Data Structures

Performance

Ja-Hee Kim@seoultech

Contents

Data Structures . Performance

Motivation

Example

$$sum = \sum_{i=1}^{n} i$$

Algorithm A	Algorithm B	Algorithm C

sum = 0
for i = 1 *to* n
    sum = sum + i
	sum = 0
	for i = 1 *to* n
	{
	    for j = 1 *to* i
	        sum = sum + 1
	}
		sum = n * (n + 1) / 2

Most Efficient

- Lower complexity is better
- Usually the "best" solution to a problem balances various criteria such as time, space, generality, programming effort, and so on.
- Time complexity:
  - Time requirement
  - the time it takes to execute
- Space complexity:
  - Space requirements
  - the memory it needs to execute

Sum from 1 to n

- How much time to add 1 … n

- Algorithm A

```java
long n = 10000;

// Algorithm A
long t0 = System.currentTimeMillis();
long sum = 0;
for(long i = 1; i <= n ; i++)
    sum = sum+i;
long t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

- Algorithm B

```java
//Algorithm B
t0 = System.currentTimeMillis();
sum = 0;
for(long i = 1; i <= n ; i++)
    for(long j = 1; j<=i; j++)
        sum = sum+1;
t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

- Algorithm C

```java
// Algorithm C
t0 = System.currentTimeMillis();
sum = n*(n+1)/2;
t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

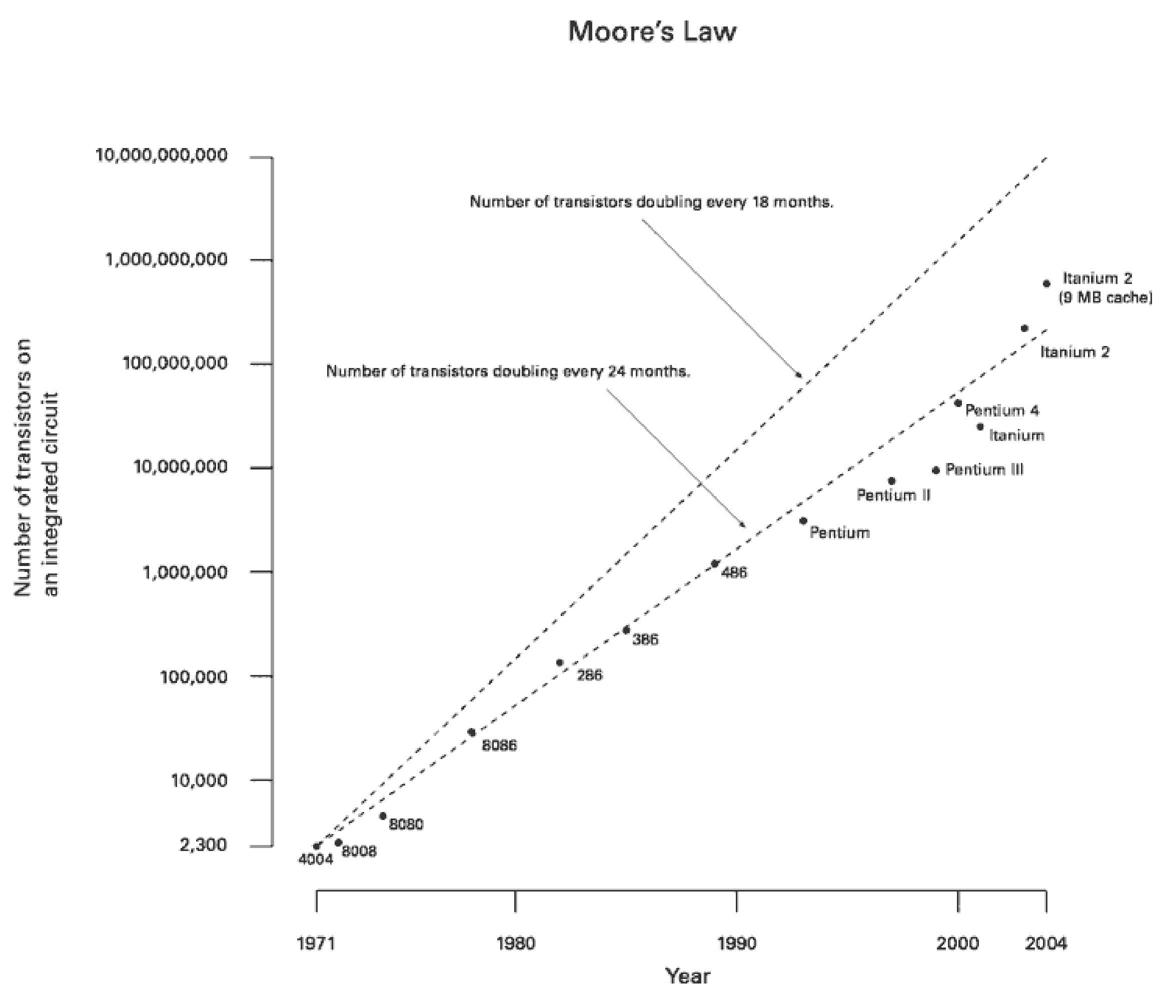
- Result

```
Problems  @ Javadoc  Declaration  Console
<terminated> AlgorithmABC [Java Application] C:₩Program Files₩Java₩
50005000 : 0
50005000 : 95
50005000 : 0
```

Factors

- Factors that determine running time of a program
  - problem size
  - basic algorithm / actual processing
  - memory access speed
  - CPU/processor speed
  - the of processors?
  - compiler/linker optimization?

Moore's law


Moore's Law
10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
Number of transistors on an integrated circuit
Number of transistors doubling every 18 months.
Number of transistors doubling every 24 months.
Itanium 2 (9 MB cache)
Itanium 2
Pentium 4
Itanium
Pentium III
Pentium II
Pentium
486
386
286
8086
8080
8008
4004
1971
1980
1990
2000
2004
Year

Data Structures . Performance

Measuring an Algorithm Efficiency

Terminology

- Analysis of algorithms
  - the process of measuring the complexity of algorithms
- Problem size
  - the number of items that an algorithm processes
- Basic operation
  - the most significant contributor to its total time requirement
  - the most frequent operation is not necessarily the basic operation such as assignments, control loop
  - Simplified analysis can be based on : number of arithmetic operations performed, Number of comparisons made, Number of times through a critical loop, Number of array elements accessed, etc
- directly proportional
  - the time requirement increases by some factor
- growth-rate function: T(n)
  - how an algorithm's
  - The number of basic operations for n

Constant time

- T(n)=3

```
// Algorithm C
t0 = System.currentTimeMillis();
sum = n*(n+1)/2;
t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

Linear time

```
long n = 10000;

// Algorithm A
long t0 = System.currentTimeMillis();
long sum = 0;
for(long i = 1; i <= n ; i++)
    sum = sum+i;
long t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

- The number of basic operations

n	0	1	2	3	4	5
T(n)						

- T(n) = n

Quadratic time

```
//Algorithm B
t0 = System.currentTimeMillis();
sum = 0;
for(long i = 1; i <= n ; i++)
    for(long j = 1; j<=i; j++)
        sum = sum+1;
t1 = System.currentTimeMillis();
System.out.println(sum+" : "+(t1-t0));
```

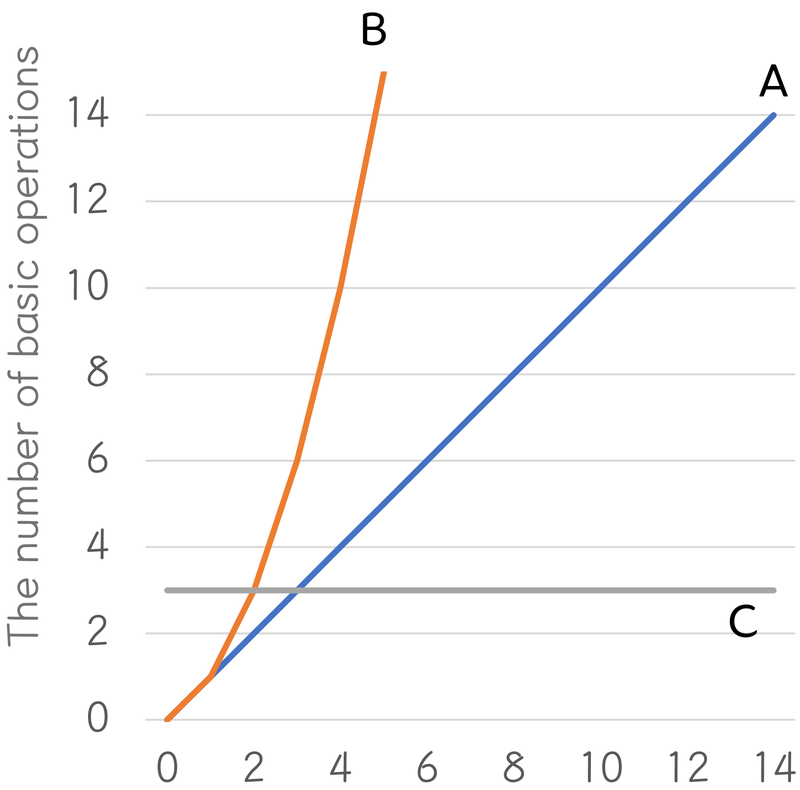
- The number of basic operations

n	0	1	2	3	4	5
i						
j						
T(n)						

- $T(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

Necessary time

	A	B	C
Additions	n	n(n+1)/2	1
Multiplications			1
Divisions			1
Total	n	$\frac{n^2}{2}+\frac{n}{2}$	3



Growth-rate function

- Dominant term: the term the one dominating as n gets bigger

Polynomial-time algorithm

f(n)	$n=10^3$	$n=10^5$	$n=10^6$
$\log_2(n)$	10^{-5} sec	$1.7 * 10^{-5}$ sec	$2 * 10^{-5}$ sec
n	10^{-3} sec	0.1 sec	1 sec
$n*\log_2(n)$	0.01 sec	1.7 sec	20 sec
n^2	1 sec	3 hr	12 days
n^3	17 min	32 yr	317 centuries
2^n	10^{285} centuries	10^{10000} years	10^{100000} years

Logarithmic algorithm

Linear algorithm

Quadratic algorithm

Exponential algorithm

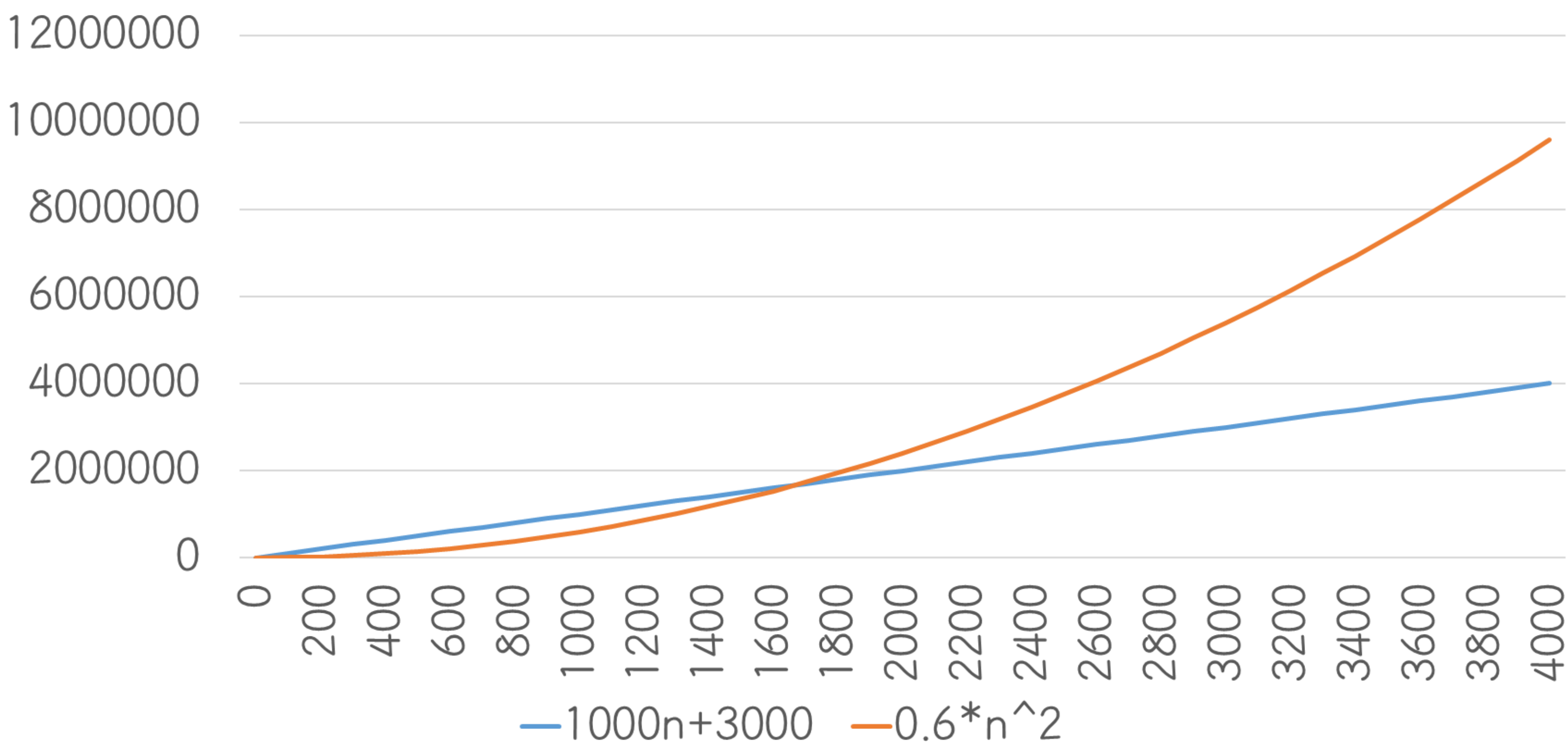
fast

slow

Data Structures . Performance

Asymptotic Notation

Asymptotic



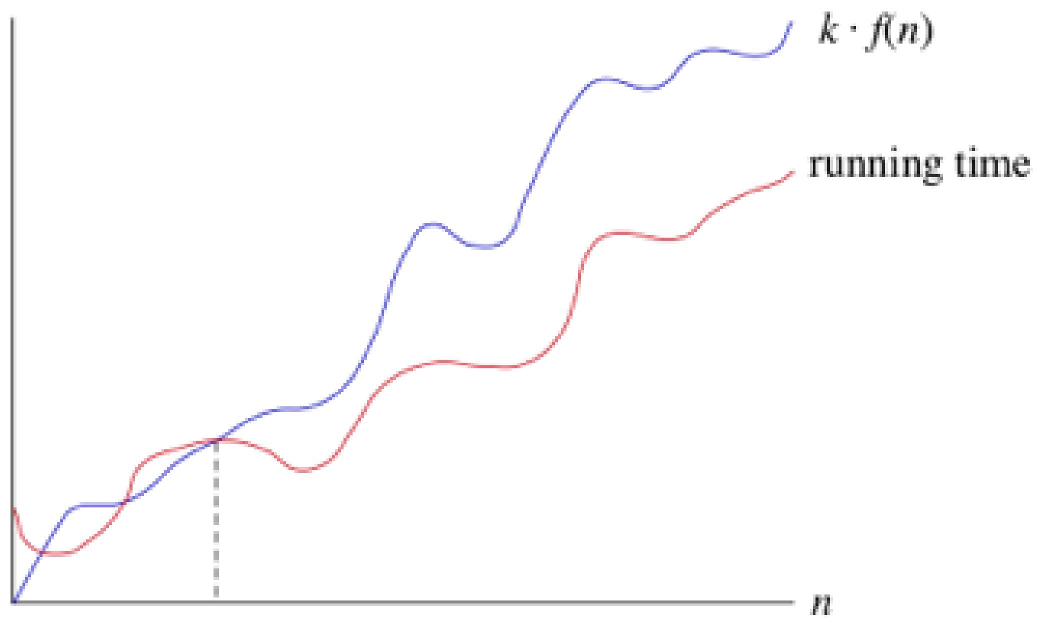
12000000
10000000
8000000
6000000
4000000
2000000
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
2400
2600
2800
3000
3200
3400
3600
3800
4000
1000n+3000
0.6*n^2

Asymptotic notations

- O
  - Big O notation
  - Asymptotic upper bound
- Ω
  - Big Omega
  - Asymptotic lower bound
- Θ
  - Big Theta notation
  - Asymptotic upper and the lower bound

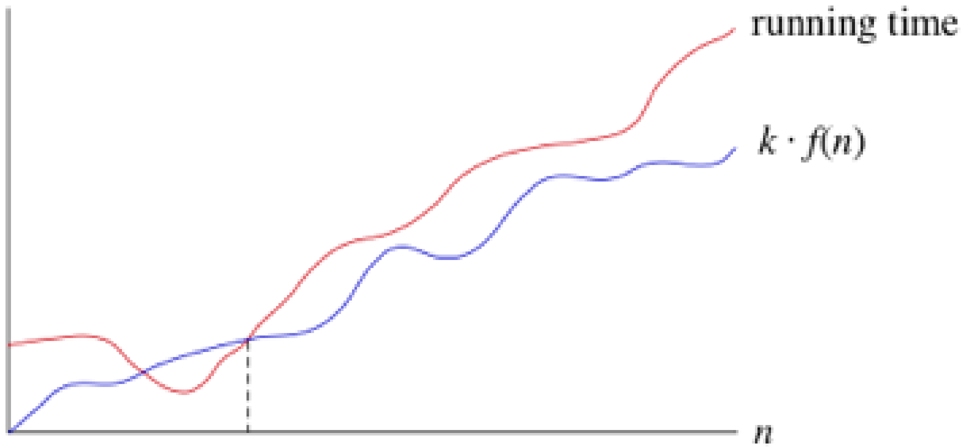
O-notation

- Upper bound of the running time of an algorithm
- $f(n) \in O(n^2)$
    - n^2
    - $3n^2+2n$
    - $3n^2+n \log n$
    - $n \log n$
    - $3n$



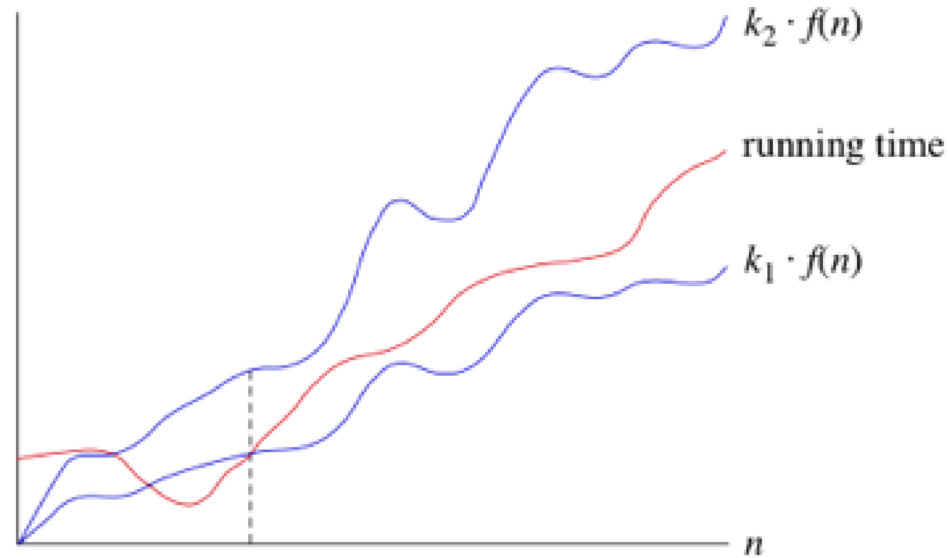
Ω-notation

- Upper bound of the running time of an algorithm
- $f(n) \in \Omega(n^2)$
  - n^2
  - $3n^2+2n$
  - $3n^2+n \log n$
  - $7n^3+5n$



Θ-notation

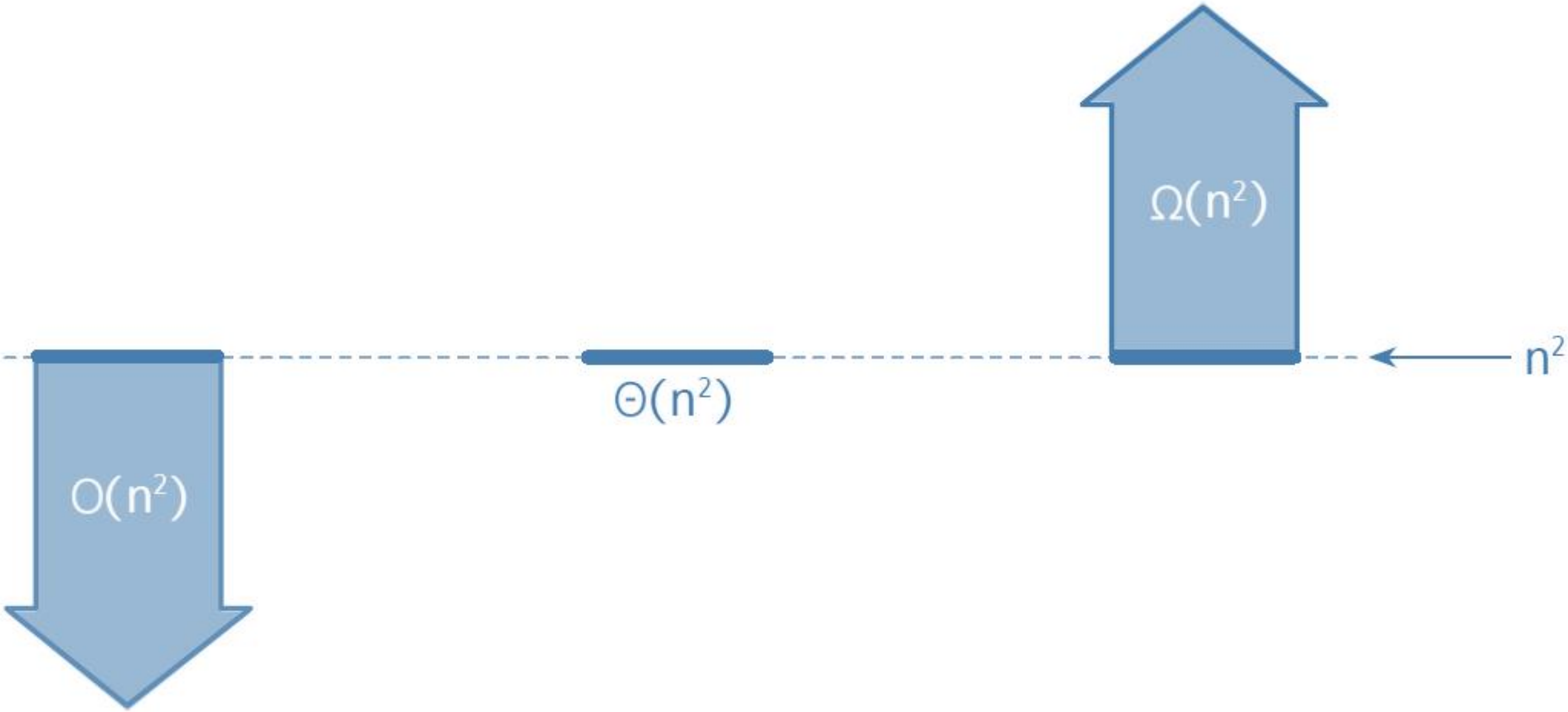
- Tight bound of the running time of an algorithm
- $\Theta(n^2) = O(n^2) \cap \Omega(n^2)$



$\in \rightarrow =$

- Proper notation
  - $T(n) \in O(n^2)$
- General notation
  - $T(n) = O(n^2)$
- Wrong notation
  - $O(n^2) = T(n)$

Relation of notations



Thank you!

Questions?

Exit