Complexity

Why do we need big O ?

Nếu chúng ta sử dụng hai cỗ máy khác nhau, một cái A rất mạnh, một cái B rất yếu. Cùng chạy một thuật toán C nhất định. Thì kết quả về thời gian chạy của máy A sẽ phải khác với máy B. Do vậy nếu chúng ta dùng cách đo thời gian chạy của một thuật toán PHỤ THUỘC trên một máy tính nào đó, thì sẽ không đủ để làm tiêu chuẩn đánh giá độ nhanh chậm của thuật toán.

Nói theo nghĩa khác, một thuật toán sẽ phải được đo lường thông qua một cách nào đó không phụ thuộc vào chiếc máy tính đang chạy nó. Gọi là **Machine Independent**.

Một kịch bản khác, thuật toán không thể đo lường chính xác nếu như dùng hai ngôn ngữ khác nhau để implement nó. Chẳng hạn như ta dùng C++ để implement thuật toán Binary Search, và cũng dùng Python để implement thuật toán tương tự. Với các ngôn ngữ khác nhau, thời gian chạy thuật toán có thể có sự sai khác. Vì vậy mà chúng ta cần đo lường chúng theo cách độc lập với ngôn ngữ, gọi là **Language Independent**.

Từ đó, cách đo lường sử dụng Big O notation ra đời.

Definition

Big O Notation (Ký hiệu O lớn) là một số để đo lường mức độ hiệu quả của thuật toán. Mức độ hiệu quả chính là độ phức tạp thời gian và độ phức tạp không gian. Độ phức tạp thời gian là để chỉ thời gian và độ phức tạp không gian là để chỉ bộ nhớ dùng thêm tương ứng với dữ liệu đầu vào trong lúc chạy chương trình. Thời gian thực thi và bộ nhớ thực thi chính là hai yếu tố cần thiết góp phần vào mức độ hiệu quả của thuật toán.

Ngoài ra, ký hiệu O lớn dùng để mô tả mức độ tăng lên của thời gian hoặc không gian thực thi bởi chương trình, khi đối số đầu vào tiến về một số nào đó. Ký hiệu O lớn ta có thể gọi chung là độ phức tạp thuật toán.

Chú ý rằng Big O là đo cận trên của thời gian chạy thuật toán, ngoài ra còn Omega Notation để đo cận dưới và Theta Notation để đo trung bình giữa hai cận trên và dưới.

Để so sánh hai thuật toán thì ta dùng Big O Notation cho lượng dữ liệu đầu vào lớn.

Formal Definition

Một Big O là một hàm số biểu thị mức độ tăng lên của thời gian hay các bước thực hiện thuật toán. Xét hai hàm số $F(n)$ và $G(n)$ cùng hằng số $C > 0$.

$$F = O(G)$$

khi và chỉ khi

$$F(n) \leq C.G(n)$$

Tức là hàm Big O chỉ tồn tại khi hằng số C tồn tại.

Ví dụ, có một hàm số thực hiện các phép so sánh phụ thuộc vào dữ liệu đầu vào với số lần so sánh là

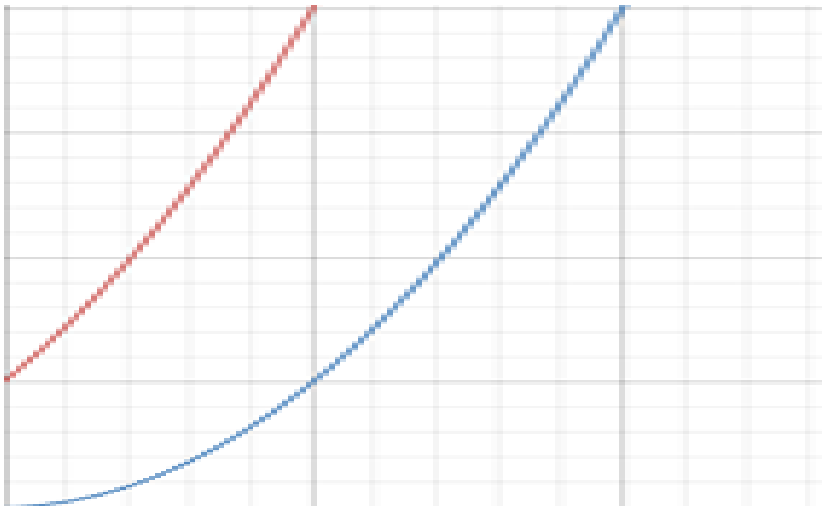
$$F(n) = (x+1)^2$$

Khai triển ta được

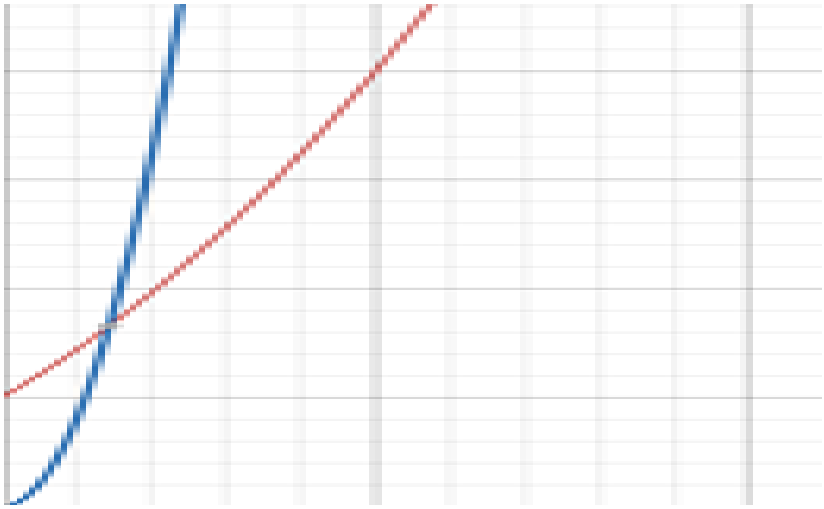
$$F(n) = x^2 + 2x + 1$$

Nếu theo quy luật áp đảo khi n trở nên rất lớn, bậc lớn nhất sẽ là Big O của hàm số. Nghĩa là $F = O(n^2)$. Suy ra $G(n)$ là n^2.

Khi vẽ hai đồ thị này lên hình, ta sẽ có hình ảnh như sau:



Với đường màu đỏ là $F(n)$ và màu xanh là $G(n)$. Dễ dàng tìm được một hằng số C bất kỳ làm cho $G(n) \geq F(n)$ khi n tiến ra vô cùng. Hằng số đó có thể là 20, cho ra kết quả như hình dưới đây.



Ta có thể dùng công thức giới hạn để chứng minh hàm Big O tồn tại.

$$\lim_{n\to\infty} \frac{G(n)}{F(n)} = \lim_{n\to\infty} \frac{n^2}{n^2 + 2n + 1} = 1 = C > 0$$

Do $C > 0$ nên hàm Big O tồn tại.

Ta tự hỏi rằng tại sao không phải là $O(n)$. Cũng dùng một giới hạn để chứng minh điều này

$$\lim_{n\to\infty} \frac{G(n)}{F(n)} = \lim_{n\to\infty} \frac{n}{n^2 + 2n + 1} = 0 = C$$

Do C không lớn hơn 0 nên hàm Big O không tồn tại. Ngụ ý rằng, hàm $G(n) = n$ không có cách nào có thể nhanh hơn hàm $F(n)$ khi n tiến ra vô

cùng được, nên vì thế hàm $F(n)$ sẽ áp đảo và độ phức tạp sẽ là $O(n)$.

Có thể thấy hình trên, ban đầu $G(n) = n$ có thể lớn hơn, nhưng khi n tiến về vô cùng, cụ thể là từ $n = 13$ trở đi, hàm $G(n)$ luôn tăng chậm hơn hàm $F(n)$.

Measurement

Để tính toán độ phức tạp của một thuật toán, ta cần tuân thủ theo các bước sau:

1. Understand how algorithm works

Mục đích của thuật toán.

Dữ liệu đầu vào, dữ liệu đầu ra.

2. Determine the criterial unit that algorithm base on.

Các dòng xuất thông tin.

Vòng lặp/phép gán.

Lời gọi đệ quy (đây là đơn vị cơ bản cho các hàm đệ quy).

Đồng thời tập trung vào trường hợp tệ nhất.

3. Counting the unit and regconize the pattern. Conclusion the complexity.

Cases

Độ phức tạp thuật toán chia thành ba trường hợp: Worst Case – xấu nhất, Best Case – tốt nhất và Average Case – trung bình. Thông thường thường người ta sẽ lưu tâm đến Worst Case hoặc Average Case vì Best Case không có ý nghĩa gì với chúng ta khi xét về độ hiệu quả của thuật toán.

Order of Big O