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 ith integer is stored in the ith 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.
- $S_{\text{abc}}(I) = 0$. (for only fixed space requirements)

Example

```c
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, $S_{\mathrm{sum}}(I) = 0$.
- 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: $S_{\mathrm{rsum}}(I) = 12n$.
  - 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., $T_P(n) = c_a \cdot \mathrm{ADD}(n) + c_s \cdot \mathrm{SUB}(n) + c_l \cdot \mathrm{LDA}(n) + c_{st} \cdot \mathrm{STA}(n)$.
  - 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:
    \# program steps per execution $\times$ frequency.
  - Add up the contribution of all statements.

Example

```c
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;
}
```

- 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;`	**1**	**1**	**1**
`int i;`	0	0	0
`for(i=0; i <n; i++)`	**1**	**n+1**	**n+1**
`tempsum += list[i];`	**1**	**n**	**n**
`return tempsum;`	**1**	**1**	**1**
`}`	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: $O, \Omega, \Theta, \ldots$

Asymptotic Notation

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

▷ Using asymptotic notations: $O, \Omega, \Theta, \ldots$

- A motivating example:

Asymptotic Notation

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

▷ Using asymptotic notations: $O, \Omega, \Theta, \ldots$

- A motivating example:
- $c_3 n < c_1 n^2 + c_2 n$ when n is sufficiently large.
  - For $c_1 = 1$, $c_2 = 2$, $c_3 = 100$, $c_1 n^2 + c_2 n \leq c_3 n$ for $n \leq 98$.
  - For $c_1 = 1$, $c_2 = 2$, $c_3 = 1000$, $c_1 n^2 + c_2 n \leq c_3 n$ for $n \leq 998$.

Asymptotic Notation

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

▷ Using asymptotic notations: $O, \Omega, \Theta, \ldots$

- A motivating example:
- $c_3 n < c_1 n^2 + c_2 n$ when n is sufficiently large.
  - For $c_1 = 1$, $c_2 = 2$, $c_3 = 100$, $c_1 n^2 + c_2 n \leq c_3 n$ for $n \leq 98$.
  - For $c_1 = 1$, $c_2 = 2$, $c_3 = 1000$, $c_1 n^2 + c_2 n \leq c_3 n$ for $n \leq 998$.

  ⋆ For small values of n, either one could be faster.

Big-O Notation

Definition ($O(\cdot)$)

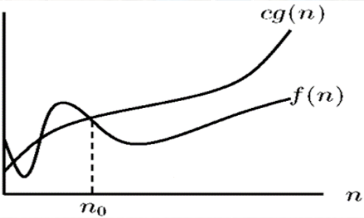
$f(n) = O(g(n))$ (or write $f(n) \in O(g(n))$) iff there exist positive constants c and $n_0 \in \mathbb{N}$ such that $f(n) \le cg(n)$ for all $n \ge n_0$.

Big-O Notation

Definition ($O(\cdot)$)

$f(n) = O(g(n))$ (or write $f(n) \in O(g(n))$) 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-Ω Notation

Definition ($\Omega(\cdot)$)

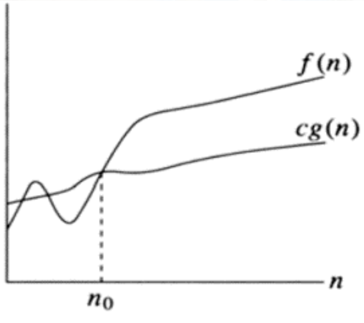
$f(n) = \Omega(g(n))$ (or write $f(n) \in \Omega(g(n))$) iff there exist positive constants c and $n_0 \in \mathbb{N}$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Big-Ω Notation

Definition ($\Omega(\cdot)$)

$f(n) = \Omega(g(n))$ (or write $f(n) \in \Omega(g(n))$) 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-Ω)

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

Examples (Big-Ω)

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

Examples (Big-Ω)

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

Examples (Big-Ω)

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

Examples (Big-Ω)

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

Examples (Big-Ω)

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

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

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

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

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

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

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

A Disproof Example

- **Claim:** $3n + 2$ is *not* $\Omega(n^2)$.

Proof (by contradiction). Assume $3n + 2 \in \Omega(n^2)$. Then there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that

$$\forall n \geq n_0 : \quad 3n + 2 \ \geq \ c\,n^2. \tag{$*$}$$

However, for all $n \geq 1$, we have $3n + 2 \ < \ 3n + 2n \ = \ 5n$. Moreover, if $n \geq \frac{5}{c}$, then $5n \leq c\,n^2$. Hence for

$$N \ = \ \max\{\,1,\ \tfrac{5}{c},\ n_0\,\}$$

and all $n \geq N$,

$$3n + 2 \ < \ c\,n^2,$$

which contradicts $(*)$. Therefore $3n + 2 \notin \Omega(n^2)$. $\square$

Polynomial

Theorem 1.2

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

Polynomial

Theorem 1.2

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

Proof:

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

Note that $n^{i-k} \le 1$ if $i \le k$ 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 $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$, then $f(n) = \Omega(n^k)$.

Polynomial (Lower Bound)

Theorem 1.3

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

Proof:

- Skipped and left as an exercise.

Theta Notation (Θ)

Definition (Θ)

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

- More precise than simply using big-O or big-Ω notations.

Theta Notation (Θ)

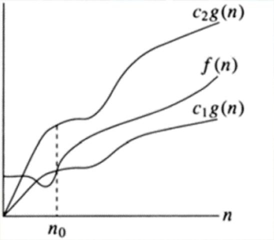
Definition (Θ)

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

- More precise than simply using big-O or big-Ω notations.

Theorem 1.4

If $f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$, 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++)	**1**	**rows+1**	**rows+1**	$\Theta(\text{rows})$
for (j=0; j< cols; j++)	**1**	**rows*(cols+1)**	**rows*(cols+1)**	$\Theta(\text{rows} \cdot \text{cols})$
c[i][j] = a[i][j]+b[i][j];	**1**	**rows*cols**	**rows*cols**	$\Theta(\text{rows} \cdot \text{cols})$
}	0	0	0	0
Total	$2 \cdot \text{rows} \cdot \text{cols} + 2 \cdot \text{rows} + 1$			$\Theta(\text{rows} \cdot \text{cols})$

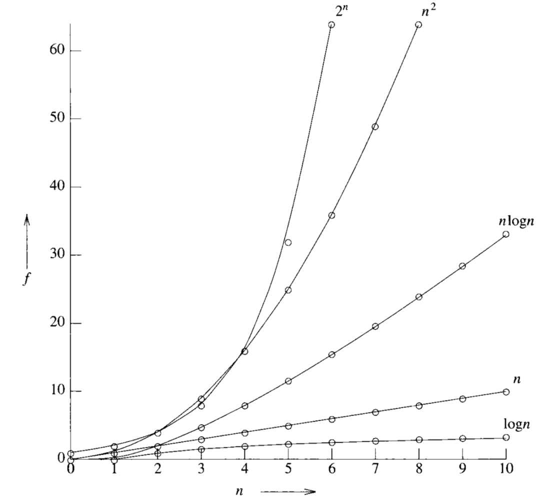
Function Values & Plots

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	26313×10^{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	`start = clock();`	`start = time(NULL);`
stop timing	`end = clock();`	`end = time(NUL);`
type returned	`clock_t`	`time_t`

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