

# Basic Concepts

## Performance Analysis & Measurement

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,  
National Taiwan Ocean University

Fall 2025



# Outline

- 1 Performance Analysis
- 2 Performance Measurement



# Outline

- 1 Performance Analysis
- 2 Performance Measurement



## Criteria for judging a program:

- Meet the original specification?
- Work correctly?
- The documentation.
- Does the program effectively use functions to create logic units?
- Code readability.
- Efficient usage of storage?
- Acceptable running time?



# Performance Analysis

machine **independent**

- Space complexity
  - The amount of memory that it needs to run to completion.
- Time complexity
  - Computing time



# Space complexity

$$S(P) = c + S_P(I),$$

$P$ : the program;  $I$ : the input.

- ★  $S_P(I)$  can be represented by  $S_P(n)$  if  $n$  is the only instance characteristic.



# Space complexity

$$S(P) = c + S_P(I),$$

$P$ : the program;  $I$ : the input.

- ★  $S_P(I)$  can be represented by  $S_P(n)$  if  $n$  is the only instance characteristic.
- Fixed space requirement:  $c$ .
  - Independent of the characteristics of the inputs and outputs.
    - Instruction space.
    - Space for simple variables, fixed-size structured variable and constants.
- **Variable Space Requirement ( $S_P(I)$ )**
  - depend on the instance characteristic  $I$ .
    - For instance, additional space when the program uses recursion.
  - values of inputs and outputs associated with  $I$ .



## Example

- Assume that the integers are stored in an **array** 'list', such that the  $i$ th integer is stored in the  $i$ th position `list[i]`.

```
float abc(float a, float b, float c) {  
    return a + b + b * c + (a + b - c) / (a + b) + 4.00;  
}
```

- Fixed space requirement (c): 16.
  - Three float numbers: a, b, c and one return float number.
- $S_{abc}(I) = 0$ . (for only fixed space requirements)





## Example

```
float sum(float list[ ], int n) {  
    float temp = 0;  
    int i;  
    for (i=0; i<n; i++)  
        temp += list[i];  
    return temp;  
}
```

- In this program,  $S_{\text{sum}}(I) = 0$ .
- C Programming Language: passing the address of the first element of `list[]` (instead of copying).



## Example (recursive)

```
float rsum(float list[ ], int n) {  
    if (n) return list[n] + rsum(list, n-1);  
    return list[0];  
}
```

- Total variable space:  $S_{\text{rsum}}(l) = 12n$ .
  - parameter list[]: array pointer: 4 bytes.
  - parameter n: integer: 4 bytes
  - return address (internally used): 4 bytes.
- The recursive version has a **far greater** overhead than its iterative counterpart.



# Time Complexity: $T(P) = c + T_P(I)$

- Compile time:  $c$ 
  - Independent of the characteristics of the input and output.
  - Once the correctness of the program is verified, it can run without recompilation.
- Run time:  $T_P(I)$  (what we are really concerned about)
  - E.g.,  $T_P(n) = c_a \cdot \text{ADD}(n) + c_s \cdot \text{SUB}(n) + c_l \cdot \text{LDA}(n) + c_{st} \cdot \text{STA}(n)$ .
  - ADD, SUB, LDA, STA: the number of additions, subtractions, loads and stores.
  - $c_a, c_s, c_l, c_{st}$ : the time needed to perform each operation (constants).



# Time Complexity - Program Step (1/2)

- ★ machine independent

## Program Step

a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics.

- Example of **ONE program step**
  - $a = 2;$
  - $a = 2*b + 3*c/d - e + f/g/a/b/c;$



## Time Complexity - Program Step (1/2)

### Methods to compute the number of program steps

- Creating a global variable, say, count.
- Tabular method:
  - Compute the contribution of a statement:  
 $\# \text{ program steps per execution} \times \text{frequency}.$
  - Add up the contribution of all statements.

## Example

```
float sum(float list[ ], int n) {  
    float tempSum = 0; count++; /* for assignment */  
    int i;  
    for (i = 0; i < n; i++) {  
        count++; /* for the "for" loop */  
        tempSum += list[i]; count++; /* for assignment */  
    }  
    count++; /* last execution of "for" */  
    count++; /* for return */  
    return tempSum;  
}
```

- $\text{count} = 2n + 3$  (steps).



# Example (Tabular Method)

Statements	s/e	Frequency	Total Steps
float sum(float list[ ], int n) {	0	0	0
float tempsum = 0;	<b>1</b>	<b>1</b>	<b>1</b>
int i;	0	0	0
for(i=0; i <n; i++)	<b>1</b>	<b>n+1</b>	<b>n+1</b>
tempsum += list[i];	<b>1</b>	<b>n</b>	<b>n</b>
return tempsum;	<b>1</b>	<b>1</b>	<b>1</b>
}	0	0	0
Total			$2n + 3$

# Asymptotic Notation

- Issues: determining the “exact” step count of a program could be difficult or complicated.





# Asymptotic Notation

- Issues: determining the “exact” step count of a program could be difficult or complicated.
- ▶ Using asymptotic notations:  $O, \Omega, \Theta, \dots$



# Asymptotic Notation

- Issues: determining the “exact” step count of a program could be difficult or complicated.
- ▶ Using asymptotic notations:  $O, \Omega, \Theta, \dots$
- A motivating example:



# Asymptotic Notation

- Issues: determining the “exact” step count of a program could be difficult or complicated.
- ▷ Using asymptotic notations:  $O, \Omega, \Theta, \dots$
- A motivating example:
- $c_3n < c_1n^2 + c_2n$  when  $n$  is sufficiently large.
  - For  $c_1 = 1, c_2 = 2, c_3 = 100, c_1n^2 + c_2n \leq c_3n$  for  $n \leq 98$ .
  - For  $c_1 = 1, c_2 = 2, c_3 = 1000, c_1n^2 + c_2n \leq c_3n$  for  $n \leq 998$ .



# Asymptotic Notation

- Issues: determining the “exact” step count of a program could be difficult or complicated.
- ▷ Using asymptotic notations:  $O, \Omega, \Theta, \dots$
- A motivating example:
- $c_3n < c_1n^2 + c_2n$  when  $n$  is sufficiently large.
  - For  $c_1 = 1, c_2 = 2, c_3 = 100, c_1n^2 + c_2n \leq c_3n$  for  $n \leq 98$ .
  - For  $c_1 = 1, c_2 = 2, c_3 = 1000, c_1n^2 + c_2n \leq c_3n$  for  $n \leq 998$ .
  - ★ For small values of  $n$ , either one could be faster.



# Big-O Notation

## Definition ( $O(\cdot)$ )

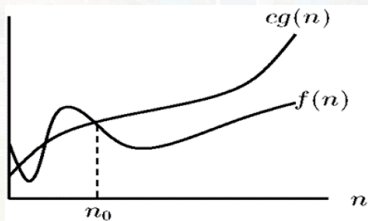
$f(n) = O(g(n))$  iff there exist positive constants  $c$  and  $n_0 \in \mathbb{N}$  such that  $f(n) \leq cg(n)$  for all  $n \geq n_0$ .

# Big-O Notation

## Definition ( $O(\cdot)$ )

$f(n) = O(g(n))$  iff there exist positive constants  $c$  and  $n_0 \in \mathbb{N}$  such that  $f(n) \leq cg(n)$  for all  $n \geq n_0$ .

- $g(n)$  serves as an **upper bound** on  $f(n)$ .
  - The smaller  $g(n)$  is, the more informative it would be!



# Big- $\Omega$ Notation

## Definition ( $\Omega(\cdot)$ )

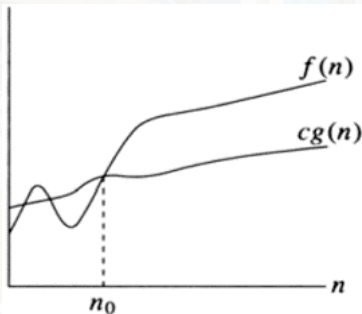
$f(n) = \Omega(g(n))$  iff there exist positive constants  $c$  and  $n_0 \in \mathbb{N}$  such that  $f(n) \geq cg(n)$  for all  $n \geq n_0$ .

# Big- $\Omega$ Notation

## Definition ( $\Omega(\cdot)$ )

$f(n) = \Omega(g(n))$  iff there exist positive constants  $c$  and  $n_0 \in \mathbb{N}$  such that  $f(n) \geq cg(n)$  for all  $n \geq n_0$ .

- $g(n)$  serves as an **lower bound** on  $f(n)$ .
  - The larger  $g(n)$  is, the more informative it would be!





# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?

# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .
  - $100n + 6 \leq 101n$  for  $n \geq 10$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .
  - $100n + 6 \leq 101n$  for  $n \geq 10$ .
- $10n^2 + 4n + 2 = O(n^2)$ .





## Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .
  - $100n + 6 \leq 101n$  for  $n \geq 10$ .
- $10n^2 + 4n + 2 = O(n^2)$ .
  - $10n^2 + 4n + 2 \leq 11n^2$  for  $n \geq 5$ .

## Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .
  - $100n + 6 \leq 101n$  for  $n \geq 10$ .
- $10n^2 + 4n + 2 = O(n^2)$ .
  - $10n^2 + 4n + 2 \leq 11n^2$  for  $n \geq 5$ .
- $6 \cdot 2^n + n^2 = O(2^n)$ .



# Examples (Big- $O$ )

- $3n + 2 = O(n)$ .
  - $3n + 2 \leq 4n$  for  $n \geq 2$ .
- $3n + 3 = O(n^2)$ .
  - $3n + 3 \leq 4n^2$  for  $n \geq 2$ .
  - Why and how can we make sure this?
- $100n + 6 = O(n)$ .
  - $100n + 6 \leq 101n$  for  $n \geq 10$ .
- $10n^2 + 4n + 2 = O(n^2)$ .
  - $10n^2 + 4n + 2 \leq 11n^2$  for  $n \geq 5$ .
- $6 \cdot 2^n + n^2 = O(2^n)$ .
  - $6 \cdot 2^n + n^2 \leq 7 \cdot 2^n$  for  $n \geq 4$ .



# Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .



# Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .



# Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .





## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .



# Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .
  - $10n^2 + 4n + 2 \geq 10n^2$  for  $n \geq 1$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .
  - $10n^2 + 4n + 2 \geq 10n^2$  for  $n \geq 1$ .
- $6 \cdot 2^n + n^2 = \Omega(2^n)$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .
  - $10n^2 + 4n + 2 \geq 10n^2$  for  $n \geq 1$ .
- $6 \cdot 2^n + n^2 = \Omega(2^n)$ .
  - $6 \cdot 2^n + n^2 \geq 2^n$  for  $n \geq 1$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .
  - $10n^2 + 4n + 2 \geq 10n^2$  for  $n \geq 1$ .
- $6 \cdot 2^n + n^2 = \Omega(2^n)$ .
  - $6 \cdot 2^n + n^2 \geq 2^n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(1)$ .



## Examples (Big- $\Omega$ )

- $3n + 2 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $3n + 3 = \Omega(n)$ .
  - $3n + 2 \geq 3n$  for  $n \geq 1$ .
- $100n + 6 = \Omega(n)$ .
  - $100n + 6 \geq 100n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(n^2)$ .
  - $10n^2 + 4n + 2 \geq 10n^2$  for  $n \geq 1$ .
- $6 \cdot 2^n + n^2 = \Omega(2^n)$ .
  - $6 \cdot 2^n + n^2 \geq 2^n$  for  $n \geq 1$ .
- $10n^2 + 4n + 2 = \Omega(1)$ .
- $6 \cdot 2^n + n^2 = \Omega(1)$ .



# Polynomial

## Theorem 1.2

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = O(n^k)$ .





# Polynomial

## Theorem 1.2

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = O(n^k)$ .

Proof:

$$f(n) \leq \sum_{i=0}^k |a_i| n^i = n^k \sum_{i=0}^k |a_i| n^{i-k} \leq n^k \sum_{i=0}^k |a_i|, \text{ for } n \geq 1.$$

# Polynomial

## Theorem 1.2

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = O(n^k)$ .

Proof:

$$f(n) \leq \sum_{i=0}^k |a_i| n^i = n^k \sum_{i=0}^k |a_i| n^{i-k} \leq n^k \sum_{i=0}^k |a_i|, \text{ for } n \geq 1.$$

Note that  $n^{i-k} \leq 1$  if  $i \leq k$  and  $\sum_{i=0}^k |a_i|$  is a constant.



# Most often seen big- $O$ complexities

\* with respect to the input of size  $n$ .

- $O(1)$ : constant.
- $O(n)$ : linear.
- $O(n^2)$ : quadratic.
- $O(n^3)$ : cubic.
- $O(2^n)$ : exponential.



# Most often seen big- $O$ complexities

\* with respect to the input of size  $n$ .

- $O(1)$ : constant.
- $O(n)$ : linear.
- $O(n^2)$ : quadratic.
- $O(n^3)$ : cubic.
- $O(2^n)$ : exponential.
- $O(2^{\sqrt{n}})$ : sub-exponential.



# Most often seen big- $O$ complexities

\* with respect to the input of size  $n$ .

- $O(1)$ : constant.
- $O(n)$ : linear.
- $O(n^2)$ : quadratic.
- $O(n^3)$ : cubic.
- $O(2^n)$ : exponential.
- $O(2^{\sqrt{n}})$ : sub-exponential.
- $O(\log n)$ : logarithmic.



# Most often seen big- $O$ complexities

\* with respect to the input of size  $n$ .

- $O(1)$ : constant.
- $O(n)$ : linear.
- $O(n^2)$ : quadratic.
- $O(n^3)$ : cubic.
- $O(2^n)$ : exponential.
- $O(2^{\sqrt{n}})$ : sub-exponential.
- $O(\log n)$ : logarithmic.
  - $O(\lg n)$ ?
- $O(n \log n)$ : log linear.



# Polynomial (Lower Bound)

## Theorem 1.3

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = \Omega(n^k)$ .



# Polynomial (Lower Bound)

## Theorem 1.3

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = \Omega(n^k)$ .

Proof:

- Skipped and left as an exercise.



# Theta Notation ( $\Theta$ )

## Definition ( $\Theta$ )

$f(n) = \Theta(g(n))$  iff  $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$ .

- More precise than simply using big- $O$  or big- $\Omega$  notations.

# Theta Notation ( $\Theta$ )

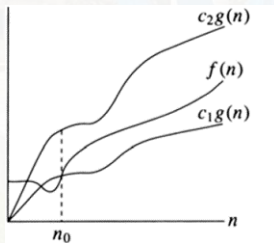
## Definition ( $\Theta$ )

$f(n) = \Theta(g(n))$  iff  $f(n) = O(g(n))$  and  $f(n) = \Omega(g(n))$ .

- More precise than simply using big- $O$  or big- $\Omega$  notations.

## Theorem 1.4

If  $f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ , then  $f(n) = \Theta(n^k)$



# Example (Tabular Method)

Statements	s/e	Frequency	Total Steps	Asymptotic Complexity
void add (int a[ ][MAX_SIZE],...) {	0	0	0	0
int i, j;	0	0	0	0
for (i = 0; i < row; i++)	<b>1</b>	<b>rows+1</b>	<b>rows+1</b>	$\Theta(\text{rows})$
for (j=0; j< cols; j++)	<b>1</b>	<b>rows*(cols+1)</b>	<b>rows*(cols+1)</b>	$\Theta(\text{rows} \cdot \text{cols})$
c[i][j] = a[i][j]+b[i][j];	<b>1</b>	<b>rows*cols</b>	<b>rows*cols</b>	$\Theta(\text{rows} \cdot \text{cols})$
}	0	0	0	0
Total	$2 \cdot \text{rows} \cdot \text{cols} + 2 \cdot \text{rows} + 1$			$\Theta(\text{rows} \cdot \text{cols})$



# Function Values & Plots

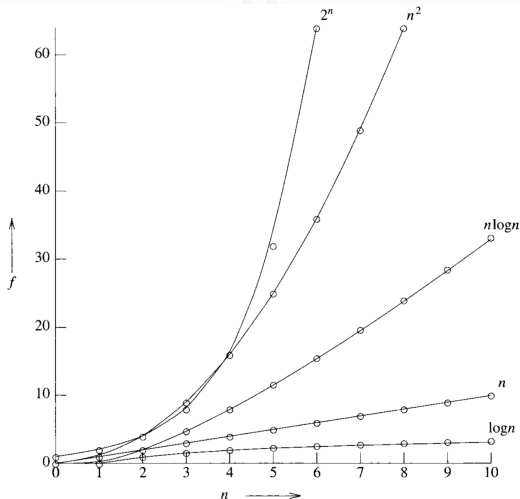
Refer to Fig. 1.7 & 1.8 in the textbook.

		Instance characteristic $n$					
Time	Name	1	2	4	8	16	32
1	Constant	1	1	1	1	1	1
$\log n$	Logarithmic	0	1	2	3	4	5
$n$	Linear	1	2	4	8	16	32
$n \log n$	Log linear	0	2	8	24	64	160
$n^2$	Quadratic	1	4	16	64	256	1024
$n^3$	Cubic	1	8	64	512	4096	32768
$2^n$	Exponential	2	4	16	256	65536	4294967296
$n!$	Factorial	1	2	24	40326	20922789888000	$26313 \times 10^{33}$



# Function Values & Plots

Refer to Fig. 1.7 & 1.8 in the textbook.



# Outline

- 1 Performance Analysis
- 2 Performance Measurement



# Motivations

- Sometimes we still need to consider how long an algorithm executes **on our machine**.
- In order to obtain accurate times, we can repeatedly run the programs for several times (and take the average running time).



# The Tricks

```
#include<time.h>
```

	1st Method	2nd Method
start timing	<code>start = clock();</code>	<code>start = time(NULL);</code>
stop timing	<code>end = clock();</code>	<code>end = time(NUL);</code>
type returned	<code>clock_t</code>	<code>time_t</code>

Result (in seconds):

- 1st Method: `duration = (double)(stop-start))/(CLOCKS_PER_SEC);`
- 2nd Method: `duration = (double)difftime(stop, start);`



# The Tricks (Example)

```
... // previous code omitted
clock_t start, stop;
double duration;
printf("n time\n");
for(i=0; i< ITERATIONS; i++) {
    for(j=0; j<sizeList[i]; j++)
        list[j] = sizeList[i]-j; /* worst case */
    start = clock();
    sort(list, sizeList[i]);
    stop = clock();
    /* number of clock ticks per second */
    duration = ((double) (stop-start));
    printf("%6d %f\n", sizeList[i], duration);
}
}
```

⇒ sample code.



# Discussions

