ĐỘ PHỨC TẠP THUẬT TOÁN

I- Thuật toán là gì:

Thuật toán là tập hợp hữu hạn các bước giải quyết một vấn đề theo một trình tự nhất định.

Input đầu vào  **Thuật toán**  **Output**

Thuật toán có mặt trong tất cả các công việc xung quanh chúng ta

II- Độ phức tạp thuật toán:

1. Đặt vấn đề:

Chúng ta mỗi ngày đều tiếp xúc với rất nhiều sản phẩm tin học. Trong đó có những sản phẩm có một số tính năng giống nhau.

Tuy nhiên, ta có thể nhận thấy giữa 2 tính năng giống nhau của 2 sản phẩm khác nhau, chúng có sự khác biệt nhau về mặt tốc độ, hoặc thậm chí là về mặt chính xác về kết quả. Vậy, làm sao để đánh giá một sản phẩm có phù hợp trong cuộc sống hay không ?

Ta cần một đại lượng để đo độ hiệu quả của một thuật toán trong một sản phẩm.

Độ phức tạp thuật toán ra đời, là một đại lượng dùng để đánh giá độ hiệu quả của một thuật toán trong một sản phẩm.

II- Độ phức tạp thuật toán:

2. Độ phức tạp thuật toán về thời gian (Time Complexity):

Độ phức tạp thuật toán về mặt thời gian phụ thuộc vào 4 yếu tố:

- Cấu hình máy tính và bộ dịch
- Ngôn ngữ lập trình
- Kích thước dữliệu đầu vào
- Khả năng lập trình của lập trình viên

II- Độ phức tạp thuật toán:

2. Độ phức tạp thuật toán về thời gian (Time Complexity):

Có 3 loại độ phức tạp thuật toán:

- BigO (O): Độ phức tạp của thuật toán trong trường hợp chương trình chạy chậm nhất

- Big Theta (Θ): Độ phức tạp của thuật toán trong thời gian trung bình

- Big Omega (Ω): Độ phức tạp của thuật toán trong thời gian nhanh nhất

II- Độ phức tạp thuật toán:

2. Độ phức tạp thuật toán về thời gian (Time Complexity):

Big-O: Là độ phức tạp thuật toán trong trường hợp thời gian thực thi chậm nhất.

Trong thực tế, khi ước lượng độ phức tạp thuật toán, đại lượng BigO là đại lượng được sử dụng nhiều nhất, vì ta chỉ nên đánh giá một thuật toán là tốt hay xấu dựa vào thời gian chạy chậm nhất.

II- Độ phức tạp thuật toán:

2. Độ phức tạp thuật toán về thời gian (Time Complexity):

Big-Omega: Là độ phức tạp thuật toán trong trường hợp thời gian thực thi nhanh nhất.

Thường chỉ được dùng để tham khảo, không có quá nhiều ý nghĩa trong thực tiễn.

II- Độ phức tạp thuật toán:

2. Độ phức tạp thuật toán về thời gian (Time Complexity):

Big-Theta: Là độ phức tạp thuật toán trong trường hợp trung bình. Ta lấy trung bình của tất cả thời gian thực thi của thuật toán trong mọi trường hợp.

Thường rất khó để đánh giá chính xác. Tuy nhiên, nếu ta đánh giá thuật toán bằng BigO không khả thi trong thực tiễn, ta có thể nghĩ đến việc đánh giá thuật toán dựa trên Big Theta, vì dữ liệu trong thực tế thường ngẫu nhiên và phân phối đều.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

a. Độ phức tạp hằng số (O(1)):

Là đơn vị được đánh giá đối với những thuật toán chỉ thực hiện trong một số lần hữu hạn bước nhất định (Lưu ý: số bước phải là một con số nhỏ).

Ví dụ:
- Thuật toán kiểm tra 3 số nguyên dương có phải là độ dài 3 cạnh tam giác hay không ?
- Thuật toán kiểm tra 1 số nguyên dương là số chẵn hay lẻ ….

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

a. Độ phức tạp hằng số (O(1)):

Dù dữ liệu input đầu vào có là bất kỳ giá trị nào, số lượng bước để thực hiện các bài toán vẫn là một số bước hữu hạn và không đổi.

Không tồn tại sản phẩm nào trong thực tế có độ phức tạp thuật toán là độ phức tạp hằng số.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

b. Độ phức tạp logarit (O(log(N)):

Là đơn vị được đánh giá đối với những thuật toán mà số bước thực thi thuật toán tăng theo hàm Logarit.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

b. Độ phức tạp logarit (O(log(N)):

```
int sum = 0;
for (int i = 1; i <= N; i *= 2) {
   sum += i;
}
```


```
sum = 0
i = 1
while i <= N:
    sum += i
    i *= 2
```

Số bước chỉ thực hiện trong log(N) bước. Nếu N = 1000000, số lượng bước cần thiết để thực hiện xong những vòng lặp trên chỉ tối đa logN(1000000) ~ 20 bước.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

b. Độ phức tạp logarit (O(log(N)):

Thuật toán có độ phức tạp Logarit thường có tốc độ thực thi rất nhanh vì số lượng bước thực thi rất ít.

Một số thuật toán có độ phức tạp Logarit:
- Thuật toán tìm kiếm nhị phân
- Thuật toán thêm - xóa của cấu trúc dữ liệu Heap hay cây nhị phân cân bằng.

Một số sản phẩm ứng dụng thực tế có độ phức tạp Logarit:
- Thuật toán tìm kiếm một tài xế xe ôm trên bản đồ (Grab)
- Đo mắt cận
- Tìm bóng đèn bị hỏng khi mạch bóng đèn được mắc nối tiếp

...

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

c. Độ phức tạp tuyến tính (O(N)):

Là độ phức tạp tương ứng đối với những thuật toán có số lượng bước tăng tuyến tính dựa trên dữ liệu đầu vào.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

c. Độ phức tạp tuyến tính (O(N)):

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


```
sum = 0
i = 1
for i in range(N):
    sum += i
```

Số bước được thực hiện trong N bước. Nếu N = 1000000, số lượng bước cần thiết để thực hiện xong những vòng lặp trên là 1000000 bước.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

c. Độ phức tạp tuyến tính (O(N)):

Một số thuật toán có độ phức tạp tuyến tính:
- Thuật toán tìm một số nguyên trong một dãy số
- Thuật toán so khớp chuỗi trong văn bản
….

Một số sản phẩm ứng dụng thực tế có độ phức tạp tuyến tính:
- Thuật toán KMP dùng để so khớp chuỗi văn bản trong Microsoft Word.
- Thuật toán DFS / BFS là công cụ thực hiện thuật toán luồng cực đại trong việc phân phối các chuyến bay.
- Hiển thị các gợi ý trên thanh search Google.
...

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

d. Độ phức tạp tuyến tính logarit (O(N.log(N))):

Là độ phức tạp tương ứng đối với những thuật toán gồm 2 vòng lặp lồng nhau, một vòng lặp có độ phức tạp tuyến tính và một vòng lặp có độ phức tạp Logarit.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

d. Độ phức tạp tuyến tính logarit (O(N.log(N))):

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

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

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

d. Độ phức tạp tuyến tính logarit (O(N.log(N))):

Một số thuật toán có độ phức tạp tuyến tính logarit:
- Thuật toán sắp xếp một dãy số nguyên dương theo chiều tăng dần
- Thuật toán Dijkstra tối ưu bởi hàng đợi ưu tiên
….

Một số sản phẩm ứng dụng thực tế có độ phức tạp tuyến tính logarit:
- Google Maps
- Sắp xếp dữ liệu
...

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

e. Độ phức tạp đa thức (O(N^c)):

Là độ phức tạp tương ứng đối với những thuật toán gồm c vòng lặp tuyến tính lồng nhau.

Những thuật toán có độ phức tạp từ độ phức tạp đa thức trở lên ít được ứng dụng rộng rãi trong thực tế khi tập dữ liệu lớn.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

e. Độ phức tạp đa thức (O(N^c)):

```
for (int i = 1; i <= N; i++) {
   for (int j = 1; j <= N; j++) {
      cout << i << ' ' << j << endl;
   }
}
```

```
for i in range(N):
    for j in range(N):
        print(i, j)
```

Độ phức tạp O(N^2).

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

e. Độ phức tạp đa thức (O(N^c)):

Một số thuật toán có độ phức tạp đa thức:
- Thuật toán sắp xếp Buble sort
- Tìm khoảng cách Leverstein trong 2 chuỗi AND
- Thuật toán Floyd - Warshall

….

Một số sản phẩm ứng dụng thực tế có độ phức tạp tuyến tính:
- Floyd Warshall
- Seam Carving trong Photoshop
- Luồng cực đại tách Object chính ra khỏi nền xung quanh trong Photoshop

...

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

f. Độ phức tạp lũy thừa (O(c^N)):

Là độ phức tạp tương ứng đối với những thuật toán có số bước tăng rất nhanh theo cấp độ của hàm lũy thừa.

Trong thực tế gần như không có sản phẩm nào có thuật toán với độ phức tạp này.

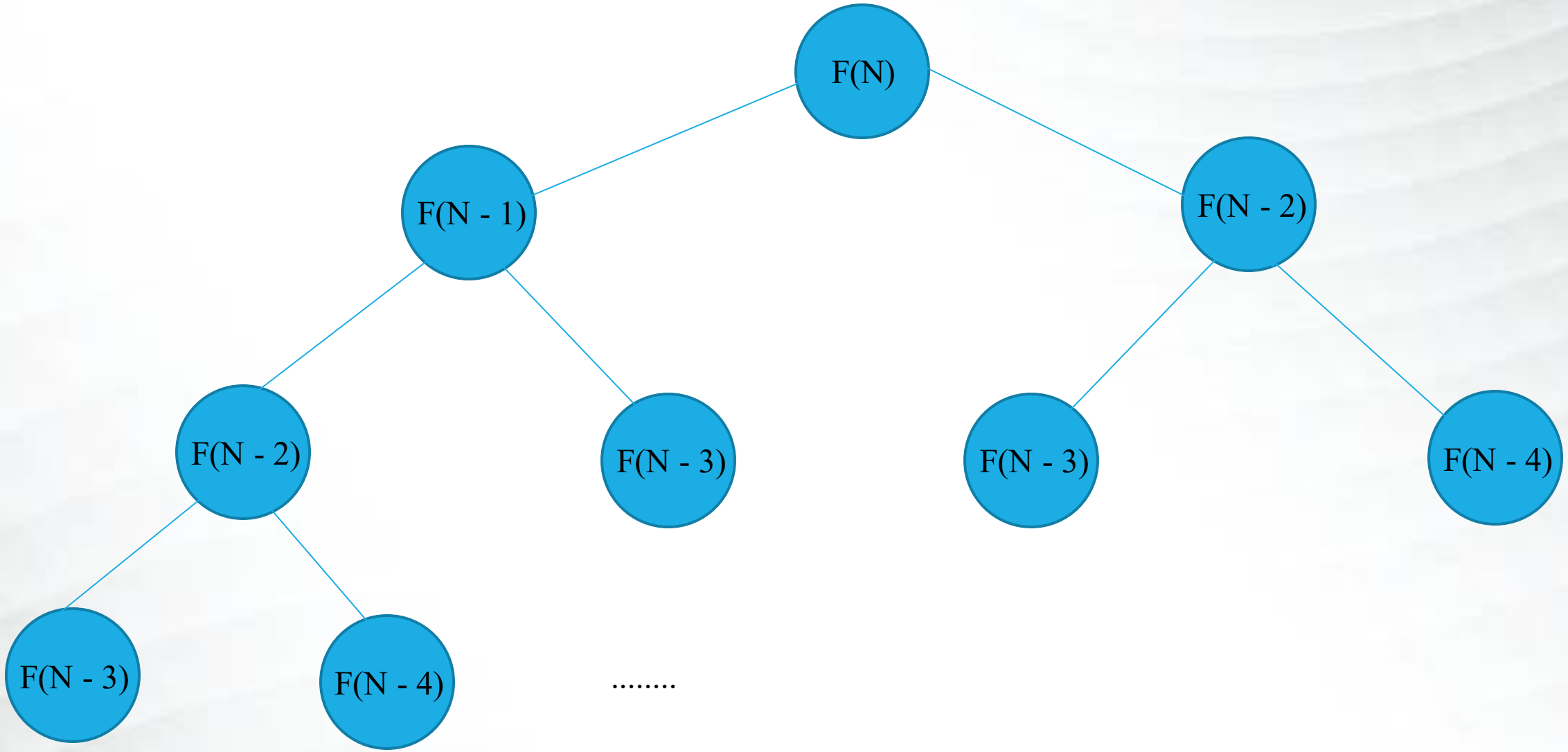
II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

f. Độ phức tạp lũy thừa (O(c^N)):

```
long long fibo(int N) {
  if (N == 0 || N == 1) {
    return 1;
  }
  return fibo(N - 1) + fibo(N - 2);
}
```

```
def fibo(N):
    if N == 0 || N == 1:
        return 1
    return fibo(N - 1) + fibo(N - 2)
```



Độ phức tạp thuật toán tính số Fibonacci: O(2^N)

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

g. Độ phức tạp giai thừa (O(N!)):

Là độ phức tạp tương ứng đối với những thuật toán cần phải sinh hết toàn bộ trường hợp có thể xảy ra để xét.

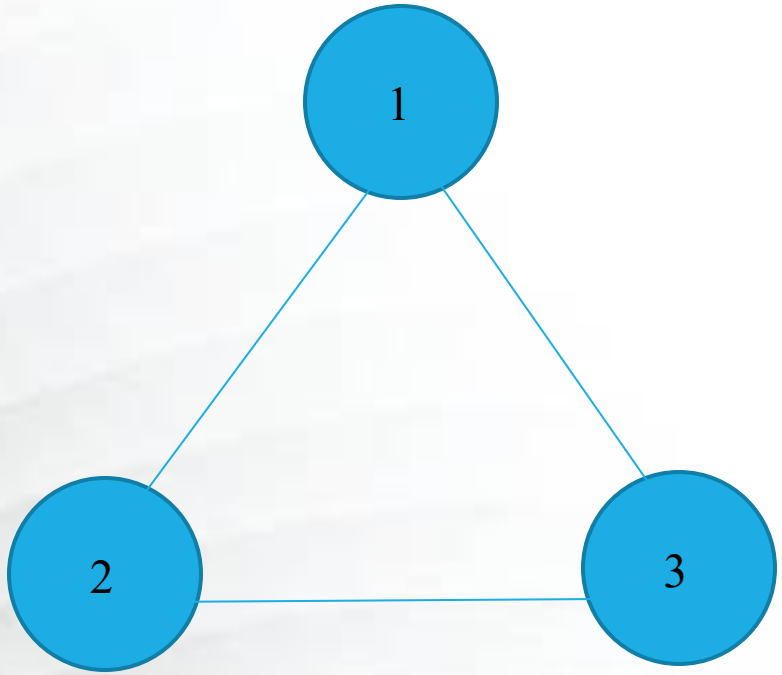
Vì độ phức tạp quá lớn, do đó trong thực tế gần như không có sản phẩm nào có thuật toán với độ phức tạp này.

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

g. Độ phức tạp giai thừa (O(N!)):

Bài toán Người đưa thư (Traveling Saleman):



Các đường đi thỏa mãn:

1 - 2 - 3

1 - 3 - 2

2 - 1 - 3

2 - 3 - 1

3 - 1 - 2

3 - 2 - 1

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

g. Độ phức tạp giai thừa (O(N!)):

Vì không thể giải được bài toán Traveling Saleman khi kích thước bài toán lớn. Do đó đã có một số thuật toán dưới dạng Heuristics, chỉ cho ra kết quả gần đúng. Một vài thuật toán tiêu biểu :

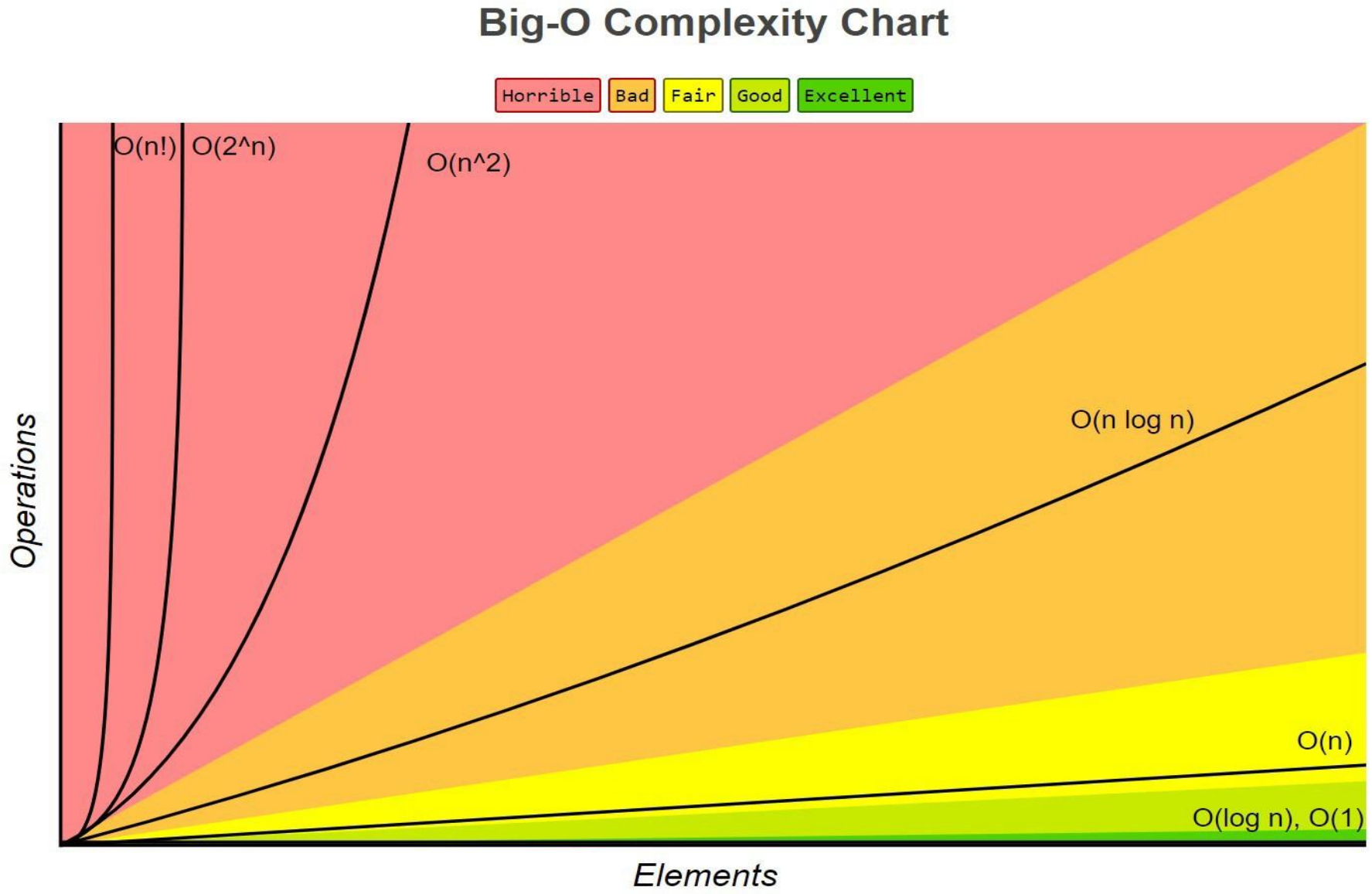
- Tabu Search
- Harmony Search
....

Bảng thống kê ước lượng thời gian thực thi thuật toán tương ứng với kích thước dữ liệu đầu vào tương ứng.

Bảng thống kê được thực hiện khi thực thi trên máy tính core i5, 1GHz.

	Also called	$n = 100$	$n = 10{,}000$	$n = 1{,}000{,}000$
$O(1)$	Constant time	0.000001 sec.	0.000001 sec.	0.000001 sec.
$O(\lg n)$	Logarithmic time	0.000007 sec.	0.000013 sec.	0.00002 sec.
$O(n)$	Linear time	0.0001 sec.	0.01 sec.	1 sec.
$O(n \lg n)$		0.00066 sec.	0.13 sec.	20 sec.
$O(n^2)$	Quadratic time	0.01 sec.	100 sec.	278 hours
$O(n^3)$	Cubic time	1 sec.	278 hours	317 centuries
$O(2^n)$	Exponential time	10^{14} centuries	10^{2995} centuries	10^{30087} centuries
$O(n!)$	Factorial time	10^{143} centuries	10^{35645} centuries	N/A

Link bài viết gốc: https://www.sciencedirect.com/topics/computer-science/o-notation



Link bài viết gốc: https://medium.com/swlh/basics-of-big-o-notation-7d5d905d058d

Bảng thống kê những thuật toán có thể chấp nhận được đối với input đầu vào tương ứng trong thời gian 1 giây

Acceptance Complexity by Inputs:-

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

II- Độ phức tạp thuật toán:

3. Các đơn vị đánh giá độ phức tạp thuật toán về thời gian:

Một số quy tắc để tính độ phức tạp thuật toán.

- **Quy tắc hằng số: O(k.f(n)) = O(f(n)) với k là một hằng số nhỏ.**
  Ví dụ: O(2n + 3) = O(n)
- **Quy tắc cộng : O(f(n) + g(n)) = Max(O(f(n)), O(g(n)))**
  Ví dụ: O(3n^2 + 2n + 5) = O(n^2)
- **Quy tắc nhân: O(f(n) * g(n)) = O(f(n)) * O(g(n))**
  Ví dụ: O((2n + 5) * (3n^2 + 4n)) = O(2n + 5) * O(3n^2 + 4n) = O(n) * O(n^2) = O(n^3)

II- Độ phức tạp thuật toán:

4. Độ phức tạp thuật toán về không gian (Space Complexity):

Độ phức tạp thuật toán về mặt không gian phụ thuộc vào 4 yếu tố:

- Cấu hình máy tính và bộ dịch
- Ngôn ngữ lập trình
- Cấu trúc dữ liệu được sử dụng
- Khả năng lập trình của lập trình viên

II- Độ phức tạp thuật toán:

5. Các đơn vị được dùng để đánh giá độ phức tạp không gian:

a. Độ phức tạp hằng số (O(1)):

Được đánh giá là độ phức tạp hằng số khi và chỉ khi chương trình sử dụng một hoặc nhiều biến số rời rạc để tính toán.

Ví dụ: Bài toán tính tổng các số từ 1 đến N.
Ta chỉ sử dụng một biến N đầu vào, một biến tính tổng và một biến chạy dùng để duyệt vòng lặp.

II- Độ phức tạp thuật toán:

5. Các đơn vị được dùng để đánh giá độ phức tạp không gian:

b. Độ phức tạp tuyến tính (O(N)):

Được đánh giá là độ phức tạp tuyến tính khi và chỉ khi chương trình sử dụng một hoặc nhiều mảng gồm N biến số để tính toán.

III- Luyện tập:

Bài 1:

Hãy đánh giá độ phức tạp về thời gian cũng như không gian trong đoạn code sau:

```
int sumA = 0, sumB = 0;
for (int i = 0; i < N; i++) {
    sum A += A[i];
}

for (int j = 0; j < M; j += 2) {
    sumB += B[j];
}
```

```
sumA = 0
sumB = 0;
for i in range(N):
    sumA += A[i]
for j in range(0, M, 2):
    sumB += B[j]
```

Độ phức tạp về thời gian: O(N + M)
Độ phức tạp về không gian: O(N + M)

III- Luyện tập:

Bài 2:

Hãy đánh giá độ phức tạp về thời gian cũng như không gian trong đoạn code sau:

```
long long res = 1;
for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= (int) sqrt(i); j++) {
        res *= j;
    }
}
```

```
res = 1
for i in range(1, N + 1):
    for j in range(1, int(i ** 0.5) + 1):
        res *= j
```

Độ phức tạp về thời gian: O(N.sqrt(N))

Độ phức tạp về không gian: O(1)

III- Luyện tập:

Bài 3:

Hãy đánh giá độ phức tạp về thời gian trong đoạn code sau:

```
long long res = 1;
for (int i = N; i >= 1; i /= 2) {
    for (int j = 1; j <= i; j++) {
        res *= j;
    }
}
```

```
res = 1
i = N
while i >= 1:
    for j in range(1, i + 1):
        res *= j
    i //= 2
```

Độ phức tạp về thời gian: O(N)