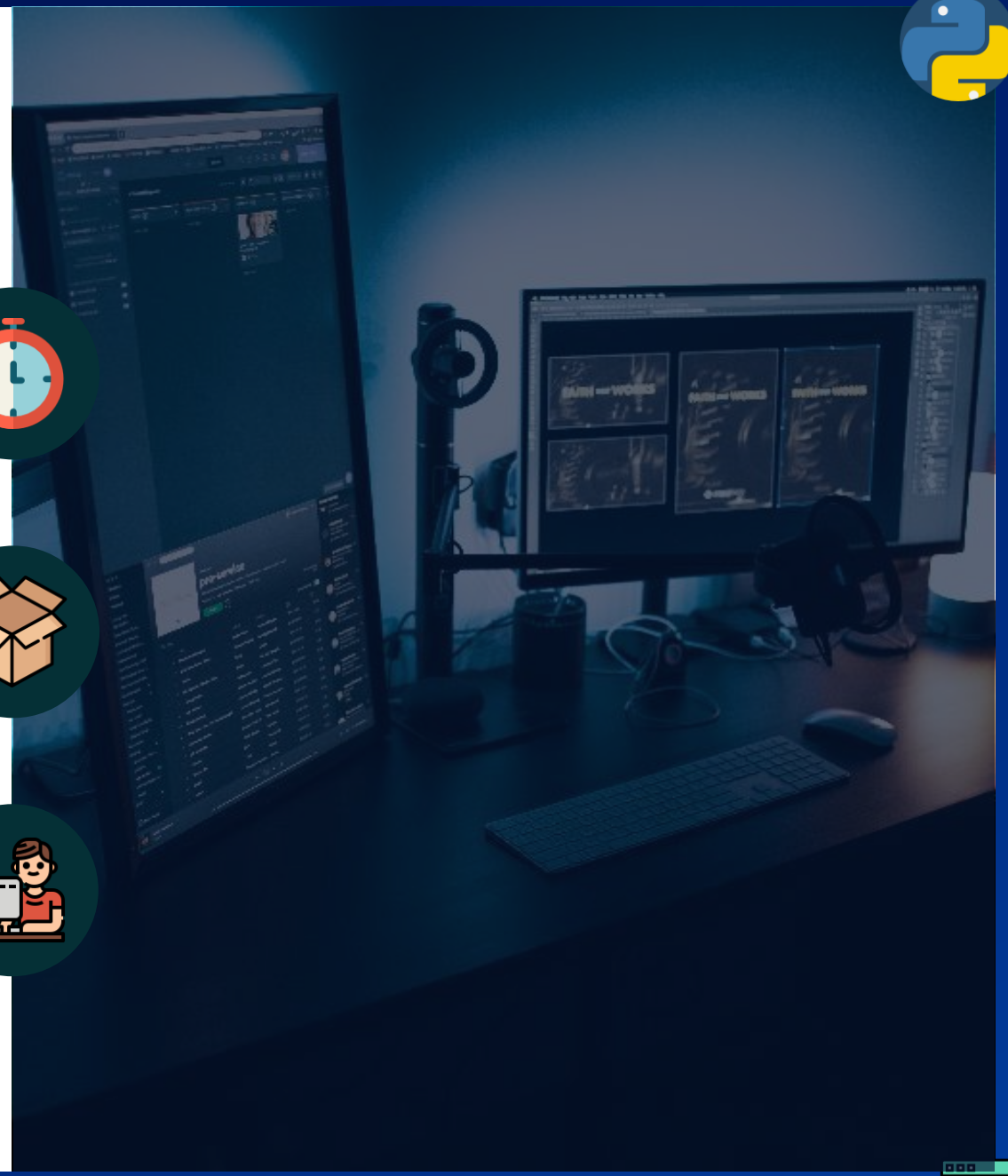
ĐỘ PHỨC TẠP CỦA THUẬT TOÁN (TIME COMPLEXITY)

Đánh giá một thuật toán

Độ phức tạp về thời gian (time complexity) và độ phức tạp về không gian (space complexity) là 2 yếu tố để quyết định một thuật toán có thích hợp để giải quyết một bài toán nào đó hay không.

Trong đó độ phức tạp về thời gian được quan tâm nhiều hơn khi các bạn tham gia vào các contest về lập trình.

Độ phức tạp thời gian là thời gian mà thuật toán của bạn cần để thực thi, nó là một hàm của input, tức là dựa vào đầu vào ta sẽ tính toán số lượng thao tác mà thuật toán cần thực thi từ đó tính ra được thời gian thực thi của thuật toán.



Trên các trang chấm bài online

Giới hạn thời gian của bài toán là **1 - 2s** đối với:

C/C++

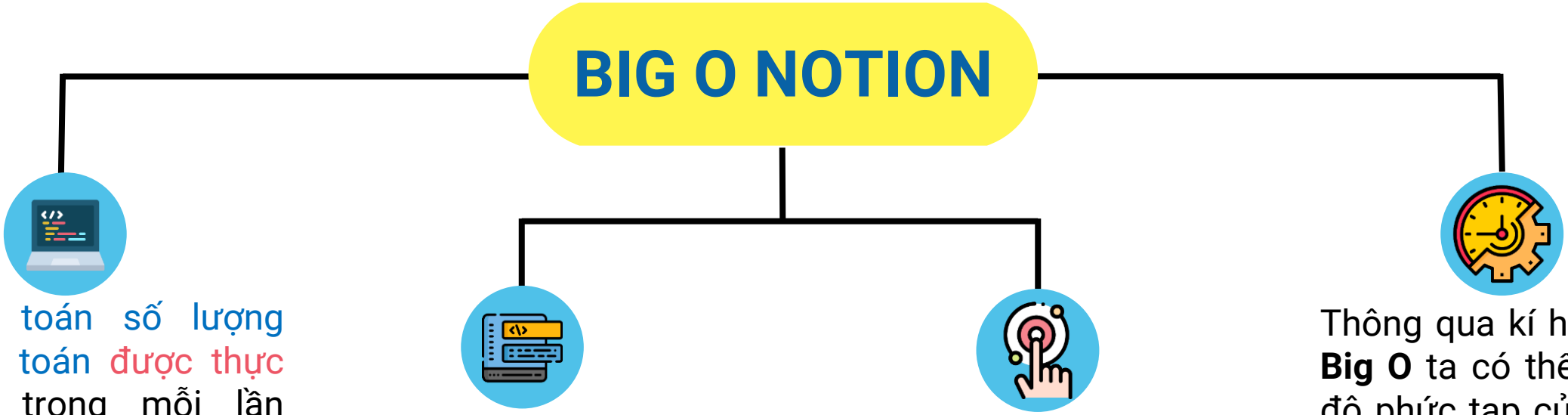
Giới hạn thời gian của một bài toán là **4s** đối với:

Java

Python

Thông thường **1s** các bạn có thể thực hiện được từ **10^8 - 5.10^8** phép toán.

1. Tính toán độ phức tạp và kí hiệu Big O:



Tính toán số lượng phép toán được thực hiện trong mỗi lần thuật toán được khởi chạy dựa trên input đầu vào của bài toán.

Kí hiệu **Big O** mô tả trường hợp tệ nhất của thuật toán thông qua một hàm của input đầu vào n.

Độ phức tạp của thuật toán càng nhỏ thì thuật toán chạy càng nhanh

Thông qua kí hiệu của **Big O** ta có thể mô tả độ phức tạp của thuật toán là O(f(n))

1. Tính toán độ phức tạp và kí hiệu Big O:

Ví dụ mô tả độ phức tạp của thuật toán là O(f(n)):
O(n), O(1), O(logn), O(nlogn), O(n^2), O(n!)...

Chú ý: Trong trường hợp hàm f(n) có chứa hằng số và các bậc khác nhau của n thì ta chọn bậc cao nhất để đại diện cho hàm f(n).
VD: O(n^2 +2n +3) được thay bằng O(n^2).

2. Một số ví dụ về phân tích độ phức tạp của thuật toán:

a) Độ phức tạp của các phép toán và nhập xuất:

Code sau có độ phức tạp là O(1) vì nó thực thi một số phép toán cố định


```
a, b, c = 100, 200, 300
sum = a + b + c
print(sum)
tich = a * b * c
print(tich)
```

Chú ý: Các phép toán như +, -, *, /, % hay các phép gán, so sánh và nhập xuất đều được coi là O(1).

2. Một số ví dụ về phân tích độ phức tạp của thuật toán:

b) Độ phức tạp của vòng lặp:

Độ phức tạp của vòng lặp lồng nhau chính là số lượng của lặp của vòng lặp nhân với độ phức tạp của code bên trong vòng lặp.

Code sau đều có độ phức tạp là O(n)

EXAMPLE

```
n = 1000
for i in range(n):
    #O(1) code
```

EXAMPLE

```
n = 1000
i = 1
while i <= n:
    #O(1) code
```

EXAMPLE

```
n = 10000
for i in range(3 * n + 2804):
    print(i)
```

2. Một số ví dụ về phân tích độ phức tạp của thuật toán:

b) Độ phức tạp của vòng lặp:

Code sau có độ phức tạp là O(nlogn), trong đó độ phức tạp của hàm nguyên tố là O(sqrt(n))


```python
import math
#O(n)
def ngto1(n):
    for i in range(2, n):
        if n % i == 0:
            return False
    return n > 1

#O(sqrt(n))
def ngto2(n):
    for i in range(2, math.isqrt(n) + 1):
        if n % i == 0:
            return False
    return n > 1
```

2. Một số ví dụ về phân tích độ phức tạp của thuật toán:

c) Độ phức tạp của vòng lặp lồng nhau:

Trong trường hợp thuật toán của bạn sử dụng vòng lặp lồng nhau, độ phức tạp của thuật toán được tính bằng cách nhân số lần lặp của từng vòng lặp với nhau.

Code sau có độ phức tạp là O(n*m) hay O(n^2)

EXAMPLE

```
n = 1000
m = 3000
#O(n * m)
for i in range(n):
    for j in range(m):
        #O(1) code
```

Code sau có độ phức tạp là O(n^3)

EXAMPLE

```
n = 1000
m = 3000
k = 5000
#O(n * m * k)
for i in range(n):
    for j in range(m):
        for l in range(k):
            #O(1) code
```

CHÚ Ý

Các bạn càng sử dụng nhiều vòng lặp lồng nhau thì thuật toán sẽ càng lớn và thời gian chạy càng lâu.

2. Một số ví dụ về phân tích độ phức tạp của thuật toán:

d) Độ phức tạp khi chương trình chứa nhiều khối thực thi:

Trong trường hợp thuật toán của bạn có nhiều khối thực thi thì độ phức tạp của thuật toán sẽ được xét bằng với độ phức tạp của khối có độ phức tạp lớn nhất

Code sau có độ phức tạp là O(n^2)


```
n, m = map(int, input().split())
#O(n^2)
for i in range(n):
    for j in range(m):
        print(i, j)
#O(n)
for i in range(n + 29):
    print(i)
```

3. Một vài độ phức tạp thường gặp:

Thao tác, thuật toán	Độ phức tạp
Sử dụng công thức toán học để tìm ra ngay lời giải	O(1)
Tìm kiếm nhị phân	O(logn)
Các thao tác của set, map, hàng đợi ưu tiên	O(logn)
Kiểm tra số nguyên tố, phân tích thừa số nguyên tố	O(sqrt(n)
Đọc n số từ input	O(n)
Duyệt qua mảng	O(n)
Hàm sort trong thư viện	O(nlogn)
Sàng số nguyên tố	O(nloglogn)
Duyệt các tập con cỡ k của tập có n phần tử	O(n^k)
Duyệt mọi tập con	O(2^n)
Duyệt mọi hoán vị	O(n!)