Asymptotic Notation

In computer science it is often difficult to precisely measure certain quantities, and even when they can be measured accurately, their exact values are not essential. Accurate, but simple looking approximations could be more useful than complex exact formulas.

Asymptotic Notation

A typical example is that of a running time of an algorithm. The actual running time depends on the

- implementation,
- the compiler,
- the machine,
- the conditions under which the program is executed.

Thus it is simply not possible to tell what is the running time of an algorithm based only on its description. But, in fact, the exact formula is not even necessary.

If an algorithm makes $4n^3 + 3n + 7$ steps on inputs of length n, for large values of n,

the terms 3n and 7 are insignificant.

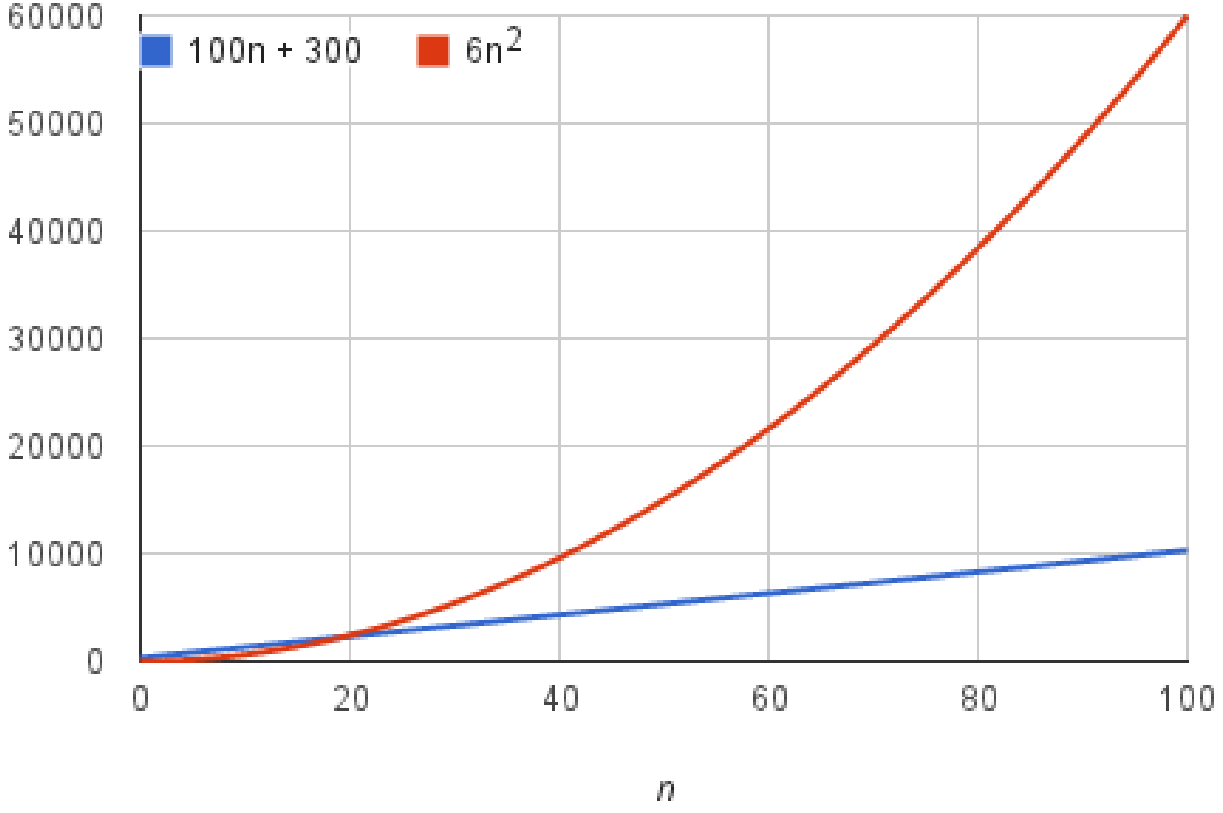
It makes more sense to say that the running time is approximately $4n^3$.

In fact, it makes sense to even ignore the constant 4, since this constant is implementation- and platform-dependent.

So we can say that the running time will be proportional to n^3.

In other words, it will be roughly cn^3 , for some constant c.

We focus on how fast this function grows with the input size. We call that the **rate of growth** of the running time. For example, suppose that an algorithm, running on an input of size n, takes $6n^2 + 100n + 300$ machine instructions. The $6n^2$ term becomes larger than the remaining terms, $100n + 300$, once n becomes large enough (20 in this case). We would say that *the running time of this algorithm grows as* n^2,

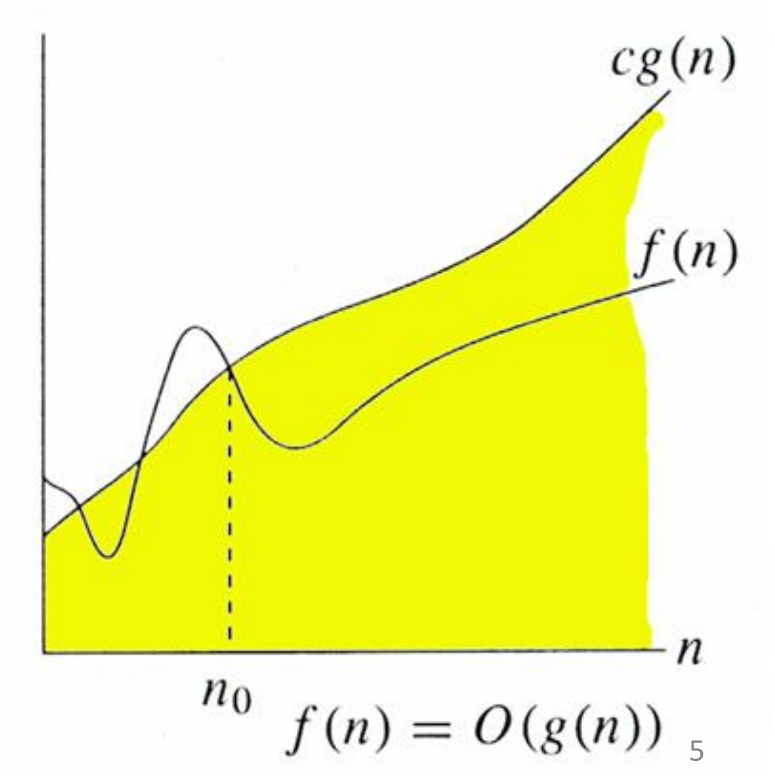


Big O

Definition. For two functions, $f(n)$ and $g(n)$, we say that $f(n)$ is of order $g(n)$, and write

$$f(n) = O\big(g(n)\big),$$

if there are positive constants c and n_0, such that $f(n) \leq cg(n)$ for all $n \geq n_0$.



Big O

<u>Definition</u>. For two functions $f(n)$ and $g(n)$, we say that $f(n)$ is of order $g(n)$, and write $f(n) = O\big(g(n)\big)$, if there are constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

<u>Examples:</u> 1. $c = O(?)$

2. $10x = O(?)$

3. $10x + 5 = O(?)$

4. $2n^3 + 6n^2 - 2 = O(?)$

Big O

Definition. For two functions $f(n)$ and $g(n)$, we say that $f(n)$ is of order $g(n)$, and write $f(n) = O(g(n))$, if there are constants c and n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Examples:

1. $c = O(1)$
2. $10x = O(x)$
3. $10x + 5 = O(x)$

   for $x \geq 1$

   $10x + 5 \leq 10x + 5x = 15x = O(x)$
4. $2n^3 + 6n^2 - 2 = O(n^3)$

   for $n \geq 1$

   $2n^3 + 6n^2 - 2 \leq 2n^3 + 6n^3 = 8n^3 = O(n^3)$

Big O

There is no unique set of values for n_0 and c in proving the asymptotic bounds;

$f(n) = n^2 + 100n + 5$

Prove that $100n + 5 = O(n^2)$

$100n + 5 \leq 100n + n = 101n \leq 101n^2$

for all $n \geq 5$

$n_0 = 5$ and $c = 101$ is a solution

$100n + 5 \leq 100n + 5n = 105n \leq 105n^2$

for all $n \geq 1$

$n_0 = 1$ and $c = 105$ is also a solution

Must find **SOME** constants c and n_0 that satisfy the asymptotic notation relation

- The main reason for using the asymptotic notation is that it allows us to estimate a complicated or difficult to determine function in terms of a simple looking function.

$$197n^5 + 13n^3 - n^2 + 21 = O(n^5)$$

- Combinations of functions.

Many functions can be expressed as sums or products of other more elementary functions. To determine the growth rate of more complicated functions, we can often find the growth rate of their components, and combine them using the following theorems.

Big O

Theorem Suppose that $f_1(n) = O\big(g_1(n)\big)$ and

$f_2(n) = O\big(g_2(n)\big)$.

Then

(a) $f_1(n) + f_2(n) \;=\; \max(O(g_1(n)), O(g_2(n)))$

(b) $f_1(n) f_2(n) = O\big(g_1(n)\big) O(g_2(n))$

Polynomials are among the most common functions that appear in the analysis of algorithms.

Big O

Theorem: Let $f(x) = \sum_{i=0}^{k} a_i x^i$. Then $f(x) = O\left(x^k\right)$.

Proof: Let $A = max|a_i|$, be the maximum absolute value of the coefficient in $f(x)$. We can estimate $f(x)$ as follows. For $x \geq 1$ we have

$$\begin{aligned} f(x) &= a_k x^k + a_{k-1} x^{k-1} + \cdots + a_1 x + a_0 \\ &\leq A(x^k + x^{k-1} + \cdots + x + 1) \\ &\leq A(k+1)x^k \end{aligned}$$

Thus $f(x) \leq cx^k$ for $c = A(k+1)$ and $x \geq 1$.The theorem follows.

Required

Asymptotic Notation
Big - O

Theorem

Let a > 0, b > 0, c > 1. Then

(a) $1 = O(\log^a n)$. (b) $\log^a n = O(n^b)$. (c) $n^b = O(c^n)$.

Proof: (c) Let $d = c^{1/b}$, then $d > 1$, and

$$\begin{aligned} n &\le 1 + d + d^2 + \ldots + d^{n-1} && \text{, since } d > 1 \\ &= \frac{d^n - 1}{d - 1} && \text{- sum of the geom. series,} \\ &\le Ad^n, && \text{where } A = 1 / (d - 1) \\ &= Ac^{(1/b)n} \end{aligned}$$

$n^b \le Bc^n$, where $B = A^b$

$n^b = O(c^n)$

Required (for EC)

Asymptotic Notation

Claim: $2^n \geq n + 1$ for all integers $n \geq 1$.

Proof (by induction) :

base case is clear;

assume the claim holds for all $n = k$, where k is an integer, $k \geq 1;$

prove that it also holds for $n = k + 1.$

$$2^{k+1} = 2 * 2^k \geq 2(k + 1) \geq 2k + 2 \geq k + 2;$$

The claim holds by the principle of induction.

This claim is true for all integers n ≥ 1.

We have: $2^n \geq n + 1$

Then: $\log (2^n) \geq \log (n + 1) > \log (n)$

$$n > \log (n)$$

and $n = O (2^n);$ $\log n = O(n)$ for n ≥ 1

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)} = \frac{1}{\log_b(a)} \log_b(n) = B \log_b(n), \text{ where } B = \frac{1}{\log_b(a)}$$

Big O

Example. Determine the asymptotic value of function $f(n) = 5n^2 log^6 n + n^3 \log n + 2\sqrt{n} + 1$.

1. We can ignore all constant coefficients.
2. Then we eliminate low order terms one by one.

   $1 = O(n^3 \log n)$, so we can discard 1

   $\sqrt{n} = n^{1/2} = O(n^3)$, so $\sqrt{n} = O(n^3 \log n)$ as well

Finally, we compare $n^2 log^6 n$ and $n^3 \log n$. Since $log^5 n = O(n)$, multiplying both sides by $n^2 \log n$ we get

$$n^2 log^6 n = O(n^3 \log n).$$

Putting it all together, we get $f(n) = O(n^3 \log n)$.

Big O

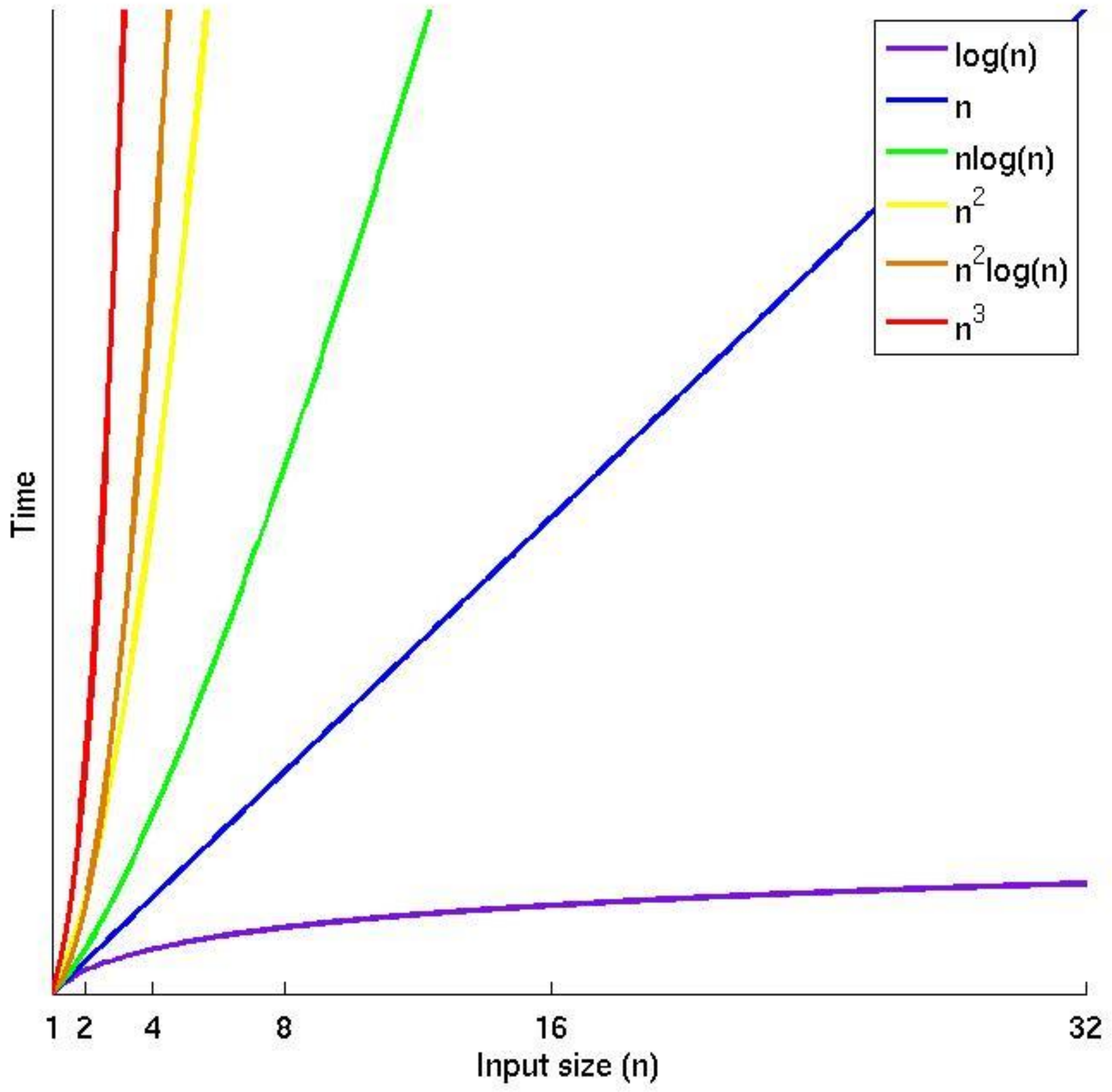
Example. Determine the asymptotic value of function $f(n) = 7n^5 2^n + 3^n$.

$$\begin{aligned} f(n) &= 7n^5 2^n + 3^n \\ &= 2^n(7n^5 + 1.5^n) \\ &= O(2^n)\,(O(1.5^n) + O(1.5^n)\,) \\ &= O(2^n)\,(O(1.5^n)\,) \\ &= O(3^n). \end{aligned}$$

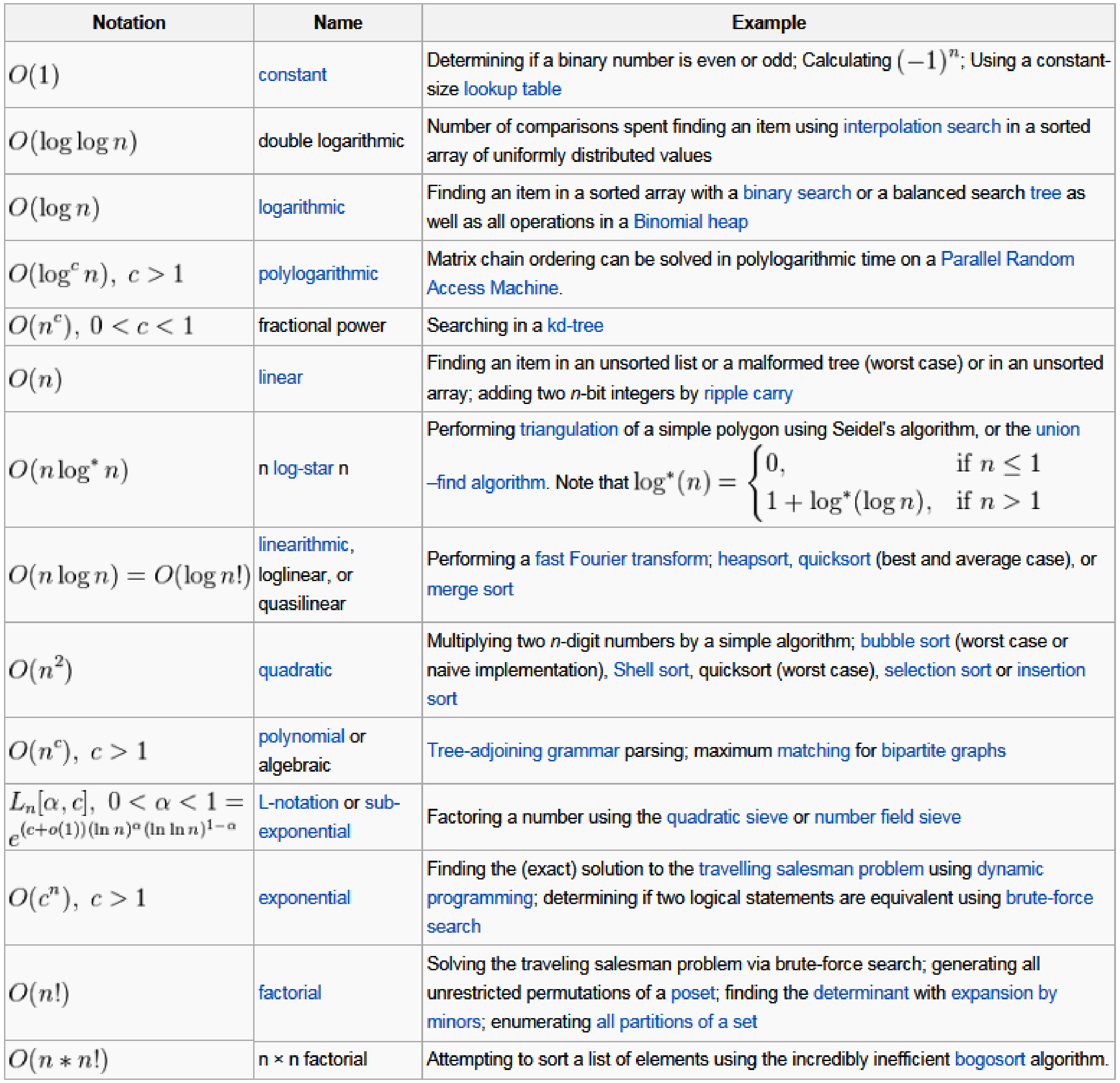
since $n^5 = O(1.5^n)$:

Thus $f(n) = O(3^n)$.

Common functions



Orders of Common functions



Notation	Name	Example
$O(1)$	constant	Determining if a binary number is even or odd; Calculating $(-1)^n$; Using a constant-size lookup table
$O(\log \log n)$	double logarithmic	Number of comparisons spent finding an item using interpolation search in a sorted array of uniformly distributed values
$O(\log n)$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$O(\log^c n),\ c > 1$	polylogarithmic	Matrix chain ordering can be solved in polylogarithmic time on a Parallel Random Access Machine.
$O(n^c),\ 0 < c < 1$	fractional power	Searching in a kd-tree
$O(n)$	linear	Finding an item in an unsorted list or a malformed tree (worst case) or in an unsorted array; adding two *n*-bit integers by ripple carry
$O(n \log^* n)$	n log-star n	Performing triangulation of a simple polygon using Seidel's algorithm, or the union–find algorithm. Note that $\log^*(n) = \begin{cases} 0, & \text{if } n \le 1 \\ 1 + \log^*(\log n), & \text{if } n > 1 \end{cases}$
$O(n \log n) = O(\log n!)$	linearithmic, loglinear, or quasilinear	Performing a fast Fourier transform; heapsort, quicksort (best and average case), or merge sort
$O(n^2)$	quadratic	Multiplying two *n*-digit numbers by a simple algorithm; bubble sort (worst case or naive implementation), Shell sort, quicksort (worst case), selection sort or insertion sort
$O(n^c),\ c > 1$	polynomial or algebraic	Tree-adjoining grammar parsing; maximum matching for bipartite graphs
$L_n[\alpha, c],\ 0 < \alpha < 1 = e^{(c+o(1))(\ln n)^\alpha (\ln \ln n)^{1-\alpha}}$	L-notation or sub-exponential	Factoring a number using the quadratic sieve or number field sieve
$O(c^n),\ c > 1$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$O(n!)$	factorial	Solving the traveling salesman problem via brute-force search; generating all unrestricted permutations of a poset; finding the determinant with expansion by minors; enumerating all partitions of a set
$O(n * n!)$	n × n factorial	Attempting to sort a list of elements using the incredibly inefficient bogosort algorithm.

Big O

Example. A sequence $\{a_i\}$ is defined by

$$a_0 = 3, a_1 = 8, a_n = 2a_{n-1}+a_{n-2}$$

Prove that $a_n = O(2.75^n)$.

Proof: In order to prove this, we show by induction that $a_n \leq 3(2.75)^n$.

1. *Base cases:* $a_0 = 3 = 3(2.75)^0$ and $a_1 = 8 < 3(2.75)^1$.
2. *Inductive step*: assume the claim holds for numbers less than n.
3. For n, by the inductive assumption, we get

$$\begin{aligned} a_n &= 2a_{n-1} + a_{n-2} \\ &\leq 2 \cdot 3(2.75)^{n-1} + 3(2.75)^{n-2} \\ &\leq 3(2.75)^{n-2}(2 \cdot 2.75 + 1) \\ &\leq 3(2.75)^{n-2}(2.75)^2 \\ &\leq 3(2.75)^n \end{aligned}$$

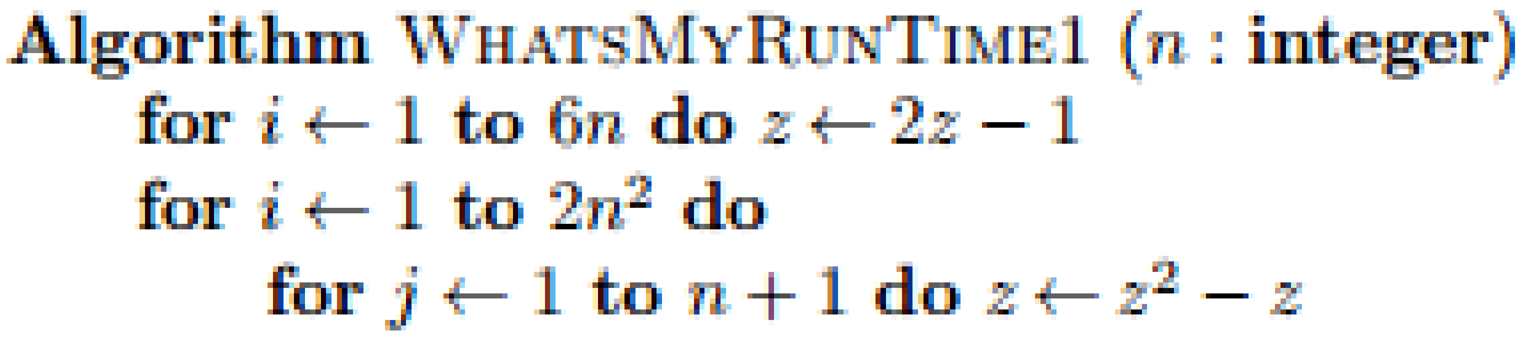
and thus the claim holds for n as well. Therefore it holds for all values of n.

Big O

What’s the running time of the algorithm below?

```
Algorithm WhatsMyRunTime1 ($n$ : integer)
    for $i \leftarrow 1$ to $6n$ do $z \leftarrow 2z - 1$
    for $i \leftarrow 1$ to $2n^2$ do
        for $j \leftarrow 1$ to $n + 1$ do $z \leftarrow z^2 - z$
```

Big O



```
Algorithm WhatsMyRunTime1 ($n$ : integer)
    for $i \leftarrow 1$ to $6n$ do $z \leftarrow 2z - 1$
    for $i \leftarrow 1$ to $2n^2$ do
        for $j \leftarrow 1$ to $n + 1$ do $z \leftarrow z^2 - z$
```

The first loop makes $6n$ iterations,

The second double loop makes

$2n^2 \cdot (n + 1) = 2n^3 + 2n^2$ iterations,

So, the total is $2n^3 + 2n^2 + 6n$ iterations.

For $n \geq 1$, this is at most $2n^3 + 2n^3 + 6n^3 \leq 10n^3$, so the running time is $O(n^3)$.

Big O

Example: determine the number of words printed by the following pseudo-code

```
Algorithm HowManyHellos ($n$ : integer)
    for $i \leftarrow 1$ to $6n$ do print("HELLO")
    for $i \leftarrow 1$ to $4n + 1$ do
        for $j \leftarrow 1$ to $3i + 2$ do print("HELLO")
```

```
Algorithm HowManyHellos ($n$ : integer)
    for $i \leftarrow 1$ to $6n$ do print("HELLO")
    for $i \leftarrow 1$ to $4n+1$ do
        for $j \leftarrow 1$ to $3i+2$ do print("HELLO")
```

The first loop will print 6n words.

Note: Unlike in Algorithm WhatsMyRunTime1, we cannot simply multiply the ranges of the two loops, because the number of iterations in the inner loop depends on i; it changes from one iteration to another.

The second nested loop, for any given i, will print $3i + 2$ words.

The total number of printed words is

$$6n + \sum_{i=1}^{4n+1} (3i + 2) = 6n + \sum_{i=1}^{4n+1} (3i) + \sum_{i=1}^{4n+1} 2$$

$$= 6n + 3\sum_{i=1}^{4n+1} i + 2(4n + 1)$$

$$= 6n + \frac{3(4n+1)(4n+2)}{2} + 2(4n + 1)$$

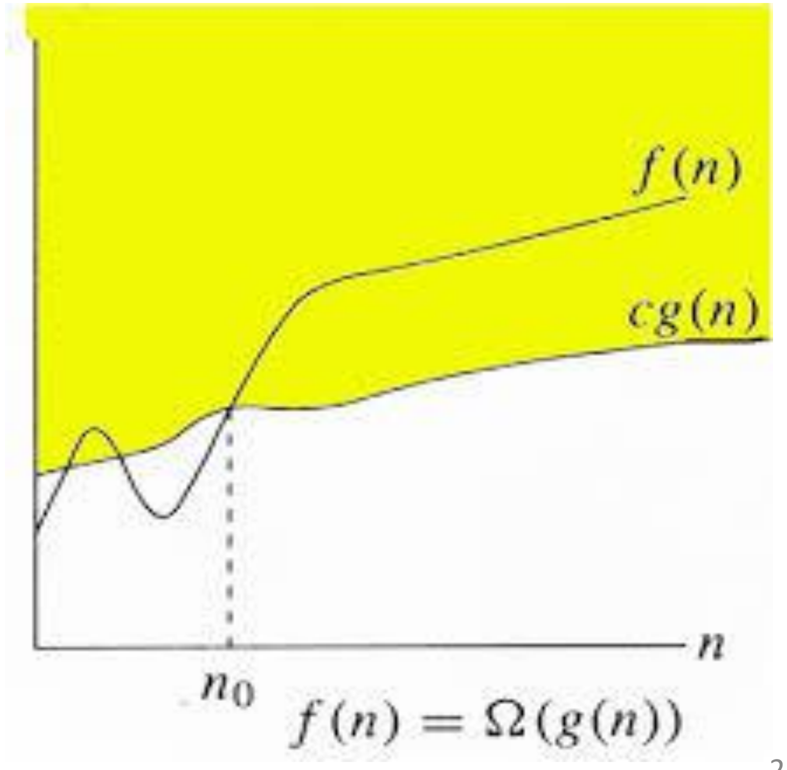
$$= 24n^2 + 32n + 5 = O(n^2)$$

Big Ω

Definition. We say that $f(n)$ is $\Omega(g(n))$, and write

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

if there are positive constants c and n_0, such that $f(n) \geq cg(n)$ for all $n \geq n_0$.



Big Ω

Definition. We say that $f(n)$ is $\Omega(g(n))$, and write $f(n) = \Omega\big(g(n)\big)$ if there are positive constants c and n_0, such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Example: $3n^2 + 2n - 7 = \Omega(n^2)$

for $n \geq 4$,

$2n \geq 7$, so

$3n^2 + 2n - 7 \geq 3n^2 = \Omega(n^2)$

Big θ

Definition. We say that $f(n)$ is $\theta(g(n))$, and write

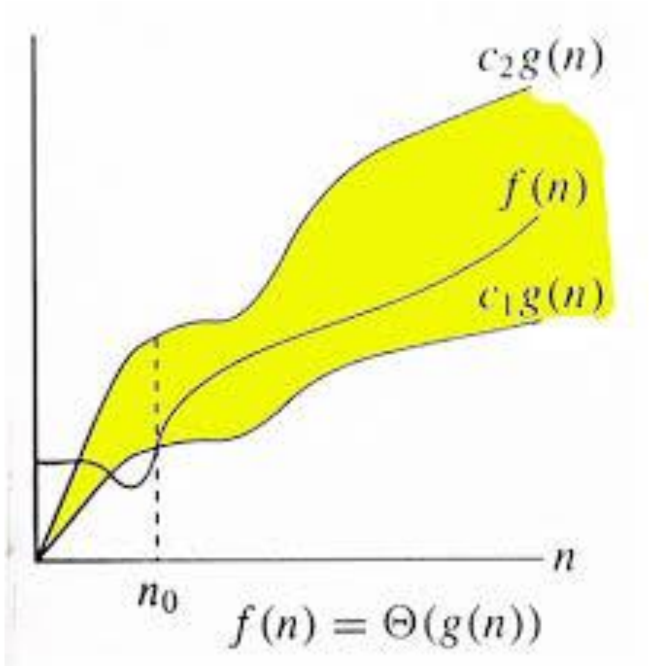
$$f(n) = \theta\big(g(n)\big),$$

if there are positive constants

c_1, c_2, and n_0, such that

$c_1 g(n) \geq f(n) \geq c_2 g(n)$

for all $n \geq n_0$.



Theorem. For any two functions $f(n)$ and $g(n)$,

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

if and only if $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega(g(n))$.

Big θ

Example: $3n^2 + 2n - 7 = O(n^2)$

and

$3n^2 + 2n - 7 = \Omega(n^2)$,

so

$3n^2 + 2n - 7 = \theta(n^2)$

Big θ

Examples: Give the asymptotic values of the following functions:

(a) $394n^2 + n^5 + \frac{1}{2} \cdot n^4 + 3n$

(b) $3/n + n^{-3} + 2/\log n$

(c) $n^2(n^2 \log n + 8n\, log^5 n) + 10n^3$

(d) $n^6 2^n + n3^n + 43n^{11}$

Big θ

(a) $394n^2 + n^5 + \frac{1}{2} \cdot n^4 + 3n = \theta(n^5)$ because it is a polynomial of degree 5.

Big θ

(b) $f(n) = 3/n + n^{-3} + 2/\log n$

Let's show that $f(n) = O(1/\log n)$

For $n \geq 2,\ \ n \geq \log n$, so $1/\text{n} \leq 1/\log n \Rightarrow$

$$3/n = O(1/\log n).$$

Similarly $n^{-3} \leq O(1/\log n) \Rightarrow n^{-3} = O(1/\log n)$

$$\begin{aligned} f(n) &= 3/n + n^{-3} + 2/\log n \\ &= O(1/\log n) + O(1/\log n) + O(1/\log n) = O(1/\log n) \end{aligned}$$

We have shown that $f(n) = O(1/\log n)$. We can similarly show that $f(n) = \Omega(1/\log n)$. Therefore $f(n) = \theta(1/\log n)$.

Big θ

(c) $f(n) = n^2(n^2 \log n + 8n\, log^5 n) + 10n^3$
$= n^4 \log n + 8n^3\, log^5 n + 10n^3$

For n $\geq$ 1:

1. $f(n) \geq n^4 \log n \Rightarrow f(n) = \Omega(n^4 \log n)$

2. Let’s show that $f(n) = O(n^4 \log n)$

   $n^3\, log^5 n = n^3\, (\log n)(log^4 n) = n^3\, (\log n) O(n) = O(n^4 \log n)$

   $n^3 \leq n^4 \log n$ for $n \geq 2$, so $10n^3 = O(n^4 \log n)$

   $f(n) = n^4 \log n + 8n^3\, log^5 n + 10n^3 = O(n^4 \log n) + O(n^4 \log n) + O(n^4 \log n) = O(n^4 \log n)$

We have shown that $f(n) = \Omega(n^4 \log n)$ and $f(n) = O(n^4 \log n)$.
Therefore $f(n) = \theta(n^4 \log n)$

Big θ

$(d) f(n) = n^6 2^n + n3^n + 43n^{11}$

For n $\geq$ 1:

1. $f(n) \geq n3^n \Rightarrow f(n) = \Omega(n3^n)$
2. Let's show that $f(n) = O(n3^n)$

$$n^{11} = n \cdot n^{10} = nO(3^n) = O(n3^n)$$
$$n^6 2^n = n \cdot n^5 2^n = nO(1.5^n \cdot 2^n) = O(n3^n)$$

We have shown that $f(n) = \Omega(n3^n)$ and $f(n) = O(n3^n)$.
Therefore $f(n) = \theta(n3^n)$