CSE102
Algorithmic Foundations And Problem Solving

Algorithm efficiency: asymptotic analysis

Dr Yushi Li

Department of Computer Science

西交利物浦大学
Xi'an Jiaotong-Liverpool University

Learning outcomes

- Understand the notations of asymptotic analysis.
- Able to carry out simple **asymptotic analysis** of algorithms

Time Complexity Analysis

How fast is the algorithm?

Code the algorithm and run the program, then measure the running time

1. Depend on the speed of the computer
2. Waste time coding and testing if the algorithm is slow

Identify some important operations/steps and count how many times these operations/steps needed to executed

Time Complexity Analysis

How to measure efficiency?

Number of operations usually expressed in terms of input size

- If we doubled the input size, how much longer would the algorithm take?
  If we trebled the input size, how much longer would it take?

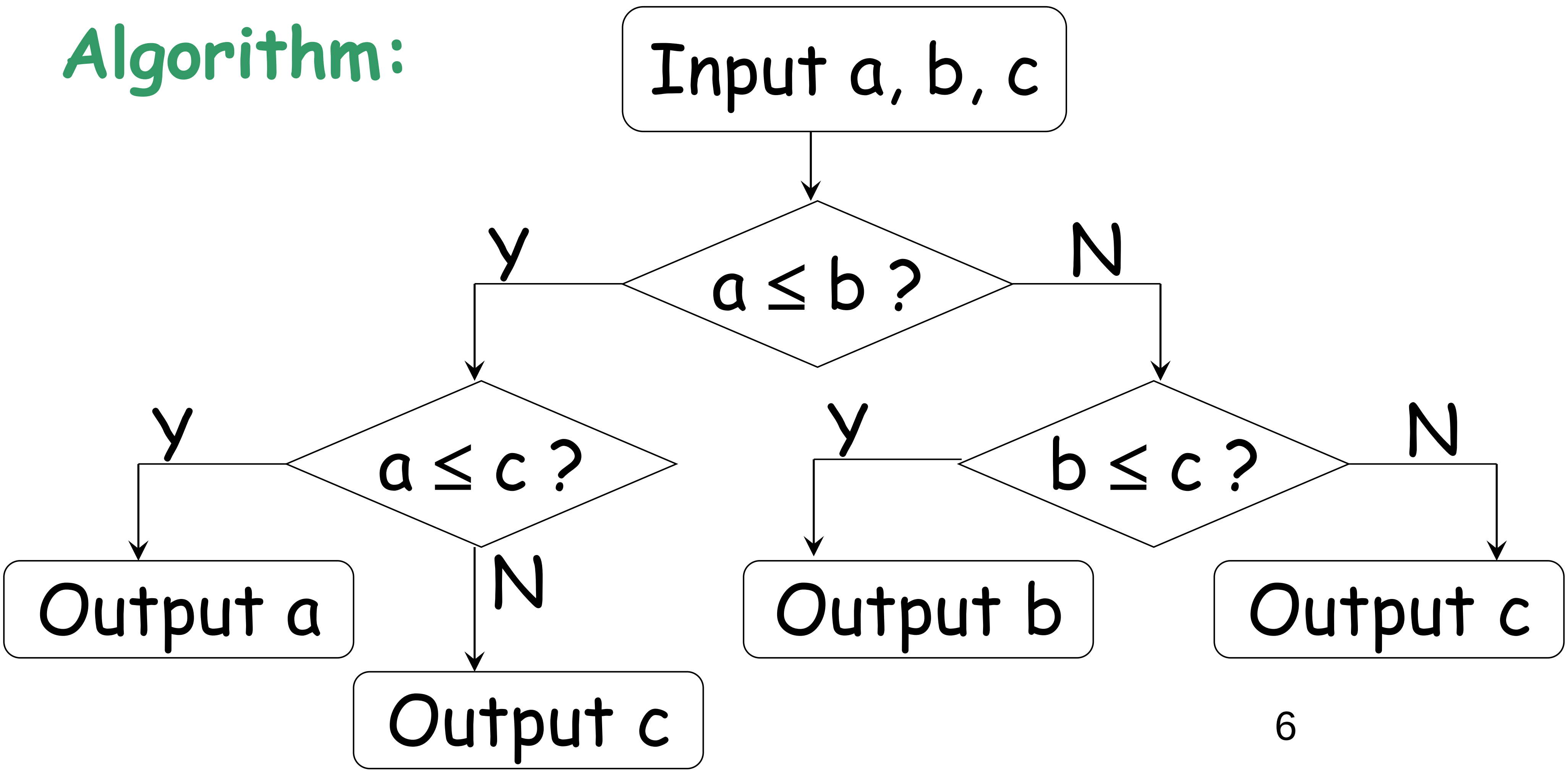
Finding the minimum…

Finding min among 3 numbers

Input: 3 numbers **a, b, c**

Output: the min value of these 3 numbers

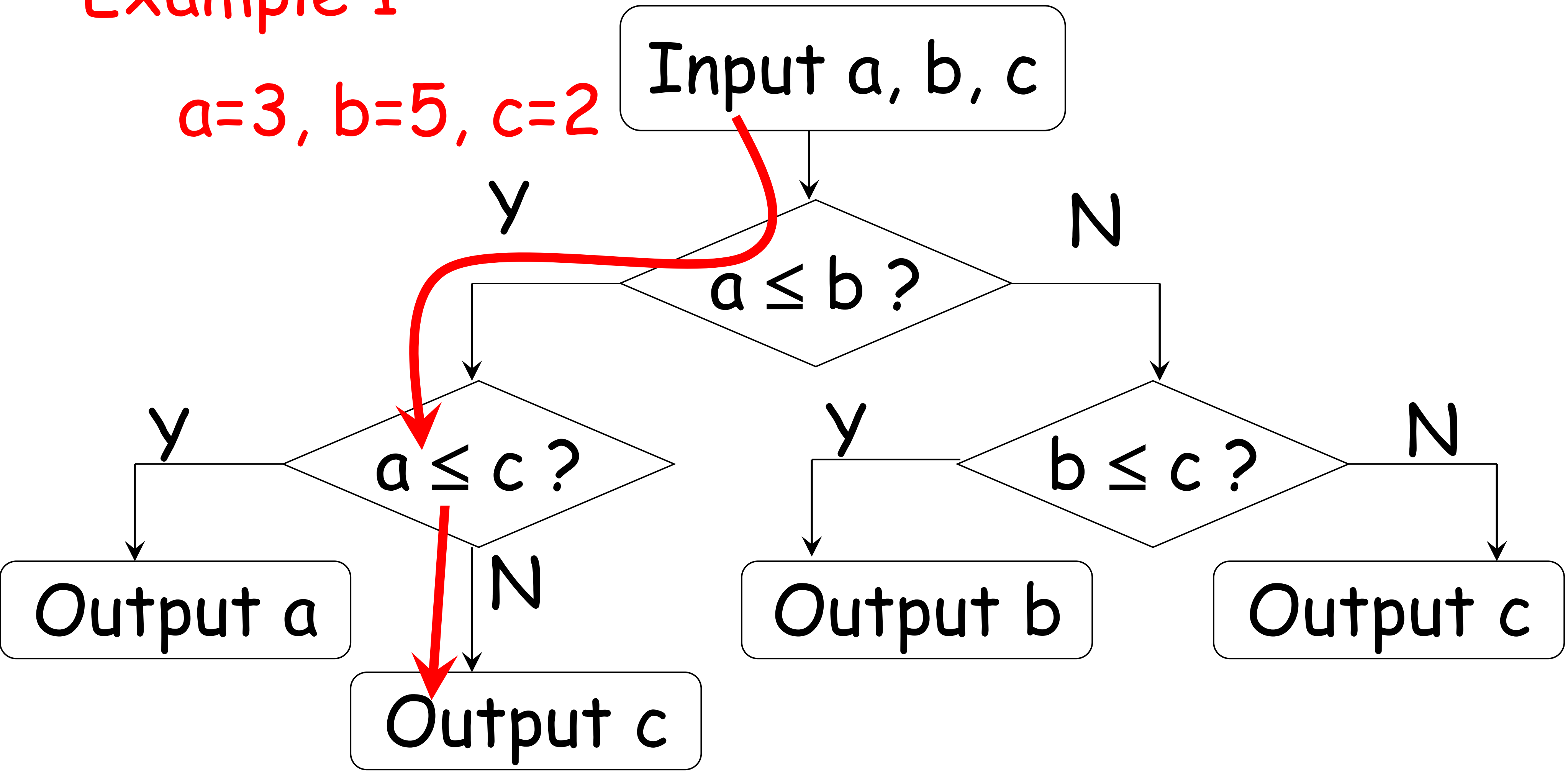
Algorithm:



Finding min among 3 numbers

Example 1

a=3, b=5, c=2

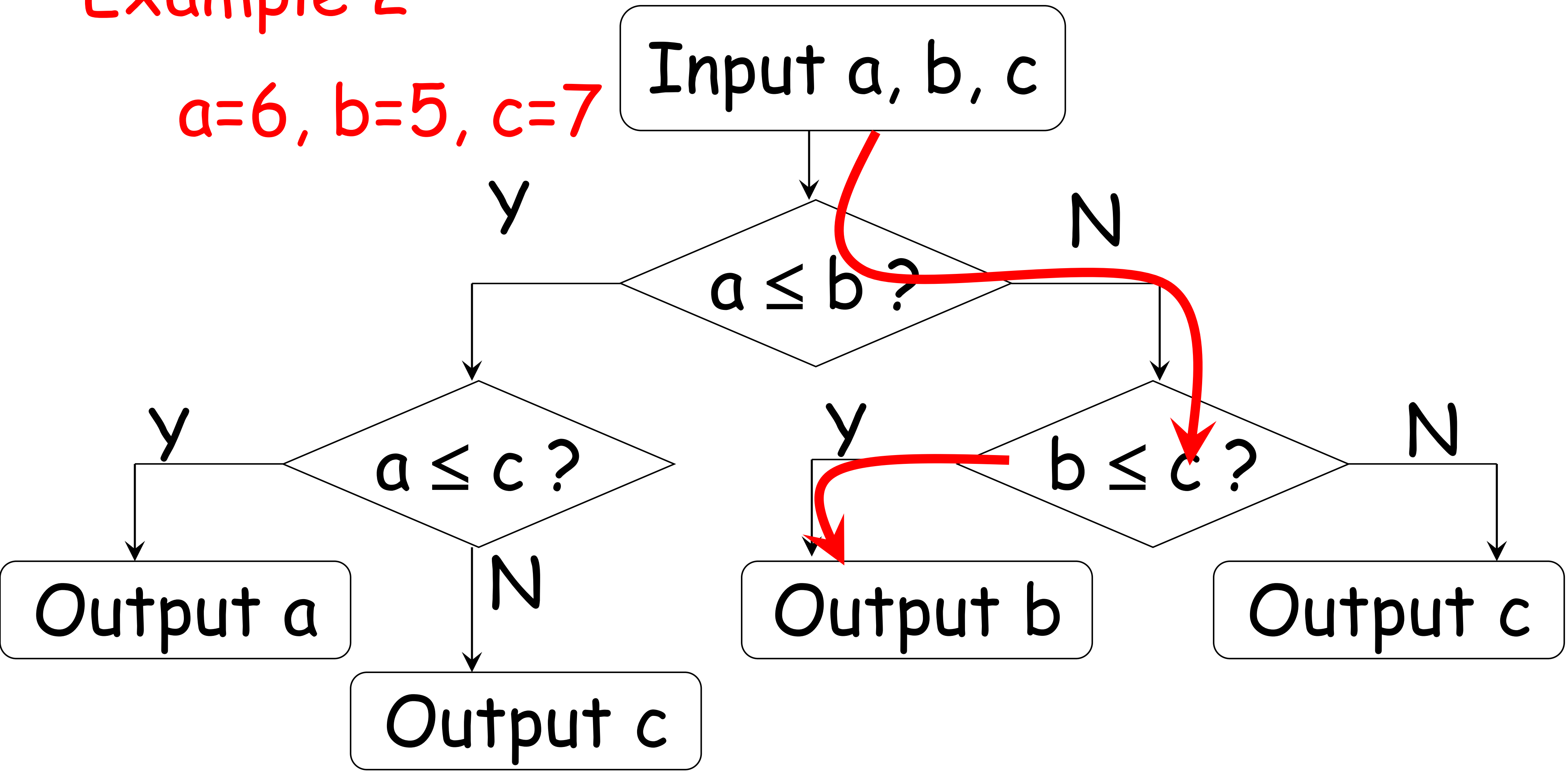


2 is output

Finding min among 3 numbers

Example 2

a=6, b=5, c=7



5 is output

Pseudo Code

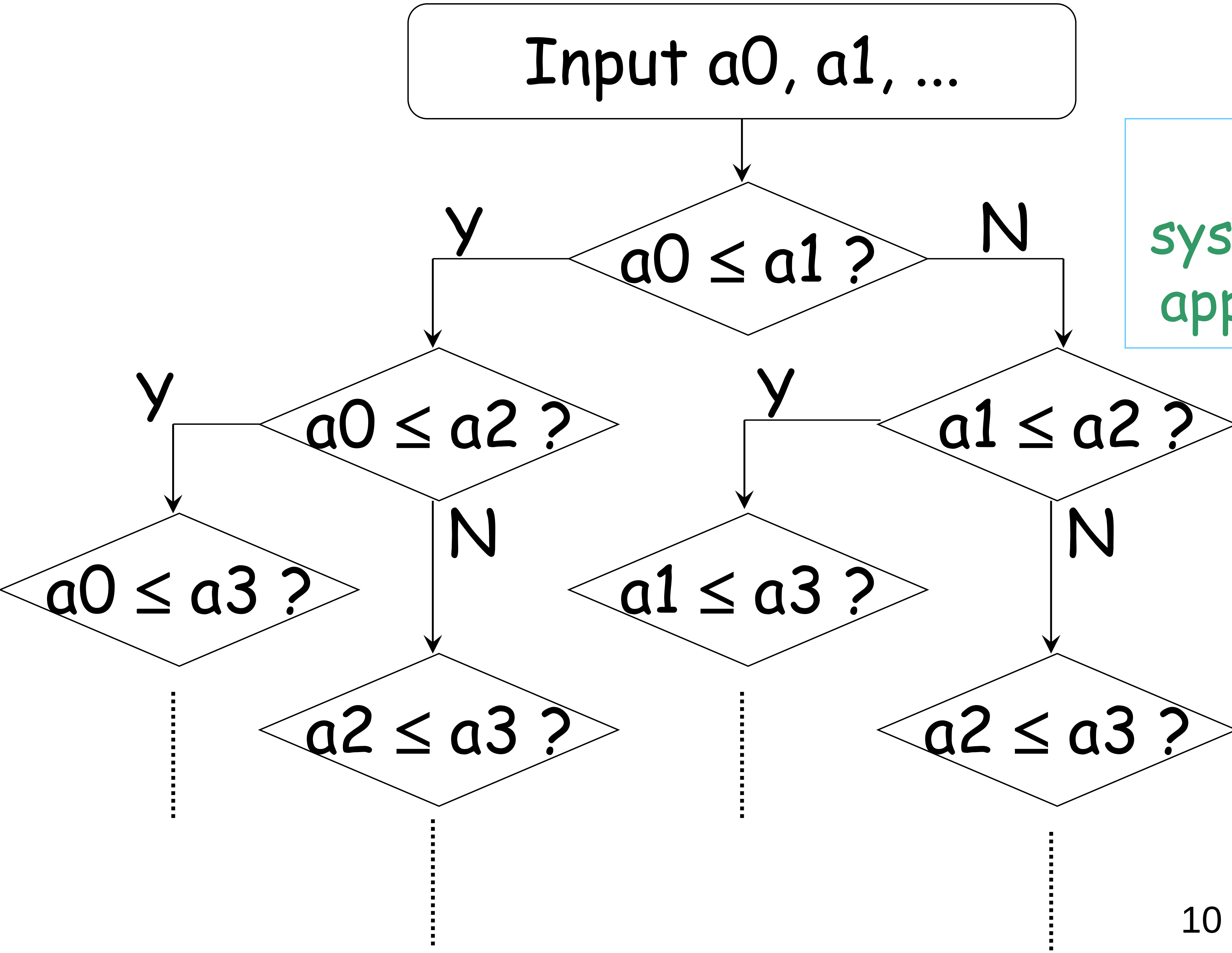
Important operation:
comparison

How many comparison this algorithm requires?

```
input a, b, c
if (a ≤ b) then
   if (a ≤ c) then
      output a
   else
      output c
else
   if (b ≤ c) then
      output b
   else
      output c
```

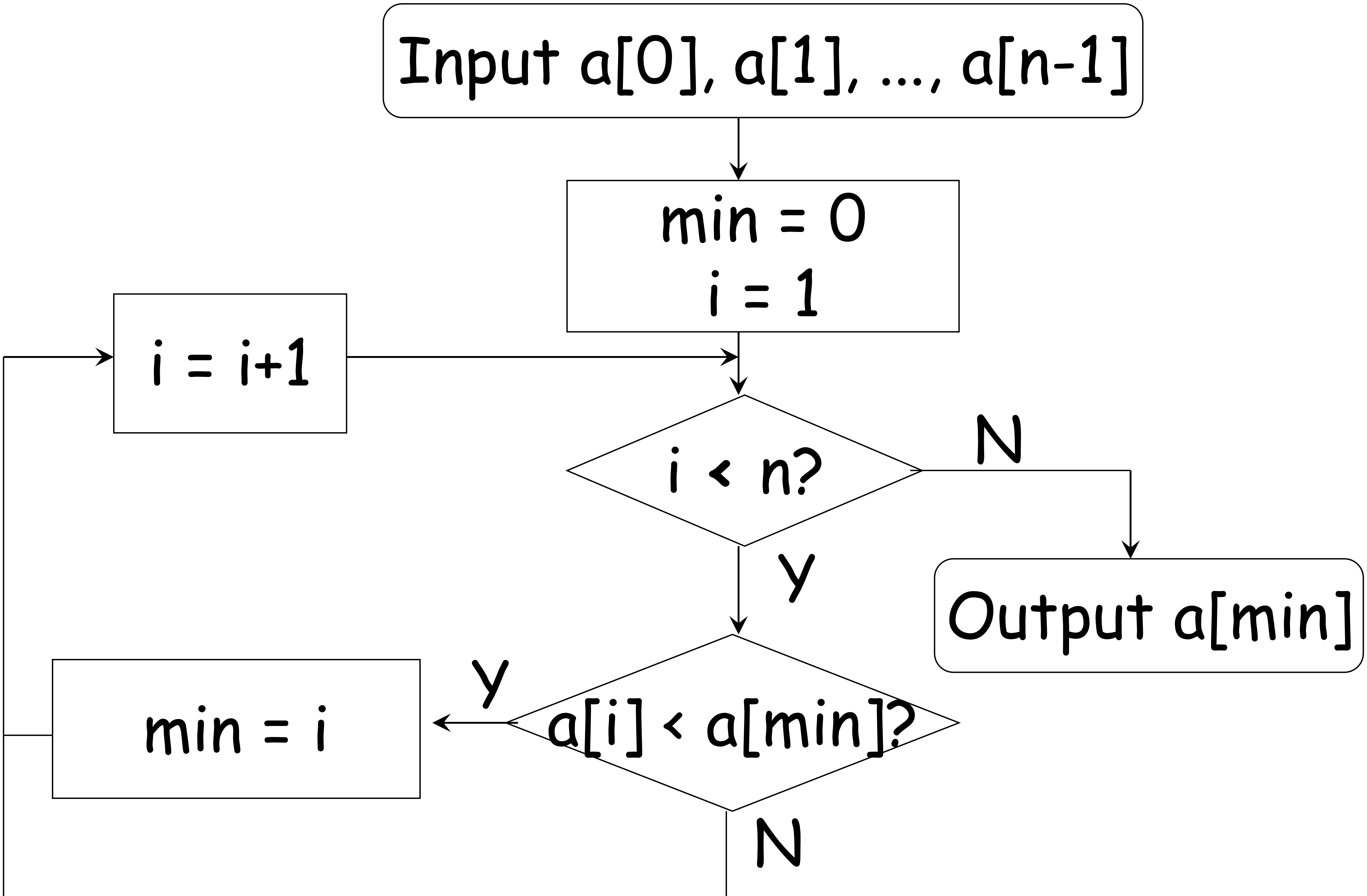
It takes 2 comparisons

Finding min among n numbers



Any systematic approach?

Finding min among n numbers



Finding min among n numbers

```
input a[0], a[1], ..., a[n-1]
min = 0
i = 1
while (i < n) do
begin
  if (a[i] < a[min]) then
    min = i
  i = i + 1
end
output a[min]
```

Example

a[]={50,30,40,20,10}

iteration	min	a[min]
before	0	50
1	1	30
2	1	30
3	3	20
4	4	10

Finding min among n numbers

- Time complexity: ?? comparisons

n-1

```
input a[0], a[1], ..., a[n-1]
min = 0
i = 1
while (i < n) do
begin
   if (a[i] < a[min]) then
      min = i
   i = i + 1
end
output a[min]
```

Finding min using for-loop

Rewrite the above while-loop into a for-loop

```
input a[0], a[1], ..., a[n-1]
min = 0
for i = 1 to n-1 do
begin
  if (a[i] < a[min]) then
    min = i
end
output a[min]
```

Why efficiency matters?

- speed of computation by hardware has been improved
- efficiency still matters
- ambition for computer applications grow with computer power
- demand a great increase in speed of computation

Amount of data handled matches speed increase?

When computation speed vastly increased, can we handle much more data?

Suppose

- an algorithm takes n^2 comparisons to sort n numbers
- we need *1* sec to sort ***5*** numbers (25 comparisons)
- computing speed *increases by factor of* ***100***

Using 1 sec, we can now perform ***100x25*** comparisons, i.e., to sort ***50*** numbers

With ***100*** times speedup, only sort ***10*** times more numbers!

Time complexity
- Big O notation …

Which algorithm is the fastest?

Consider a problem that can be solved by 5 algorithms A_1, A_2, A_3, A_4, A_5 using different number of operations (time complexity).

$f_1(n) = 50n + 20$

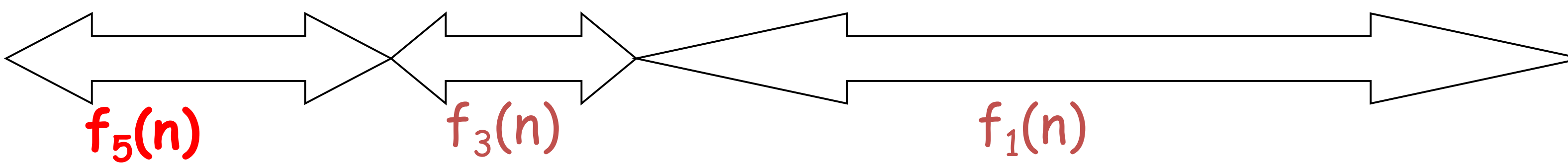
$f_2(n) = 10\ n \log_2 n + 100$

$f_3(n) = n^2 - 3n + 6$

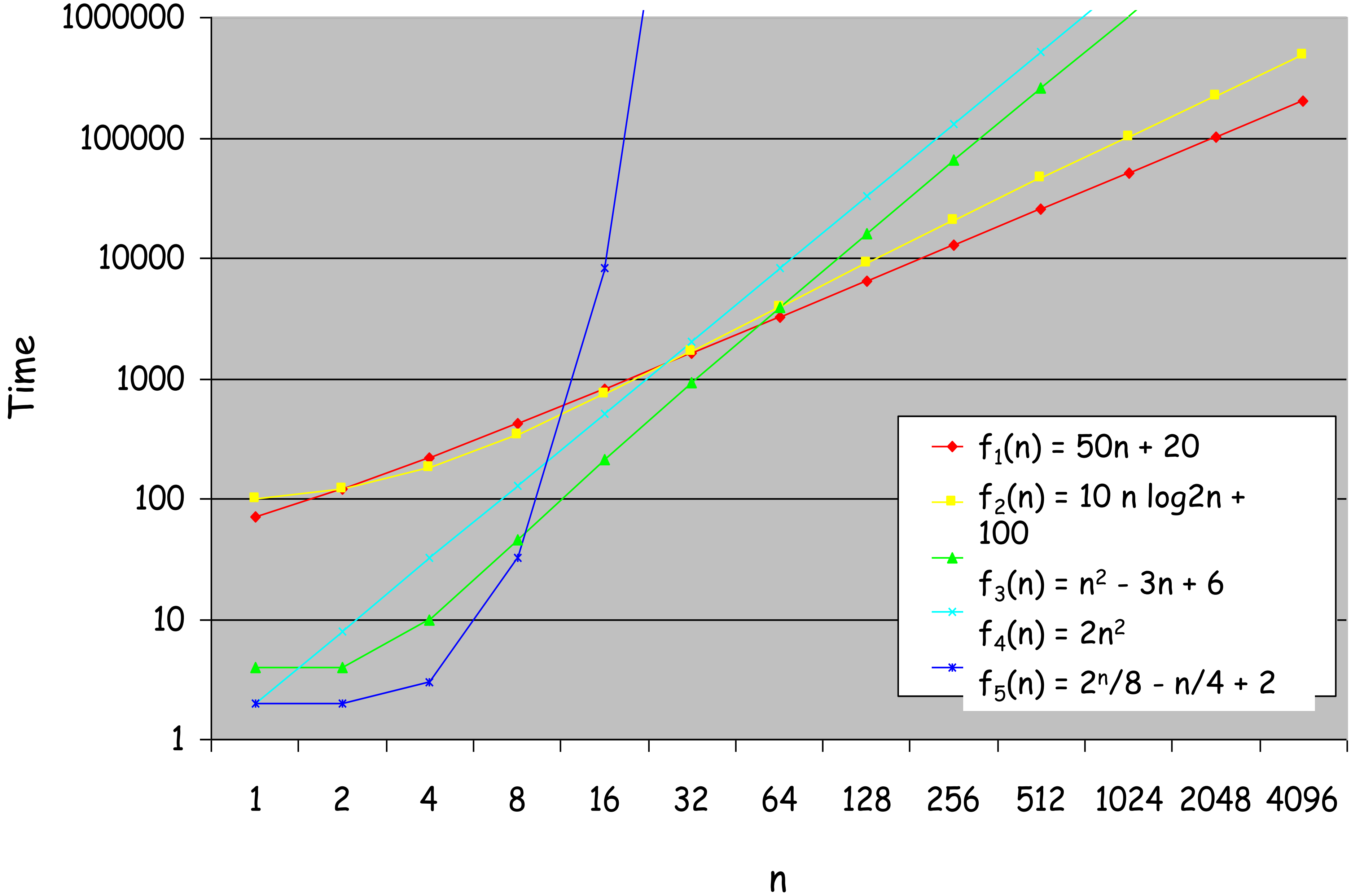
$f_4(n) = 2n^2$

$f_5(n) = 2^n/8 - n/4 + 2$

n	1	2	4	8	16	32	64	128	256	512	1024	2048
$f_1(n) = 50n + 20$	70	120	220	420	820	1620	3220	6420	12820	25620	51220	102420
$f_2(n) = 10\ n\ \log2n + 100$	100	120	180	340	740	1700	3940	9060	20580	46180	102500	225380
$f_3(n) = n^2 - 3n + 6$	4	4	10	46	214	934	3910	16006	64774	3E+05	1E+06	4E+06
$f_4(n) = 2n^2$	2	8	32	128	512	2048	8192	32768	131072	5E+05	2E+06	8E+06
$f_5(n) = 2^n/8 - n/4 + 2$	2	2	3	32	8190	5E+08	2E+18					



Depends on the size of the problem!


Time
1000000
100000
10000
1000
100
10
1
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
n
f1(n) = 50n + 20
f2(n) = 10 n log2n + 100
f3(n) = n2 - 3n + 6
f4(n) = 2n2
f5(n) = 2n/8 - n/4 + 2

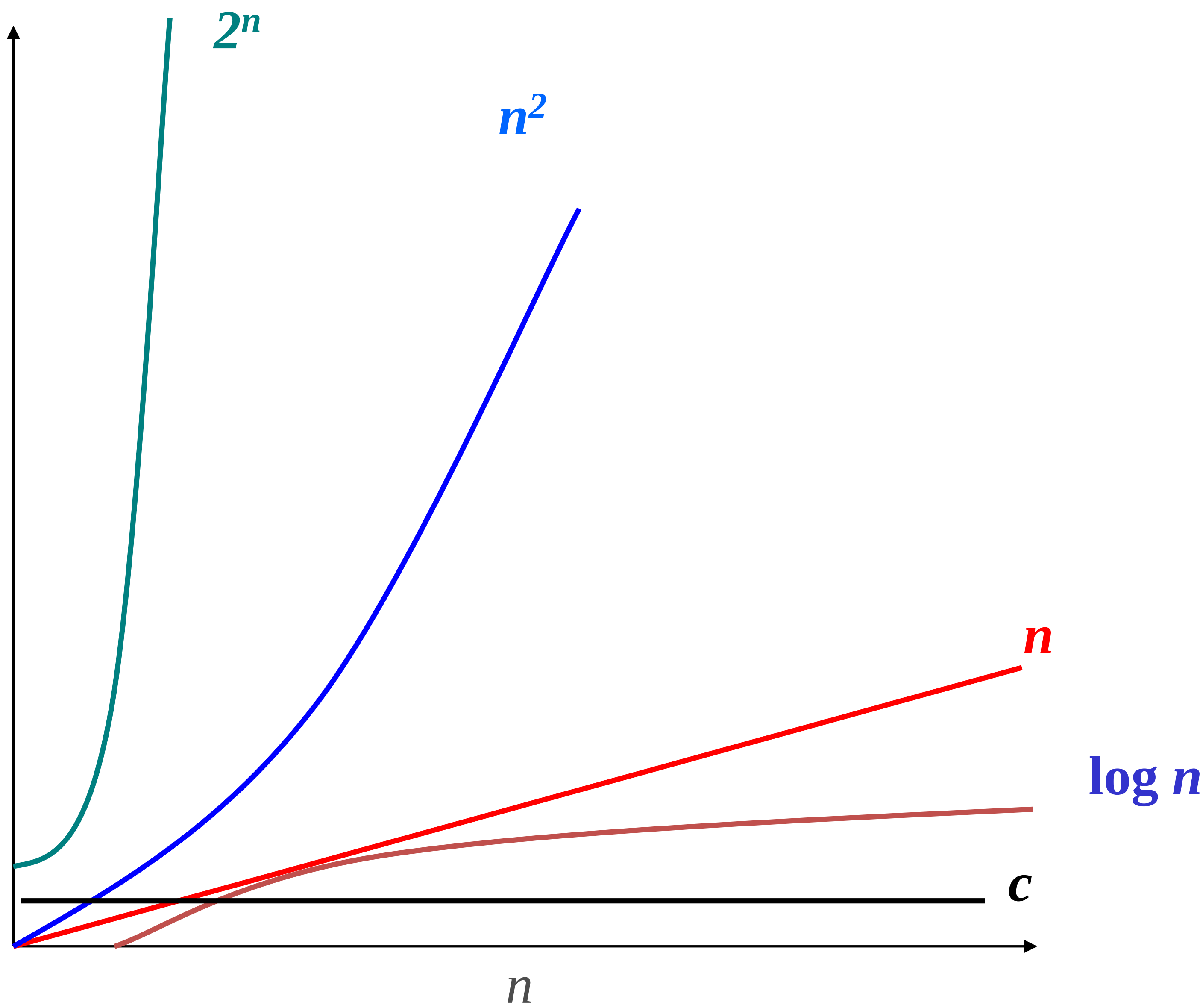
What do we observe?

- There is huge difference between
  - functions involving powers ***of*** n (e.g., n, n log n, n^2, called **polynomial** functions) and
  - functions involving powering ***by*** n (e.g., 2^n, called **exponential** functions)
- Among polynomial functions, those with same order of power are more comparable
  - e.g., $f_3(n) = n^2 - 3n + 6$ and $f_4(n) = 2n^2$

Growth of functions

n	$\log n$	$\sqrt{n}$	n	$n \log n$	n^2	n^3	2^n
2	1	1.4	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2.8	8	24	64	512	256
16	4	4	16	64	256	4096	65536
32	5	5.7	32	160	1024	32768	4294967296
64	6	8	64	384	4096	262144	1.84×10^{19}
128	7	11.3	128	896	16384	2097152	3.40×10^{38}
256	8	16	256	2048	65536	16777216	1.16×10^{77}
512	9	22.6	512	4608	262144	134217728	1.34×10^{154}
1024	10	32	1024	10240	1048576	1073741824	

Relative growth rate



Hierarchy of functions

- We can define a hierarchy of functions each having a **greater** order of magnitude than its predecessor:

1	$\log n$	$\underbrace{n \;\; n^2 \;\; n^3 \; \ldots \; n^k \; \ldots}$	2^n
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
constant	logarithmic	polynomial	exponential

- We can further refine the hierarchy by inserting $\mathbf{n \log n}$ between $\mathbf{n}$ and $\mathbf{n^2}$, $\mathbf{n^2 \log n}$ between $\mathbf{n^2}$ and $\mathbf{n^3}$, and so on.

Hierarchy of functions (2)

1	$\log n$	$\underbrace{n \quad n^2 \ n^3 \ \dots \ n^k \dots}$	2^n
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$
constant	logarithmic	polynomial	exponential

Important: as we move from left to right, successive functions have **greater order of magnitude** than the previous ones.

As **n** increases, the values of the later functions increase **more rapidly** than the values of the earlier ones.

$\Rightarrow$ Relative growth rates increase

Hierarchy of functions (3)

1	$\log n$	$\underbrace{n \;\; n^2 \;\; n^3 \;\ldots\; n^k \ldots}$	2^n
constant	logarithmic	polynomial	exponential

- Now, when we have a function, we can assign the function to some function in the hierarchy:
  - For example, $f(n) = 2n^3 + 5n^2 + 4n + 7$

    The term with the highest power is $2n^3$.

    The growth rate of $f(n)$ is dominated by n^3.
- This concept is captured by **Big-O notation**

Big-O notation

$f(n) = O(g(n))$ *[read as f(n) is of order g(n)]*

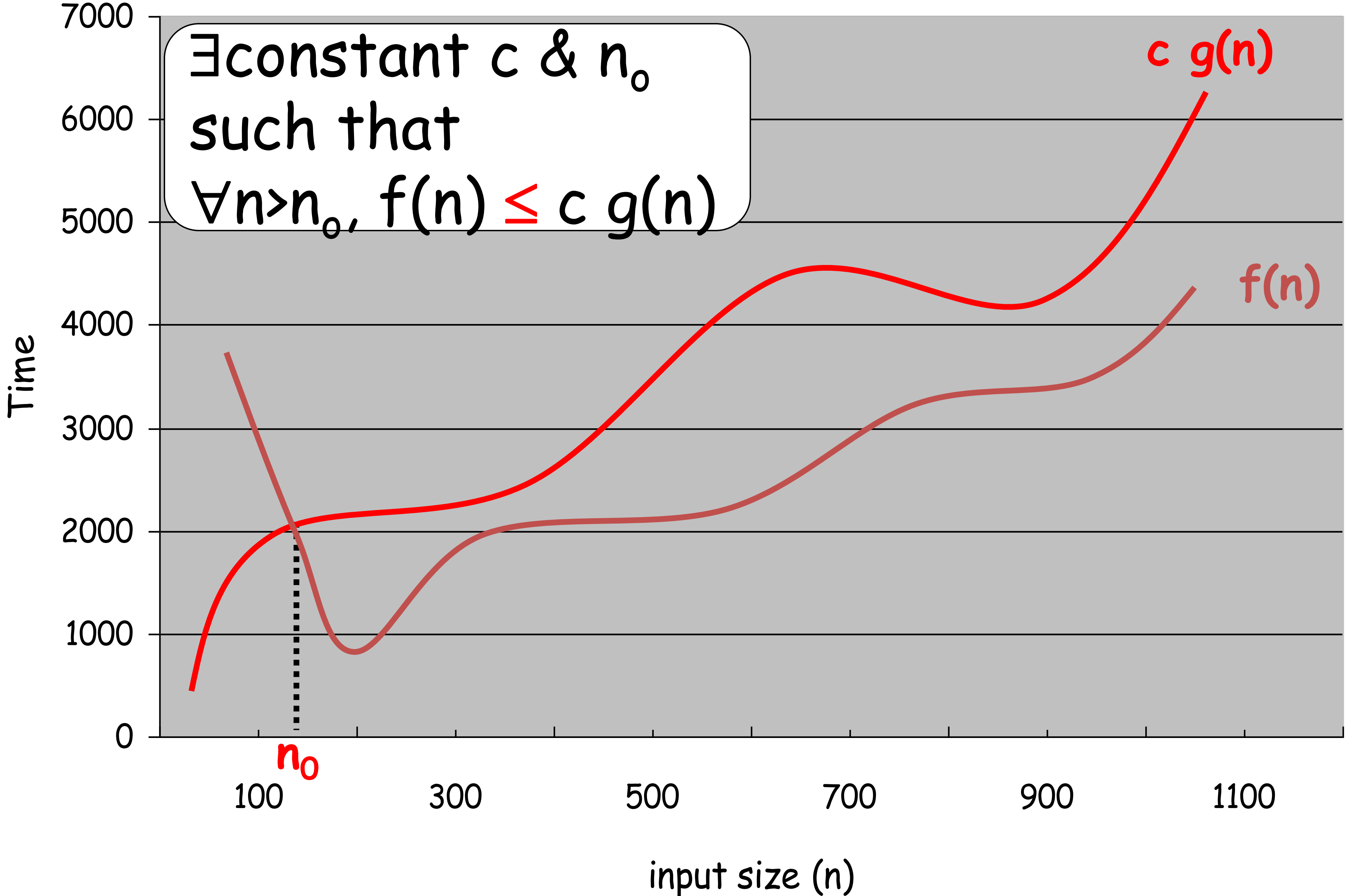
- Roughly speaking, this means f(n) is at most **a constant times g(n)** for all large n
- Examples
  - $2n^3 = O(n^3)$
  - $3n^2 = O(n^2)$
  - $2n \log n = O(n \log n)$
  - $n^3 + n^2 = O(n^3)$
- function on L.H.S and function on R.H.S are said to have the **same order of magnitude**

Big-O notation - formal definition

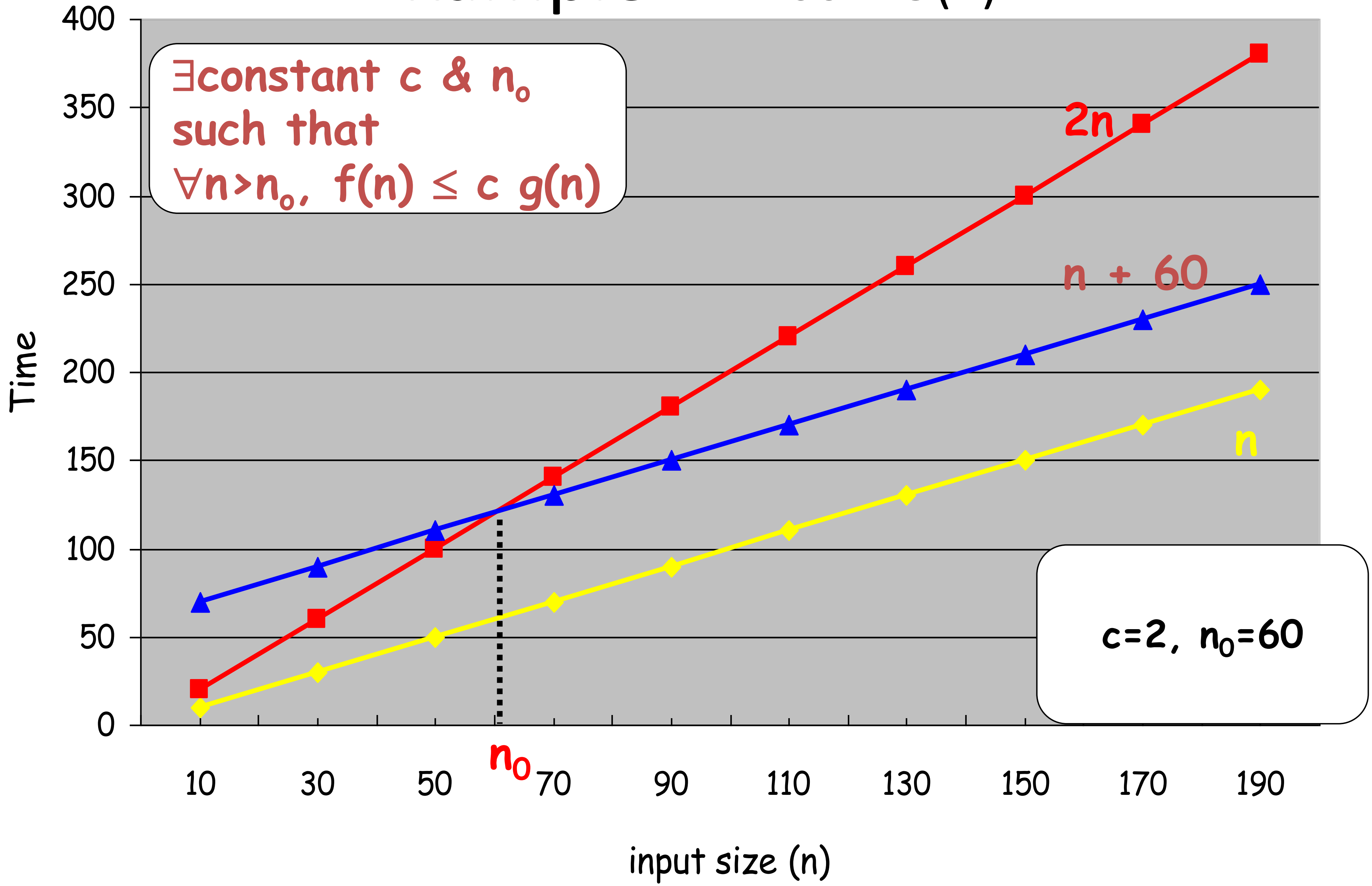
$f(n) = O(g(n))$: There exists a constant **c** and **n_o**

such that **$f(n)$** $\leq$ **c** **$g(n)$** for all $n > n_o$

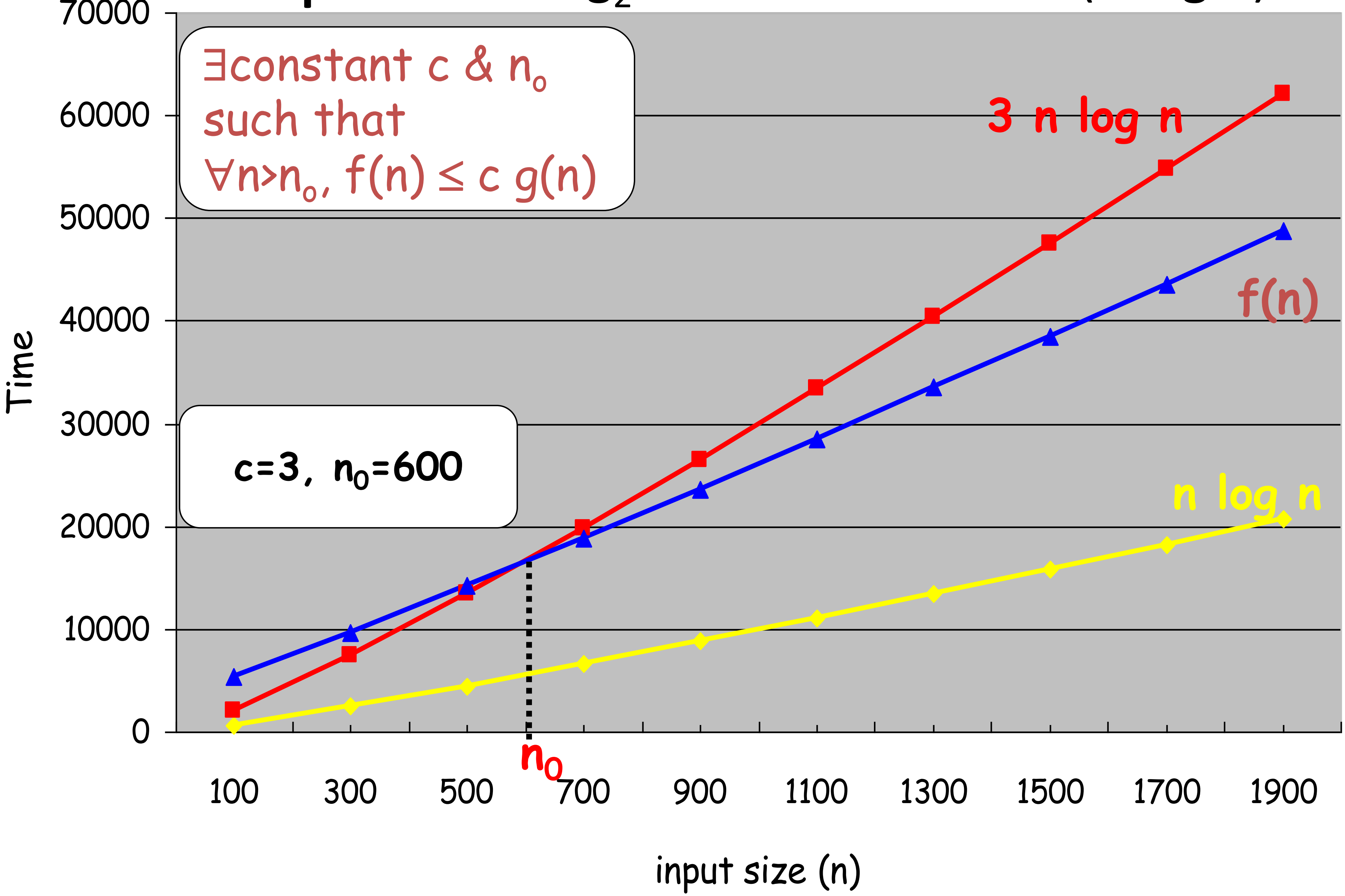
f(n) = O(g(n) – Graphical illustration



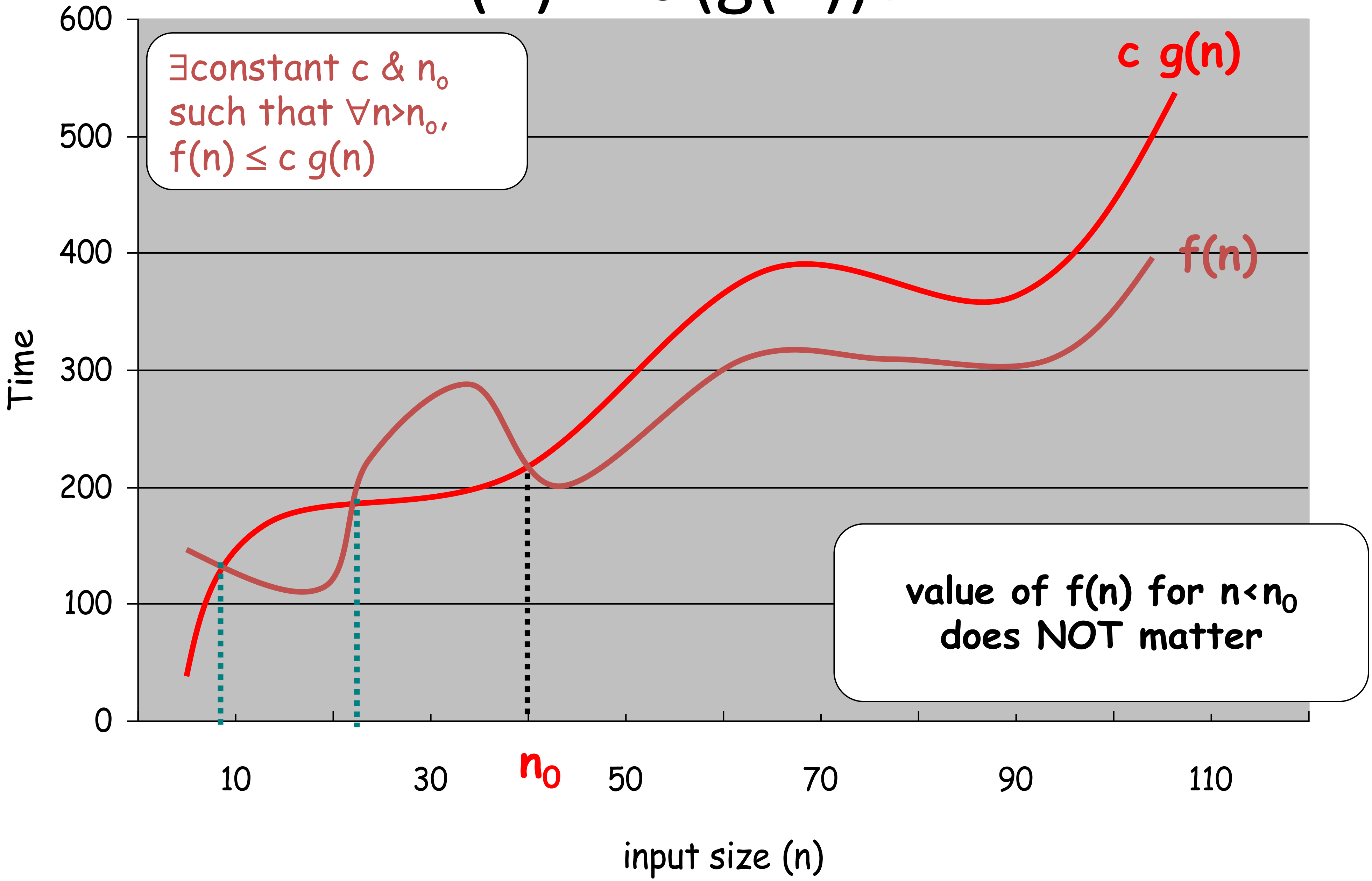
Example – $n+60 = O(n)$



Example – $2n\log_2 n + 100\overline{\sqrt{n}} + 1000 = O(n \log n)$



f(n) = O(g(n))?


600
500
400
300
200
100
0
Time
∃constant c & n_o
such that ∀n>n_o,
f(n) ≤ c g(n)
c g(n)
f(n)
value of f(n) for n<n_0
does NOT matter
10
30
n_0
50
70
90
110
input size (n)

Which one is the fastest?

- Usually we are only interested in the ***asymptotic*** time complexity, i.e., when n is large

$O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(2^n)$

Proof of order of magnitude

- Show that $2n^2 + 4n$ is $O(n^2)$
  - ✓ Since $n \le n^2$ for all $n>1$,
    we have $2n^2 + 4n \le 2n^2 + 4n^2 \le 6n^2$ for all $n>1$.
  - ✓ Therefore, by definition, $2n^2 + 4n$ is $O(n^2)$.

- Show that $n^3 + n^2 \log n + n$ is $O(n^3)$
  - ✓ Since $n \le n^3$ and $\log n \le n$ for all $n>1$,
    we have $n^2 \log n \le n^3$, and
    $n^3 + n^2 \log n + n \le 3n^3$ for all $n>1$.
  - ✓ Therefore, by definition, $n^3 + n^2 \log n + n$ is $O(n^3)$.

Exercise

Determine the order of magnitude of the following functions.

1. $n^3 + 3n^2 + 3$
2. $4n^2 \log n + n^3 + 5n^2 + n$
3. $2n^2 + n^2 \log n$
4. $6n^2 + 2^n$

Exercise (2)

Prove the order of magnitude

1. $n^3 + 3n^2 + 3$

2. $4n^2 \log n + n^3 + 5n^2 + n$

Exercise (2) cont'd

Prove the order of magnitude

3. $2n^2 + n^2 \log n$

4. $6n^2 + 2^n$

More Exercise

Are the followings correct?

1. $n^2 \log n + n^3 + 3n^2 + 3$ $O(n^2)$?
2. $n + 1000$ $O(n)$?
3. $6n^{20} + 2^n$ $O(n^{20})$?
4. $n^3 + 5n^2 \log n + n$ $O(n^2 \log n)$?

Some algorithms we learnt

Computing sum of the first n numbers

```
input n
sum = n*(n+1)/2
output sum
```

O(?)

```
input n
sum = 0
for i = 1 to n do
begin
   sum = sum + i
end
output sum
```

O(?)

Finding the min value among n numbers

```
min = 0
for i = 1 to n-1 do
   if (a[i] < a[min]) then
      min = i
output a[min]
```

O(?)

More exercise

What is the time complexity of the following pseudo code?

```
for i = 1 to 2n do
   for j = 1 to n do
      x = x + 1
```

$O(?)$