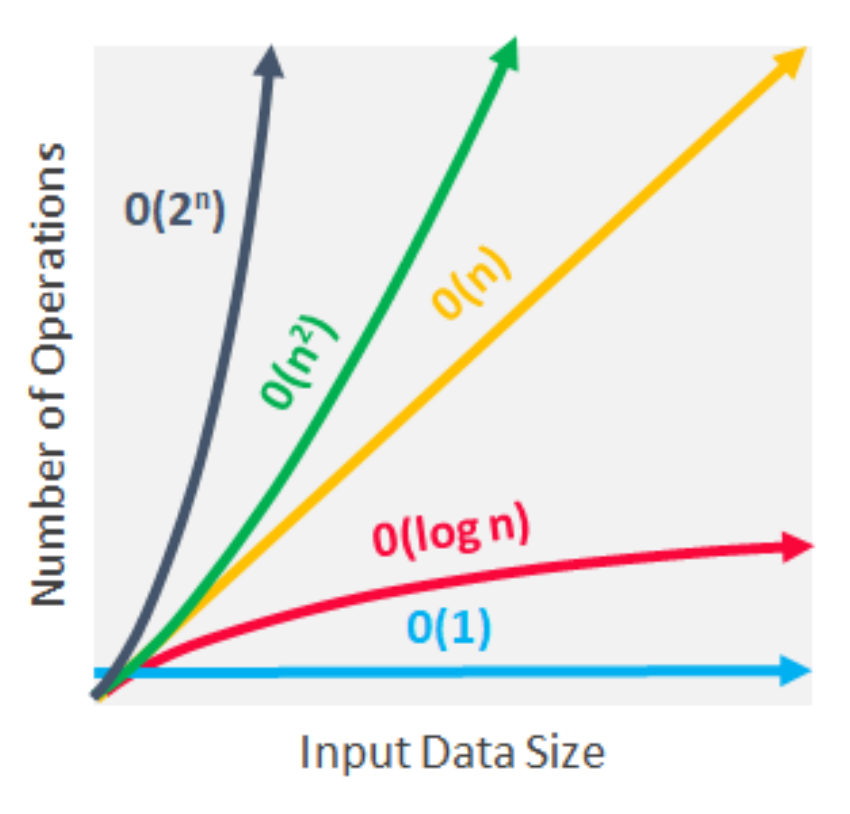


Phân tích độ phức tạp thuật toán

GV: Nguyễn Thanh Sơn

Nhóm 7:

- Nguyễn Trung Tuấn – 19522477
- Nguyễn Khả Tiến – 19522337
- Trịnh Nhật Tân – 19522179

Nội dung

1. Độ phức tạp của thuật toán là gì?
2. Tại sao cần đo độ phức tạp của thuật toán?
3. Cách tính độ phức tạp của thuật toán.
4. Tổng quan về độ phức tạp của các thuật toán.
5. Quiz.
6. Q&A.

Độ phức tạp của thuật toán

- Một thuật toán được xây dựng phải kèm theo những đánh giá mang tính định lượng về nó.
- Để so sánh với các thuật toán liên quan khác.
- Có hai cách tiếp cận, tương ứng với mỗi tiêu chí đánh giá:
  - Độ phức tạp về thời gian.
  - Độ phức tạp về không gian.

Độ phức tạp của thuật toán

Độ phức tạp về thời gian:

- Trực quan nhất để lượng hóa tính hiệu quả của một thuật toán.
- Trong cùng một điều kiện hoạt động, thuật toán nào cho ra kết quả sớm nhất sẽ là tốt nhất.

Độ phức tạp của thuật toán

```
 1 int Fib1(int n){
 2     int a, b, c;
 3     if(n <= 1)
 4         return n;
 5     a = 0; b = 1; c = 0;
 6     for(int k = 2; k <= n; k++) {
 7         c = a + b;
 8         a = b;
 9         b = c;
10     }
11     return c;
12 }
```

```
1 int Fib2(int n){
2     if(n <= 1)
3         return n;
4     return Fib2(n - 1) + Fib2(n - 2);
5 }
```

So sánh hai thuật toán cho dãy Fibonacci

Độ phức tạp của thuật toán

N	Fib1	Fib2
40	41 ns	1048 µs
60	61 ns	1s
80	81 ns	18 phút
100	101 ns	13 ngày
120	121 ns	36 năm
160	161 ns	$3.8 * 10^7$ năm
200	201 ns	$4 * 10^{13}$ năm

So sánh thời gian thực hiện của hai thuật toán cho dãy Fibonacci

Độ phức tạp của thuật toán

Độ phức tạp về không gian:

- Dựa trên mức độ tiêu thụ tài nguyên của hệ thống.
- Dựa vào cấu trúc dữ liệu được sử dụng.
- Độ phức tạp về không gian không được chú ý nhiều.

Độ phức tạp của thuật toán

Thuật toán thứ nhất	Thuật toán thứ hai
Temp ← a; a ← b b ← temp;	a ← a + b; b ← a – b; a ← a – b;

Hai thuật toán hoán chuyển giá trị lưu trong hai biến

Tại sao cần đo độ phức tạp của thuật toán?

- Giúp ta chọn thuật toán phù hợp nhất.
- Đơn giản mà vẫn tổng quát, có thể áp dụng trong nhiều vấn đề khác nhau.
- Tiết kiệm thời gian, không bị giới hạn bởi các yếu tố như cấu hình máy tính, ngôn ngữ sử dụng, trình biên dịch, dữ liệu đầu vào, ...

Cách tính độ phức tạp của thuật toán

- Các độ phức tạp thường gặp đối với các thuật toán thông thường gồm có:
  - Hằng số, $\boldsymbol{O(1)}$.
  - Logarithm, $\boldsymbol{O(log\ n)}$.
  - Tuyến tính, $\boldsymbol{O(n)}$.
  - Đa thức, $\boldsymbol{O(P(n))}$.
  - Hàm mũ, $\boldsymbol{O(2^n)}$.

Cách tính độ phức tạp của thuật toán

- Hằng số, $O(1)$:

```
1 a = 10                    # O(1)
2 a = 10000000000           # O(1)
3 print(a)                  # O(1)
```

```
1 age = int(input())                                    # O(1)
2 visitors = 0                                          # O(1)
3 if age < 17:
4     status = "Not allowed!"                           # O(1)
5 else:
6     status = "Welcome! Please come in"                # O(1)
7     visitors += 1                                     # O(1)
```

Cách tính độ phức tạp của thuật toán

- Logarithm, $\boldsymbol{O(logn)}$: Thường trong các thuật toán chia nhỏ vấn đề thành các vấn đề nhỏ hơn.
- Chẳng hạn: cây nhị phân,…

Cách tính độ phức tạp của thuật toán

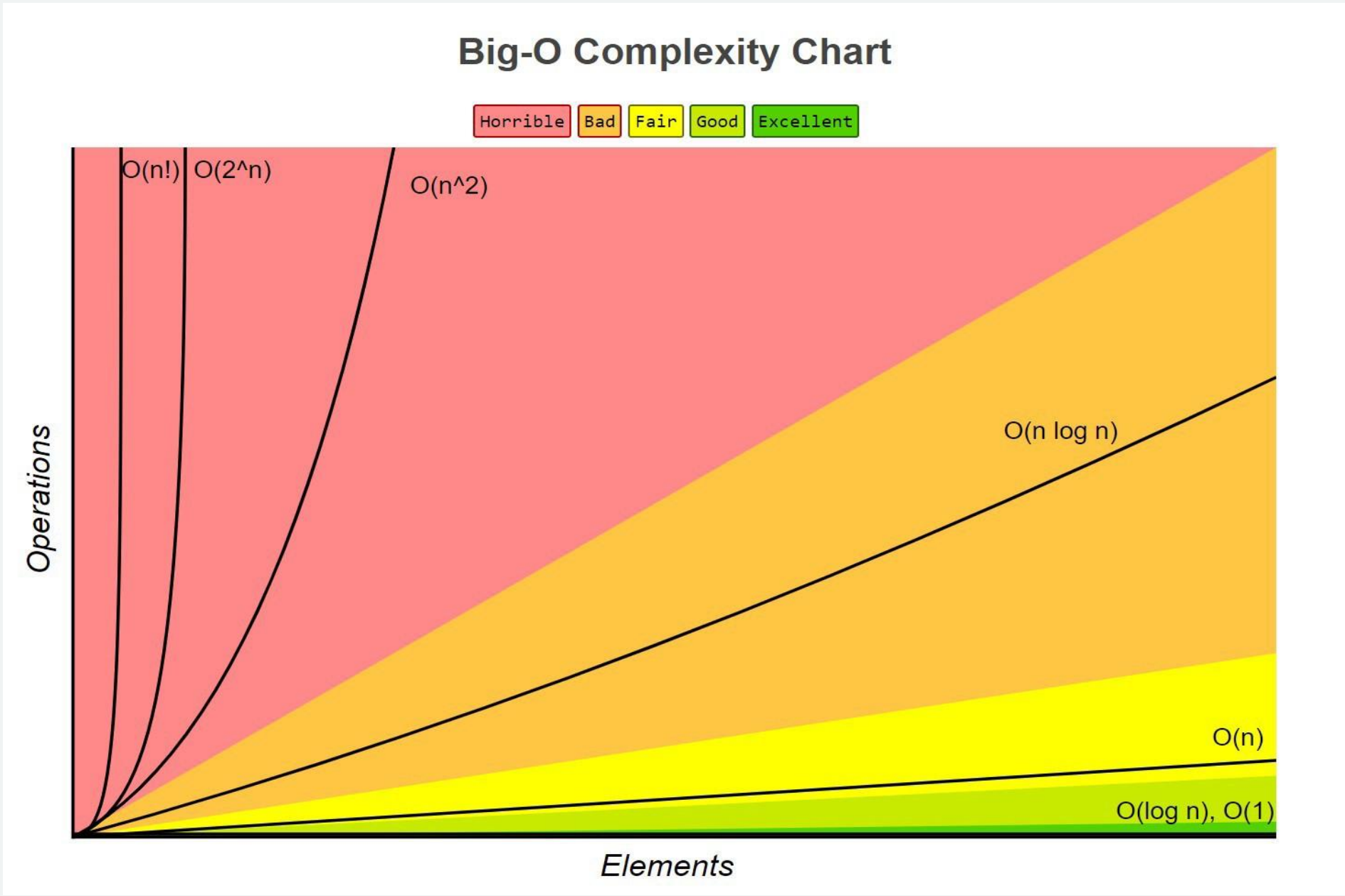
- Tuyến tính, $O(n)$:

```
1 total = 0                    # O(1)
2 for i in range(n):
3     total += i               # O(n)
4 print(total)                 # O(1)
```

Cách tính độ phức tạp của thuật toán

- Đa thức, $O(P(n))$

```
1 x = 0                                          # O(1)
2 for i in range(n):
3     for j in range(n):
4         for k in range(n):
5             x += 2                             # O(n^3)
```


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Cách tính độ phức tạp của thuật toán

- **Quy tắc bỏ hằng số:**

$$T(n) = O(c.f(n)) = O(f(n)) \qquad c = const,\ c \geq 0$$

- **Quy tắc max:**

$$O(f(n)) + O(g(n)) = O(max(f(n)), O(g(n))$$

- **Quy tắc nhân:** $T(n) = O(f(n))$

Nếu thực hiện $k(n)$ lần với $k(n) = O(g(n))$, thì độ phức tạp sẽ là $O(g(n).f(n))$.[3]

Cách tính độ phức tạp của thuật toán

```
1 s = 0;
2 for (i = 0; i <= n; i++){
3     p = 1;
4     for (j = 1; j <= i; j++)
5         p = p * x / j;
6     s += p;
7 }
```

Số lần thực hiện phép toán `p = p * x / j` là $n(n+1)/2$

Độ phức tạp của đoạn code này là $O(1) + O\left(\frac{1}{2}(n^2+n)\right) + O(1) + O(1) = O(n^2)$

Cách tính độ phức tạp của thuật toán

```
1  def function():
2      num1 = 50000
3      n = num1 / 1000
4      m = 2
5      for i in range(n):
6          for j in range(m):
7              print("This is 1st string")
8      num2 = 10000
9      x = num2 / 1000
10     for i in range(x):
11         for j in range(x):
12             print("This is 2nd string")
13             print("This is 3rd string")
14     return "Returned string"
```

- Dòng 2, 3, 4, 7, 8, 9, 12, 13, 14: mỗi dòng có $\boldsymbol{O(1)}$, nên đô phức tạp của 9 dòng đó là $\mathbf{9}$.
- Dòng 5, 6: $\boldsymbol{n * m}$
- Dòng 10, 11: $\boldsymbol{x * x = x^2}$
- Áp dụng quy tắc cộng: $\mathbf{9} + \boldsymbol{n * m + x^2}$
- Áp dụng quy tắc bỏ hằng số: $\boldsymbol{n * m + x^2}$
- Áp dụng quy tắc Max: $\boldsymbol{max(n * m,\ x^2)}$ là độ phức tạp của hàm trên.[4]

Tổng quan về độ phức tạp của các thuật toán

- Độ phức tạp thuật toán được dùng để đánh giá các thuật toán khác nhau từ đó chọn ra thuật toán tối ưu nhất.
- Thuật toán có độ phức tạp càng cao thông thường sẽ tốn nhiều thời gian.
- Việc đánh giá một thuật toán trước khi đem nó vào thực tiễn sẽ giúp tiết kiệm thời gian, tiền bạc.
- Có 3 nguyên tắc cần nhớ đó là bỏ hằng số, nhân và lấy max.

Tài liệu tham khảo

[1] Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, Trần Minh Triết (2018). Nhập môn lập trình, NXB Khoa học và Kỹ thuật.

[2] wikipedia.com. Độ phức tạp thuật toán.

[3] viblo.asia. Độ phức tạp của thuật toán.

[4] medium.com, Nguyễn Yên Bảo. Time Complexity - Độ phức tạp của thuật toán.

[5] stackoverflow.com, How to find time complexity of an algorithm.

[6] philipstel.wordpress.com, Determining The Complexity Of Algorithm (The Basic Part).