# Chapter 2
# Complexity of Algorithms

*Data Structures and Algorithms*

**Luong The Nhan, Tran Giang Son**
*Faculty of Computer Science and Engineering*
*University of Technology, VNU-HCM*

# Outcomes

- **L.O.1.1** - Define concept "computational complexity" and its sepcial cases, best, average, and worst.
- **L.O.1.2** - Analyze algorithms and use Big-O notation to characterize the computational complexity of algorithms composed by using the following control structures: sequence, branching, and iteration (not recursion).
- **L.O.1.3** - List, give examples, and compare complexity classes, for examples, constant, linear, etc.
- **L.O.1.4** - Be aware of the trade-off between space and time in solutions.
- **L.O.1.5** - Describe strategies in algorithm design and problem solving.

# Contents

# Algorithm Efficiency

## Algorithm Efficiency

- A problem often has many algorithms.
- Comparing two different algorithms $\Rightarrow$ Computational complexity: measure of the difficulty degree (time and/or space) of an algorithm.
    - How fast an algorithm is?
    - How much memory does it cost?

## Algorithm Efficiency

**General format**

**efficiency = f(n)**

n is the size of a problem (the key number that determines the size of input data)

# Linear Loops

```
for (i = 0; i < 1000; i++)
    application code
```

The number of times the bodyof the loop is replicated is 1000.

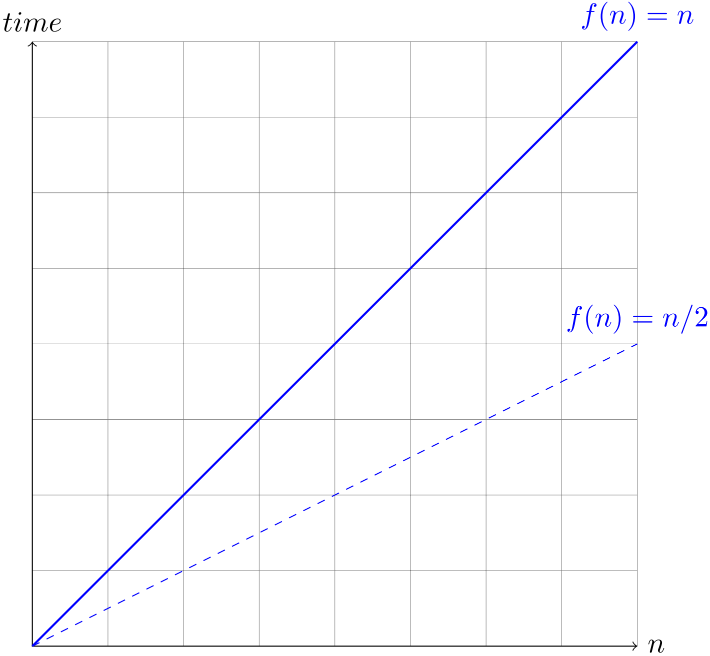
$$f(n) = n$$

```
for (i = 0; i < 1000; i += 2)
    application code
```

The number of times the bodyof the loop is replicated is 500.

$$f(n) = n/2$$

# Linear Loops



# Logarithmic Loops

## Multiply loops

```
i = 1
while (i <= 1000)
   application code
   i = i x 2
```

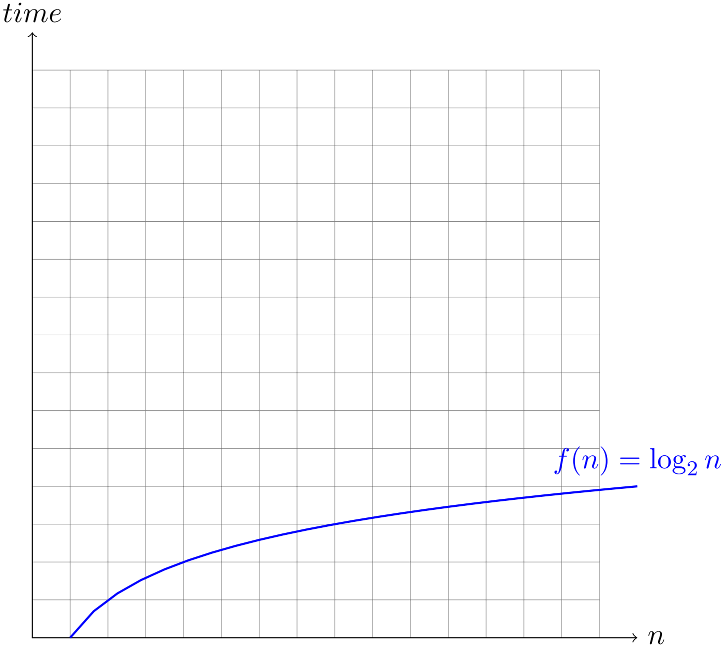
## Divide loops

```
i = 1000
while (i >= 1)
   application code
   i = i / 2
```

The number of times the body of the loop is replicated is

$$\mathbf{f(n)} = \log_2 n$$

# Logarithmic Loops



## Nested Loops

Iterations = Outer loop iterations × Inner loop iterations

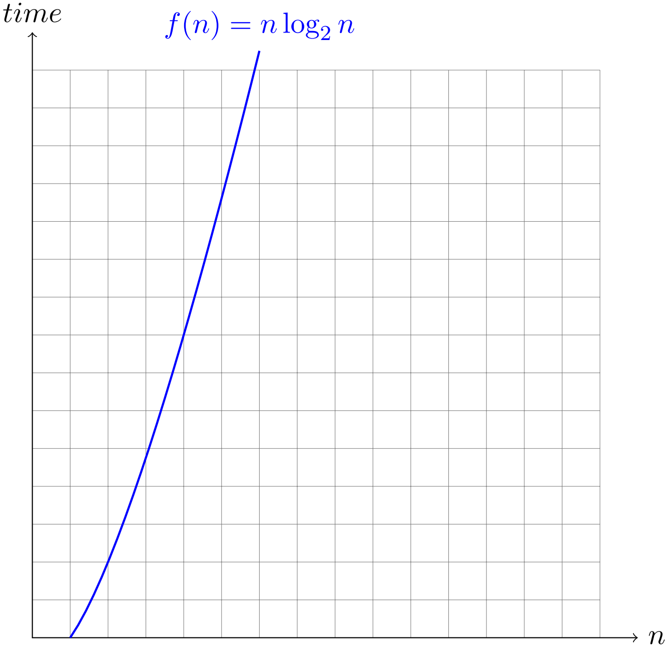
### Example

```
i = 1
while (i <= 10)
   j = 1
   while (j <= 10)
      application code
      j = j * 2
   i = i + 1
```

The number of times the body of the loop is replicated is

$$f(n) = n \log_2 n$$

# Nested Loops



## Quadratic Loops



### Example

```
i = 1
while (i <= 10)
   j = 1
   while (j <= 10)
      application code
      j = j + 1
   i = i + 1
```

The number of times the body of the loop is replicated is

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

# Dependent Quadratic Loops



Luong The Nhan, Tran Giang Son



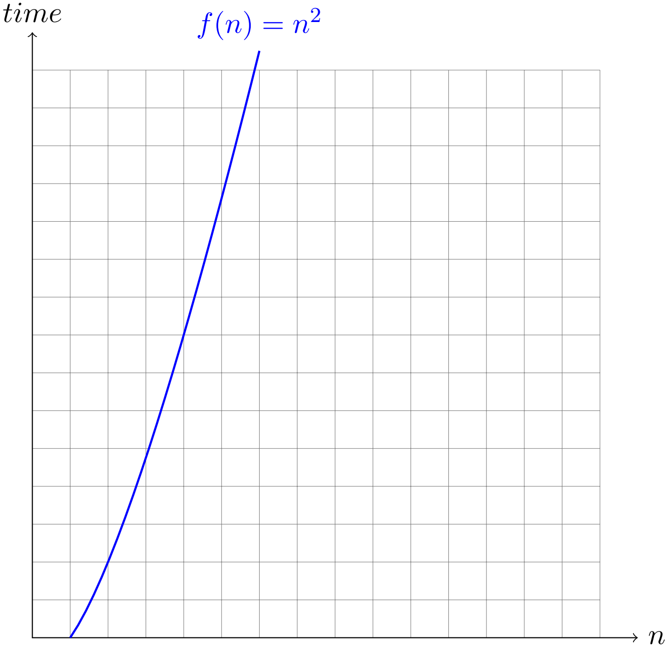
### Example

```
i = 1
while (i <= 10)
   j = 1
   while (j <= i)
      application code
      j = j + 1
   i = i + 1
```

The number of times the body of the loop is replicated is

$$1 + 2 + \ldots + n = n(n + 1)/2$$

# Quadratic Loops



## Asymptotic Complexity

- Algorithm efficiency is considered with only big problem sizes.
- We are not concerned with an exact measurement of an algorithm's efficiency.
- Terms that do not substantially change the function's magnitude are eliminated.

# Big-O notation

# Big-O notation

**Example**

$f(n) = c.n \Rightarrow f(n) = O(n)$
$f(n) = n(n+1)/2 = n^2/2 + n/2 \Rightarrow f(n) = O(n^2)$

- Set the coefficient of the term to one.
- Keep the largest term and discard the others.

Some example of Big-O:

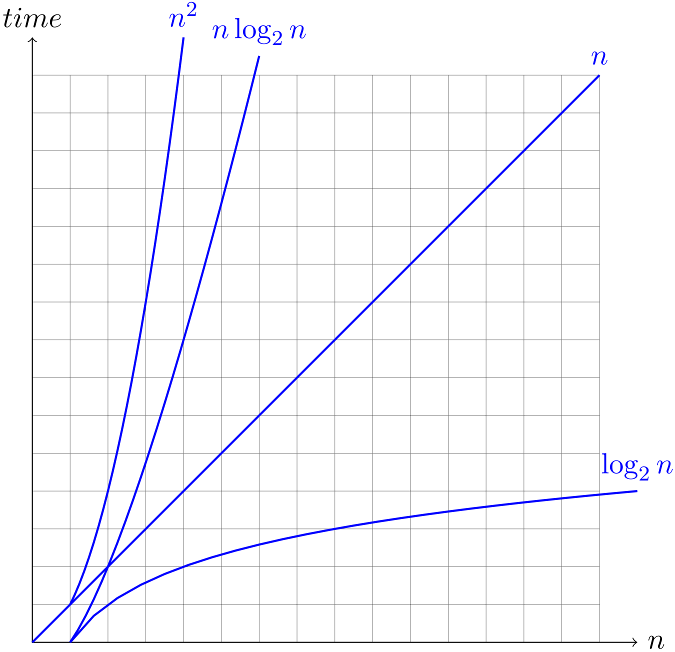
$\log_2 n \quad n \quad n\log_2 n \quad n^2 \ \dots \ n^k \ \dots \ 2^n \quad n!$

## Standard Measures of Efficiency

| Efficiency | Big-O | Iterations | Est. Time |
|---|---|---|---|
| logarithmic | $O(\log_2 n)$ | 14 | microseconds |
| linear | $O(n)$ | 10 000 | 0.1 seconds |
| linear log | $O(n \log_2 n)$ | 140 000 | 2 seconds |
| quadratic | $O(n^2)$ | $10000^2$ | 15-20 min. |
| polynomial | $O(n^k)$ | $10000^k$ | hours |
| exponential | $O(2^n)$ | $2^(10000)$ | intractable |
| factorial | $O(n!)$ | $10000!$ | intractable |

Assume instruction speed of 1 microsecond and 10 instructions in loop.
$n = 10000$

# Standard Measures of Efficiency



## Big-O Analysis Examples

**Algorithm** addMatrix(val matrix1<matrix>, val matrix2<matrix>, val size<integer>, ref matrix3<matrix>)
Add matrix1 to matrix2 and place results in matrix3
**Pre:** matrix1 and matrix2 have data
size is number of columns and rows in matrix
**Post:** matrices added - result in matrix3

```
r = 1
while r <= size do
    c = 1
    while c <= size do
        matrix3[r, c] = matrix1[r, c] + matrix2[r, c]
        c = c + 1
    end
    r = r + 1
end
return matrix3
End addMatrix
```

## Big-O Analysis Examples

Nested linear loop:

$$f(size) = O(size^2)$$

## Time Costing Operations

- The most time consuming: data movement to/from memory/storage.
- Operations under consideration:
  - Comparisons
  - Arithmetic operations
  - Assignments

# Problems and common complexities

## Binary search

**Recurrence Equation (Phương trình hồi quy)**

An equation or inequality that describes a function in terms of its value on smaller input.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 |

$$T(n) = 1 + T(n/2) \Rightarrow T(n) = O(\log_2 n)$$

## Binary search

- Best case: when the number of steps is smallest. $T(n) = O(1)$
- Worst case: when the number of steps is largest. $T(n) = O(\log_2 n)$
- Average case: in between. $T(n) = O(\log_2 n)$

## Sequential search

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 |

- Best case: $T(n) = O(1)$
- Worst case: $T(n) = O(n)$
- Average case: $T(n) = \sum_{i=1}^{n} i.p_i$
  $p_i$ : probability for the target being at a[i]
  $p_i = 1/n \Rightarrow T(n) = (\sum_{i=1}^{n} i)/n = O(n(n+1)/2n) = O(n)$

## Quick sort

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 19 | 8 | 3 | 15 | 28 | 10 | 22 | 4 | 12 | 83 |

**Recurrence Equation**

$$T(n) = O(n) + 2T(n/2)$$

- Best case: $T(n) = O(n \log_2 n)$
- Worst case: $T(n) = O(n^2)$
- Average case: $T(n) = O(n \log_2 n)$

# P and NP Problems

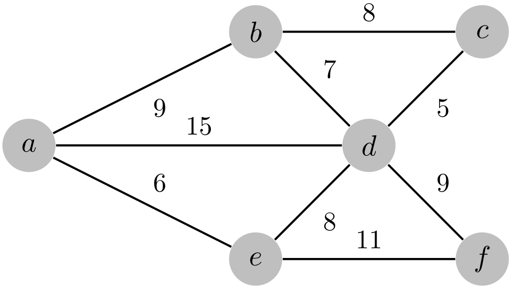
## P and NP Problems

- P: Polynomial (can be solved in polynomial time on a deterministic machine).
- NP: Nondeterministic Polynomial (can be solved in polynomial time on a nondeterministic machine).

# P and NP Problems

### Travelling Salesman Problem:

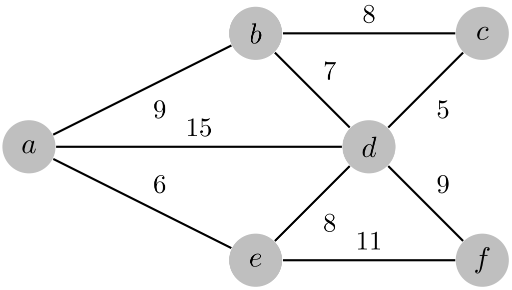
A salesman has a list of cities, each of which he must visit exactly once. There are direct roads between each pair of cities on the list.
Find the route the salesman should follow for the shortest possible round trip that both starts and finishes at any one of the cities.



# P and NP Problems

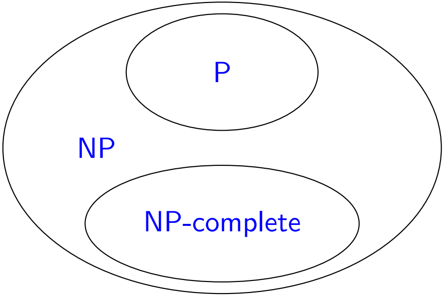
**Travelling Salesman Problem:**

Deterministic machine: $f(n) = n(n-1)(n-2)\ldots 1 = O(n!)$
$\Rightarrow$ NP problem



## P and NP Problems

NP-complete: NP and every other problem in NP is polynomially reducible to it.




$$P = NP?$$