LECTURE 02

ALGORITHMIC COMPLEXITY

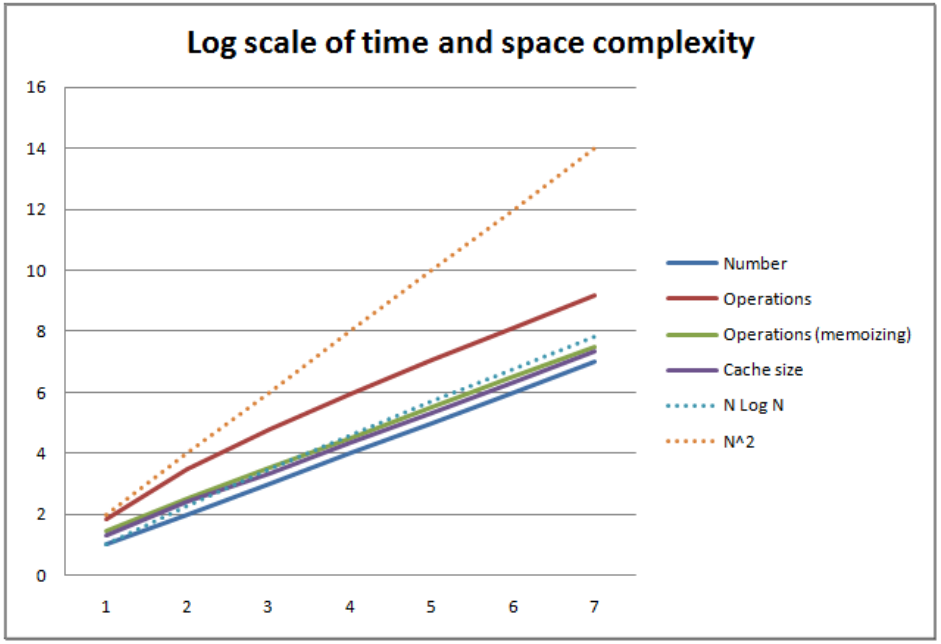
Big-O Coding

Website: www.bigocoding.com

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).



**** Time and space complexity depends on lots of things like hardware, operating system, processors, etc.*

Độ phức tạp thời gian

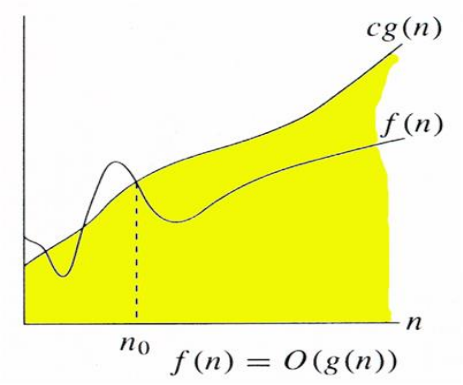
Độ phức tạp thời gian (Time complexity): một khái niệm liên quan đến tốc độ thực thi của một thuật toán.

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

Khái niệm trong phân tích độ phức tạp

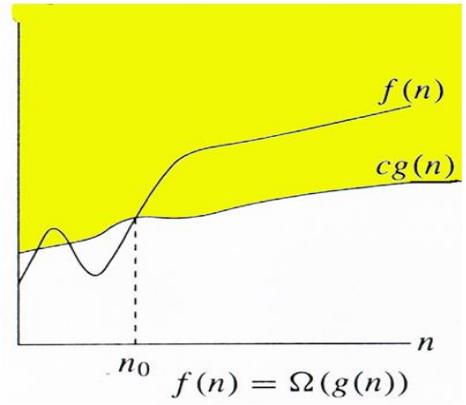
Để ước lượng độ phức tạp thuật toán người ta dùng một số khái niệm sau:

Cận trên (Upper bound)



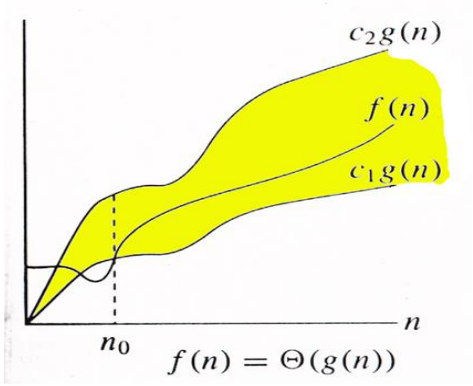
Big-O Notation (Big-Oh)

Cận dưới (Lower bound)



Big-Ω Notation (Big-Omega)

Cận chặt (Tight bound)

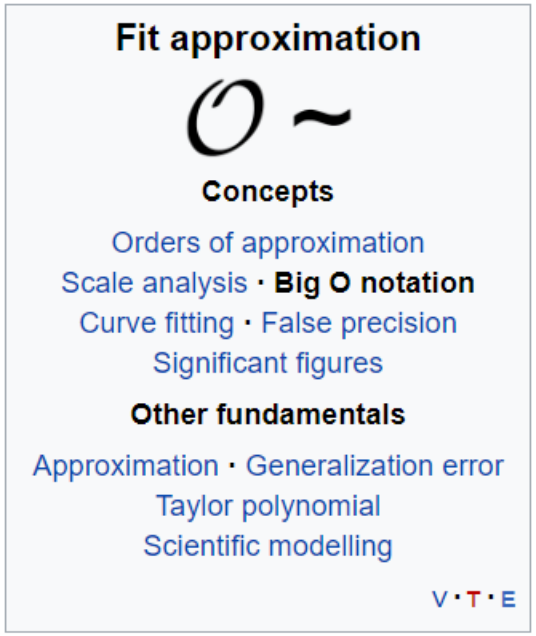


Big-Θ Notation (Big-Theta)

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.



Nguồn: https://en.wikipedia.org/wiki/Big_O_notation

Độ phức tạp của một số thuật toán sắp xếp

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)

Nguồn: http://bigocheatsheet.com/

Hàm đánh giá độ phức tạp thời gian (1)

Constant Time Complexity: O(1)

- Độ phức tạp hằng số là độ phức tạp mà 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 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(log(n))

- Độ phức tạp logarit là độ phức tạp có thời gian thực hiện tăng theo kích thước dữ liệu đầu vào với tốc độ hàm logarit.

Ví dụ 1: Biến đếm count sẽ có giá trị bao nhiêu?

C++

```cpp
int count = 0, c = 2;
for (int i = 1; i < n; i *= c)
{
    count += 1;
}
```

Python

```python
count = 0
c = 2
i = 1
while i < n:
    count += 1
    i *= c
```

Java

```java
int i = 1, count = 0, c = 2;
while (i < n) {
    count += 1;
    i *= c;
}
```

Hàm đánh giá độ phức tạp thời gian (2)

Ví dụ 2: Biến đếm count có giá trị bao nhiêu?

C++

```cpp
int count = 0
int c = 2;
for (int i = n; i >= 1; i /= c)
{
    count++;
}
```

Python

```python
count = 0
c = 2
i = n
while i >= 1:
    count += 1
    i //= c
```

Java

```java
int count = 0;
int c = 2;
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 phụ thuộc vào dữ liệu đầu vào, với vòng lặp tăng/giảm một cách tuần tự.

Ví dụ: Tính tổng các phần tử của mảng a.

C++

```
int sumArray(int a[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return sum;
}
```

Python

```
def sumArray(a, n):
    sum = 0
    for i in range(n):
        sum += a[i]
    return sum
```

Java

```
static int sumArray(int a[], int n) {
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += a[i];
    return sum;
}
```

Hàm đánh giá độ phức tạp thời gian (4)

Log-Linear Time Complexity: O(nlog(n))

- Độ phức tạp tuyến tính logarit là độ phức tạp thường xuất hiện trong các bài toán lớn được giải bằng cách kết hợp kết quả của nhiều bài toán nhỏ hơn được giải độc lập.

Ví dụ: Tính giá trị của biến count.

C++

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

Python

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

Java

```
int x = n;
int count = 0;
while (x > 0) {
    int y = n;
    while (y > 0) {
        y = y - 1;
        count += 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 trong các vòng lặp lồng nhau.

Ví dụ: Tính tổng các phần tử trong 2 mảng a và b.

C++

```
int sum = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        sum += a[i] + b[j];
```

Python

```
sum = 0
for i in range(n):
    for j in range(n):
        sum += a[i] + b[j]
```

Java

```
int sum = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        sum += a[i] + b[j];
```

Hàm đánh giá độ phức tạp thời gian (6)

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

- Độ phức tạp hàm mũ là độ phức tạp số phép tính phụ thuộc vào hàm mũ n của dữ liệu đầu vào, đây là độ phức tạp rất 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.

$$F(n) := \begin{cases} 0, khi\ n = 0 \\ 1, khi\ n = 1 \\ F(n-1) + F(n-2), khi\ n > 1 \end{cases}$$

F(0)	F(1)	F(2)	F(3)	F(4)	F(5)	F(6)	...	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

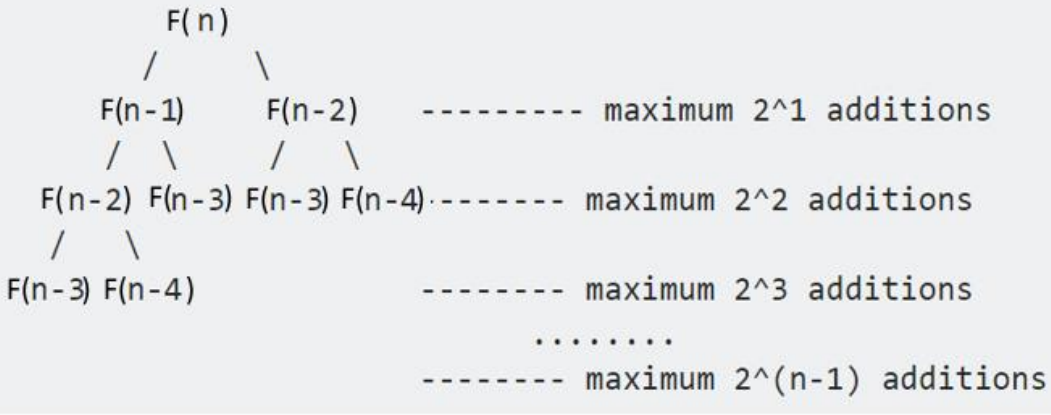
```python
def F(n):
    if n == 0:
        return 0
    else:
        if n == 1:
            return 1
        else:
            return F(n-1) + F(n-2)
```

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)

Minh họa sự phát triển của cây F(n).



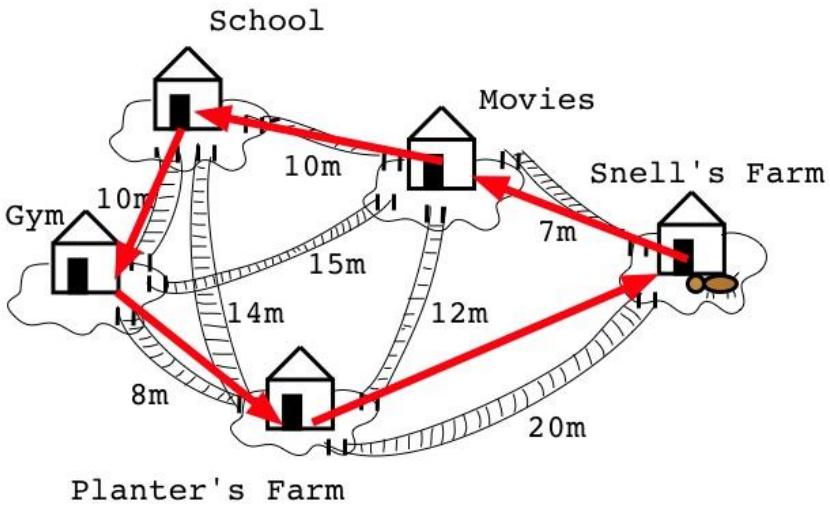
Cây này tiếp tục phát triển theo cấp số nhân khi chúng ta tăng n. Do đó Độ phức tạp ~ $O(2^n)$.

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 bài toán quay lui, vét cạn.

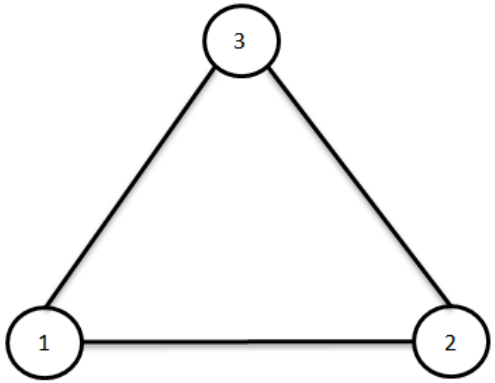
Ví dụ: Bài toán Traveling Salesman Problem (người giao hàng)



- Snell's Farm → Movies → School → Gym → Planter's Farm → Snell's Farm = 55m.
- Snell's Farm → Movies → Gym → Planter's Farm → Snell's Farm = 50.
- Gym → Movies → Snell's Farm → Planter's Farm → School → Gym = 66m.
- …

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

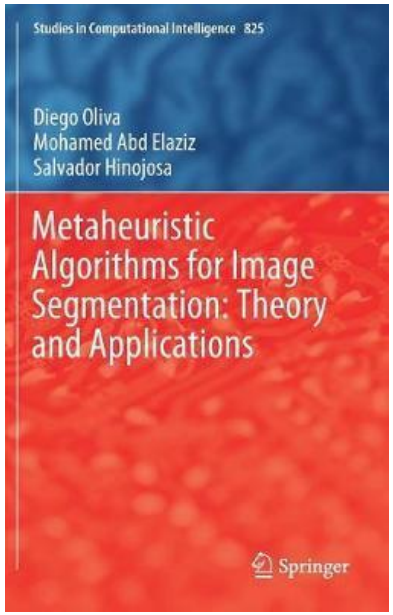
2 → 3 → 1

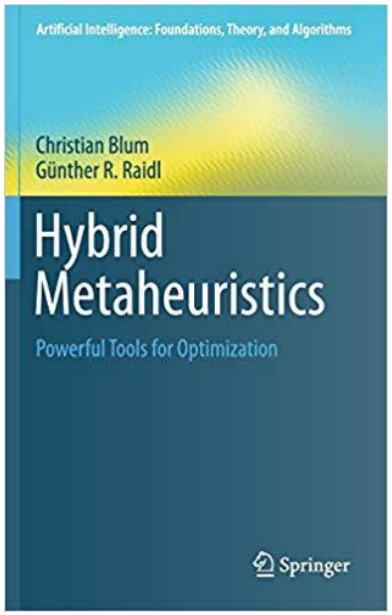
3 → 1 → 2

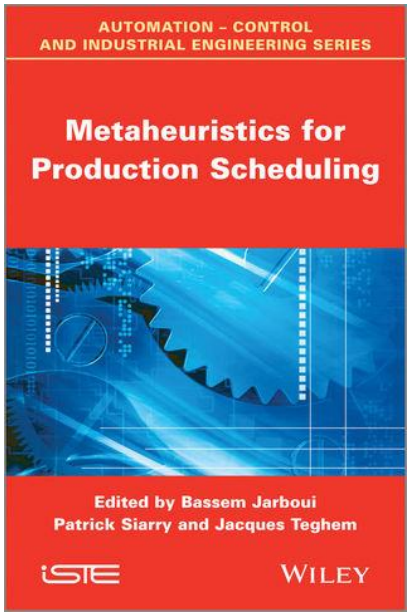
3 → 2 → 1

- N ngôi nhà → khoảng $N!$ hành trình khác nhau.
- 100 ngôi nhà, kiểm tra 1 hành trình/giây.
- → $100!$ hành trình $= 2.96 \times 10^{148}$ **thế kỷ.**

Các thuật toán Metaheuristic







Simulated annealing

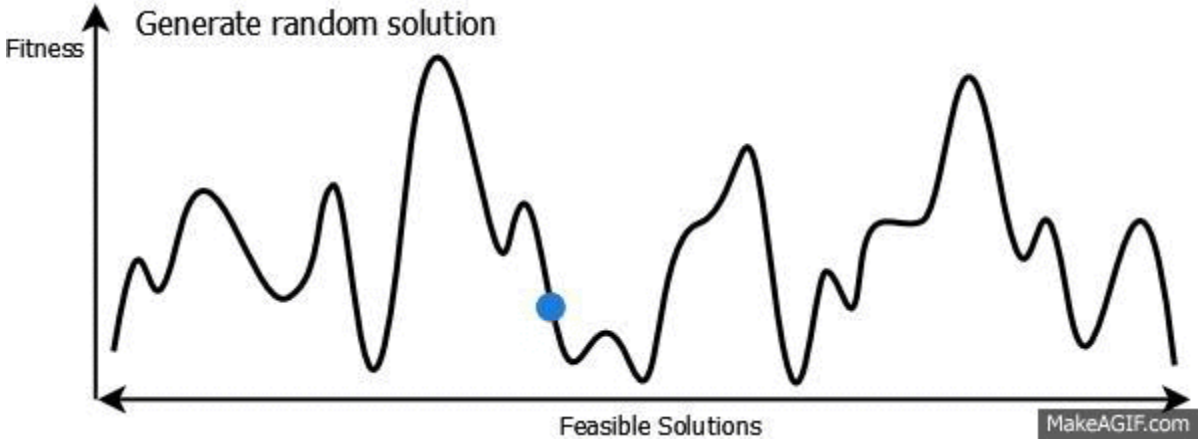
Tabu search

Ant colony

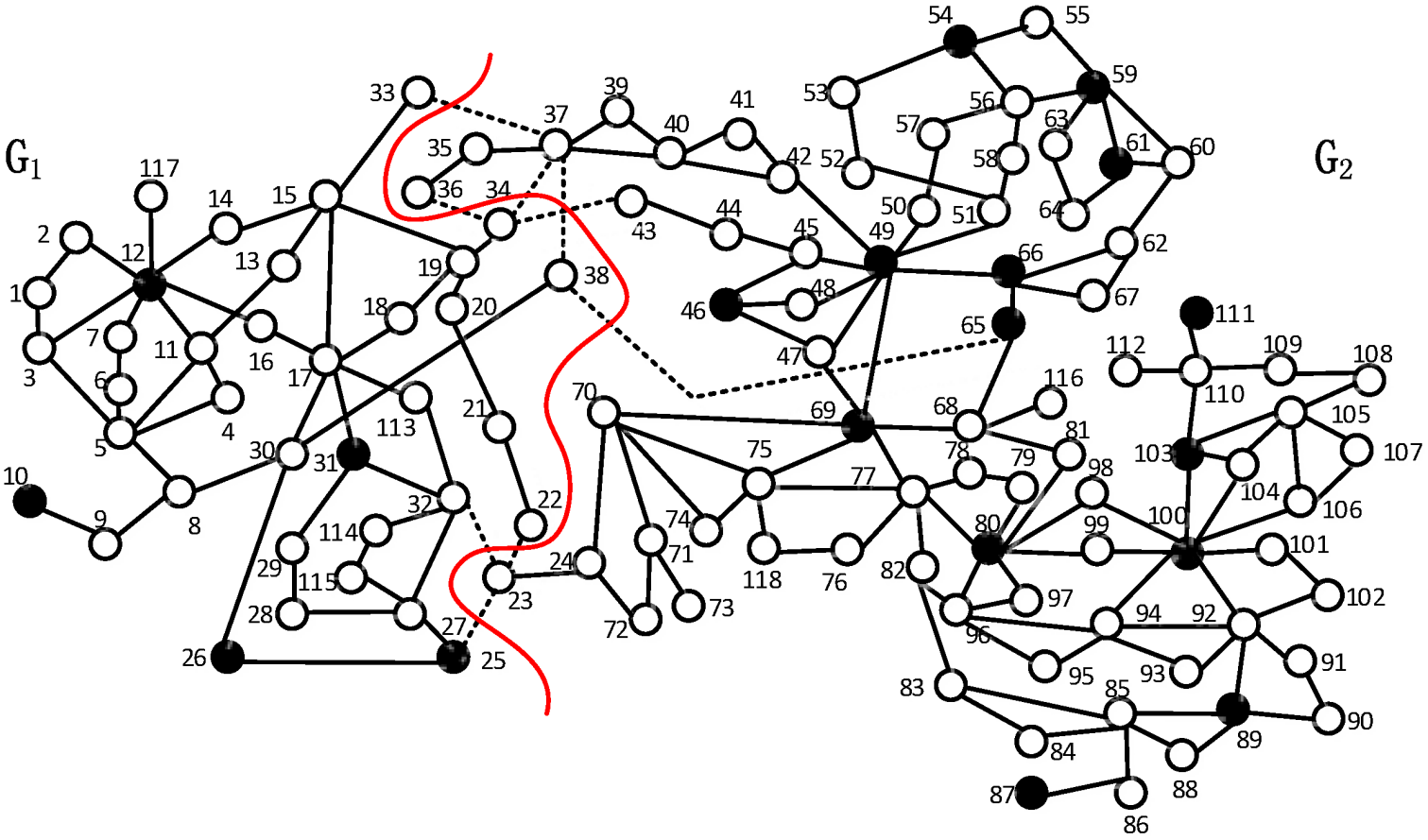
Genetic algorithm

Harmony search

Simulated Annealing

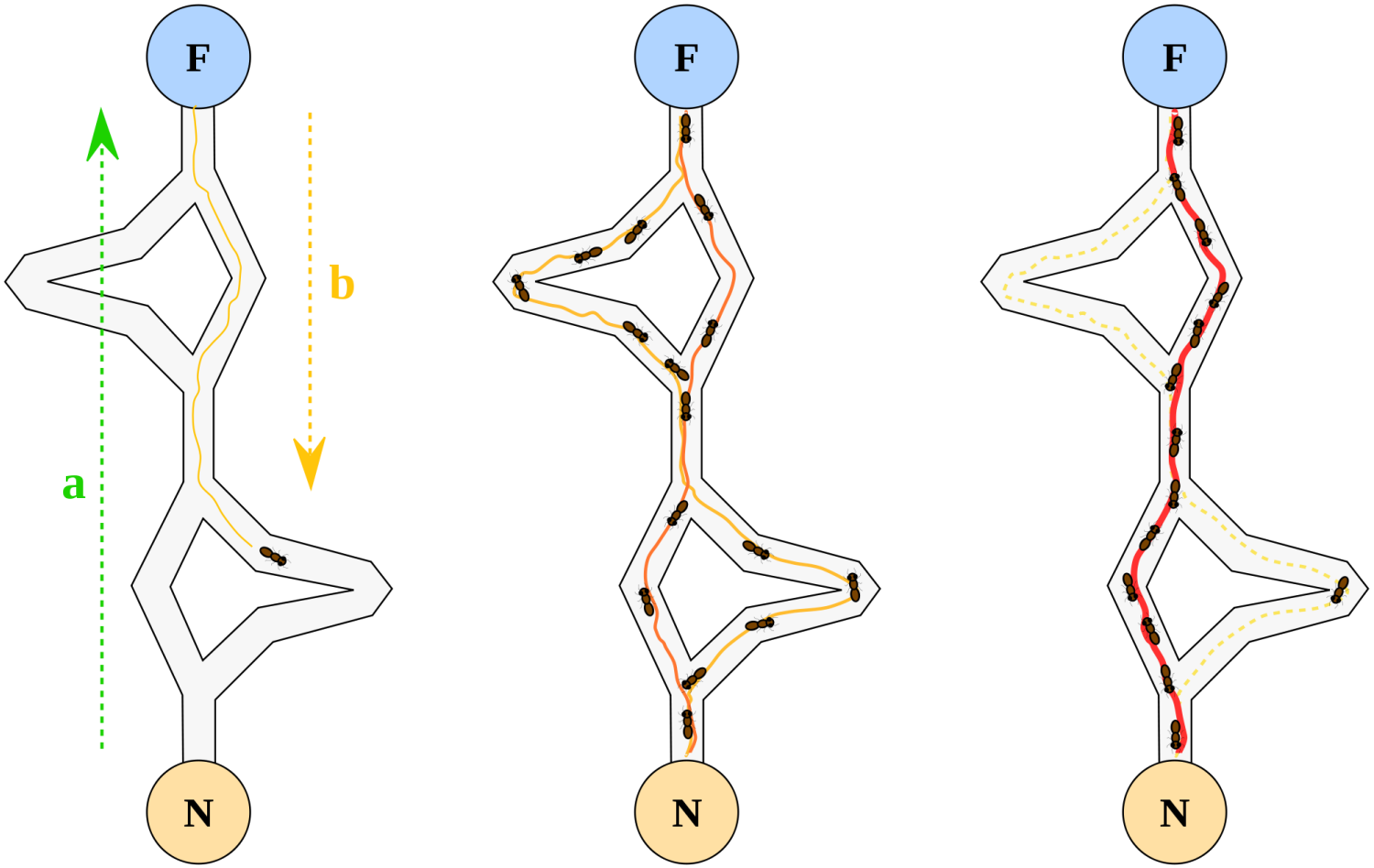


Tabu search



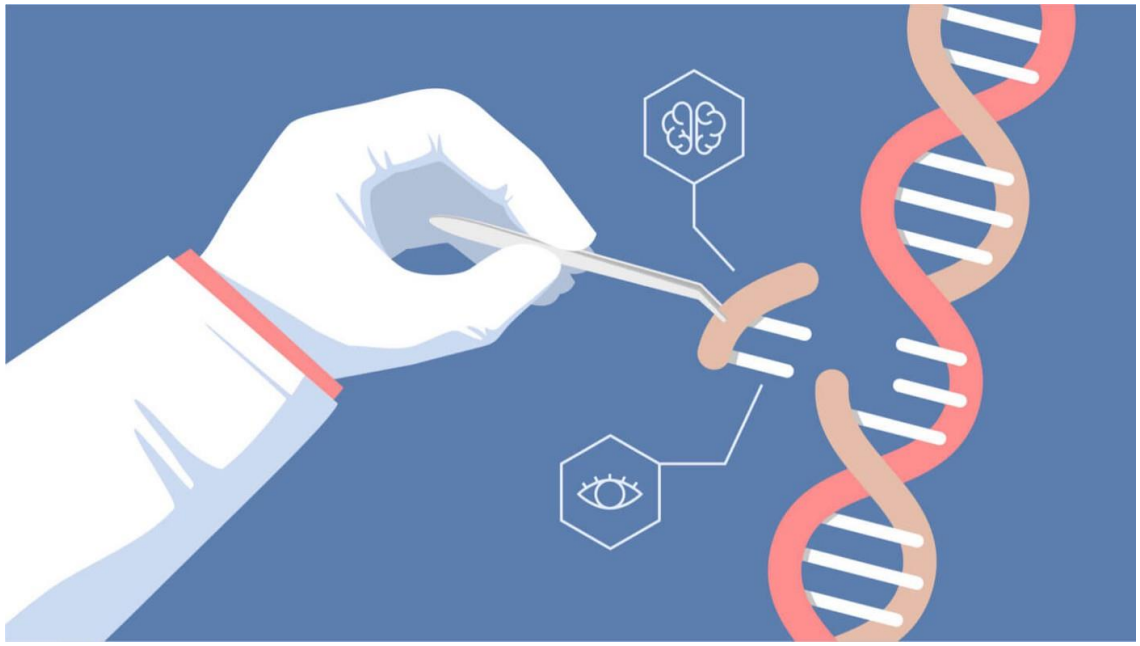
The tripped edges are: 22-23, 23-25, 23-32, 33-37, 34-36, 34-37, 34-43, 37-38, 38-65

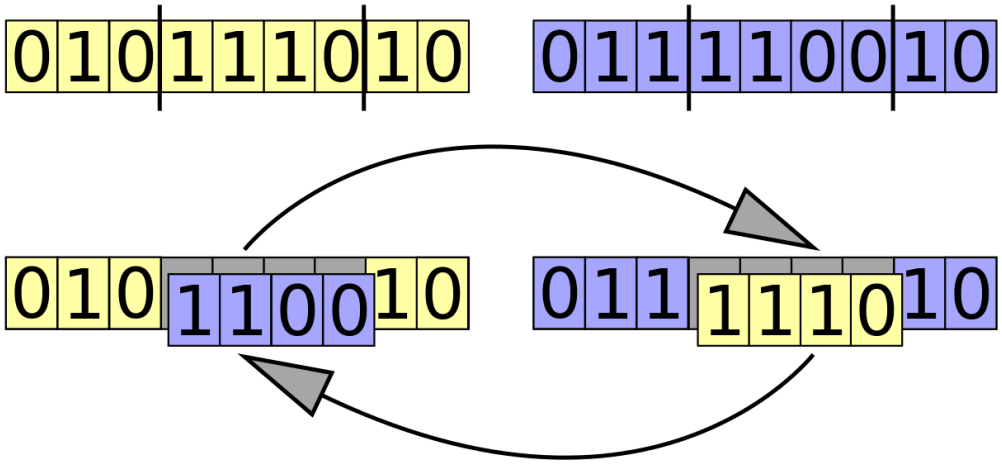
Ant colony



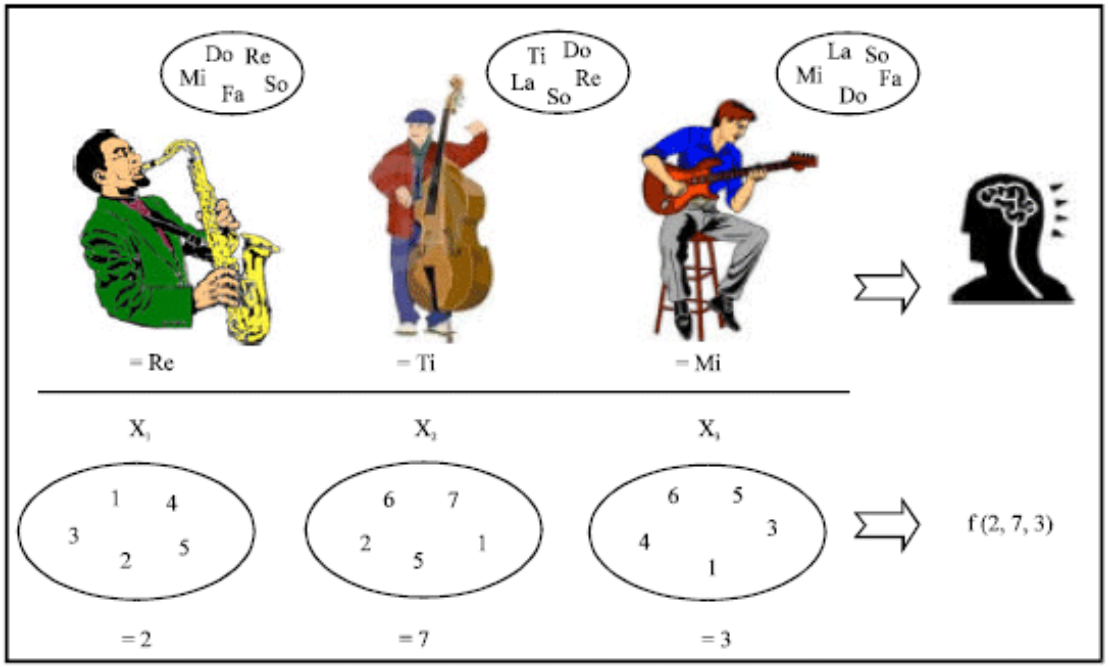
1 2 3

Genetic algorithm





Harmony search

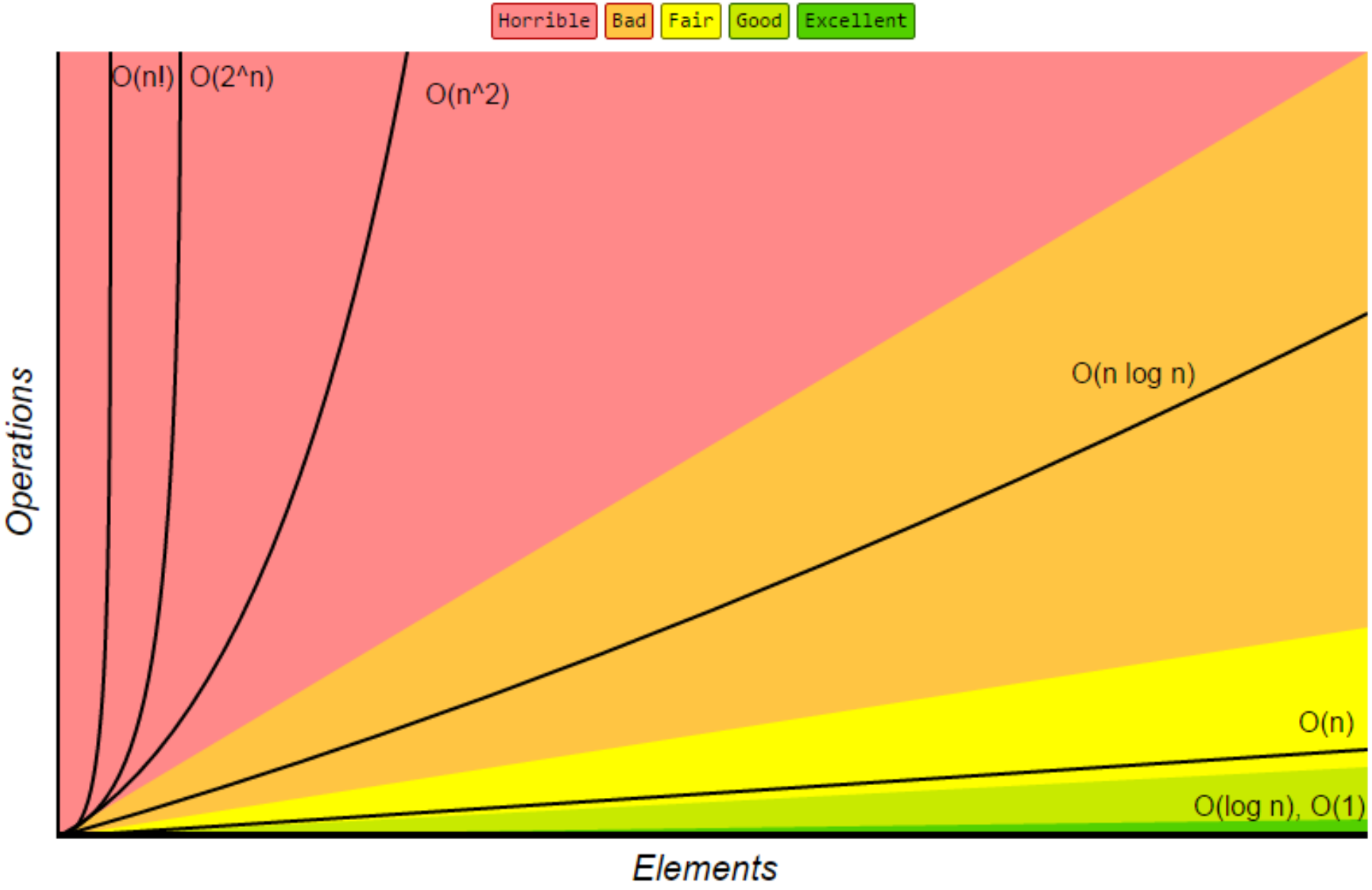


MỘT SỐ ĐỘ PHỨC TẠP THỜI GIAN KHÁC

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 thực tế độ phức tạp Thuật toán

n	$O(1)$	$O(\log_2 n)$	$O(n)$	$O(n \log_2 n)$	$O(n^2)$
10^2	1 μsec	1 μsec	1 μsec	1 μsec	1 μsec
10^3	1 μsec	1.5 μsec	10 μsec	15 μsec	100 μsec
10^4	1 μsec	2 μsec	100 μsec	200 μsec	10 msec
10^5	1 μsec	2.5 μsec	1 msec	2.5 msec	1 sec
10^6	1 μsec	3 μsec	10 msec	30 msec	1.7 min
10^7	1 μsec	3.5 μsec	100 msec	350 msec	2.8 hr
10^8	1 μsec	4 μsec	1 sec	4 sec	11.7 d

n	$O(n^2)$	$O(2^n)$
100	1 μsec	1 μsec
110	1.2 μsec	1 msec
120	1.4 μsec	1 sec
130	1.7 μsec	18 min
140	2.0 μsec	13 d
150	2.3 μsec	37 yr
160	2.6 μsec	37,000 yr

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)) = O(**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 thi 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 một vài biến.

Ví dụ: Tính tổng từ 1 đến n

C++

```
int 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: Tính tổng giá trị phần tử của mảng theo công thức cho trước

C++

```cpp
int sum = 0;
vector<int> a; 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 PHÂN TÍCH VỀ ĐỘ PHỨC TẠP THUẬT TOÁN

Bài tập 1

Tính 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 += rand();
for (int j = 0; j < M; j++)
    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);
```

Bài tập 2

Tính 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 += j;
for (int k = 0; k < N; k++)
    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;
```

Bài tập 3

Tính 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;
}
```

Bài tập 4

Tính độ phức tạp thời gian của đoạn code sau:

C++

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

Bài tập 5

Tính độ phức tạp không gian của đoạn code sau:

C++

```cpp
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

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

Bài tập 5

Tính độ phức tạp không gian của đoạn code sau:

Java

```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;
    }
}
```

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
```

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;
```

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)
```

Hỏi đáp