Big-O Blue

LECTURE 02

ALGORITHMIC COMPLEXITY

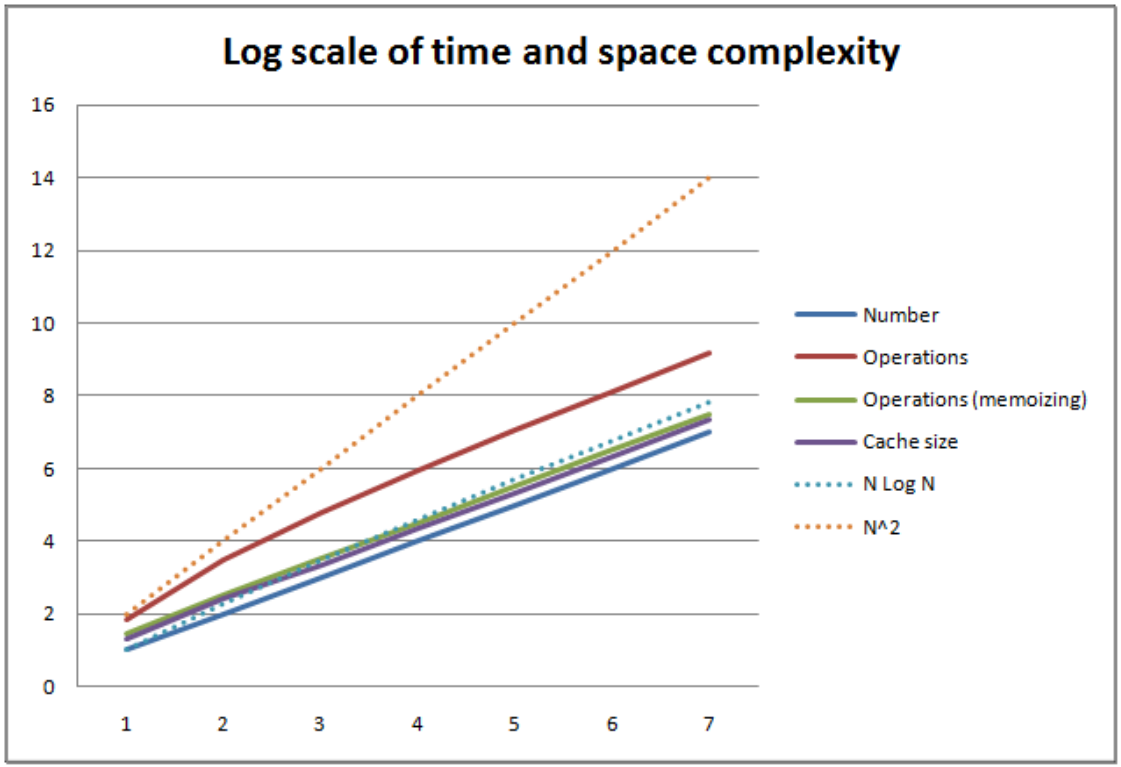
Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Định nghĩa độ phức tạp thuật toán

Đối với **một thuật toán** có **2 độ phức tạp** quan trọng cần chú ý:

- **Độ phức tạp về thời gian** (thời gian thuật toán chạy).
- **Độ phức tạp về không gian** (dung lượng bộ nhớ sử dụng).



Độ phức tạp thời gian

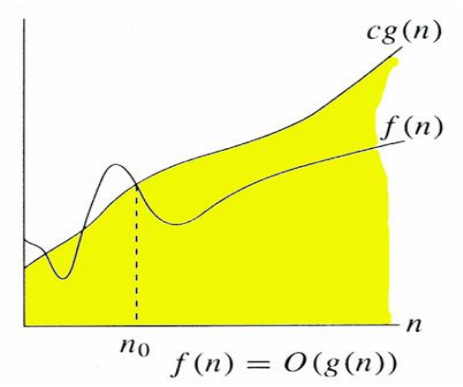
Độ phức tạp thời gian (time complexity): là một khái niệm liên quan đến tốc độ nhanh chậm của một thuật toán khi nó thực thi.

- Kỹ năng lập trình.
- Chương trình dịch mã nguồn.
- Tốc độ xử lý của bộ vi xử lý.
- Bộ dữ liệu đầu vào.

Phân tích độ phức tạp thời gian

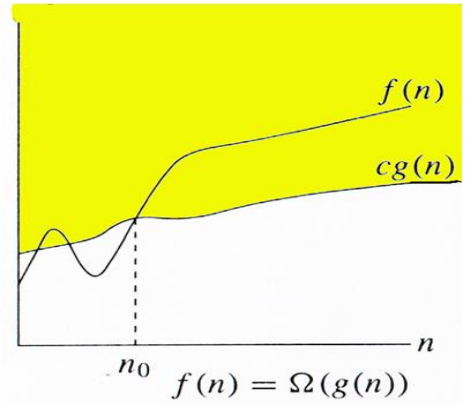
Để ước lượng độ phức tạp thuật toán người ta dùng một số khái niệm sau:

Cận trên (Upper bound)



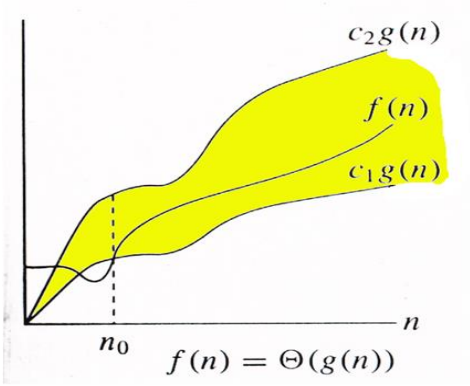
Big O Notation (Big-Oh)

Cận dưới (Lower bound)



Big Ω Notation (Big-Omega)

Cận chặt (Tight bound)

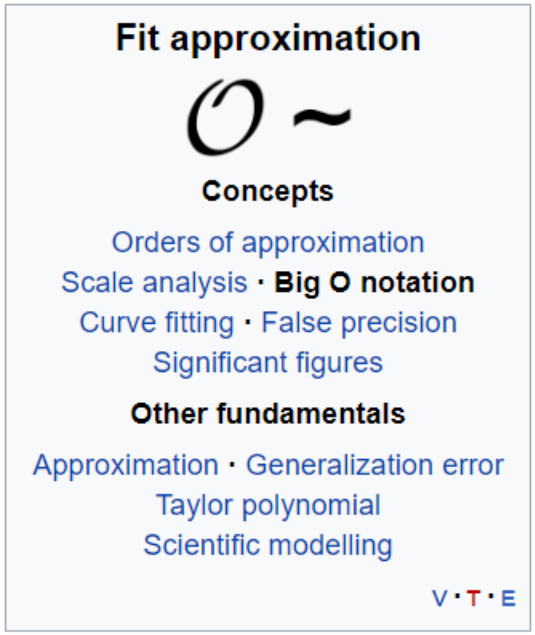


Big Θ Notation (Big-Theta)

Phân tích độ phức tạp thời gian

Trong Big-O có 3 trường hợp quan trọng cần xét:

- **Worst-case performance:** trường hợp xấu nhất.
- **Best-case performance:** trường hợp tốt nhất.
- **Average performance:** trường hợp trung bình.



Phân tích độ phức tạp thời gian

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Hàm đánh giá độ phức tạp thời gian (1)

Constant Time Complexity: O(1)

- Độ phức tạp hằng số là độ phức tạp số phép tính không phụ thuộc vào dữ liệu đầu vào. Thuật toán hữu hạn các thao tác thực hiện 1 lần hoặc vài lần.
- Ví dụ: Tính tổng kết quả của x và y

C++

```cpp
int x = 15 + (10 * 30);
int y = 59 - x;
cout << x + y;
```

Python

```python
x = 15 + (10 * 30)
y = 59 - x
print(x + y)
```

Java

```java
int x = 15 + (10 * 30);
int y = 59 - x;
System.out.println(x + y);
```

Hàm đánh giá độ phức tạp thời gian (2)

Logarithmic Time Complexity: O(logn)

- Độ phức tạp logarit là độ phức tạp có thời gian thực hiện tăng theo kích thước dữ liệu vào với tốc độ hàm logarit.
- Ví dụ 1: Đếm biến count sẽ có giá trị bao nhiêu?

C++

```cpp
for (int i = 1; i < n; i *= c) {
    count++;
}
```

Python

```python
i = 1
while (i < n):
    count += 1
    i *= c
```

Java

```java
int i = 1;
while (i < n) {
    count += 1;
    i *= c;
}
```

Hàm đánh giá độ phức tạp thời gian (2)

- Ví dụ 2: Đếm biến count có giá trị bao nhiêu?

C++

```
for (int i = n; i >= 1; i /= c) {
    count++;
}
```

Python

```
i = n
while (i >= 1):
    count += 1
    i //= c
```

Java

```
for (int i = n; i >= 1; i /= c) {
    count++;
}
```

Hàm đánh giá độ phức tạp thời gian (3)

Linear Time Complexity: O(n)

- Độ phức tạp tuyến tính là độ phức tạp số phép tính phục thuộc vào dữ liệu đầu vào, với vòng lặp tăng/giảm tuần tự theo 1 biến.
- Ví dụ: Tính tổng các phần tử của mảng a.

C++

```
int a[100];
int sum = 0;
for (int i = 1; i <= n; i++) {
    sum += a[i];
}
```

Python

```
a = [0]*100
sum = 0
for i in range(1, n+1):
    sum += a[i]
```

Java

```
int a[] = new int[100];
int sum = 0;
for (int i = 1; i <= n; i++)
    sum += a[i];
```

Hàm đánh giá độ phức tạp thời gian (4)

Log-Linear Time Complexity: O(nlogn)

- Độ phức tạp tuyến tính logarit là độ phức tạp được tính bằng việc chia bài toán lớn thành các bài toán nhỏ hơn, giải một cách độc lập rồi hợp lại để nhận được kết quả của bài toán lớn.
- Ví dụ: Tính giá trị của x và y.

C++

```
int x = n;
while(x > 0) {
    y = n;
    while(y > 0) {
        y = y - 1;
    }
    x = x / 2;
}
```

Python

```
x = n
while x > 0:
    y = n
    while y > 0:
        y = y - 1
    x = x // 2
```

Java

```
int x = n;
while (x > 0) {
    int y = n;
    while (y > 0)
        y -= 1;
    x /= 2;
}
```

Hàm đánh giá độ phức tạp thời gian (5)

Polynomial Time Complexity: $O(n^c)$

- Độ phức tạp đa thức (với c là hằng số) là độ phức tạp với các thao tác được thực hiện với trong các vòng lặp lồng nhau.
- Ví dụ: Tính tổng.

C++

```
int sum = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++) {
        sum += i + j;
    }
```

Python

```
sum = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        sum += i + j
```

Java

```
int sum = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        sum += i + j;
```

Hàm đánh giá độ phức tạp thời gian (6)

Exponential Time Complexity: $O(c^n)$

- Độ phức tạp hàm mũ là lớp thuật toán có độ phức tạp lớn. Khi *n* đủ lớn, có thể xem như bài toán không giải được theo nghĩa là không nhận được lời giải trong một thời gian hữu hạn.
- Ví dụ: Bài toán tìm số Fibonacci thứ n.

$$F(n) := \begin{cases} 1, khi\ n = 1 \\ 1, khi\ n = 2 \\ F(n-1) + F(n-2), khi\ n > 2 \end{cases}$$

F(0)	F(1)	F(1)	F(2)	F(3)	F(4)	F(5)	...	F(n)
0	1	1	2	3	5	8	...	?

Hàm đánh giá độ phức tạp thời gian (6)

C++

```cpp
int F(int n) {
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return F(n-1) + F(n-2);
}
```

Python

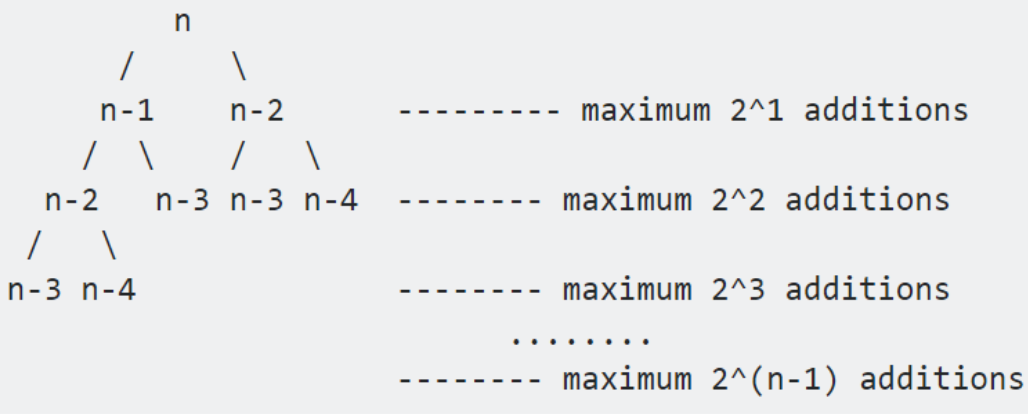
```python
def F(n):
    if n == 0:
        return 0
    else:
        if n == 1:
            return 1
        else:
            return F(n-1) + F(n-2)
```

Java

```java
public static int F(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return F(n - 1) + F(n - 2);
}
```

Hàm đánh giá độ phức tạp thời gian (6)

Để tính độ phức tạp F(n) ta có.

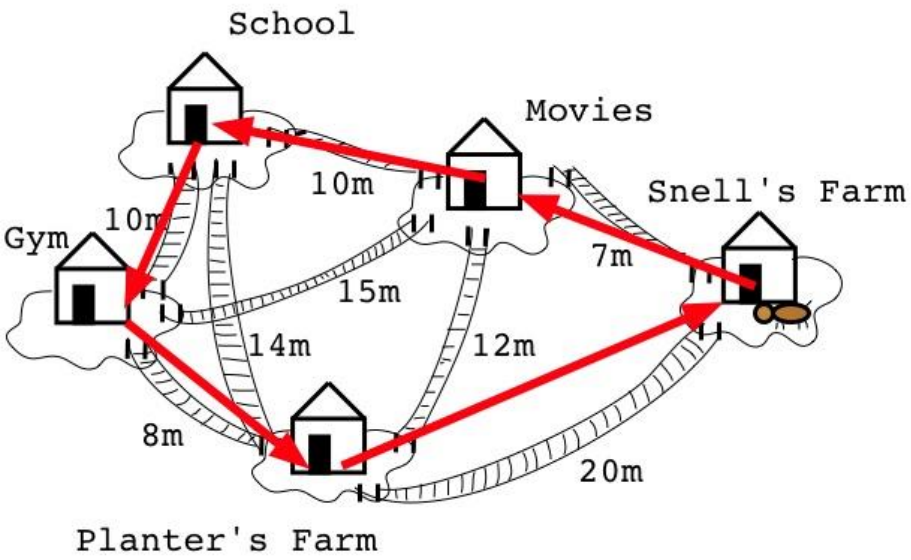


Cây này tiếp tục phát triển theo cấp số nhân khi chúng ta tăng N. Do đó Độ phức tạp ~ $O(2^n)$.

Hàm đánh giá độ phức tạp thời gian (7)

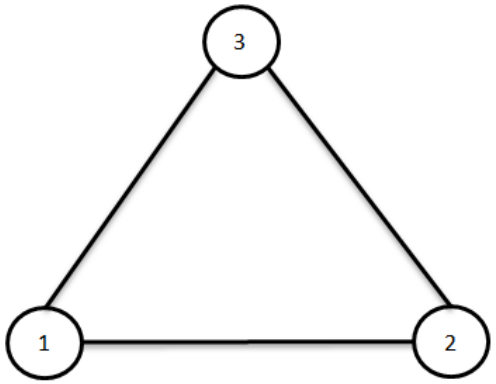
Factorial Time Complexity: O(n!)

- Độ phức tạp giai thừa cũng tương tự như độ phức tạp hàm mũ, đây là lớp thuật toán có độ phức tạp lớn, thường gặp trong các bày toán quay lui, vét cạn.
- Ví dụ: Bài toán TSP (Traveling salesman problem) là ví dụ rõ nét nhất cho độ phức tạp này.



Hàm đánh giá độ phức tạp thời gian (7)

Giả sử chúng ta có 3 thành phố người này cần đi qua. Vậy tổng cộng sẽ có 3! trường hợp có thể xảy ra:



1 → 2 → 3

1 → 3 → 2

2 → 1 → 3

2 → 3 → 1

3 → 1 → 2

3 → 2 → 1

Simulated annealing

Tabu search

Ant colony

Genetic algorithm

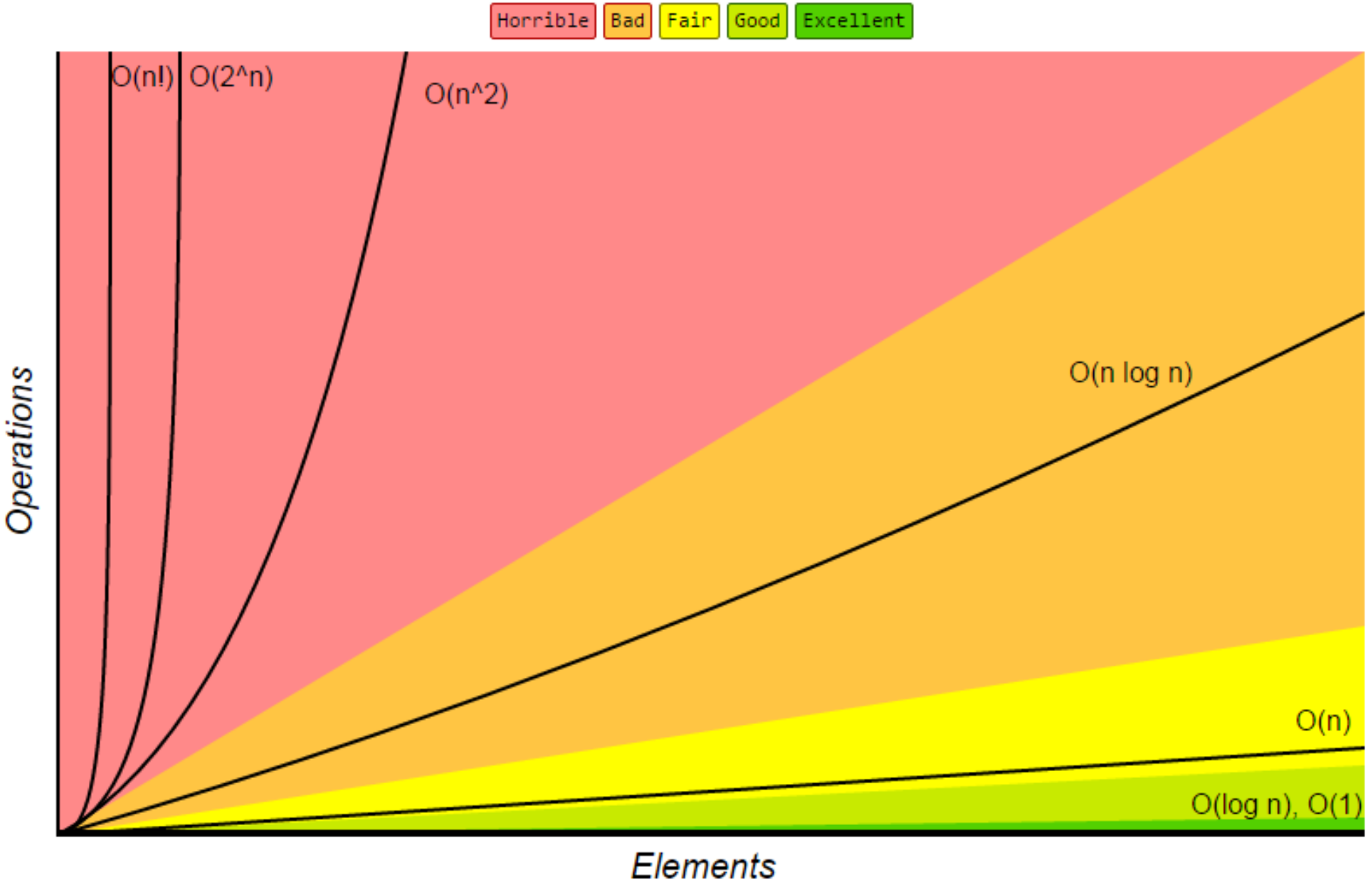
Harmony search

MỘT SỐ ĐỘ PHỨC TẠP THỜI GIAN KHÁC

Thứ tự các độ phức tạp thời gian

Function	Common name
$n!$	factorial
2^n	exponential
n^d, $d > 3$	polynomial
n^3	cubic
n^2	quadratic
$n\sqrt{n}$	
$n \log n$	quasi-linear
n	linear
$\sqrt{n}$	root - n
$\log n$	logarithmic
1	constant

Big-O Complexity Chart



Tính toán độ phức tạp thực tế trên thực nghiệm

Consider an Intel Core i7 Extreme Edition 980X (Hex core), 3.33GHz, top speed $< 150 \cdot 10^9$ instructions per second (IPS). For simplicity, say it's 10^9 IPS.

	10	20	30	40	50	1000
$\lg \lg n$	1.7 ns	2.17 ns	2.29 ns	2.4 ns	2.49 ns	3.3 ns
$\lg n$	3.3 ns	4.3 ns	4.9 ns	5.3 ns	5.6 ns	9.9 ns
n	10 ns	20 ns	3 ns	4 ns	5 ns	1 μs
n^2	0.1 μs	0.4 μs	0.9 μs	1.6 μs	2.5 μs	1 ms
n^3	1 μs	8 μs	27 μs	64 μs	125 μs	1 sec
n^5	0.1 ms	3.2 ms	24.3 ms	0.1 sec	0.3 sec	277 h
2^n	1 μs	1 ms	1 s	18.3 m	312 h	$3.4 \cdot 10^{282}$ Cent.
3^n	59 μs	3.5 s	57.2 h	386 y	227644 c	$4.2 \cdot 10^{458}$ Cent.

1.6^{100} ns is approx. 82 centuries (Recall `fib1()`).

$$\lg 10^{10} = 33, \quad \lg \lg 10^{10} = 4.9$$

Tính toán thực tế độ phức tạp Thuật toán

Trung bình các máy chấm hiện nay 10^8 phép tính / giây.

Length of Input (N)	Worst Accepted Algorithm
$\leq [10..11]$	$O(N!), O(N^6)$
$\leq [15..18]$	$O(2^N * N^2)$
$\leq [18..22]$	$O(2^N * N)$
≤ 100	$O(N^4)$
≤ 400	$O(N^3)$
$\leq 2K$	$O(N^2 * logN)$
$\leq 10K$	$O(N^2)$
$\leq 1M$	$O(N * logN)$
$\leq 100M$	$O(N), O(logN), O(1)$

* Of course, these limits are not precise. They are just approximations, and will vary depending on the specific task.

Akash Sharma– hackerrank

Quy tắc đánh giá độ phức tạp Thuật toán

Quy tắc hằng (Multiplicative Constants):

- O(k * f(n)) = O(f(n)).
- Ví dụ: O(1000n) = O(n)

Quy tắc cộng (Addition Rule):

- O(f(n) + g(n)) = **max**(f(n), g(n))
- Ví dụ: $O(n^2 + 3n + 2) = O(n^2)$

Quy tắc nhân (Multiplication Rule):

- O(f(n) * g(n)) = O(f(n)) * O(g(n))
- Ví dụ: $O(n^2) * O(\log n) = O(n^2 \log n)$

Độ phức tạp không gian

Độ phức tạp không gian (space complexity): là dung lượng bộ nhớ ước tính phát sinh khi thực hiện thuật toán.

- Kỹ năng lập trình.
- Các biến cần lưu thực hiện chương trình.
- Các cấu trúc dữ liệu cần lưu khi thực hiện chương trình.
- Thuật toán.

Hàm đánh giá độ phức tạp không gian (1)

Constant Space Complexity: O(1)

- Độ phức tạp hằng số là độ phức tạp số phép tính không phụ thuộc vào dữ liệu đầu vào. Chỉ thao tác trên 1 biến hoặc 1 vài biến.
- Ví dụ:

C++

```
s = 0;
for(int i=1; i <= n; i++){
    s += i;
}
```

Python

```
s = 0
for i in range(1, n+1):
    s += i
```

Java

```
int s = 0;
for (int i= 1; i<= n; i++)
    s += i;
```

Hàm đánh giá độ phức tạp không gian (2)

Linear Space Complexity: O(n)

- Độ phức tạp không gian tuyến tính là độ phức tạp thao tác trên biến của mảng có n phần tử.
- Ví dụ:

C++

```cpp
int sum = 0; a[0] = 1;
for (int i = 1; i <= n; ++i) {
    a[i] = a[i - 1] * 2;
    sum += a[i];
}
```

Python

```python
sum = 0
a = [1]
for i in range(1, n+1):
    a.append(a[-1]*2)
    sum += a[i]
```

Java

```java
int sum = 0;
ArrayList <Integer> a = new ArrayList<Integer>();
a.add(1);
for (int i = 1; i <= n; i++) {
        a.add(a.get(a.size() - 1) * 2);
        sum += a.get(i);
}
```

MỘT SỐ BÀI TẬP ỨNG DỤNG VỀ ĐỘ PHỨC TẠP

Bài tập 1

Tìm độ phức tạp thời gian và không gian của bài toán sau:

C++

```cpp
int a = 0, b = 0;
for (int i = 0; i < N; i++) {
    a = a + rand();
}
for (int j = 0; j < M; j++) {
    b = b + rand();
}
```

Python

```python
a = b = 0
for i in range(N):
    a += random.randint(1, 1000)
for j in range(M):
    b += random.randint(1, 1000)
```

Java

```java
int a = 0, b = 0;
Random rn = new Random();
for (int i = 0; i < N; i++)
    a += rn.nextInt(1000);
for (int j = 0; j < M; j++)
    b += rn.nextInt(1000);
```

O(N + M) time, O(1) space.

Bài tập 2

Tìm độ phức tạp thời gian và không gian:

C++

```cpp
int a = 0, b = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        a = a + j;
for (int k = 0; k < N; k++)
    b = b + k;
```

Python

```python
a = b = 0
for i in range(N):
    for j in range(N):
        a += j
for k in range(N):
    b += k
```

Java

```java
int a = 0, b = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        a += j;
for (int k = 0; k < N; k++)
    b += k;
```

O(N * N) time, O(1) space.

Bài tập 3

Tìm độ phức tạp thời gian của đoạn code sau:

C++

```cpp
int count = 0;
for (int i = N; i > 0; i /= 2) {
    for (int j = 0; j < i; j++) {
        count += 1;
    }
}
```

Python

```python
count = 0
i = N
while i > 0:
    for j in range(i):
        count += 1
    i //= 2
```

Java

```java
int count = 0;
int i = N;
while (i > 0) {
    for (int j = 0; j < i; j++)
        count += 1;
    i /= 2;
}
```

O(N)

Bài tập 4

Tìm độ phức tạp thời gian của đoạn code sau:

C++

```cpp
int i, j, k = 0;
for (i  = n/2; i <= n; i++)
    for (j = 2; j <= n; j = j * 2)
        k = k + n/2;
```

Python

```python
k = 0
for i in range(n//2,n+1):
    j = 2
    while j <= n:
        k = k + n/2
        j *= 2
```

Java

```java
int k = 0;
for (int i = n/2; i < n + 1; i++) {
    int j = 2;
    while (j <= n) {
        k = k + n/2;
        j *= 2;
    }
}
```

O(NLogN)

Bài tập 5

Tìm độ phức tạp không gian của đoạn code sau:

C++

```
double foo(int n) {
    int i;
    double sum;
    if (n == 0) return 1.0;
    else {
        sum = 0.0;
        for (i = 0; i < n; i++)
            sum += foo (i);
        return sum;
    }
}
```

Python

```
def foo(n):
    if n == 0:
        return 1.0
    else:
        sum = 0
        for i in range(n):
            sum += foo(i)
        return sum
```

O(n)

Bài tập 5

Tìm độ phức tạp không gian của đoạn code sau:

Java

```
public static double foo(int n) {
    if (n == 0)
        return 1.0;
    else {
        double sum = 0.0;
        for (int i = 0; i < n; i++)
            sum += foo(i);
        return sum;
    }
}
```

O(n)

Bài tập 6

Vòng lặp nào kết thúc nhanh nhất?

C++

```
A) for(i = 0; i < n; i++)

B) for(i = 0; i < n; i += 2)

C) for(i = 1; i < n; i *= 2)

D) for(i = n; i > -1; i /= 2)
```

Python

```
A) for i in range(n):

B) for i in range(0,n,2):

C) i = 1
   while i < n:
        i *= 2

D) i = n
   while i > -1:
        i //= 2
```

C

Bài tập 6

Vòng lặp nào kết thúc nhanh nhất?

Java

```
A) for (int i = 0; i < n; i++)

B) for (int i = 0; i < n; i += 2)

C) int i = 1;
   while (i < n)
       i *= 2;

D) int i = n;
   while (i > -1)
       i /= 2;
```

C

Bài tập 7

Sắp xếp độ phức tạp tăng dần?

C++ / Python / Java

```
f1(n) = 2^n
f2(n) = n^(3/2)
f3(n) = nLogn
f4(n) = n^(Logn)
```

f3, f2, f4, f1

Hỏi đáp