

# Worst & Best case analysis

For some algos, running time is dependent on both input data size & nature of input data

There may be different running times for different inputs. Can have  $T(1)$  for some case,  $T(N)$  for another,  $T(N^2)$  for another.

In the following example the running time depends on the input size only

## FIND MAXIMUM

```
function max(A)
  max=0  ---  $C_0$ 
  for 1 <= i < N ---  $C_1N + C_2$ 
    if (A[i]>A[max]) ---  $C_3N + C_4$ 
      max=i ---  $C_5$ 
  return max ---  $C_6$ 
```

This algo has to check every element of the array before it can return the max

$T(N) = C_0N + C_1$ , Running time function

No matter how many elements in the array, the algo must check every one. There is only 1 case to analyse.

## LINEAR SEARCH

```
function L_Search(A,x)
```

```
  for 0 ≤ i < N
```

```
    if(A[i]==x)
```

```
      return i
```

```
  return -1
```

$C_1 N + C_2$   
 $C_3 N$   
 $C_4$  } only 1 is  
 $C_5$  } executed

Checks every item in an array (A) in sequence and returns the position as soon as x is found; or -1 if x is never found

There are multiple cases to analyse. To analyse the different cases, we can try different inputs.

arr =

13	8	2	24	5	17	6	9
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

$L\_search(arr, 7)$

This is the worst case scenario where the item we're searching for isn't in the array

## LINEAR SEARCH

```
function L_Search(A,x)
```

```
  for 0 ≤ i < N
```

```
    if(A[i]==x)
```

```
      return i
```

```
  return -1
```

The for loop is executed N times, and it will return -1.

The growth of the running time is a linear function in the worst case.

$$T_w(N) \propto N$$

# LINEAR SEARCH

```
function L_Search(A, x)
  for 0 ≤ i < N
    if(A[i]==x)
      return i
  return -1
```

$$T_b(N) \propto 1$$

arr =

13	8	2	24	5	17	6	9
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

L\_search(arr, 13)

This is the best case scenario, where the first item is the one we're searching for

$T(N) = C$  This case has a constant running time  
The loop only executes once.

When the item we're searching for appears as the first element in the array, the growth of the running time doesn't depend on the input data size. It will always be constant time.

The WORST case represents the maximum # of instruction that the algo must execute.

The BEST case represents the MINIMUM # of instruction that the algo must execute.

# Asymptotic Analysis

- Is a different way of describing the time and/or memory requirements of an algo.
- It's the analysis of the growth of a function and the input size grows larger and larger.
- We're only concerned with the fastest growing term in the function, we can drop all lower ordered terms, because they're not significant for large inputs

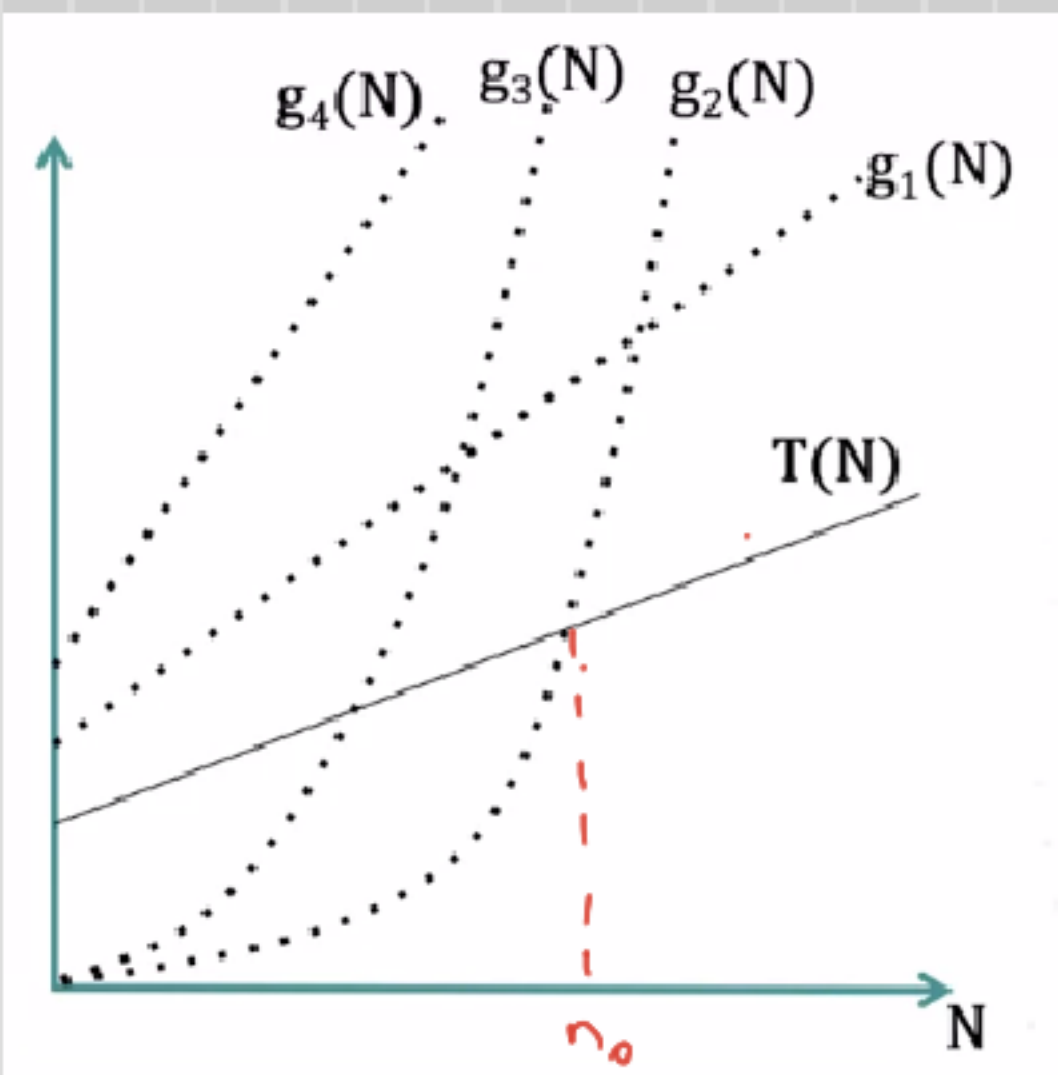


# Big-O notation

$$T(N) \in \mathcal{O}(g(N)) \rightarrow C \cdot g(N) \geq T(N) \forall N \geq n_0$$

$$C > 0, n_0 > 0$$

Big-O represents a set of functions that act as an upper bound for  $T(N)$ , which is a running time function.



$T(N)$  is  $\mathcal{O}(g_1(N))$   
 $\mathcal{O}(g_2(N))$   
 $\mathcal{O}(g_3(N))$   
 $\mathcal{O}(g_4(N))$


1) Our aim is to find the function  $g(N)$ , which acts as an upper bound for  $T(N)$ .

2)  $g(n)$  can be any of the dotted functions. It doesn't matter that for some values of  $N$ , that  $g(N)$  is less than  $T(N)$ .

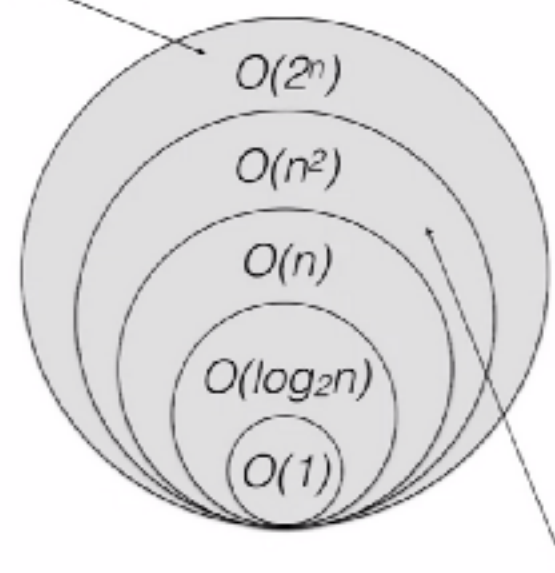
3) What's most important is that for large values of  $N$ ,  $g(n)$  is always greater than or equal to  $T(N)$ .

4) For any of these functions, we can say that  $T(N)$  is  $\mathcal{O}(g(n))$  because they all comply with being equal to or greater than  $T(N)$  from  $n_0$ .

$T(N)$  is  $N^2$   
 $\mathcal{O}(N^3), \mathcal{O}(N^4)$

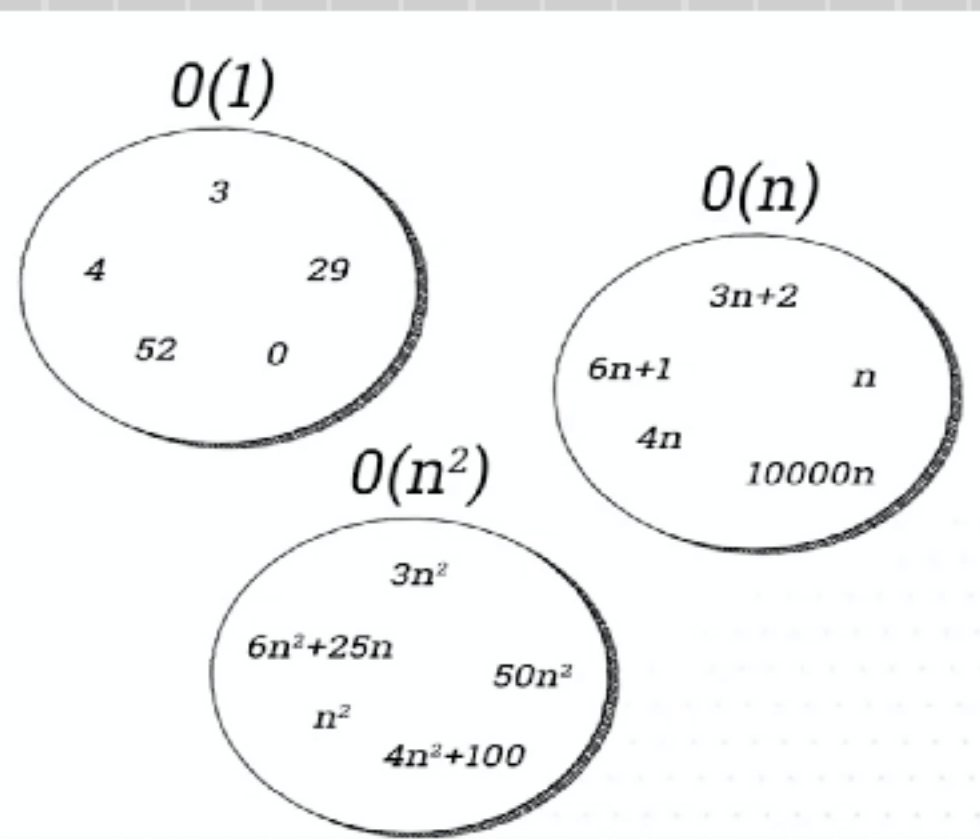
→  ← Big-O classes

$$f(n) = 2^n + 3n$$



$$g(n) = 1000n^2 + n$$

- 1) Big-O defines a set of functions that act as an upper bound.
- 2) For any given function, there's a set of functions that can be considered an upper bound for the growth of a function,  $T(N)$ .

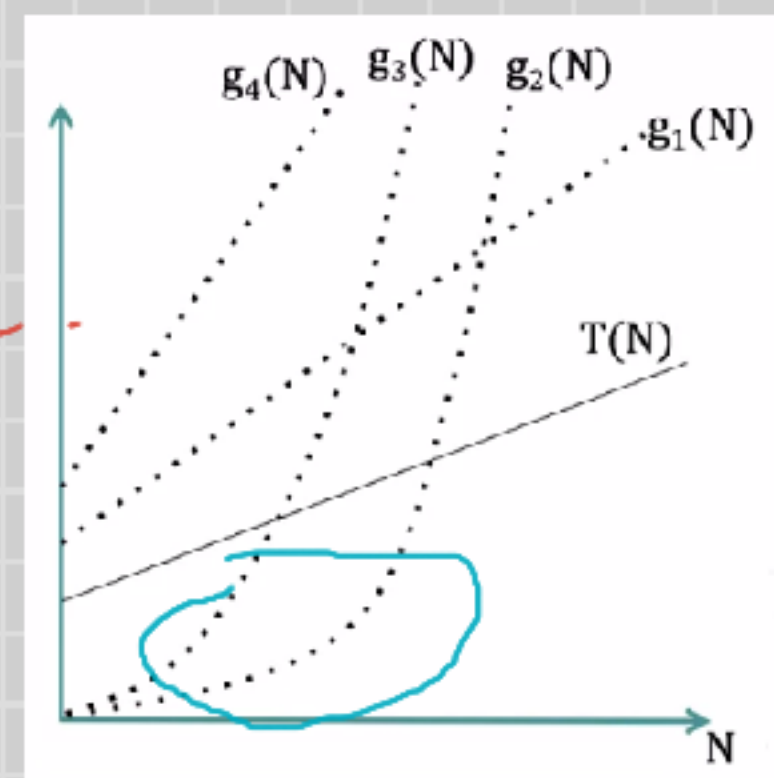


$$O(n^2) \subseteq O(n^3) \subseteq O(2^n) \dots$$

Formally defined as:  
 $T(N)$  is  $O(g(N))$  if:

$$\hookrightarrow c * g(N) \geq T(N), \text{ for all } N \geq n_0.$$

$\hookrightarrow c$  and  $n_0$  must always be positive



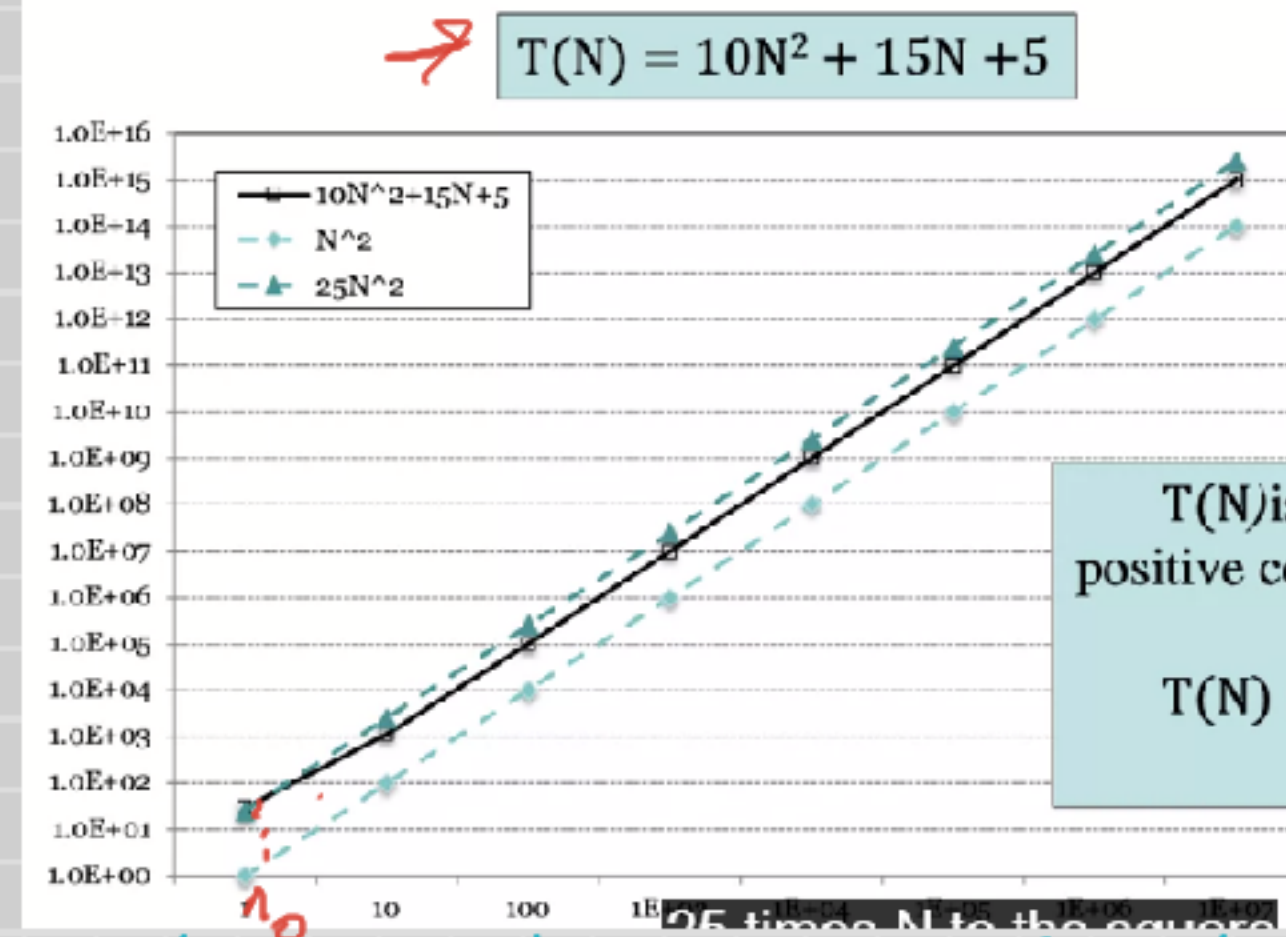
It's not always necessary for  $c * g(N)$  to be equal to or greater than  $T(N)$  for all values of  $N$ , where  $N < n_0$ . There can be values for which  $c * g(N)$  is less than that of  $T(N)$ . For large values of  $N$ ,  $c * g(N)$  is an upper bound for  $T(N)$ .



1. We need to find a function,  $g(n)$ , such that, when it's multiplied by a positive constant,  $c$ , it will comply with being  $\geq T(N)$ , for all values where  $N \geq n_0$ .

2. We'll choose the highest ordered term as  $g(n)$ ,  $N^2$ , we'll choose an arbitrary constant,  $c=1$ .  
 $g(N) = N^2$ ,  $c=1$ .

So, in every case, when we plug in our chosen values,  $T(N)$  is always larger than  $c \cdot g(N)$ . This value of  $c$  is not suitable.



$$c \cdot g(N) \geq T(N) \Rightarrow c \cdot g(N) - T(N) \geq 0$$

$T(N)$  is  $O(g(N))$  if there exist positive constants  $c$  and  $n_0$  such that:  
 $T(N) \leq c \cdot g(N)$  for all  $N \geq n_0$

3. Since the numbers chosen for  $c$  was not suitable, we'll choose another arbitrary number  $c=25$ .  $N=1$ .  $c \cdot g(N)$

$$25N^2. \quad 25(1)^2 = 25, \quad T(N) = 10(1)^2 + 15(1) + 5 = 30$$

We found that  
 value of  $N=2$ .

$g(N) < T(N)$  for  $N=1$  BUT if we change the

we can say the  $T(N)$  is  $O(N^2)$  because  
 $T(N) \leq 25 \cdot g(N), \forall N \geq 2, n_0 = 2$

$$c \cdot g(N) \geq T(N) \\ 25(2)^2 \geq T(2)$$

$$c \cdot g(N) \geq T(N)$$

$$25N^2 \geq 10N^2 + 15N + 5$$

$$15N^2 - 15N - 5 \geq 0$$

Show that  $T(N)$  is  $O(N^3)$ ,  $T(N) = 10N^2 + 15N + 5$

$$1. g(N) = N^3$$

$$2) N = 1, c = 1$$

$$3) N^3 = (1)^3 = 1$$

$$T(N) = 10(1)^2 + 15(1) + 5 = 30$$

$$T(N) \geq c \cdot g(N)$$

Choose another  $c = 20$

$$T(N) = 10(1)^2 + 15(1) + 5 = 30$$

$$c \cdot g(N) = 20N^3$$

$$= 20(1)^3$$

$$= 20$$

$$T(N) \leq c \cdot g(N), N \geq 1$$

∴ We can say that  $T(N)$  is  $O(N^3)$  ✓



1. The running time of an algorithm is defined by  $T(N) = 3N^3 + 2N + 13$ . To prove that  $T(N) = O(N^3)$ , you have to find values for the constants  $c$  and  $n_0$  for which  $T(N) \leq c \cdot N^3$  for all  $N \geq n_0$ . Which of the following values work?

- 1)  $c = 4$   
 $n_0 = 3$   $T(3) = 3(3)^3 + 2(3) + 13 = \underline{100}$   
 $c \cdot g(N) = 4(3)^3 = 108$  }  $T(N) \leq c \cdot g(N) \checkmark \checkmark$
- 2)  ~~$c = 0$~~   
 ~~$n_0 = 10$~~  eliminate because  $c$  must be greater than 0
- 3)  $c = 10$   
 $n_0 = 2$   $T(2) = 3(2)^3 + 2(2) + 13 = 41$   
 $c \cdot g(N) = 10(2)^3 = 80$   $T(N) \leq c \cdot g(N) \checkmark \checkmark$
- 4)  $c = 1$   
 $n_0 = 1$   $T(1) = 3(1)^3 + 2(1) + 13 = 18$   
 $c \cdot g(N) = (1)^3 = 1$  is not suitable  
 $T(N) \geq c \cdot g(N)$
- 5) No such  $c$  and  $n_0$

4. The memory space function  $S(x) = 5x^3 + 3x + 6$  can be described as being in:

$O(x^3)$

$O(x^2)$

None of the others

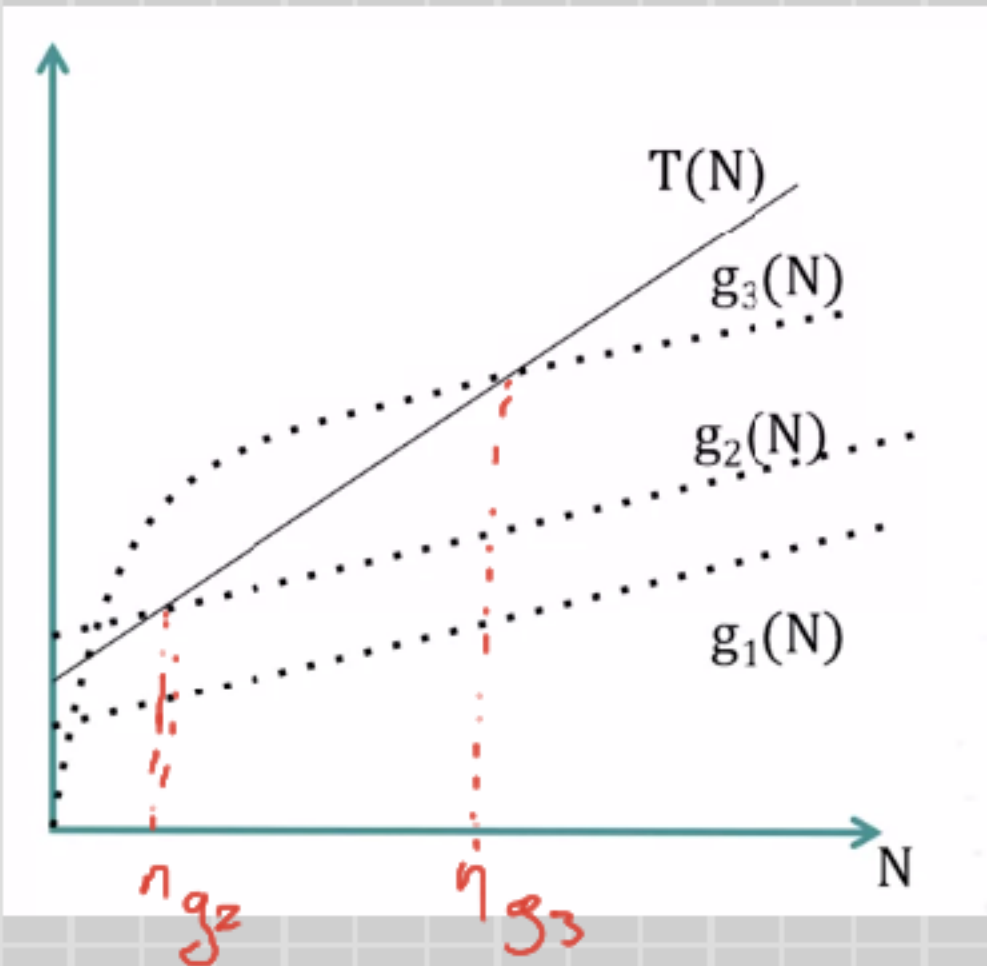
$O(x^4)$

$O(x)$

$O(2^x)$

# Big-Omega

Similar to Big-O notation, but instead of looking for an upper bound, we look for a lower bound.



$g(n)$  can be any of the dotted line functions. It doesn't matter that  $g(N)$  has values greater than  $T(N)$  for some smaller values of  $N$ . It's only important that for very large values of  $N$ , that  $g(N)$  is always equal to or greater than  $T(N)$ .

Our aim is to find a function,  $g(N)$  that acts as a lower bound for  $T(N)$ .

For any of these functions, we can say that  $T(N)$  is  $\Omega(g(n))$ , as they comply with being equal to or less than  $T(N)$ , from a certain value of  $N$ .

It's not necessary that  $c \cdot g(N)$  is equal to or less than  $T(N)$ . There can be values for which  $c \cdot g(N)$  is greater than  $T(N)$ .

Big- $\Omega$  defines a set of functions that act as a lower bound for  $T(N)$ .

Formally, we say that:  $T(N)$  is  $\Omega(g(n))$  if:  $c \cdot g(N) \leq T(N)$ , for all  $N \geq n_0$ .  $c > 0, n_0 > 0$

2. The memory space requirement of an algorithm is described by the function  $S(N) = 4N^2 + 17$ . To prove that  $S(N) = \Omega(N)$  you have to find positive constants  $c$  and  $n_0$  such that  $c \cdot N \leq 4N^2 + 17$  for all  $N \geq n_0$ . Which of the following constants work?

1)  ~~$c = 0$   
 $n_0 = 0$~~

$c$  &  $n$  must be greater than 0.

2)  $c = 1$   
 $n_0 = 1$

3)  $c = 25$   
 $n_0 = 1$

4)  $c = 25$   
 $n_0 = 6$

$$S(N) = 4(1)^2 + 17 = 21$$

$$c \cdot g(N) = (1)(1) = 1$$

$$S(N) = 4(1)^2 + 17 = 21$$

$$c \cdot g(N) = 25(1)^2 = 25$$

$$S(N) = 4(6)^2 + 17 = 161$$

$$c \cdot g(N) = 25(6) = 150$$

$$c \cdot g(N) \leq T(N) \quad \checkmark \checkmark$$

$S(N) \leq c \cdot g(N)$   
doesn't meet requirements

$$S(N) \geq c \cdot g(N)$$

$\checkmark \checkmark$

$$g(N) = N$$

1)  $\Omega(x^3)$   $\checkmark$

4)  $\Omega(x)$   $\checkmark$

2)  $\Omega(1)$   $\checkmark$

3)  $\Omega(x^4)$

3.  $5x^3 + 3x + 6$  can be described as being in:

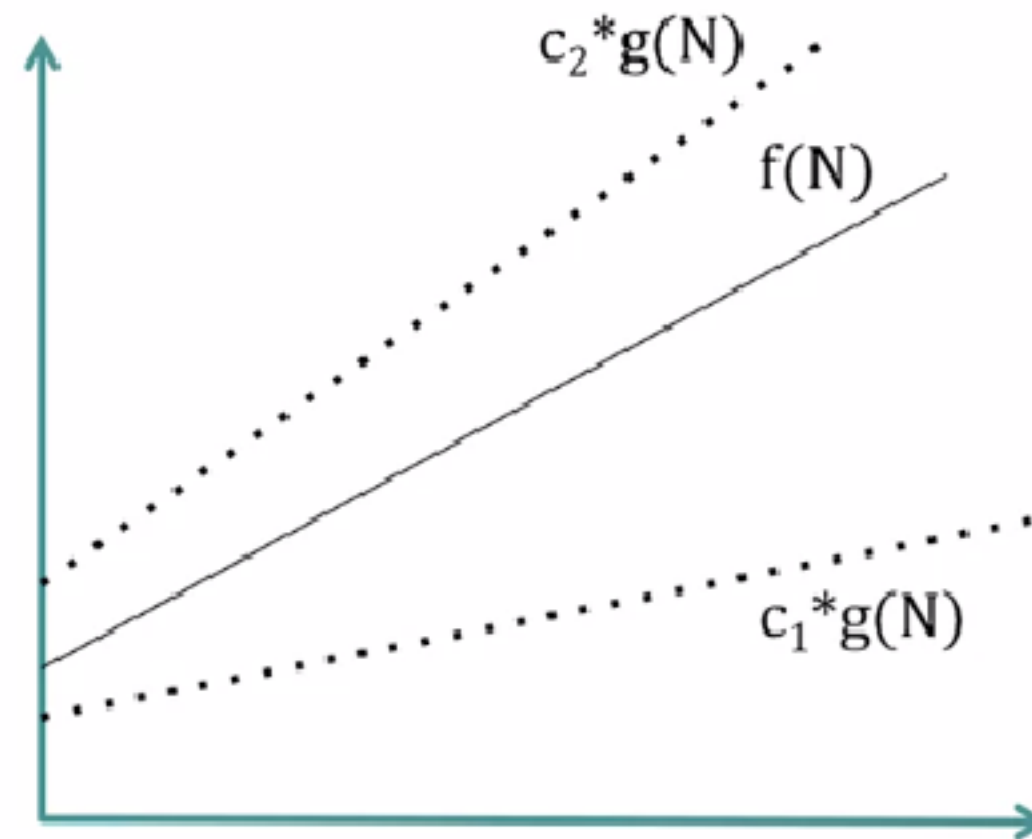


# Theta Notation

Theta notation finds a single function that acts simultaneously as an upper and lower bound for a function.

Our aim with theta notation is find a function,  $g(N)$  that acts as an upper and a lower bound simultaneously

Theta notation

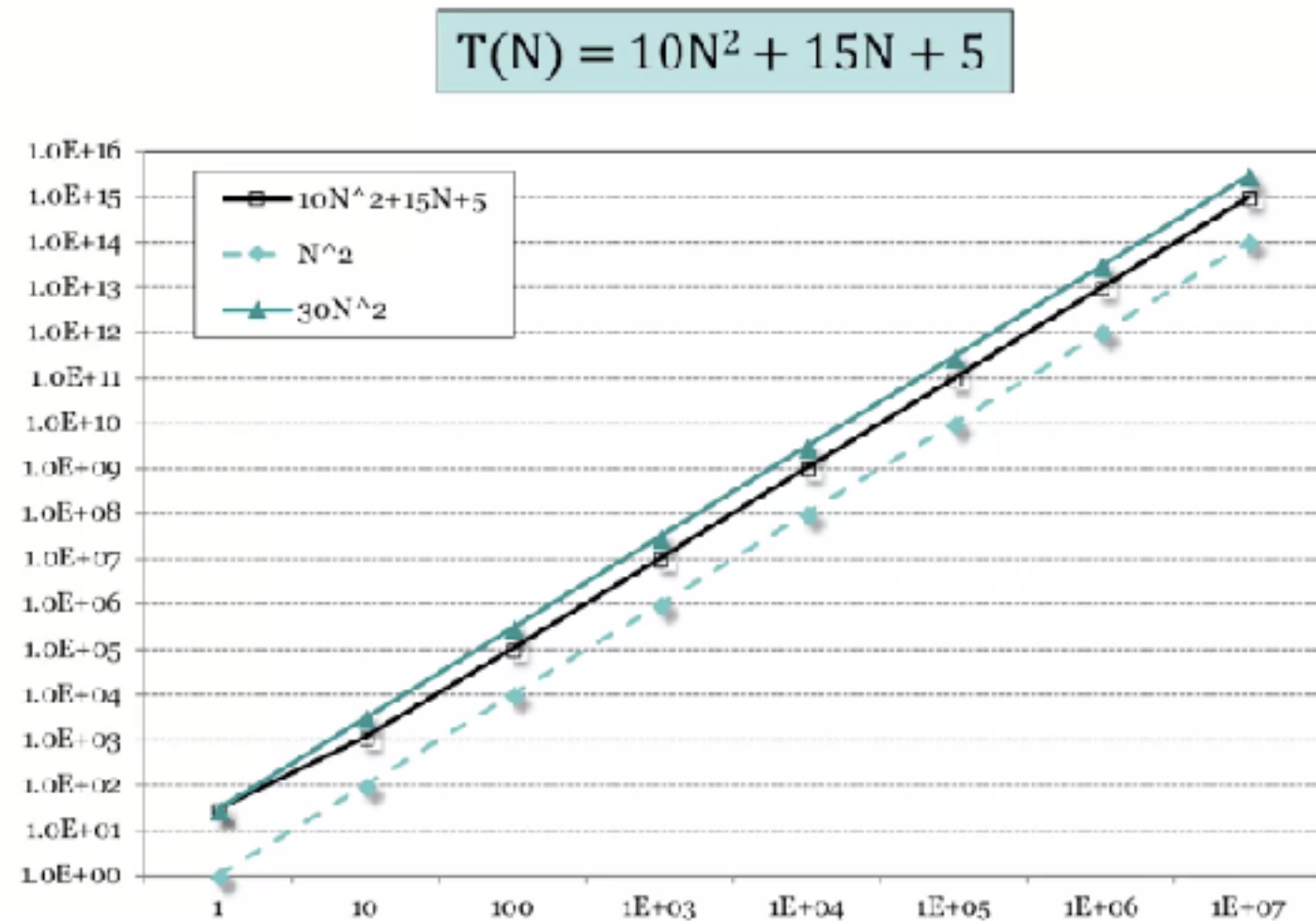


↳ If  $g(N)$  is multiplied by  $c_1$ ,  $c * g(N)$  acts as a lower bound of  $T(N)$ .

↳ If  $g(N)$  is multiplied by  $c_2$ ,  $c * g(N)$  act as an upper bound for  $T(N)$ .

$T(n) = \Theta(n^2)$ . Let us define what this notation means. For a given function  $g(n)$ , we denote by  $\Theta(g(n))$  the set of functions

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}.$



If we choose  $g(N)$  to be  $N^2$ :

$$\left. \begin{array}{l} g(N) = N^2 \\ g(N) = 30N^2 \end{array} \right\} c * g(N)$$

↳ Where  $c * g(N)$  is  $N^2$ , for large values of  $N$ , it acts as **lower bound** for  $T(N)$ .

↳ Where  $c * g(N)$  is  $30N^2$ , for large values of  $N$ , it acts as an **upper bound**.

Formally we say that:

$T(N)$  is  $\Theta(g(N))$  if two conditions are met:

$$\left. \begin{array}{l} 1) c_0 * g(N) \geq T(N), \text{ for all } N \geq n_0 \\ 2) c_1 * g(N) \leq T(N), \text{ for all } N \geq n_0 \end{array} \right\} \begin{array}{l} c_0, c_1 > 0 \\ n_0 > 0 \\ c_0 \leq c_1 \end{array}$$

1. A function described as being  $\Theta(N^4)$  means

It grows at a rate of  $N^4$

2.  $3x\log(x)+17+9x^2$  can be described as being in

$\Theta(x^4)$

$\Theta(x\log x)$

$\Theta(x^3)$

$\Theta(x)$

$\Theta(x^2)$