Applied Algorithms

CSCI-B505 / INFO-I500

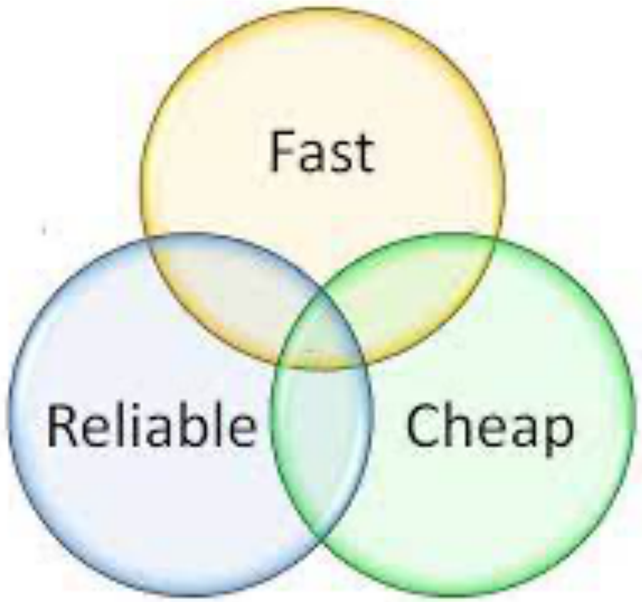
Lecture 1.

Algorithm Analysis and Asymptotic Notation

M. Oğuzhan Kulekci

How can we measure the performance of an algorithm?

The parameters



Correctness: Works on all cases, general

Elegance: Easy to communicate and code

Efficiency :

- **Time**
- **Space**

ON THE COMPUTATIONAL COMPLEXITY OF ALGORITHMS

BY

J. HARTMANIS AND R. E. STEARNS

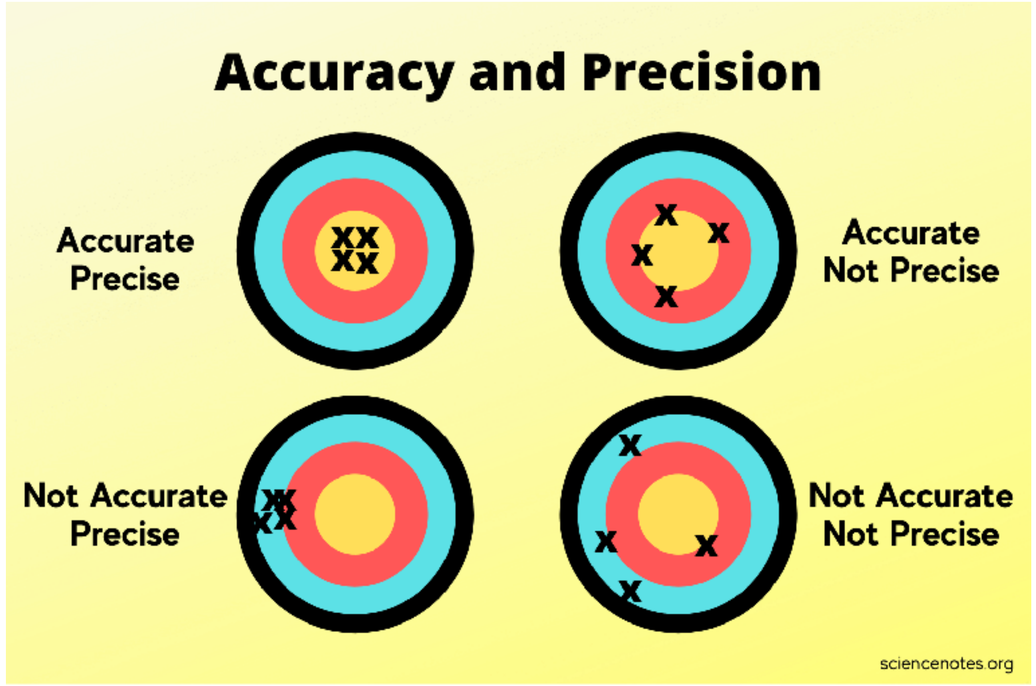
I. **Introduction.** In his celebrated paper [1], A. M. Turing investigated the computability of sequences (functions) by mechanical procedures and showed that the set of sequences can be partitioned into computable and noncomputable sequences. One finds, however, that some computable sequences are very easy to compute whereas other computable sequences seem to have an inherent complexity that makes them difficult to compute. In this paper, we investigate a scheme of classifying sequences according to how hard they are to compute. This scheme puts a rich structure on the computable sequences and a variety of theorems are established. Furthermore, this scheme can be generalized to classify numbers, functions, or recognition problems according to their computational complexity.

Hartmanis & Stearn'1965: Time and space complexity analysis of algorithms

Experimental versus **theoretical** *evaluation* ?

Experimental vs. Theoretical Measurements?

Experimental vs. Theoretical Analysis



https://sciencenotes.org/what-is-the-difference-between-accuracy-and-precision/

Experimental: PRECISE but SPECIFIC

Theoretical: ACCURATE and GENERAL

To measure the general time/space performance of an algorithm,
we follow the theoretical complexity analysis
by using the mathematical notion of
ASYMPTOTIC ANALYSIS

ACCURACY: How close your measurements to the truth

PRECISION: How close your measurements to each other

Experimental vs. Theoretical Analysis

The experimental analysis

- Implement the algorithm and execute on some inputs !
- Many specifications such as the programming language, programmer, compiler, the machine (CPU, memory, I/O channels, etc …) are required.

Very PRECISE on the SPECIFIC case, but …

Some further questions:

- How long will it take on another input size ?
- Best, average, and worst case scenarios ?
- Is it optimal ?
- …

If we have an **abstract** way of analyzing the performance, such questions would be easier to answer **without need of explicit experiments.**

Experimental vs. Theoretical Measurements?

The theoretical analysis

- Assume an abstract machine *(actually, this is the **word-RAM** model)* such that :
  - All instructions (arithmetic, memory access, control, etc…) are *atomic,* which take equal time
  - We have enough memory, and can read/write in blocks of log n bits, for an input size of n.

Without executing the code itself,
just investigate how many instructions would be
executed on an input size of n,
and return it as a function of the input size.

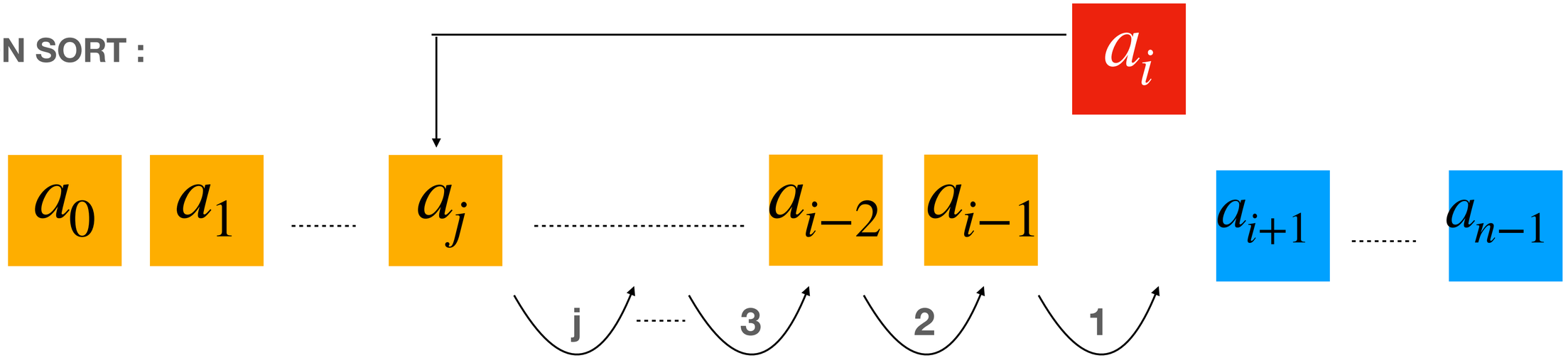
Very ACCURATE on the GENERAL case!

Let’s pass over an example case…

Experimental vs. Theoretical Measurements?

Time Complexity of Insertion Sort

INSERTION SORT :



- Assume a_0 to a_{i-1} is sorted
- Save a_i in a temporary variable
- Scan elements towards left, and shift right by one if it is smaller than a_i
- When the check fails, or the beginning of the list is reached, insert a_0 to the vacant position

Iterate through $i = 1$ to $i = n - 1$

Investigating the Time Complexity of Insertion Sort

```
      a[0..n-1] contains elements to be sorted
1     for (i = 1; i < n; i++ ) {
2         // Invariant: a[0..i-1] is sorted
          // Invariant: a[i..n-1] not yet sorted
3         int tmp = a[i];
4         for (j=i; (j>0 && a[j]>tmp); j--) {
            // Invariant: hole is at a[j]
5           a[j] = a[j-1];
6         }
7         a[j] = tmp;
      }
```

EXECUTION COUNT	COST
n	c_1
$n-1$	c_2
$\sum_{j=1}^{n-1} t_j$	c_3
$\sum_{j=1}^{n-1} (t_j - 1)$	c_4
$n-1$	c_5

t_j is the number times a_j is compared with the previous elements.

Notice that t_j is not related with the input size n, but the values in the array.

Experimental vs. Theoretical Measurements?

Investigating the Time Complexity of Insertion Sort

Total running time of insertion sort is

$$C(n) = c_1 \cdot n + (c_2 + c_5) \cdot (n-1) + c_3 \cdot \sum_{j=1}^{j=n} t_j + c_4 \cdot \sum_{j=1}^{j=n} (t_j - 1)$$

Worst case: Input sequence is in decreasing order, thus $t_j = j$ and

quadratic function

$$C(n) = c_1 \cdot n + (c_2 + c_5) \cdot (n-1) + c_3 \cdot \frac{n(n+1)}{2} + c_4 \cdot \frac{n(n-1)}{2} = A \cdot n^2 + B \cdot n + C$$

Best case: Input sequence is in increasing order, thus $t_j = 1$ and

$$C(n) = c_1 \cdot n + (c_2 + c_5) \cdot (n-1) + c_3 \cdot n + c_4 \cdot 0 = D \cdot n + E$$

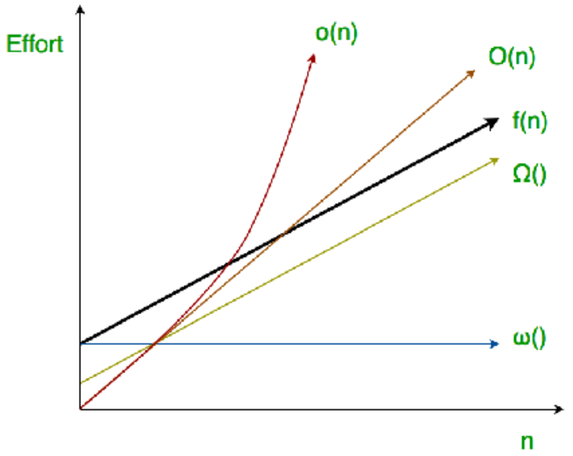
linear function

We don't know the values of the constants A, B, C, D, E exactly, but we know how $C(n)$ will change according to input size n !

Asymptotic Notation

Big-Oh, Big-Omega, Big-Theta, Little-Oh, Little-Omega

How do the running time and the memory consumption of an algorithm change according to the input size ?



$$O(), \Omega(), \Theta(), o(), w()$$

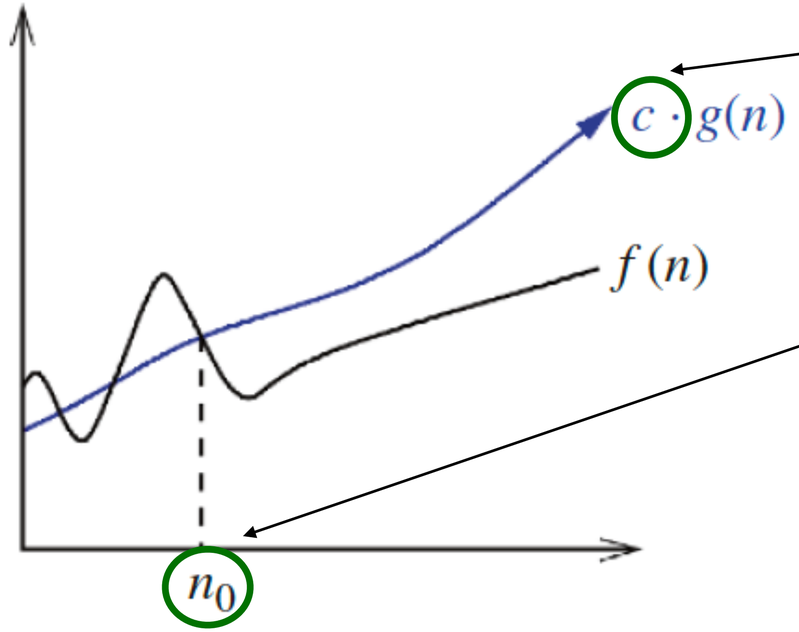
Bachman (1894), Landau (1909): Hardy&Littlewood(1914); Knuth (1970s)

- Represent the number of instructions and the memory allocation of an algorithm as a function of the input size.
- Such functions may include many terms of different orders
- **Lower order terms will not make much difference when input size gets large**
- $O(), \Omega(), \Theta(), o(), w()$ notations are used to observe the growth of functions

Big-Oh

$O(g(n))$ defines a **SET** of functions such that there exist positive constants $c > 0$ and $n_0 > 0$, where $f(n)$ is always less than or equal to $c \cdot g(n)$ for all $n > n_0$.

$$O(g(n)) = \{f(n) \mid 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$



All $f(n)$ functions are UPPER BOUNDED by $g(n)$

Big-Oh

$$O(g(n)) = \{f(n) \mid 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$

To prove $f(n) \in O(g(n))$ **we need to find explicit values of** c and n_0 ,where $f(n)$ is always less than or equal to $c \cdot g(n)$ for all $n > n_0$.

Example:

If $f(n) = 4n^2 + 1000000n + 10^{34}$, then $f(n) \in O(n^2)$.

Proof:

For $c = \mathbf{5}$, $n_0 = \mathbf{10^{18}}$; $4n^2 + 10^6 n + 10^{34} \leq \mathbf{5} \cdot n^2, \forall n > \mathbf{10^{18}}$

After a threshold value, $f(n)$ *is* ***upper bounded*** *by* $g(n) = n^2$.

Little-Oh

$o(g(n))$ defines a **SET** of functions such that **for any** $c > 0$, there is an $n_0 > 0$, where $f(n)$ is always **less than** $c \cdot g(n)$ for all $n \geq n_0$.

$$o(g(n)) = \{f(n) \mid 0 \leq f(n) < c \cdot g(n), \forall n \geq n_0, \forall c > 0, \exists n_0 > 0\}$$

$$O(g(n)) = \{f(n) \mid 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$

Examples:

- $5n + 17 \in o(n^2)$
- $n \log n \in o(n^{1.0001})$

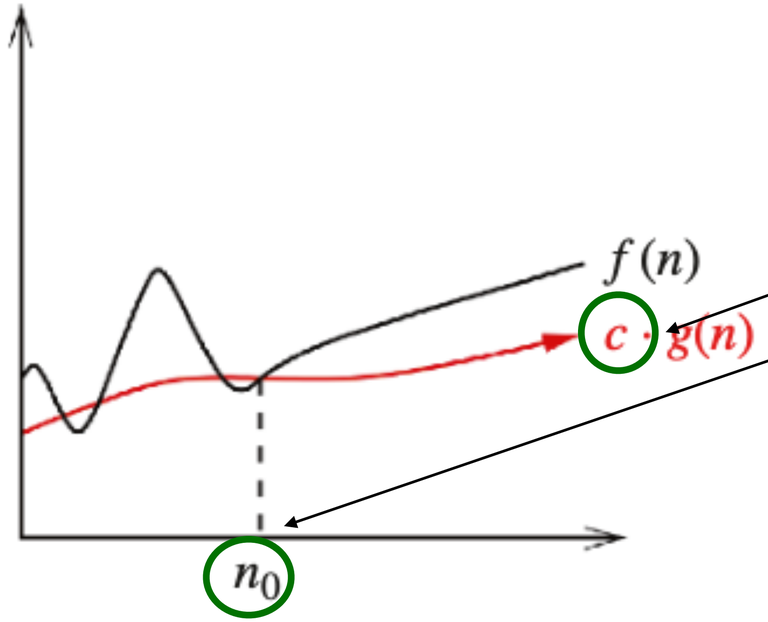
Alternative definition:

$$\{f(n) \mid \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0\}$$

Big-Omega

$\Omega(g(n))$ defines a **SET** of functions such that there exist positive constants $c > 0$ and $n_0 > 0$, where $f(n)$ is always greater than or equal to $c \cdot g(n)$ for all $n > n_0$.

$$\Omega(g(n)) = \{f(n) \,|\, 0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$



All $f(n)$ functions are LOWER BOUNDED by $g(n)$

Big-Omega

$$\Omega(g(n)) = \{f(n) \mid 0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$

To prove $f(n) \in \Omega(g(n))$ **we need to find explicit values of** c and n_0 ,where $f(n)$ is always **larger than or equal** to $c \cdot g(n)$ for all $n > n_0$.

Example:

Given $f(n) = 0.01n^2 + 0.1n + 3,\ f(n) \in \Omega(n)$.

Proof:

For $c = 1,\ n_0 = 90;\ \ 0.01n^2 + 4n + 3 \geq \mathbf{1} \cdot n, \forall n > \mathbf{90}$

*After a threshold value, $f(n)$ is **lower bounded** by $g(n) = n$.*

Little-Omega

$\omega(g(n))$ defines a **SET** of functions such that **for any** $c > 0$, there is an $n_0 > 0$, where $f(n)$ is always **greater than** $c \cdot g(n)$ for all $n \geq n_0$.

$$\omega(g(n)) = \{f(n) \,|\, 0 \leq c \cdot g(n) < f(n), \forall n \geq n_0, \forall c > 0, \exists n_0 > 0\}$$

$$\Omega(g(n)) = \{f(n) \,|\, 0 \leq c \cdot g(n) \leq f(n), \forall n \geq n_0, \exists c > 0, \exists n_0 > 0\}$$

Examples:

- $f(n) = 5n^3 + 6n - 17 \in \omega(g(n) = n^2)$
- $f(n) = n^{1.0001} \in \omega(g(n) = \log n)$

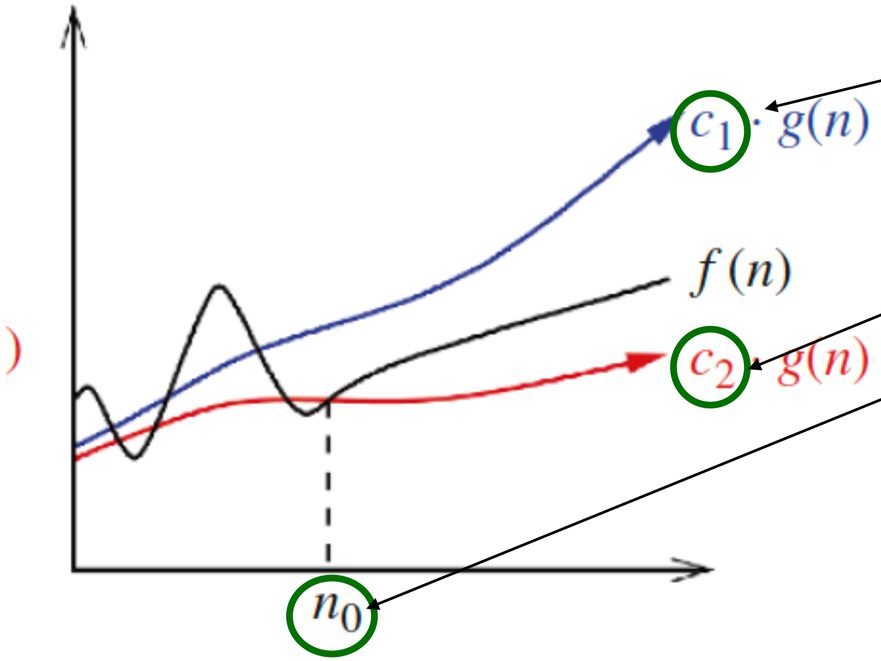
Alternative definition:

$$\{f(n) \,|\, \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0\}$$

Big-Theta $\Theta()$

$\Theta(g(n))$ defines a **SET** of functions such that there exist positive constants $c_1 > 0$ and $c_2 > 0$ and $n_0 > 0$, where $f(n)$ is always less than or equal to $c_1 \cdot g(n)$ and greater than or equal to $c_2 \cdot g(n)$ for all $n > n_0$.

$$\Theta(g(n)) = \{f(n) \mid 0 \leq c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)), \forall n \geq n_0, \exists(c_1 > 0, c_2 > 0, n_0 > 0)\}$$



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

All $f(n)$ functions that are upper and lower bounded by the SAME $g(n)$.

Big-Theta

$$\Theta(g(n)) = \{f(n) \mid 0 \leq c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)), \forall n \geq n_0, \exists(c_1 > 0, c_2 > 0, n_0 > 0)\}$$

To prove $f(n) \in \Theta(g(n))$ **we need to find explicit values of** c_1, c_2 and n_0 ,where $f(n)$ is always less than or equal to $c_1 \cdot g(n)$ and greater than $c_2 \cdot g(n)$ for all $n > n_0$.

Example:

If $f(n) = 4n^2 + 10^6 n + 10^{34}$, then $f(n) \in \Theta(n^2)$.

Proof:

For $c_1 = \mathbf{5}$ and $c_2 = \mathbf{1}$ and $n_0 = \mathbf{10^{18}}$;

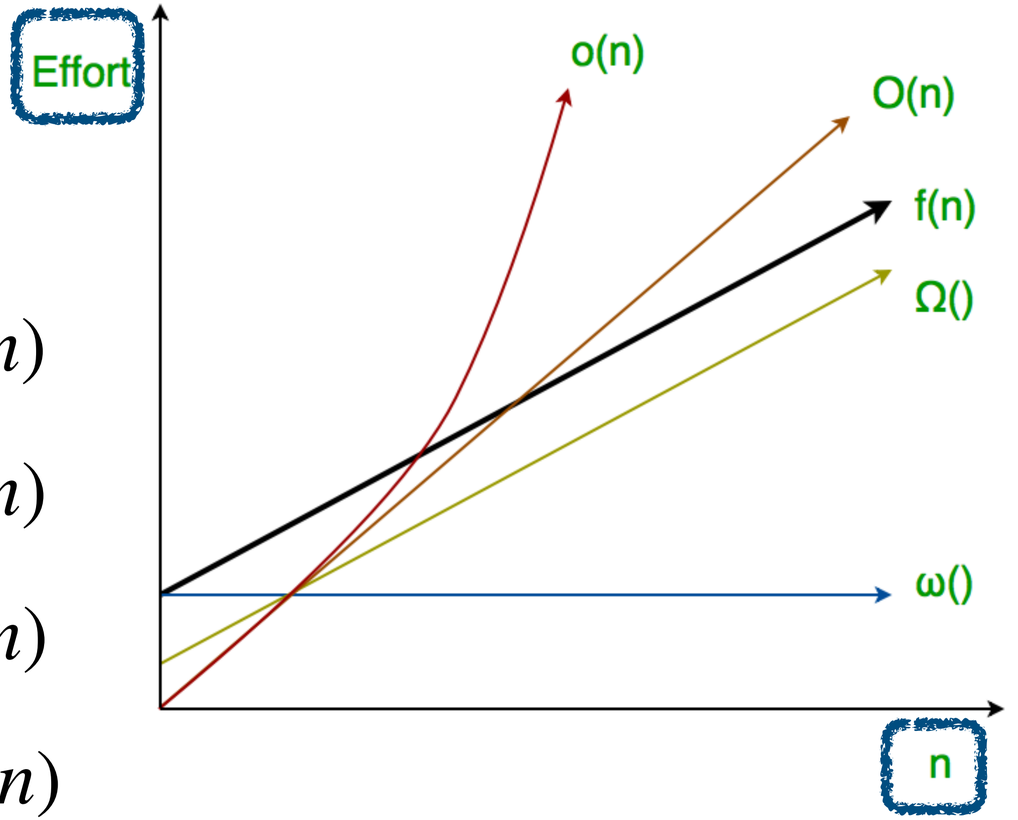
$$\mathbf{1} \cdot n^2 \leq 4n^2 + 10^6 n + 10^{34} \leq \mathbf{5} \cdot n^2, \forall n > n_0$$

*After a threshold value, $f(n)$ is **upper and lower bounded** by $g(n) = n^2$.*

Some Relationships in Asymptotic Notation

- $f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$
- $f(n) \in o(g(n)) \iff g(n) \in \omega(f(n))$
- $f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \;\; AND \;\; f(n) \in \Omega(g(n))$
- Can you prove them ?

Some Analogies in Asymptotic Notation



- $f(n) \in O(g(n))$ is like $\leq$ operator: $f(n) \leq cg(n)$
- $f(n) \in o(g(n))$ is like $<$ operator: $f(n) < cg(n)$
- $f(n) \in \Omega(g(n))$ is like $\geq$ operator: $f(n) \geq cg(n)$
- $f(n) \in \omega(g(n))$ is like $>$ operator: $f(n) > cg(n)$
- $f(n) \in \Theta(g(n))$ is like $=$ operator: $f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n))$

Questions or comments ?

- Please make sure you understand the meaning and the math behind.

Next lecture we will be studying analysis of algorithms with asymptotic notation.