Lecture 3: Asymptotic Notation

1. Introduction: Remember we wish to determine mathematically the quantity of resource 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 night refer, we shall be content to express the time taken to within a multiplication constant.
   We introduce the asymptotic notation, which per unit substantial simplifications, even when we are interested in meaning something were in a 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 $t(n)$: $N \rightarrow R^+$, where 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 of this algorithm spends t(n) μs in the worst case on an instance of size n, or perhaps t(n) represent the amount of storage. $t(n) = 27n^2 + (355/113)n + 12$ and let also f: $N \rightarrow R^+$ such that $f(n) = n^2$ we say that t(n) is in the order of f(n) if t(n) is bounded above by a positive real multiple of f(n) for all sufficiently large n. $t(n) \leq c.f(n)\ \forall n \geq n_o$

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

   $t(n) = 27n^2 + (355/113)n + 12$
   $t(n) \leq 27n^2 + (355/113)n + 12$
   $t(n) = 42\ (16/113)n^2 = 42\ (16/113)\ f(n)$

   Taking $c = 42\ (16/113)$ (or anything larger) and $n_o \geq 1$, we conclude that t(n) is in the order of f(n). We could have chosen: $c=28$ and $n_o = 6$

   $t(n) \leq 28f(n)\ \forall n \geq 6$

   This shows a trade off, between c and n_o 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)

   $O(f(n)) = \{t: N \rightarrow R^+ \mid \exists c \in R^+, \forall n \in N, t(n) \leq c.f(n)\}$

   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 happens with this notation, like when t(n) is negative as when t(n) is not defined for some values of n.
   In other terms $t(n) \in O(f(n))$ if $0 \leq t(n) \leq c.f(n)\ \forall n \geq n_o$ regardless of what happens to t & f when $n \leq n_o$.

Example: $n^3-3n^2-n-8 \in O(n^3)$ even though $n^3-3n^2-n-8<0$ when $n\leq 3$

The threshold is of ten useful to simplify arguments, but it is never necessary when we consider strictly positive functions. $t(n) \in O(f(n))$, $t(n) \leq c.f(n)\ \forall n \geq n_o$ and $t(n) \leq b\ f(n)$ where $b = \max\{t(n)/f(n)\ 0 \leq n \leq n_o\}$
$\Rightarrow t(n) \leq a\ f(n), \forall n \geq 0$ and $a = \max\{b,c\}$ (remember only when t(n), f(n) are strictly positive functions).

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

Let, f, g: $N \rightarrow R^+$ the max, rule says that: $O(f(n) + g(n)) = O(\max(f,g))$

Example: Consider an algorithm that proceeds in 3 steps: initialization, processing, 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 $0 \leq \min(f(n),g(n)) \leq \max(f(n),g(n))$.
(1) It follows that: $\max(f(n),g(n)) \leq f(n) + g(n) \leq 2 \max(f(n), g(n))$
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.
$\rightarrow \quad t(n) \in O(\max(f(n),g(n)))$

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 negative 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 the t(n) discarding the coefficient $f(n) = n^3\log n$ then: $O(t(n)) = O(f(n))$ which allows for dramatic yet automatic simplifications in asymptotic notation.

Note the following:
$O(t(u)) = O(\max(12n^3\log n, -5n^2, \log^2 n, 36) = O(12n^3\log n) = O(n^3\log n)$ IS WRONG!
Just because it doesn't use the max rule property.
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 then $n^3\log n$ for some n.

Another observation is that it is unnecessary to specify the base of the logarithm just because $\log_a n = \log_a b \cdot \log_b n$ 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 is 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 anti-symmetric, which provides a way to define a partial order on functions and consequently on the relative efficiency of different algorithm 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.

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

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 $t(n) \leq c.f(n)\ \forall n \leq n_o$ which means:

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

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

Given arbitrary f, g: $\aleph \rightarrow \Im^+$

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

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

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

3. **The Omega and Theta Notation:**

We saw before that insert and sort algorithms take a time in $O(n^2)$ and we mentioned that some other 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 the lower bounds, which is Ω notation.

Let $t(n), f(n): \aleph \to \Im^+$

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

$\Omega(f(n)) = \{t: \aleph \to \Im^+ | \exists d \in \Im^{+,*}, \overset{\infty}{\forall} n \in \aleph, t(n) \geq d\, f(n)\}$

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

Then why Ω?

It seems more natural to say: "An algorithm takes s time in $\Omega(n^2)$" that 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 possible different positive and real multiples of the same function. For this reason we introduce θ(n). We say that t(n) is in the theta of f(n) or the t(n) is in the exact order of f(n) denoted:

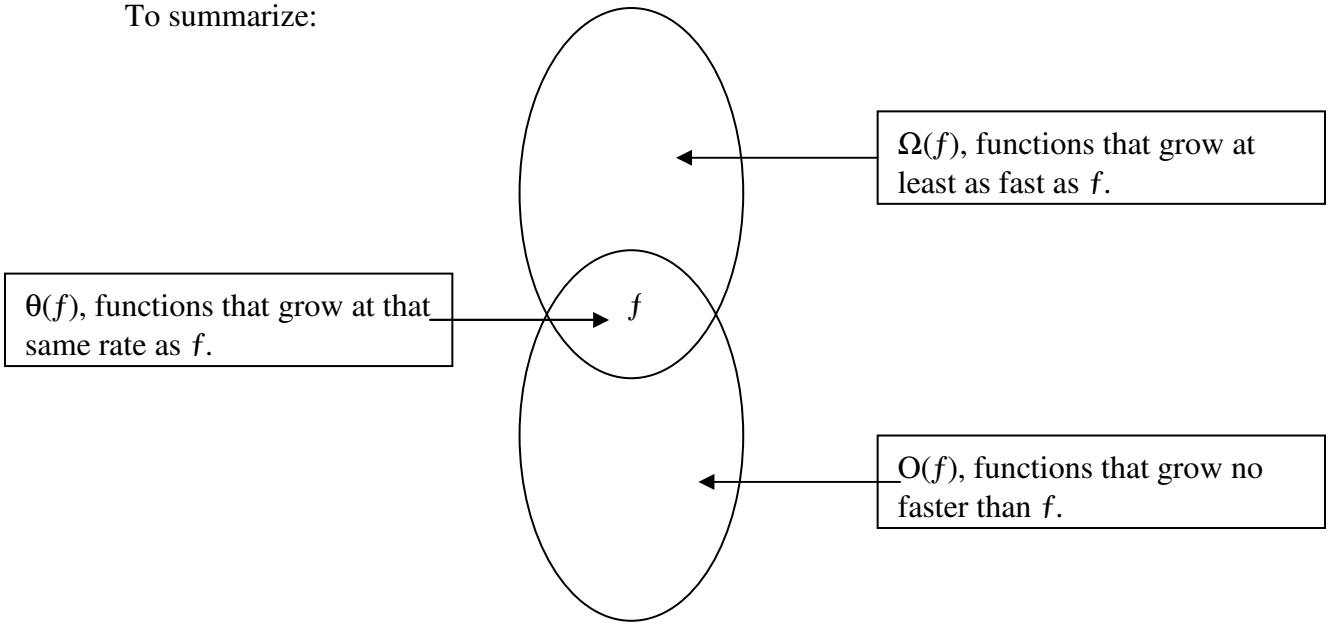
$t(n) \in \theta(f(n))$

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

$\theta(f(n)) = \{t: \aleph \to \Im^+ | \exists c,d \in \Im^{+,*}, \overset{\infty}{\forall} n \in \aleph, df(n) \leq t(n) \leq c\, f(n)\}$

To summarize:



Example of manipulating asymptotic notation,

Prove that: $\sum_{i=1}^{n} i^k \in \theta(n^{k+1})$

Fisrt "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 $\sum_{i=1}^{n} i^k \in \theta(n^{k+1})$ using c=1.

Now prove that "Ω" direction. Notice $i^k \leq (n/2)^k$ 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 $\lceil n/2 \rceil \geq 1$),

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

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

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

$f(bn) \in O(f(n))$ or in other words, there exists a constant c (depending on b) such that $f(bn) \leq cf(n)\ \forall n \geq n_o$ a function is smooth if it is b-smooth for very integer $b \geq 2$.

Example of smooth function:

Log n, n.log n, n^2 or any polynomial where the 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: $\log n! \in \theta(n \log n)$

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

we conclude the $\log n! \in O(n \cdot \log n)$

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

$$\geq \log n + \ldots + \log\lceil \frac{n}{2} \rceil = \lceil \frac{n}{2} \rceil \log\left(\frac{n}{2}\right)$$

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

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

Thus $\log n + \log (n-1) + \ldots + \log 1 \geq ¼\ n \log n$ for $n \geq 4$ therefore
$\log n! \in \Omega((\ ¼ \cdot \quad n \cdot \log n)$

It follows that $\log n! \in \theta(n \cdot \log n)$

(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 n 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 is set to 2, j runs from 1 to 2, line iii is executed 2 times and so on.

$$1 + 2 + \ldots + n = \theta(n^2)$$

(3) Same question as in (2)

```
1.  j←n
2.      while j≥1 do                    Let t(n) be the number of this we execute
3.              begin                   x←x+1 the fist time we get to the body of
4.              for i←1 to j do         the while loop, the statement x←x+1 is
5.                      x←x+1           executed n times. Thus: t(n) ≥n, t(n) ∈ Ω (n)
6.                      j ←⌊j/2⌋         then j is ⌊j/2⌋ which executes x←x+1 at
7.              end                     most n/2 times and so on if m after k times.
```

$t(n) \le n + \frac{n}{2} + \frac{n}{4} + ... + \frac{n}{2^{k+1}}$ geometric sum that is equal to

$$t(n) \le \frac{n(1-\frac{1}{2^k})}{1-\frac{1}{2}} = 2n\left(1-\frac{1}{2^k}\right) \le 2n \text{ so } t(n) \in O(n) \text{ then } t(n) \in \theta(n)$$

(4) Which of the following statements are true? Prove your answers:

1. $n^2 \in O(n^3)$ True $n^2 \le c \cdot n^3 \rightarrow 1 \le c \cdot n$, $C=1$ $n_o=1$
2. $n^2 \in \Omega(n^3)$ False $n^2 \ge c \cdot n^3 \rightarrow c \cdot n \le 1 \rightarrow c? \frac{1}{n}$ none $\exists$
3. $2^n \in \theta(2^{n+1})$ True $2^n = \frac{1}{2} \cdot 2^{n+1} or 2^n \le 2^{n+1}$ $c=1$ $\forall n$ & $2^n \le \frac{1}{4} 2^{n+1}$ $c= ¼$ $\forall n$
4. $n! \in \theta((n+1)!)$ False $n(n-1)\ldots 2x1 \le c\ (n+1)n\ (n-1)\ldots 2x1 \Rightarrow 1 \le c(n+1) \le 2cn \rightarrow n! \in O((n+1))$ but does $n! \in \Omega((n+1))$ $1 \ge c(n+1)$
5. Prove: $f(n) \in O(n) \rightarrow [f(n)]^2 \in O(n^2)$
   $f(n) \in O(n) \rightarrow f(n) \le c \cdot n \rightarrow$
   $[f(n)]^2 \in O(n^2) \rightarrow [f(n)]^2 \le c^2 \cdot n^2 = d \cdot n^2 \rightarrow [f(n)]^2 \in O(n^2)$

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 counter example, let a be an integer greater than 1, where $f(n)=a \cdot n$. 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
   $a(n-1) \le f(n) \le b(n-1)$ $\forall n \ge n_1$
   $cn \le g(n) \le dn$ $\forall n \ge n_2$
   let $h(n) = f(n) + g(n)$ and $n_o = \max(n_1, n_2)$.

   Obviously $h(n) = f(n) + g(n) \le b(n-1) + dn \le (b+d)n$ $\forall n \ge n_o$ which proves that $h(n) \in O(n)$. Also: $h(n) = f(n) + g(n) \ge a(n-1) + cn \ge cn$ 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 also $f(n) = g(n) = \frac{1}{2}h(n)$. Obviously: $f(n) \in \theta(n)$, $g(n) \in \theta(n)$ and $f(n) \in \theta(n-1)$

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

On the other hand: It is not true that $\theta(n) = \theta(n-1) - \theta(n)$ and a 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 $n>1$, $n \leq 10n-10=10(n-1)$

$\rightarrow \lceil \log n \rceil \leq \lceil \log 10(n-1) \rceil = \lceil \log (n-1) \rceil + \log 10$
$\rightarrow \lceil \log n \rceil \leq \lceil \log (n-1) \rceil + 1$
$\leq (n-1) + 1$ using the assumption (*)
$= n$, have we consider $c=n_o=1$