Chapter 2: Algorithm Complexity

Section 2.1: Big-Theta notation

Unlike Big-O notation, which represents only upper bound of the running time for some algorithm, Big-Theta is a tight bound; both upper and lower bound. Tight bound is more precise, but also more difficult to compute.

The Big-Theta notation is symmetric: `f(x) = Θ(g(x)) <=> g(x) = Θ(f(x))`

An intuitive way to grasp it is that `f(x) = Θ(g(x))` means that the graphs of f(x) and g(x) grow in the same rate, or that the graphs 'behave' similarly for big enough values of x.

The full mathematical expression of the Big-Theta notation is as follows:
Θ(f(x)) = {g: N0 -> R and c1, c2, n0 > 0, where c1 < abs(g(n) / f(n)), for every n > n0 and abs is the absolute value }

An example

If the algorithm for the input `n` takes `42n^2 + 25n + 4` operations to finish, we say that is `O(n^2)`, but is also `O(n^3)` and `O(n^100)`. However, it is `Θ(n^2)` and it is not `Θ(n^3)`, `Θ(n^4)` etc. Algorithm that is `Θ(f(n))` is also `O(f(n))`, but not vice versa!

Formal mathematical definition

`Θ(g(x))` is a set of functions.

```
Θ(g(x)) = {f(x) such that there exist positive constants c1, c2, N such that 0 <= c1*g(x) <= f(x) <= c2*g(x) for all x > N}
```

Because `Θ(g(x))` is a set, we could write `f(x) ∈ Θ(g(x))` to indicate that `f(x)` is a member of `Θ(g(x))`. Instead, we will usually write `f(x) = Θ(g(x))` to express the same notion - that's the common way.

Whenever `Θ(g(x))` appears in a formula, we interpret it as standing for some anonymous function that we do not care to name. For example the equation `T(n) = T(n/2) + Θ(n)`, means `T(n) = T(n/2) + f(n)` where `f(n)` is a function in the set `Θ(n)`.

Let `f` and `g` be two functions defined on some subset of the real numbers. We write `f(x) = Θ(g(x))` as `x->infinity` if and only if there are positive constants `K` and `L` and a real number `x0` such that holds:

`K|g(x)| <= f(x) <= L|g(x)|` for all `x >= x0`.

The definition is equal to:

`f(x) = O(g(x)) and f(x) = Ω(g(x))`

A method that uses limits

if `limit(x->infinity) f(x)/g(x) = c ∈ (0,∞)` i.e. the limit exists and it's positive, then `f(x) = Θ(g(x))`

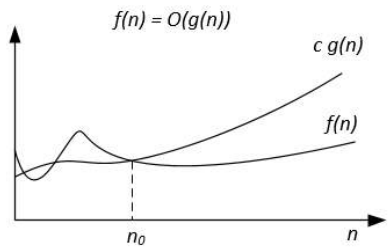
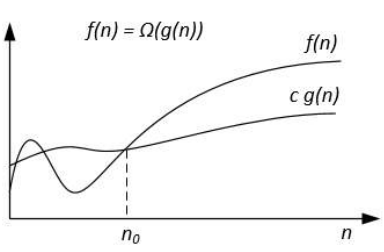
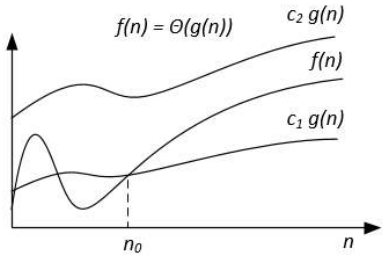
Common Complexity Classes

Name	Notation	n = 10	n = 100
Constant	Θ(1)	1	1
Logarithmic	Θ(log(n))	3	7
Linear	Θ(n)	10	100

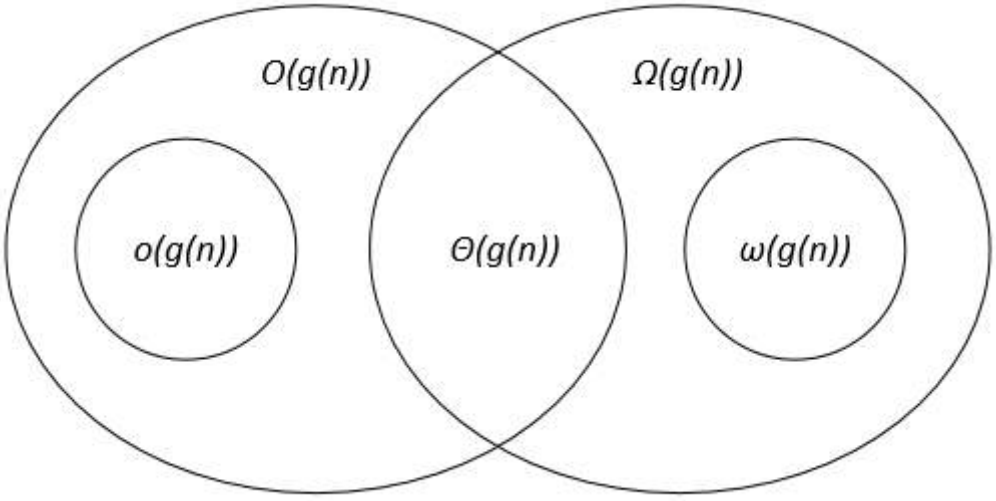
Linearithmic	Θ(n*log(n))	30	700
Quadratic	Θ(n^2)	100	10 000
Exponential	Θ(2^n)	1 024	1.267650e+ 30
Factorial	Θ(n!)	3 628 800	9.332622e+157

Section 2.2: Comparison of the asymptotic notations

Let `f(n)` and `g(n)` be two functions defined on the set of the positive real numbers, `c`, `c1`, `c2`, `n0` are positive real constants.

Notation	f(n) = O(g(n))	f(n) = Ω(g(n))	f(n) = Θ(g(n))	f(n) = o(g(n))	f(n) = ω(g(n))
Formal definition	`∃ c > 0, ∃ n0 > 0 : ∀ n ≥ n0, 0 ≤ f(n) ≤ c g(n)`	`∃ c > 0, ∃ n0 > 0 : ∀ n ≥ n0, 0 ≤ c g(n) ≤ f(n)`	`∃ c1, c2 > 0, ∃ n0 > 0 : ∀ n ≥ n0, 0 ≤ c1 g(n) ≤ f(n) ≤ c2 g(n)`	`∀ c > 0, ∃ n0 > 0 0 : ∀ n ≥ n0, 0 ≤ f(n) < c g(n)`	`∀ c > 0, ∃ n0 > 0 : ∀ n ≥ n0, 0 ≤ c g(n) < f(n)`
Analogy between the asymptotic comparison of `f`, `g` and real numbers `a`, `b`	`a ≤ b`	`a ≥ b`	`a = b`	`a < b`	`a > b`
Example	`7n + 10 = O(n^2 + n - 9)`	`n^3 - 34 = Ω(10n^2 - 7n + 1)`	`1/2 n^2 - 7n = Θ(n^2)`	`5n^2 = o(n^3)`	`7n^2 = ω(n)`
Graphic interpretation					

The asymptotic notations can be represented on a Venn diagram as follows:



Links

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms.

Section 2.3: Big-Omega Notation

Ω-notation is used for asymptotic lower bound.

Formal definition

Let `f(n)` and `g(n)` be two functions defined on the set of the positive real numbers. We write `f(n) = Ω(g(n))` if there are positive constants `c` and `n0` such that:

`0 ≤ c g(n) ≤ f(n) for all n ≥ n0`.

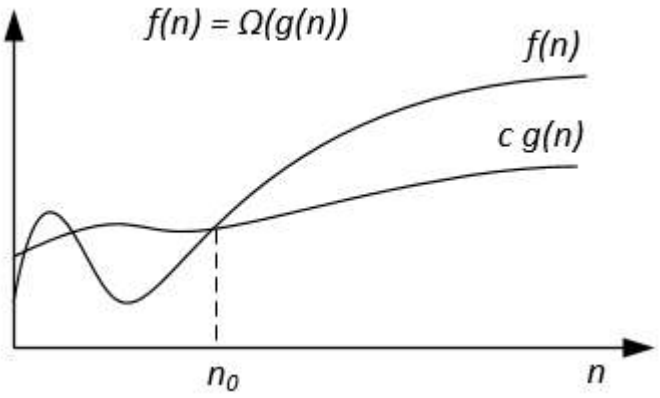
Notes

`f(n) = Ω(g(n))` means that `f(n)` grows asymptotically no slower than `g(n)`. Also we can say about `Ω(g(n))` when algorithm analysis is not enough for statement about `Θ(g(n))` or / and `O(g(n))`.

From the definitions of notations follows the theorem:

For two any functions `f(n)` and `g(n)` we have `f(n) = Θ(g(n))` if and only if `f(n) = O(g(n)) and f(n) = Ω(g(n))`.

Graphically Ω-notation may be represented as follows:



For example lets we have `f(n) = 3n^2 + 5n - 4`. Then `f(n) = Ω(n^2)`. It is also correct `f(n) = Ω(n)`, or even `f(n) = Ω(1)`.

Another example to solve perfect matching algorithm : If the number of vertices is odd then output "No Perfect Matching" otherwise try all possible matchings.

We would like to say the algorithm requires exponential time but in fact you cannot prove a `Ω(n^2)` lower bound using the usual definition of Ω since the algorithm runs in linear time for n odd. We should instead define `f(n)=Ω(g(n))` by saying for some constant `c>0`, `f(n)≥ c g(n)` for infinitely many `n`. This gives a nice correspondence between upper and lower bounds: `f(n)=Ω(g(n))` iff `f(n) != o(g(n))`.

References

Formal definition and theorem are taken from the book "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms".