Asymptotic Analysis

Dr Dayal Kumar Behera

Growth of Functions

Complexity of an algorithm, generally expressed by function of input size(n)

We generally use asymptotic notation to describe the behavior of the function.

Asymptotic Behavior of Function:

- Ignores small value of n (input size)
- Does not distinguish between f(n) and c*f(n) where c is a positive constant (ignores constant)

Asymptotic Behavior of function

Example: Let $f(n) = 5n$ and $g(n) = n^2$ are the running time of two algorithms A and B, respectively. Which algorithm exhibits better performance?

Another Example

$T_1(n) = n^2 + n$

$T_2(n) = n^2$

Lets assume each algorithm takes 1 sec to execute for n=100
How long will it take for n=1000?

$$\text{Time}_1 = 1 \text{ x } \frac{1000^2+1000}{100^2+100} = 99.1 \text{ (approx.)}$$

$$\text{Time}_2 = 1 \text{ x } \frac{1000^2}{100^2} = 100$$

Between Time1 and Time2 there is not much difference so we can ignore the lower order terms and constants.

- In general, we only worry about **growth rates** of algorithms!!!
- **Reason 1:** Our main objective is to analyze the cost performance of algorithms asymptotically
  - Reasonable in part, because computers get faster and faster every year
- **Reason 2:** Deriving the exact cost of algorithms often are not feasible, as they are quite complicated to analyze
- **Reason 3:** When analyzing an algorithm, we are not that interested in the exact time the algorithm takes to run
  - Often we only want to compare two algorithms for the same problem
  - The property that makes one algorithm preferred over another is **its growth rate relative to the other algorithm's growth rate**

Asymptotic Analysis

- Analysis of a given algorithm with larger number of input data is called asymptotic analysis.
- Asymptotic Analysis is the theory of approximation (Greek word for Approximation)
- For Bigger algorithms where finding exact time complexity is difficult by the process of counting, Asymptotic analysis can be very effective.
- Also limit theory is helpful in specifying this type of algorithm behavior.

For example

T(n) = $\frac{n^3 + 4}{n} = n^2 + \frac{4}{n}$ when n becomes larger $\frac{4}{n}$ can be neglected

T(n) = n^2 (we say n^2 is the asymptotic behavior of T(n) when n $\rightarrow \infty$)

Asymptotic Notations

Theta or Big-Theta (θ)

Big-Oh (O)

Big-Omega (Ω)

Little-Oh (o)

Little-Omega (ω)

Theta Notation(θ): Asymptotic Tight Bound

Definition: Let f and g be the two functions that map a set of natural numbers to a set of positive real number $\mathrm{N} \rightarrow R_{\geq 0}$

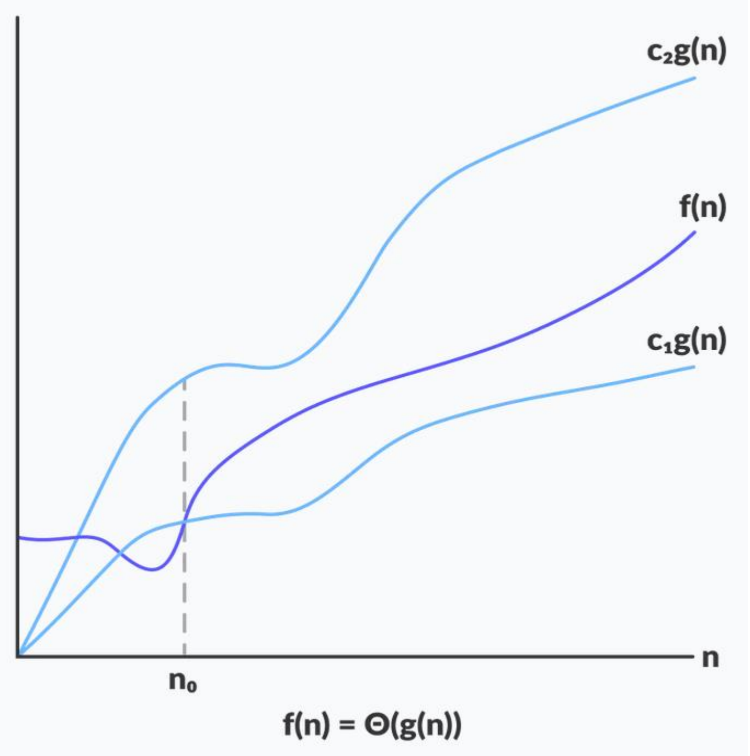
$\theta(g(n))$ = {f(n): there exist positive constant c1, c2 and n_0 such that $c1g(n) \leq f(n) \leq c2g(n)$ for all $n \geq n_0$}

Let θ(g) be the set of all functions that have a similar growth rate.

The relation *f(n) = θ(g(n))* holds good if and only if there exist two positive constants c1, c2 and n_0 such that

$$c1g(n) \leq f(n) \leq c2g(n) \qquad \forall\, n \geq n_0$$

The function f(n) is said to be in $\theta(g(n))$ *means* $f(n) \in \theta(g(n))$ But represented as *f(n) = θ(g(n))*



Both Lower and Upper Bounds of an algorithm is given by Big-Theta

Theta Notation(θ)

- Hints#

Set C1 to a value that is slightly smaller than the coefficient of higher order term and C2 to a value slightly larger. This always satisfy the inequality in the definition of θ notation.

Theta Notation(θ)

Let $f(n) = 5n^2 + 6n + 3$. Prove that f(n) of the algorithm is in $\theta(n^2)$

Big-Oh Notation (O)

Definition: Let f and g be the two functions that map a set of natural numbers to a set of positive real number $\mathrm{N} \rightarrow R_{\geq 0}$

Let O(g) be the set of all functions with a similar growth rate.

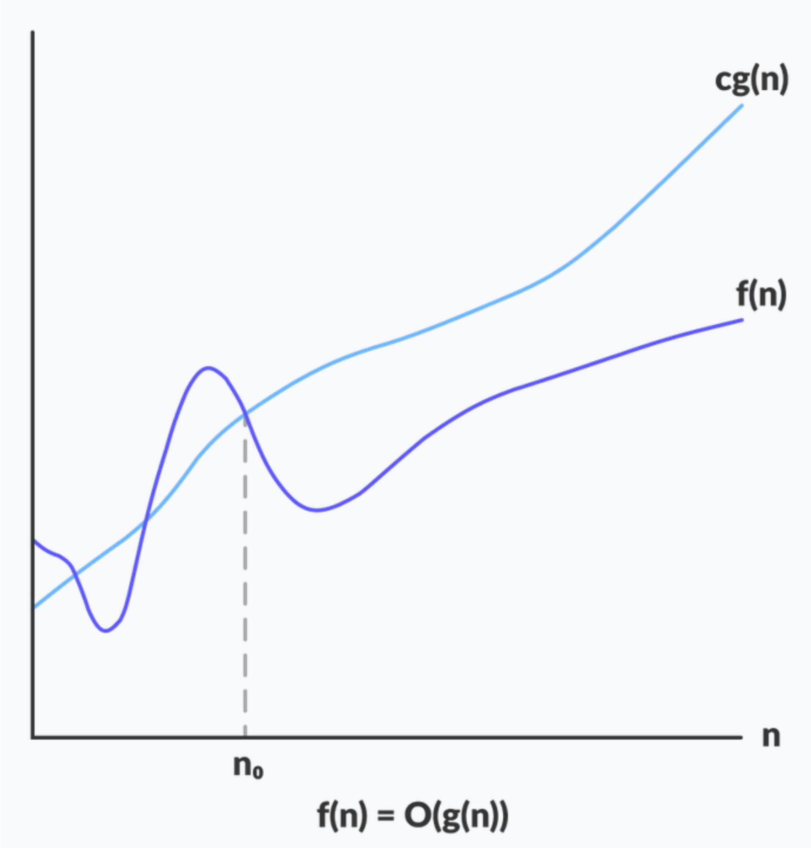
The relation *f(n) = O(g(n))* holds good if and only if there exist two positive constants c and n_0 such that

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

The function f(n) is said to be in *O(g(n))* *means* $f(n) \in O\big(g(n)\big)$ But represented as *f(n) = O(g(n))*

The Upper Bound of an algorithm is given by Big-Oh.

$O\big(g(n)\big)$ = {f(n): there exist positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$}



Big-Oh Notation (O)

Let $f(n) = 3n^3$ for an algorithm. Prove that $f(n)$ of the algorithm is in $O(n^3)$

Solution:

As per definition of Big-Oh notation $f(n) \leq c\, g(n)$

So we need to prove $3n^3 \leq cn^3$ for some positive value of c.

and we can see for all value of $c \geq 4$ this relation holds good

So $f(n)$ is in $O(g)$ or the algorithm is in $O(n^3)$

Big-Oh Notation (O)

Let $f(n) = 3n^3 + 2n^2 + 3$ for an algorithm. Let $g(n) = n^3$. Prove that f(n) of this algorithm is in O(g(n))

So we need to show

$$3n^3 + 2n^2 + 3 \leq cn^3 \textit{ for some positive integer } c$$

$$f(n) = 3n^3 + 2n^2 + 3$$

$$\leq 3n^3 + 2n^3 + 3n^3 \text{ (as } 2n^3 > 2n^2 \text{ and } 3n^3 > 3n \text{ for any positive integer n)}$$

$$\leq 8n^3$$

So $f(n) \leq 8n^3 \rightarrow f(n)$ is in $O(n^3)$

Big-Oh Notation (O)

Let $f(n) = (2n^3 + 13 \log n)/7n^2$ for an algorithm. Prove that f(n) of this algorithm is in O(n)

It can be observed that log n < n always hold good

So $13 \log n < 13n$ and $13n < 13\ n^3$

so $f(n) \leq (2n^3 + 13\ n^3) / 7n^2$

$\leq 15\ n^3 / 7n^2$

$\leq (15/7)\ n$

$\leq 3\ n$

So f(n) is in O(n)

Properties of Big-Oh notation

- For big-Oh analysis only the dominating summands maters. For example $O(4n^4+7n^2+7) = O(n^4)$, all term other than highest degree are ignored.
- In addition, in the big-Oh notation, the constant factors are not significant. For example $O(3n^2) = O(n^2)$
- Big-Oh can be used to express upper bounds.
- A bound is called tight bound or least upper bound if the difference between the bound and the actual function is a constant.
- For example n^2 can not be expressed as $O(n^3)$, it can only be expressed in $O(n^2)$, as it is the best fit.

Big-Omega Notation(Ω)

Definition: Let f and g be the two functions that map a set of natural numbers to a set of positive real number $\mathrm{N} \rightarrow R_{\geq 0}$

Let Ω(g) be the set of all functions that have a similar growth rate.

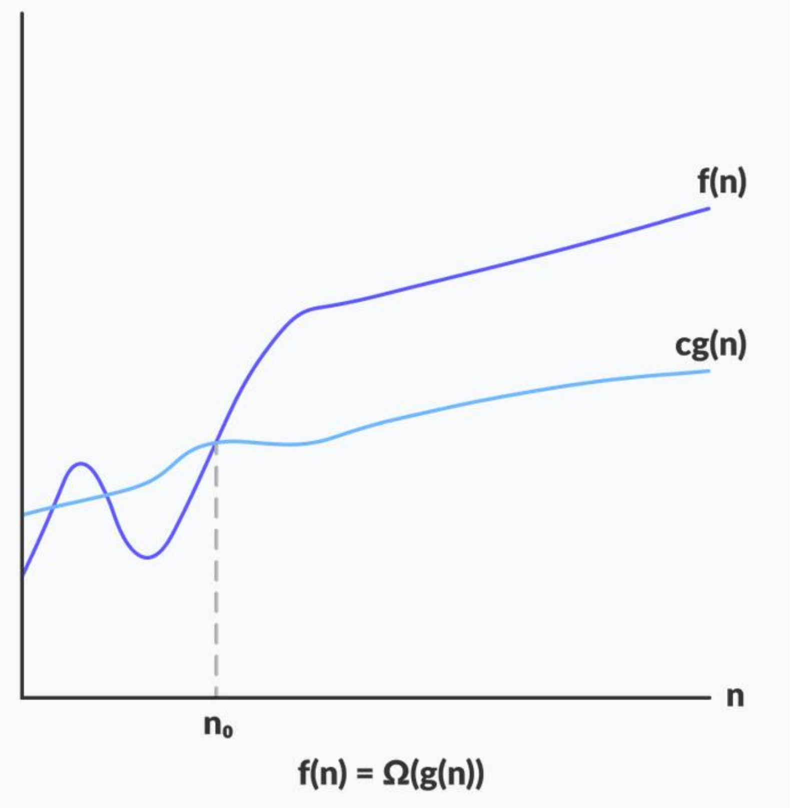
The relation $f(n) = \Omega(g(n))$ holds good if and only if there exist two positive constants c and n_0 such that

$$f(n) \geq c \times g(n) \quad \forall\, n \geq n_0$$

The function f(n) is said to be in $\Omega(g(n))$ *means* $f(n) \in \Omega(g(n))$ But represented as $f(n) = \Omega(g(n))$

The Lower Bound of an algorithm is given by Big-Omega.

$\Omega(g(n))$ = {f(n): there exist positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$}



Big-Omega Notation(Ω)

Let $f(n) = n^4 + 3n^3 + 2n + 1$ for an algorithm. Let $g(n) = n^4 + 1$. Prove that $f(n)$ of this algorithm is in $\Omega(g(n))$

So we need to show

$n^4 + 3n^3 + 2n + 1 \geq c\,(n^4 + 1)$ *for some positive integer c*

for all value of c > 0 this relation holds good

so $f(n)$ is in $\Omega(g(n))$

Big-Omega Notation(Ω)

If the relation $f(n) = 6n^2 + 7n + 8$ holds, Prove that $f(n)$ of not in $\Omega(n^3)$

$f(n)$ can be in $\Omega(n^3)$ only if

$6n^2 + 7n + 8 \geq c\, n^3$ *which can not be true for any value of* $c > 0$

so $f(n)$ is not in $\Omega(g(n))$

Properties of Theta Notation

The following are the properties of the Theta Notation

1. If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then $f(n) = \theta(g(n))$ and the vice-versa is also true.
2. If $f(n) = O(g(n))$ and $g(n) = O(f(n))$, then $f(n) = \theta(g(n))$

3. For any polynomial of the order of m, $f(n)$ is in $\theta(n^m)$

If $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots\ldots\ldots + a_m n^m$ and $a_m > 0$, then $f(n) = \theta(n^m)$

Examples of Big-Theta Notation

Supposed that an algorithm takes eight seconds to run on an input size n =12. Estimate the instances (input size) that can be processed in 56 secs. Assume that the algorithm complexity is θ(n). (Assume simplified RAM model with no hardware specific constraints)

As f(n) is in θ(n)

c*12 = 8 sec

$\Rightarrow$ c = 8/12 = 2/3

Now need to calculate n for t = 56

$\Rightarrow$ c n= 56

$\Rightarrow$ n = 56 x 3/2

$\Rightarrow$ n = 84 Hence, the maximum input that is possible is 84

Limits and Asymptotic Equality

- In general a limit of two complexity function can be written as

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

- If the limit is a positive constant "**c**", then both the complexity function are of the same order and grow at the same rate.
- If the limit is zero , g(n) grows faster
- If the limit is ∞, f(n) grows faster

Theta/Big-Oh/Big-Omega Notation

- If f(n) and g(n) are two functions and

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c$$

- If c > 0 (positive constant), then $f(n) \in \theta\left(g(n)\right)$.

Little-oh Notation

Little –Oh notation is similar to Big-Oh but it represents a loose bound

Definition :

The relation f(n) = o(g(n)) holds good, if there exist two positive constants c and n_0 such that

$$f(n) < c \times g(n)$$

The bounds can also be proved using following theorem.

if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$ then f(n) = o(g(n)) holds good

Example for little-Oh

Let $f(n) = 7n + 6$. show that $f(n)$ is in $o(n^2)$

As we know, if $f(n) = o(g(n))$ holds good when $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$$=> \quad \lim_{n\to\infty} \frac{7n+6}{n^2} = \lim_{n\to\infty} \frac{7}{n} + \frac{2}{n^2} = 0$$

This indicates $f(n)$ is in $o(n^2)$

Little-Omega notation

The relation f(n) = $\omega\left(g(n)\right)$ holds good if there exist two positive constants 'c' and 'n_0' such that f(n) > c (g(n))

The relation holds good if and only if

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

Summary of Asymptotic Notations

1. If $\mathbf{f(n) = \Theta(g(n))}$, then there exists positive constants c1, c2, n0 such that $\mathbf{0 \le c1.g(n) \le f(n) \le c2.g(n)}$, for all $n \ge n0$
2. If $\mathbf{f(n) = O(g(n))}$, then there exists positive constants c, n0 such that $\mathbf{0 \le f(n) \le c.g(n)}$, for all $n \ge n0$
3. If $\mathbf{f(n) = \Omega(g(n))}$, then there exists positive constants c, n0 such that $\mathbf{0 \le c.g(n) \le f(n)}$, for all $n \ge n0$
4. If $\mathbf{f(n) = o(g(n))}$, then there exists positive constants c, n0 such that $\mathbf{0 \le f(n) < c.g(n)}$, for all $n \ge n0$
5. If $\mathbf{f(n) = \omega(g(n))}$, then there exists positive constants c, n0 such that $\mathbf{0 \le c.g(n) < f(n)}$, for all $n \ge n0$

Tilde Notation(~)

The notation is useful when the function f(n) and g(n) grow at the same rate. The formal definition of this notation is

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 1$$

So if f(n) and g(n) grows at the same growth. Then one can write that

$$t(n) \sim g(n)$$

Asymptotic Rules

Reflexive Rule

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

$f(n) = \Omega(f(n))$

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

Transitivity Rule

If $f(n) = O(g(n))$ and $g(n) = O(h(n))$

Then $f(n) = O(h(n))$

Law of Composition

If $O(O(f(n))) = O(f(n))$

Law of Addition

Segment 1: complexity -> n

Segment 2: complexity -> log n

Segment 3: complexity -> n^2

Total Complexity = n + log n + n^2

$$t(n) + g(n) = O(\max\{t(n), g(n)\})$$
$$t(n) + g(n) = \Omega(\max\{t(n), g(n)\}))$$
$$t(n) + g(n) = \theta(\max\{t(n), g(n)\}))$$

Law of Multiplication

Multiplication of complexity of two functions equals to the product of two complexity functions

Let $t_1(n) = O(g_1(n))$ and $t_2(n) = O(g_2(n))$

$\Rightarrow t_1(n) \leq c_1 g_1(n) \;\; \forall\, n \geq n_1$ and $t_2(n) \leq c_2 g_2(n) \;\; \forall\, n \geq n_2$

Let K be a number greater then c_1 x c_2 and $n_0 = \max\{n_1, n_2\}$

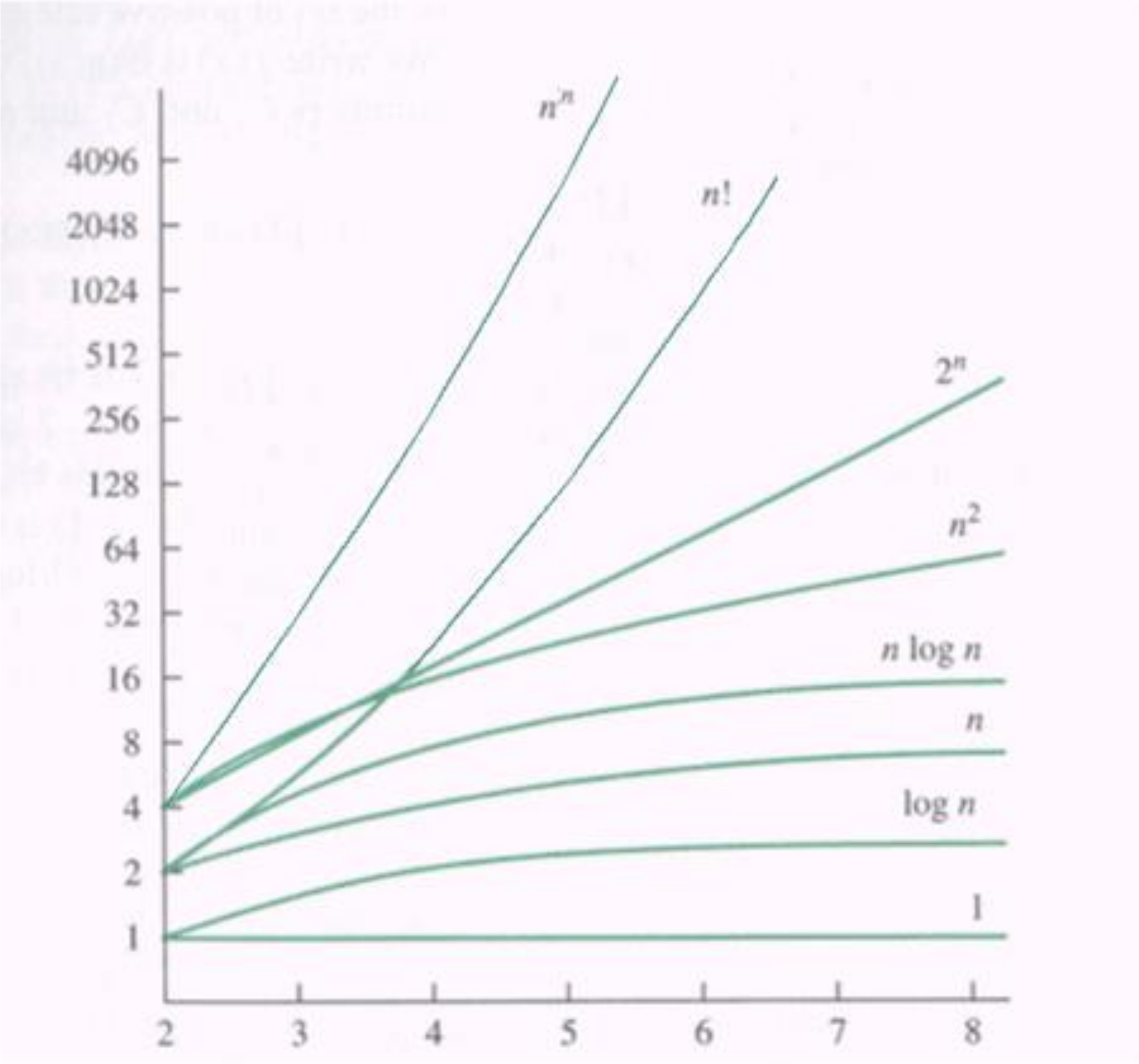
$$\begin{aligned} \Rightarrow t_1(n) \times t_2(n) &\leq c_1 g_1(n) \times c_2 g_2(n) \\ &\leq k\, g_1(n) \times g_2(n) \\ &\leq k\, (g_1(n) \times g_2(n)) \qquad \forall \;\; n \geq n_0 \end{aligned}$$

=> Therefore, $t_1(n) \times t_2(n)$ is in $O(g_1(n) \times g_2(n))$

Increasing order of Growth rate

Function Name	Name used in Algorithm analysis	Remarks
C	Constant Algorithm	Independent of input size
Log Log N	Log of Log function	Growth rate is very log
Log N	Logarithmic	Slow growth rate
N	Linear algorithm	Linear algorithms are preferred
N log N	N-Log-N	Very popular algorithm like merge sort falls here
N^2	Quadratic	General sorting algorithms
N^3	Cubic	Matrix Multiplication
N^k	Polynomial degree k	Lesser order of k are preferred
a^N	Exponential	Usage of resource is very high (Intractable)
N!	Factorial	Intractable algorithms
N^N		Intractable algorithms

Comparison of growth for classes



Execution Time

- You are executing an algorithm with time complexity order of n^2 on a CPU that can perform 10^6 operations per second. Calculate the time required to solve a worst case input of size 10? (Assume simple RAM model with cost C=1)

Time required = 0.0001 Sec

Comparison in terms of Execution Time

10^6 instructions/sec, runtimes assuming simple RAM model and C=1

N	*O(log N)*	*O(N)*	*O(N log N)*	*O(N²)*
10	0.000003	0.00001	0.000033	0.0001
100	0.000007	0.00010	0.000664	0.1000
1,000	0.000010	0.00100	0.010000	1.0
10,000	0.000013	0.01000	0.132900	1.7 min
100,000	0.000017	0.10000	1.661000	2.78 hr
1,000,000	0.000020	1.0	19.9	11.6 day
1,000,000,000	0.000030	16.7 min	18.3 hr	318 centuries

Thank You