive functions.

$t(n) \in O(f(n))$, $t(n) \leq cf(n)$ $\forall n \geq n_0$ and

$t(n) \leq b f(n)$ where $b = \max\left\{ \frac{t(n)}{f(n)} \; 0 \leq n < n_0 \right\}$

$\Rightarrow$ $t(n) \leq a f(n)$, $\forall n \geq 0$ and $a = \max\{b, c\}$

(remember only when $t(n)$, $f(n)$ are strictly positive fcts).

A useful tool for proving that one function is in the order of another is: THE MAXIMUM RULE.

Let, $f, g : \mathbb{N} \to \mathbb{R}^+$ the max. rule says that:

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

Exple:

Consider an algorithm that proceeds in 3 steps

Initialization - Procesing - Finalization

for the sake of argument assume that these steps take $O(n^2)$, $O(n^3)$, $O(n \log n)$ respectively.

from the maximum rule it is immediate that:

$$O(n^2 + n^3 + n \log n) = O(\max(n^2, n^3, n \log n))$$
$$= O(n^3)$$

Proof of the maximum rule:

observe that, $f(n) + g(n) = \min(f(n), g(n)) + \max(f(n) + g(n))$

and $0 \leq \min(f(n), g(n)) \leq \max(f(n), g(n))$,

it follows that: $\max(f(n), g(n)) \leq f(n) + g(n) \leq 2\max(f(n), g(n))$ ①

Now: $t(n) \in O(f(n) + g(n)) \Rightarrow t(n) \leq c(f(n) + g(n))$ for large n.

using ① $t(n) \leq 2.c.\ \max(f(n), g(n))$ which means that $t(n)$ is bounded by a constant $2.c$

$\rightarrow$ $t(n) \in O(\max(f(n), g(n))$

Beware of the following misuse:

$O(n) = O(n + n^2 - n^2) = O(\max(n, n^2, -n^2) = O(n^2)$

↑ incorrect use because of an infinitely neg. function

The maximum rule tells us that if $t(n)$ is a complicated function such as $t(n) = 12n^3 \log n - 5n^2 + \log^2 n + 36$ and if $f(n)$ is the most significant term of $t(n)$ discarding the coefficients $f(n) = n^3 \log n$ then: $O(t(n)) = O(f(n))$ which allows for dramatic yet automatic simplifications in asymptotic notation.

Note the following:

$O(t(n)) = O(\max(12n^3 \log n, -5n^2, \log^2 n, 36)$

$= O(12n^3 \log n) = O(n^3 \log n)$ IS WRONG!

just because it doesn't use the max rule properly.

But the following is OK:

$O(t(n)) = O(11n^3 \log n + n^3 \log n - 5n^2 + \log^2 n + 36)$

$= O(\max(11n^3 \log n, n^3 \log n - 5n^2, \log^2 n, 36)$

$= O(11n^3 \log n) = O(n^3 \log n)$.

this is OK despite the fact that:

$n^3 \log n - 5n^2$ is negative for some n and 36 is larger than $n^3 \log n$ for some n.

Another observation, is that it is unnecessary to specify the base of the logarithm just because $\log_a n = \log_a b \times \log_b n$

So: $O(\log_a n) = O(\log_b n)$

Also:

$$O\left(\frac{n^2}{\log_3 n \sqrt{n \log n}}\right) = O\left(\left(\frac{n}{\log n}\right)^{1.5}\right)$$

The base of the logarithm can't be ignored when it's smaller than 1, or when it is not a constant as in: $\log_{\sqrt{n}} n \neq O(\log n)$ or when the logarithm is in the exponent as in $O(2^{\log_a n}) \neq O(2^{\log_b n})$.

The notation "$\in O$" is Reflexive and Transitive, and obviously not symmetric - but antisymmetric, which provides a way to define ... a partial order on functions and consequently on the relative efficiency of different algorithms to solve a given problem.

How can we prove that a given $t(n)$ is not in the order of $f(n)$? The simplest way is to use contradiction.

Exple: Let $t(n) = \frac{1}{1000} n^3$ and $f(n) = 1000\, n^2$

for $n < 10^6$, $t(n) < f(n)$ you may believe that $t(n) \in O(f(n))$, where $c = 1$. Of course it is not and let's prove by assuming that $t(n) \in O(f(n)) \rightarrow \exists c$ such that $t(n) \leq c f(n)$ $\forall n \geq n_0$ which means:

$\frac{1}{1000} n^3 \leq 1000\, c\, n^2 \rightarrow n \leq 10^6 c$. which clearly shows a contradiction.

Another more powerful tool to prove that some functions are in the order of others or the opposite is the limit rule which states that

Given arbitrary $f, g : \mathbb{N} \to \mathbb{R}^+$

1. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}^+ - \{0\}$ then $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$

2. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) \in O(g(n))$ but $g(n) \notin O(f(n))$

3. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = +\infty$ then $f(n) \notin O(g(n))$ but $g(n) \in O(f(n))$

Example : given $f(n) = \log n$, $g(n) = \sqrt{n}$ determine the relative order of these functions.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1/n}{1/(2\sqrt{n})} = \lim_{n\to\infty} 2/\sqrt{n} = 0$$

3. The OMEGA and THETA notations :

We saw before that insert and sort algorithm take a time in $O(n^2)$ and we mentioned that some others take time in $O(n \log n)$. Of course it is very easy to show that $n \log n \in O(n^2)$ or even in $O(n^3)$. This is confusing at first, but remember the fact that the O notation is designed solely to give upper bounds on the amount of resources required. Thus we need a dual notation for lower bounds, which is the Ω notation.

Let $t(n), f(n) : \mathbb{N} \to \mathbb{R}^+$

$t(n) \in \Omega(f(n))$ if $\exists d \in \mathbb{R}^+ - \{0\}$, $\forall n \geq n_0$ such that $t(n) \geq d\, f(n)$.

$$\Omega(f(n)) = \{ t : \mathbb{N} \to \mathbb{R}^+ \mid \exists d \in \mathbb{R}^{+,*}, \ \overset{\infty}{\forall} n \in \mathbb{N}, \ t(n) \geq d\, f(n) \}$$

Duality : $t(n) \in \Omega(f(n)) \iff f(n) \in O(t(n))$

why then Ω ? $\natural$

It seems more natural to say: "An algorithm takes a time in $\Omega(n^2)$" than to say "n^2 is in O of the time taken by the algorithm."

When we analyze the behavior of an algorithm, we are happiest if its execution time is bounded simultaneously both above and below by possibly different positive real multiples of the same function. For this reason we introduce $\Theta(n)$.

We say that $t(n)$ is in Theta of $f(n)$ or that $t(n)$ is in the exact order of $f(n)$ denoted:

$$t(n) \in \Theta f(n)$$

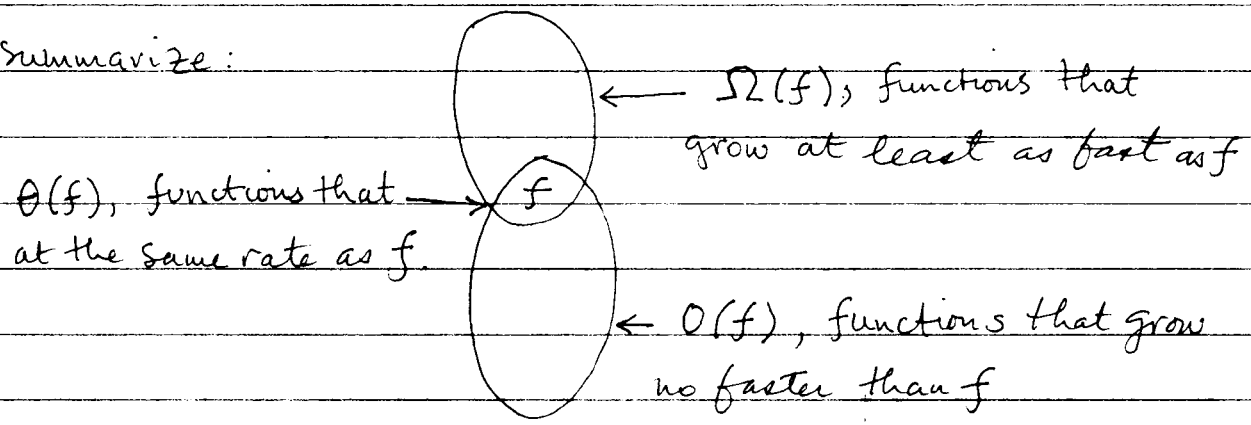
if $t(n) \in O(f(n))$ and $t(n) \in \Omega(f(n))$.

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

or:

$$\Theta(f(n)) = \left\{ t : \mathbb{N} \to \mathbb{R}^+ \mid \exists c, d \in \mathbb{R}^{++}, \overset{\infty}{\forall} n \in \mathbb{N}, \; df(n) \leq t(n) \leq cf(n) \right\}$$

To summarize:



Example of manipulating asymptotic notation,

Prove that:

$$\sum_{i=1}^{n} i^k \in \Theta(n^{k+1})$$

First the "O" direction: notice that $i^k \leq n^k$ for $1 \leq i \leq n$

Therefore: $\sum_{i=1}^{n} i^k \leq \sum_{i=1}^{n} n^k = n^{k+1}$ $\forall n \geq 1$ which means:

$$\sum_{i=1}^{n} i^k \in O(n^{k+1}) \quad \text{using } c = 1$$

Now prove the "Ω" direction.

Notice $i^k \geq (n/2)^k$ whenever $i \geq \lceil n/2 \rceil$, and the number of integers between $\lceil n/2 \rceil$ and n inclusive is greater than $n/2$

Therefore, provided that $n \geq 1$ (which implies $\lceil n/2 \rceil \geq 1$),

$$\sum_{i=1}^{n} i^k \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} i^k \geq \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \left(\frac{n}{2}\right)^k \geq \frac{n}{2} \times \left(\frac{n}{2}\right)^k = \frac{n^{k+1}}{2^{k+1}}$$

This proves $\sum_{i=1}^{n} i^k \in \Omega(n^{k+1})$ using $1/2^{k+1}$ as c.

A function f is b-smooth if (besides nondecreasing) it satisfies the condition:

$$f(bn) \in O(f(n)) \quad \text{or in other words,}$$

there exists a constant c (depending on b) such that:

$$f(bn) \leq c f(n) \quad \forall n \geq n_0$$

a function is smooth if it's b-smooth for every integer $b \geq 2$.

Example of smooth functions:

$\log n$, $n \log n$, n^2 or any polynomial whose leading coefficient is positive.

However, fast growing functions like $n^{\log n}$, 2^n, $n!$ are not smooth because $\frac{f(2n)}{f(n)}$ is unbounded

Exercises:

(1) Show that: $\log n! \in \Theta(n \log n)$

$$\log n! = \log n + \log(n-1) + \cdots + \log 2 + \log 1$$
$$\leq \log n + \log n + \cdots + \log n = n \log n$$

we conclude that $\log n! \in O(n \log n)$

Now: $\log n + \log(n-1) + \cdots + \log 1 \geq \log n + \cdots + \log \lceil \frac{n}{2} \rceil$
$$\geq \log \lceil \tfrac{n}{2} \rceil + \cdots + \log \lceil \tfrac{n}{2} \rceil = \lceil \frac{n}{2} \rceil \log(\tfrac{n}{2})$$

by mathematical induction we can prove that if $n \geq 4$

$$\frac{n}{2} \log \frac{n}{2} \geq (n \log n)/4$$

Thus $\log n + \log(n-1) + \cdots + \log 1 \geq \frac{1}{4} n \log n$ for $n \geq 4$

therefore: $\log n! \in \Omega(n \log n)$

It follows that $\log n! \in \Theta(n \log n)$

(2) Find a Theta notation in terms of n for the number of times the statement $x \leftarrow x+1$ is executed.

i. for $i \leftarrow 1$ to n do
ii. for $j \leftarrow 1$ to i do
iii. $x \leftarrow x+1$

first i set to 1, j runs from 1 to 1, thus line -iii- is executed one time
Next i " " 2, j " " 1 to 2, lines iii executed 2 times
and so on.

$$1 + 2 + \cdots + n = \Theta(n^2)$$

(3) Same Question as in (2).

```
1.  j ← n
2.  while j ≥ 1 do
3.      begin
4.      for i ← 1 to j do
5.          x ← x + 1
6.      j ← ⌊j/2⌋
7.      end
```

Let $t(n)$ be the number of times we execute $x \leftarrow x+1$

The first time we get to the body of the while loop, the statement $x \leftarrow x+1$ is executed n times

Thus: $t(n) \geq n$, $t(n) \in \Omega(n)$

Then j is $\lfloor \frac{n}{2} \rfloor$ which executes $x \leftarrow x+1$ at most $\frac{n}{2}$ times. and so on - if we after k times

$t(n) \leq n + \frac{n}{2} + \frac{n}{4} + \cdots + \frac{n}{2^{k-1}}$ Geometric Sum that is equal.

$$t(n) \leq \frac{n\left(1 - \frac{1}{2^k}\right)}{1 - \frac{1}{2}} = 2n\left(1 - \frac{1}{2^k}\right) \leq 2n \quad \text{So } t(n) \in O(n)$$

Thus $t(n) \in \Theta(n)$

4. Which of the following statements are true? Prove your answers.

1. $n^2 \in O(n^3)$ True $n^2 \leq c \cdot n^3 \rightarrow 1 \leq cn$, $c=1$, $n_0=1$

2. $n^2 \in \Omega(n^3)$ False $n^2 \geq cn^3 \rightarrow c \cdot n \leq 1 \rightarrow c \overset{?}{\leq} \frac{1}{n}$ none $\exists$

4. $2^n \in \Theta(2^{n+1})$ True $2^n = \frac{1}{2} \cdot 2^{n+1}$ or $2^n \leq 2^{n+1}$ $c=1$ $\forall n$

and $2^n \geq \frac{1}{4} 2^{n+1}$, $c = \frac{1}{4}$, $\forall n$

5. $n! \in \Theta((n+1)!)$ False $n(n-1)\cdots 2 \times 1 \leq c(n+1)n(n-1)\cdots 2 \times 1$

$\Rightarrow \quad 1 \leq c(n+1) \leq 2cn \rightarrow n! \in O((n+1)!)$

But Does $n! \in \Omega((n+1)!)$ $1 \geq c(n+1)$

5- Prove: $f(n) \in O(n) \rightarrow [f(n)]^2 \in O(n^2)$

$f(n) \in O(n) \rightarrow f(n) \leq c.n \rightarrow$

$[f(n)]^2 \leq [c.n]^2 \rightarrow [f(n)]^2 \leq c^2.n^2 = d.n^2 \rightarrow$

$[f(n)]^2 \in O(n^2)$

6. Prove that:

$2^{f(n)} \in O(2^n)$ Does not necessarily follow from $f(n) \in O(n)$.

Well - let's find a counterexample - Let a be an integer greater than 1, where $f(n) = a.n$. It is obvious that $f(n) \in O(n)$, but yet $2^{f(n)} = 2^{an} = (2^a)^n$ which is not $O(2^n)$.

7. Prove that: $\theta(n-1) + \theta(n) \subseteq \theta(n)$

if $f(n) \in \theta(n-1)$ and $g(n) \in \theta(n)$ then there exists a, b, c and d, n_1 & n_2 such that

$a(n-1) \leq f(n) \leq b(n-1) \quad \forall n \geq n_1$

$cn \leq g(n) \leq dn \quad \forall n \geq n_2$

let $h(n) = f(n) + g(n)$, and $n_0 = \max(n_1, n_2)$

obviously:

$h(n) = f(n) + g(n) \leq b(n-1) + dn \leq (b+d)n \quad \forall n \geq n_0$

which proves that $h(n) \in O(n)$. Also:

$h(n) = f(n) + g(n) \geq a(n-1) + cn \geq cn$

which proves that $h(n) \in \Omega(n)$ and therefore

$h(n) \in \theta(n) \rightarrow \theta(n-1) + \theta(n) \subseteq \theta(n)$

Actually this result can be generalized to equality - i.e.

we can easily show that: $\theta(n) \subseteq \theta(n-1) + \theta(n)$

let again $h(n) \in \theta(n)$ and let also $f(n) = g(n) = \frac{1}{2}h(n)$

obviously: $f(n) \in \theta(n)$ and $g(n) \in \theta(n)$ and also $f(n) \in \theta(n-1)$.

$h(n) = f(n) + g(n) \in \theta(n-1) + \theta(n)$ which proves that $\theta(n) \subseteq \theta(n-1) + \theta(n)$ and therefore

$$\theta(n-1) + \theta(n) = \theta(n)$$

On the other hand: It is not true that

$$\theta(n) = \theta(n) - \theta(n-1)$$

and very simple counter-example:

$$n - (n-1) = 1 \notin \theta(n) .$$

8. Prove that $\lceil \log n \rceil \in O(n)$.

For values like 1 (and other small values) it appears like $\lceil \log n \rceil \leq n$. Does it still hold for large values of n? Yes. Let's prove it by induction - it's true for the base $n=1$, and assume that it holds for $n-1$ i.e $\lceil \log(n-1) \rceil \leq n-1$ (*)

For $n > 1$, $\quad n \leq 10n - 10 = 10(n-1)$

$$\rightarrow \quad \lceil \log n \rceil \leq \lceil \log 10(n-1) \rceil = \lceil \log(n-1) \rceil + \log 10$$

$$\rightarrow \quad \lceil \log n \rceil \leq \lceil \log(n-1) \rceil + 1$$

$$\leq (n-1) + 1 \quad \text{using the assumption (*)}$$

$$= n \quad , \quad \text{here we consider } c = n_0 = 1$$