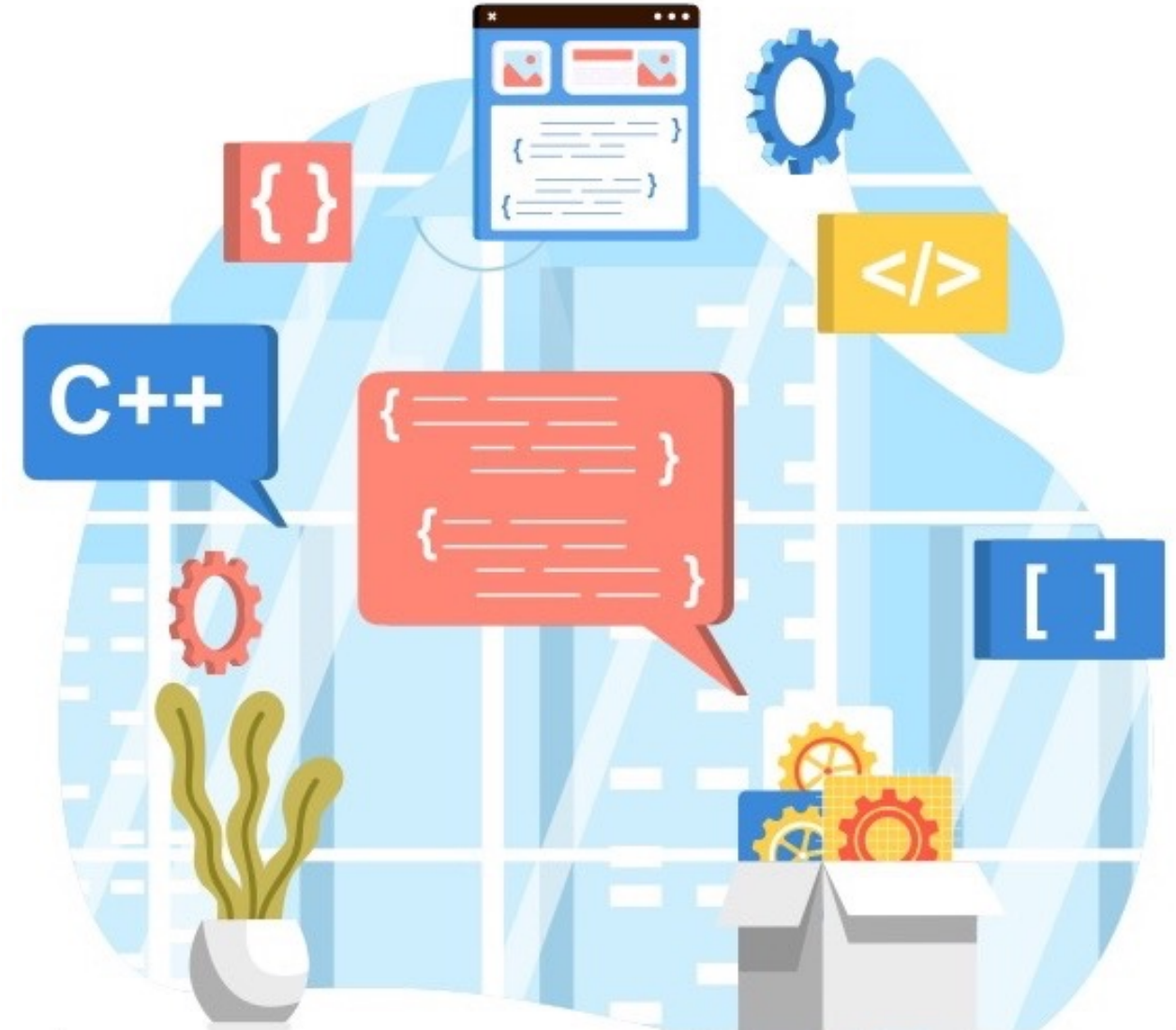
TỔNG QUAN VỀ GIẢI THUẬT VÀ CẤU TRÚC DỮ LIỆU

VŨ NGỌC THANH SANG
KHOA CÔNG NGHỆ THÔNG TIN
ĐẠI HỌC SÀI GÒN



0. MÔ TẢ HỌC PHẦN

Phương pháp đánh giá học phần

- Điểm chuyên cần, thảo luận: hệ số 10 %
- Điểm kiểm tra quá trình lý thuyết: hệ số 20 %
- Điểm kiểm tra quá trình thực hành: hệ số 20 %
- Điểm thi kết thúc học phần: hệ số 50 %

0. MÔ TẢ HỌC PHẦN

- Các bài toán thực tế có thể được đơn giản hóa thành các mô hình toán để giải quyết trên máy tính.
- Xây dựng mô hình cần chú trọng đến hai vấn đề
  1. Cấu trúc dữ liệu: biểu diễn các đối tượng thực tế, diễn tả mối quan hệ giữa các đối tượng.
  2. Giải thuật: diễn tả trình tự các thao tác xử lý dữ liệu để đạt được kết quả mong muốn.

Cấu trúc dữ liệu + Giải thuật = Chương trình

I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

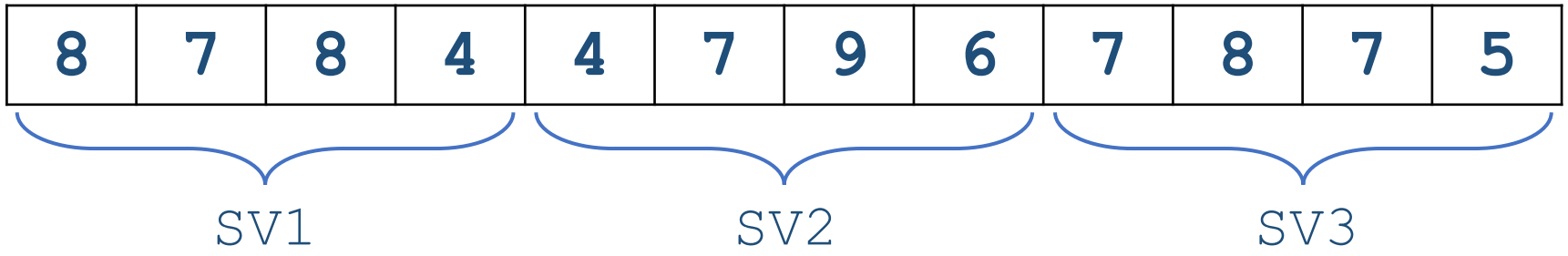
- Viết chương trình quản lý điểm thi của 3 sinh viên. Mỗi sinh viên có 4 cột điểm được mô tả như bảng dưới đây. Xuất điểm số các môn của sinh viên.

Sinh viên	Môn 1	Môn 2	Môn 3	Môn 4
SV1	8	7	8	4
SV2	4	7	9	6
SV3	7	8	7	5

I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

- Cách 1: Sử dụng mảng 1 chiều

```
int result[12] = {8,7,8,4,4,7,9,6,7,8,7,5};
```



I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

- Cách 1: Sử dụng mảng 1 chiều

```
void XuatDiem() {
  const int so_mon = 4;
  int sv, mon;

  for (int i = 0; i < 12; i++) {
    sv = i / so_mon;
    mon = i % so_mon;
    printf("Diem mon %d cua SV %d la: %d", mon, sv, result[i]);
  }
}
```

I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

- Cách 2: Sử dụng mảng 2 chiều

```
int result[3][4] = {{8,7,8,4},
                    {4,7,9,6},
                    {7,8,7,5}};
```

	Cột 0	Cột 1	Cột 2	Cột 3
Dòng 0	`result[0][0] = 8`	`result[0][1] = 7`	`result[0][2] = 8`	`result[0][3] = 4`
Dòng 1	`result[1][0] = 4`	`result[1][1] = 7`	`result[1][2] = 9`	`result[1][3] = 6`
Dòng 2	`result[2][0] = 7`	`result[2][1] = 8`	`result[2][2] = 7`	`result[2][3] = 5`

I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

- Cách 2: Sử dụng mảng 2 chiều

```
void XuatDiem() {
  int so_mon = 4, so_sv = 3;
  for (int i = 0; i < so_sv; i++)
    for (int j = 0; j < so_mon; j++)
      printf("Diem mon %d cua SV %d la: %d", j, i,
      result[i][j]);
}
```

I. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU

Cấu trúc dữ liệu là cách tổ chức dữ liệu để tăng hiệu quả và dễ dàng truy cập và xử lý. Vai trò của cấu trúc dữ liệu bao gồm:

- **Quản lý dữ liệu**: giúp tổ chức dữ liệu để dễ dàng truy cập và quản lý.
- **Tăng hiệu suất**: sử dụng các thuật toán tối ưu để xử lý dữ liệu nhanh hơn.
- **Dễ dàng sử dụng**: cho phép lập trình viên sử dụng dữ liệu một cách dễ dàng và linh hoạt.
- **Giải quyết bài toán phức tạp**: giúp giải quyết các bài toán phức tạp về dữ liệu như tìm kiếm, sắp xếp, v.v.

II. CÁC TIÊU CHUẨN ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU

Cấu trúc dữ liệu tốt phải thõa mãn các tiêu chí sau:

- Phản ánh đúng thực tế: Cần xem xét các trạng thái biến đổi của dữ liệu.

  Vd: Chọn biến số nguyên **`int`** để lưu trữ điểm trung bình của SV

- Phù hợp với các thao tác trên biến.

  Vd: Nhiều mảng 1 chiều, 1 mảng nhiều chiều, tính đồng bộ của dữ liệu.

- Tiết kiệm tài nguyên hệ thống: cấu trúc dữ liệu chỉ nên sử dụng tài nguyên hệ thống (CPU, bộ nhớ) vừa đủ.

II. CÁC TIÊU CHUẨN ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU

Đối với mỗi ứng dụng, cần lưu ý đến tác vụ nào được sử dụng nhiều nhất để sử dụng cấu trúc dữ liệu cho thích hợp.

- Nếu yêu cầu dữ liệu là các đối tượng với các thuộc tính rất nhiều và yêu cầu tìm kiếm nhanh, cấu trúc dữ liệu phù hợp có thể là một cây tìm kiếm nhị phân hoặc một bảng băm.
- Nếu số lượng dữ liệu lớn và cần phải sắp xếp theo thứ tự, cấu trúc dữ liệu phù hợp có thể là một danh sách liên kết hoặc một heap.

III. KIỂU DỮ LIỆU

Các thuộc tính của một kiểu dữ liệu:

- Tên kiểu dữ liệu (**`int, float, string, struct,`** ...)
- Miền giá trị
- Kích thước lưu trữ (**`bits, bytes,`**...)
- Tập các toán tử tác động lên kiểu dữ liệu
  - Dữ liệu số: Cộng, trừ, nhân, chia, lấy căn, lũy thừa, ...
  - Dữ liệu chuỗi ký tự: so sánh, sao chép, nhật chuỗi, xuất chuỗi, ...

III. KIỂU DỮ LIỆU

Tên kiểu	Kích thước	Miền giá trị
Char	1 byte	- 128 đến 127
unsign char	1 byte	0 đến 255
int	2 byte	- 32738 đến 32767
unsign int	2 byte	0 đến 65335
long	4 byte	-2^{32} đến $2^{31} - 1$
unsign long	4 byte	-2^{32} đến $2^{31} - 1$
float	4 byte	-3.4×10^{-38} to 3.4×10^{38}
double	8 byte	-1.7×10^{-308} to 1.7×10^{308}
long double	10 byte	-1.7×10^{-4932} to 1.7×10^{4932}

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

- Hầu hết các bài toán đề có thể được giải quyết bởi nhiều thuật toán khác nhau. Như vậy, làm thể nào để so sánh các thuật toán và chọn ra được lựa chọn tối ưu nhất?

1. Thời gian thực thi
2. Dung lượng bộ nhớ
3. Độ phức tạp
4. Độ linh hoạt
5. Độ chính xác

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

- **Thời gian thực thi**: Ví dụ, nếu chúng ta cần so sánh hai thuật toán tìm kiếm, chúng ta có thể đo thời gian cần thiết để tìm kiếm một phần tử trong một tập dữ liệu lớn và so sánh thời gian chạy của hai thuật toán.

Sorting Algorithm	Test 1 (100)	Test 2 (1000)	Test 3 (10000)
Bubble Sort	Min: 0.01008 seconds Max: 0.0206 seconds	Min: 1.0242 seconds Max: 1.0558 seconds	Min: 100.922 seconds Max: 102.475 seconds
Insertion Sort	Min: 0.00306 seconds Max: 0.00650 seconds	Min: 0.0369 seconds Max: 0.0562 seconds	Min: 100.422 seconds Max: 102.344 seconds
Selection Sort	Min: 0.00556 seconds Max: 0.00946 seconds	Min: 0.4740 seconds Max: 0.4842 seconds	Min: 40.831 seconds Max: 41.218 seconds

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

- **Dung lượng bộ nhớ**: Ví dụ, nếu chúng ta cần so sánh hai thuật toán sắp xếp, chúng ta có thể đo số lượng bộ nhớ tối thiểu cần thiết để hoàn thành quá trình sắp xếp và so sánh dung lượng bộ nhớ của hai thuật toán.

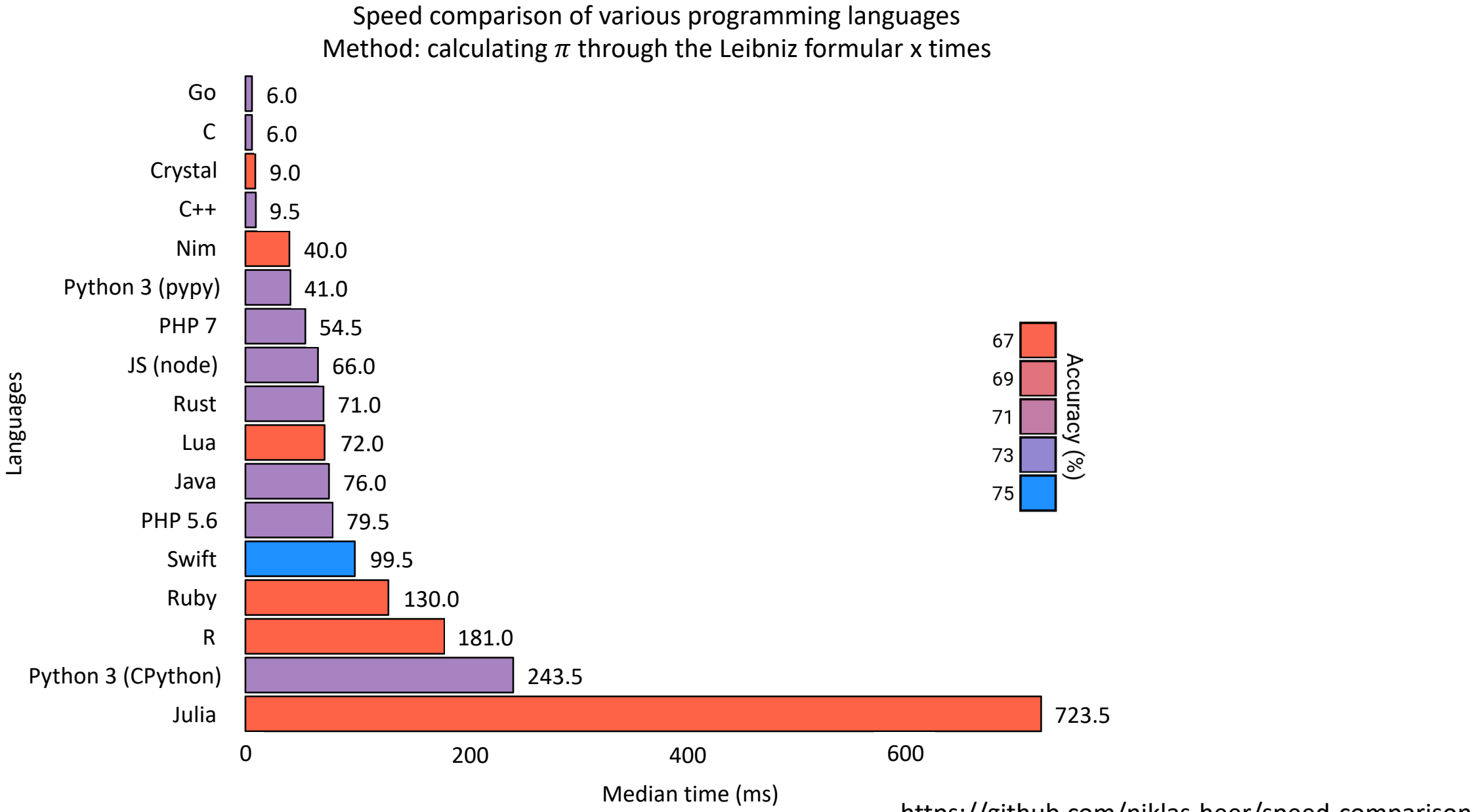
Sorting Algorithm	Test 1 (100)	Test 2 (1000)	Test 3 (10000)
Bubble Sort	Min: 0.01008 seconds Max: 0.0206 seconds	Min: 1.0242 seconds Max: 1.0558 seconds	Min: 100.922 seconds Max: 102.475 seconds
Insertion Sort	Min: 0.00306 seconds Max: 0.00650 seconds	Min: 0.0369 seconds Max: 0.0562 seconds	Min: 100.422 seconds Max: 102.344 seconds
Selection Sort	Min: 0.00556 seconds Max: 0.00946 seconds	Min: 0.4740 seconds Max: 0.4842 seconds	Min: 40.831 seconds Max: 41.218 seconds

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Một số nhược điểm khi thực hiện một thuật toán:

- Do phải cài đặt bằng một ngôn ngữ lập trình cụ thể nên thuật toán sẽ chịu sự hạn chế của ngôn ngữ lập trình này.
- Hiệu quả của thuật toán sẽ bị ảnh hưởng bởi trình độ của người cài đặt
- Việc chọn được các bộ dữ liệu cho tất cả các tập dữ liệu của thuật toán là rất khó khăn và tốn nhiều chi phí.
- Tốc độ xử lý được phụ thuộc nhiều vào phần cứng của máy tính. Dẫn tới việc so sánh các thuật toán được thực thi trên các cấu hình khác nhau trở nên khó khăn.

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT



https://github.com/niklas-heer/speed-comparison

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

- **Độ phức tạp của thuật toán** (algorithm complexity) là một hàm số mô tả hiệu quả của thuật toán về số lượng dữ liệu mà thuật toán cần phải xử lý.
- **Độ phức tạp về thời gian** (time complexity) mô tả thời gian mà một thuật toán cần xử lý dựa trên khối lượng dữ liệu đầu vào.
- **Độ phức tạp về không gian** (space complexity) mô tả số lượng bộ nhớ mà một thuật toán cần sử dụng để xử lý khối lượng dữ liệu đầu vào.
- Cả hai phép đo tập trung đến độ phức tạp tiệm cận của thuật toán. Có nghĩa là, khi N (khối lượng dữ liệu đầu vào) trở nên rất lớn thì hiệu suất của thuật toán thay đổi như thế nào?

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

- Cho hàm số $f(n) = n^2 + 100n + \log_{10} n + 1000$
- Khi n tăng, chỉ có số hạng n^2 là đáng kể

n	f(n)	n^2		100n		$\log_{10} n$		1,000	
	Value	Value	%	Value	%	Value	%	Value	%
1	1,101	1	0.1	100	9.1	0	0.0	1,000	90.83
10	2,101	100	4.76	1,000	47.6	1	0.05	1,000	47.60
100	21,002	10,000	47.6	10,000	47.6	2	0.001	1,000	4.76
1,000	1,101,003	1,000,000	90.8	100,000	9.1	3	0.0003	1,000	0.09
10,000	101,001,004	100,000,000	99.0	1,000,000	0.99	4	0.0	1,000	0.001
100,000	10,010,001,005	10,000,000,000	99.9	10,000,000	0.099	5	0.0	1,000	0.00

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Để so sánh thời gian thực thi của hai thuật toán **dựa trên thực nghiệm**, ta cần xem xét các yếu tố sau

- **Kích thước của dữ liệu đầu vào**: cần đánh giá dựa trên dữ liệu đầu vào giống nhau.
- **Năng lực của người lập trình**: mỗi người sẽ có một giải pháp cho bài toán cho trước. Khi chương trình thực thi sẽ có sự khác biệt.
- **Phần cứng**: cần so sánh hai thuật toán trên máy tính có cấu hình giống nhau. Tuy nhiên thuật toán có giải thuật tối ưu đôi khi không thể thực hiện được trên phần cứng giới hạn.

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Để so sánh thời gian thực thi của hai thuật toán **dựa trên phân tích**, ta cần xem xét các yếu tố sau

- **Kích thước của dữ liệu đầu vào**: cần đánh giá dựa trên dữ liệu đầu vào giống nhau.
- **Đếm số phép toán cơ bản (primitive operation)**: chúng ta sẽ đếm số phép toán cơ bản mà giải thuật cần thực thi với kích thước dữ liệu cố định n. Thuật toán sử dụng ít phép toán hơn thì sẽ được xem xét là tốt hơn.

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

```
int main() {                // Không tính
    int x = 0;              // Gán giá trị cho biến                  // +1
    x += 1;                 // Truy cập biến, cộng, gán giá trị      // +3
    cout << x;              // Truy cập biến, hiển thị giá trị       // +2
    return x;               // Truy cập biến, trả về giá trị         // +2
}                           // Tổng cộng                             //  8
```

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

```
#include <iostream>                          // + 0
using namespace std;                         // + 0
int main(){                                  // + 0
    int n = 10;                              // + 1
    int sum = 0;                             // + 1
    for (int i = 0; i < n; i++)              // + 6n + 4
        sum += 2;                            // + 3n
    cout << sum;                             // + 2
    return 0;                                // + 1
}
```

Độ phức tạp về thời gian $T(n) = 1 + 1 + 6n + 4 + 3n + 2 + 1 = 9n + 9$

Độ phức tạp thuật toán $O(n)$

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

```
#include <iostream>                          // + 0
using namespace std;                         // + 0
int main(){                                  // + 0
    int n = 5;                               // + 1
    int m = 7;                               // + 1
    int sum = 0;                             // + 1
    for (int i = 0; i < n; i++) {            // + 6n + 4
        for (int j = 0; j < m; j++)          // + n(6m + 4)
            sum += 1;                        // + 3mn
    }                                        // + 0
    cout << sum;                             // + 2
    return 0;                                // + 1
}
```

Độ phức tạp về thời gian $T(n,m) = 9nm + 10n + 10$

Độ phức tạp thuật toán $O(nm)$

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Ký hiệu Ω (Ω notation) – trường hợp tốt nhất

Độ phức tạp của thuật toán $f(n)$ là $\Omega(g(n))$ nếu tồn tại số dương c và N sao cho $f(n) \geq cg(n)$ với mọi $n \geq N$ (n đủ lớn).

Ký hiệu O lớn (Big-O notation) – trường hợp xấu nhất

Độ phức tạp của thuật toán $f(n)$ là $O(g(n))$ nếu tồn tại số dương c và N sao cho $f(n) \leq cg(n)$ với mọi $n \geq N$ (n đủ lớn).

Ký hiệu Θ (Θ notation) – trường hợp trung bình

Độ phức tạp của thuật toán $f(n)$ là $O(g(n))$ nếu tồn tại số dương c_1, c_2 và N sao cho $c_1 g(n) \leq f(n) \leq c_2 g(n)$ với mọi $n \geq N$ (n đủ lớn).

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Ω notation

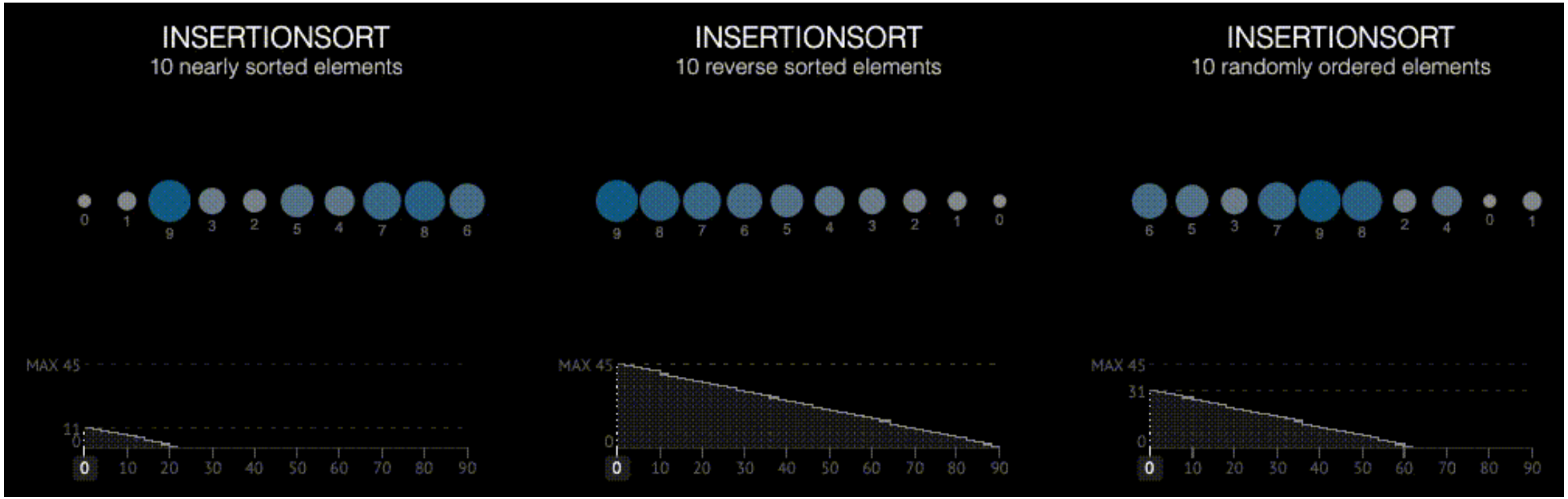
trường hợp tốt nhất

Big-O notation

trường hợp xấu nhất

Θ notation

trường hợp trung bình



IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

Tính chất của ký hiệu O lớn

- **Tính bắc cầu**: Nếu $f(n) = O(g(n))$ và $g(n) = O\big(h(n)\big)$, thì $f(n) = O(h(n))$.
- Nếu $f(n) = O(h(n))$ và $g(n) = O(h(n))$ thì $f(n) + g(n) = O\big(h(n)\big)$.
- Hàm số $an^k = O\big(n^k\big)$ với mọi giá trị $a > 0$.
- Hàm số n^k có độ phức tạp là $O\big(n^{k+j}\big)$ với mọi j là số nguyên dương.
- Nếu $f(n) = cg(n)$ thì $f(n) = O(g(n))$.
- Hàm số $\log_a(n) = O(\log_b n)$ với mọi giá trị dương a và $b \neq 1$.

IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

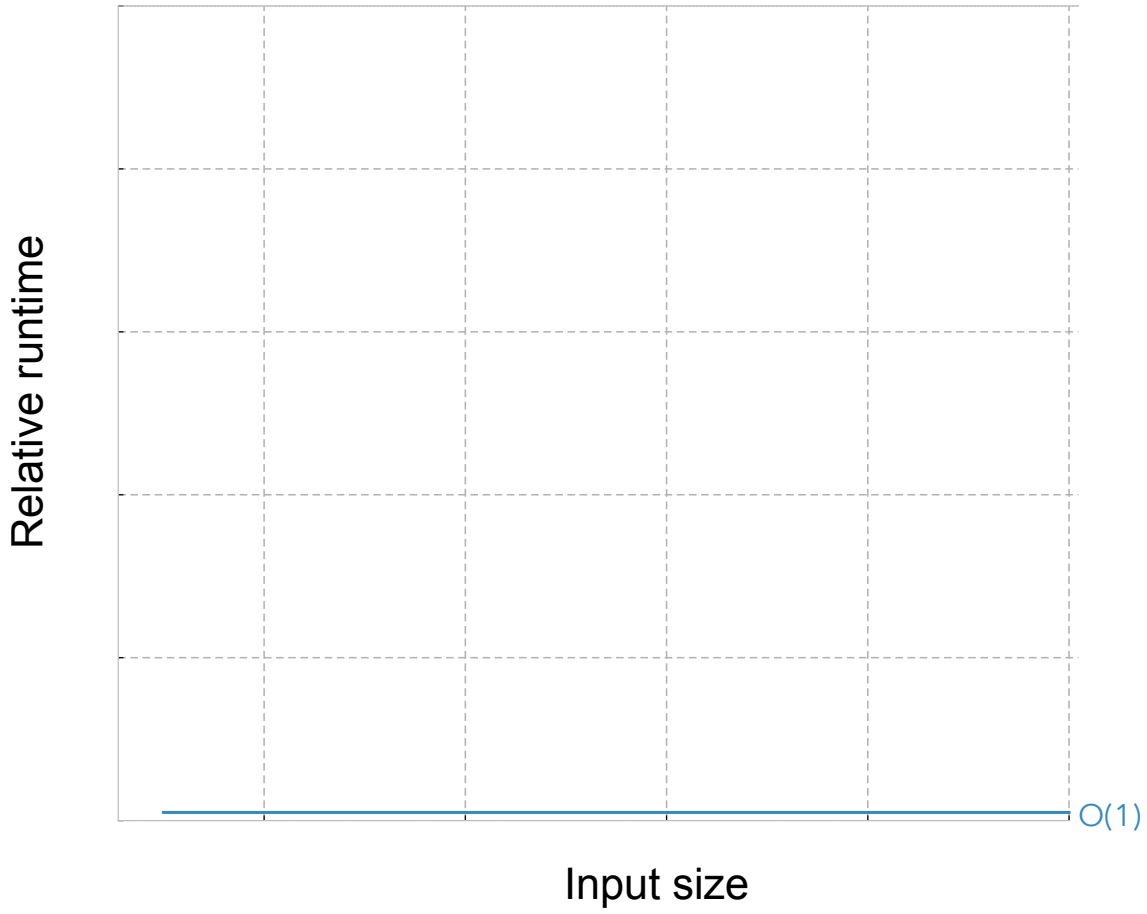
O(1) – constant (hằng số)

Thực thi trong một khoảng thời gian cố định, không phụ thuộc vào kích thước dữ liệu đầu vào.

Ví dụ:

- Truy cập giá trị thứ n trong mảng
- Xác định một số là chẵn hay lẻ

```
(n % 2 == 0) ? "even" : "odd";
```

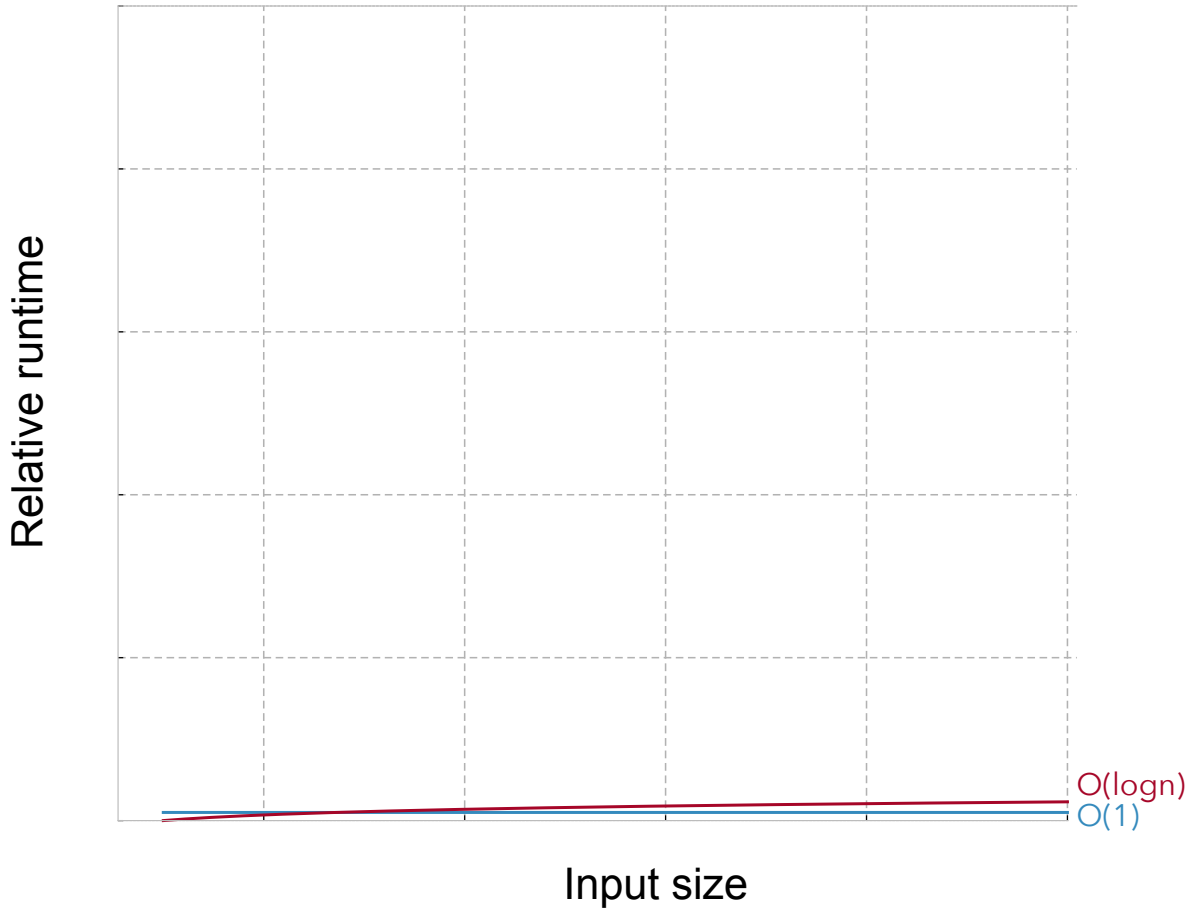


IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

O(logn)

Thời gian thực thi chương trình tỉ lệ với logarit của kích thước dữ liệu đầu vào. Tại mỗi bước của thuật toán, một phần đáng kể của dữ liệu đầu vào sẽ được bỏ qua.

Ví dụ: Tìm kiếm nhị phân



IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

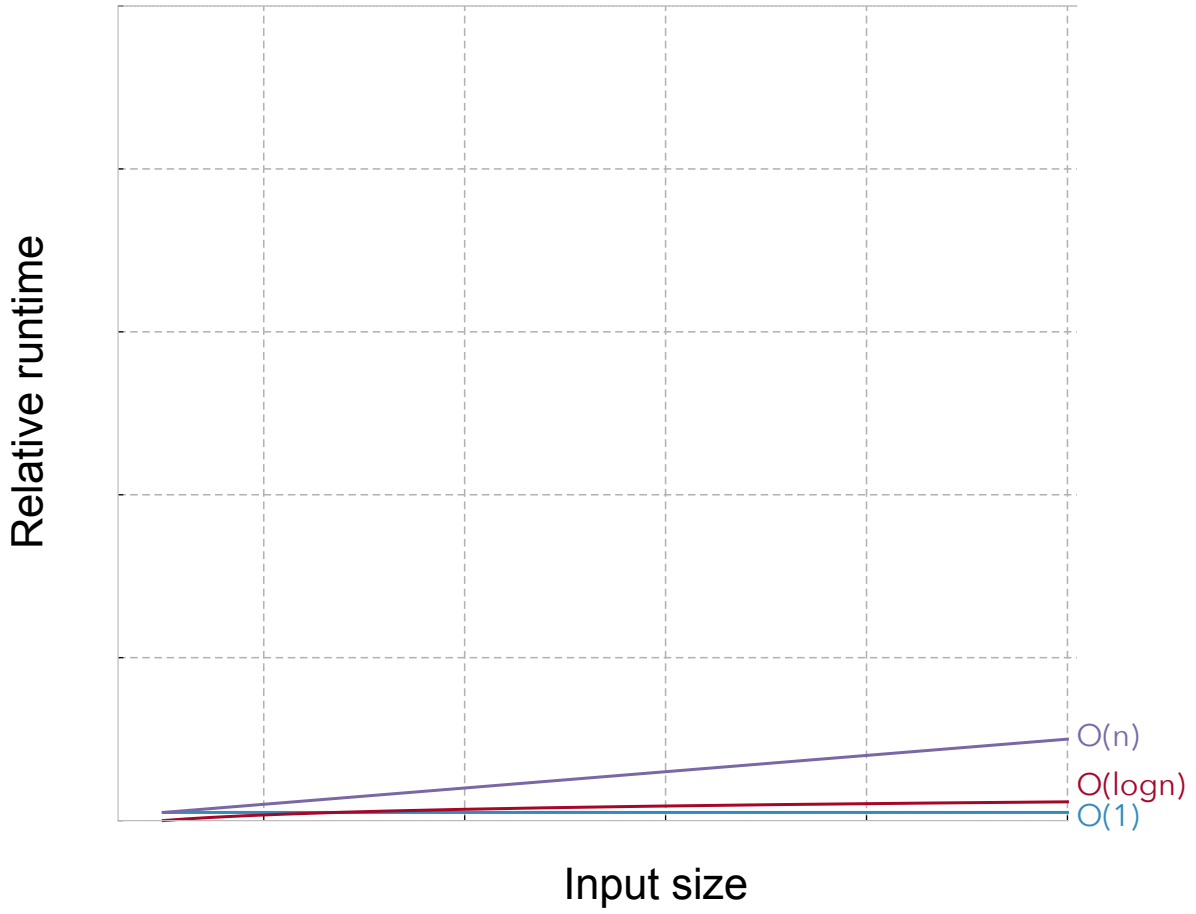
O(n) - linear complexity

Thời gian thực thi chương trình tỉ lệ với kích thước dữ liệu đầu vào.

Ví dụ:

- Tìm giá trị cụ thể, nhỏ nhất, lớn nhất trong mảng.

```
double max = a[0];
for (int i = 1; i < n; i++)
    if (a[i] > max)
        max = a[i];
```



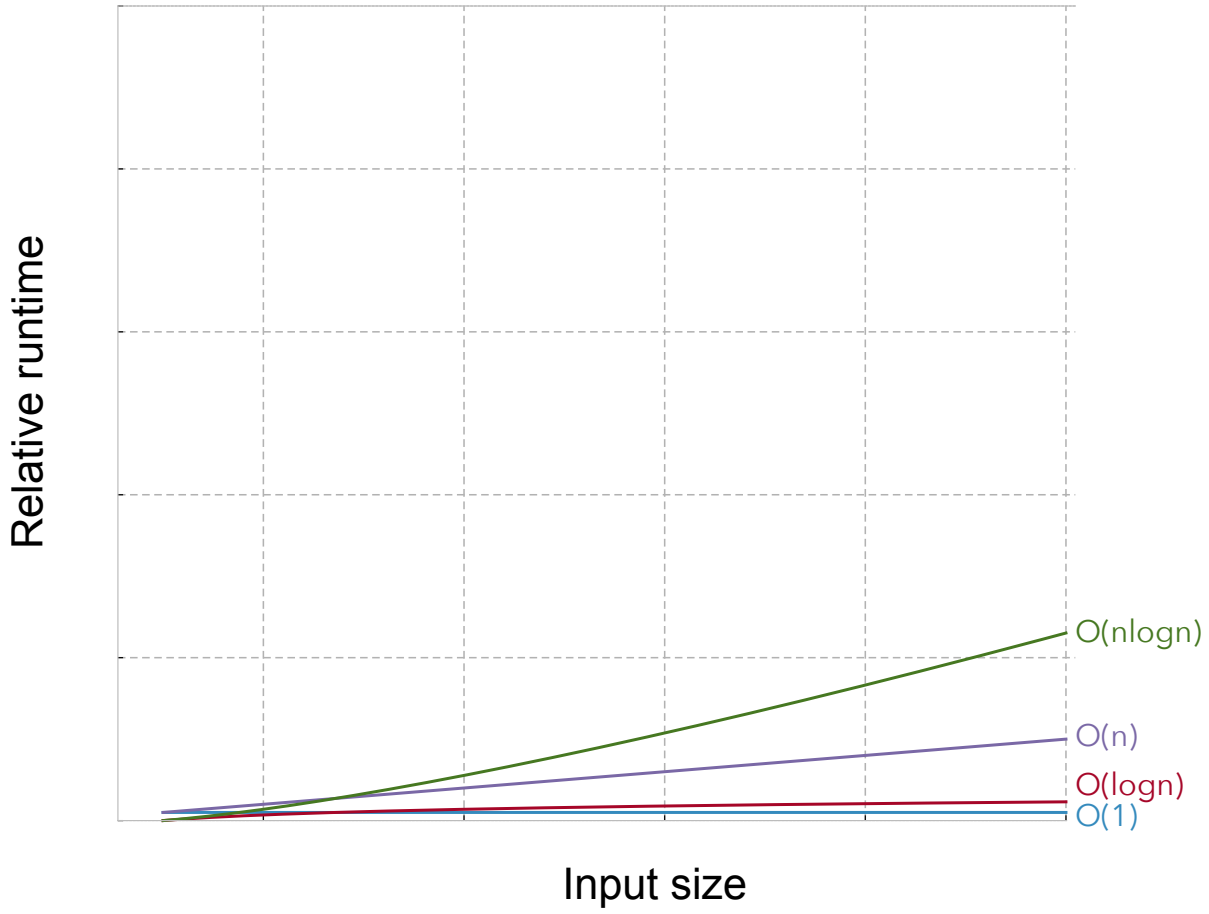
IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

O(nlogn) – logarithmic complexity

Thời gian thực thi chương trình tỉ lệ với kích thước dữ liệu đầu vào nhân với logarit của nó.

Ví dụ:

- Merge sort
- Heap sort



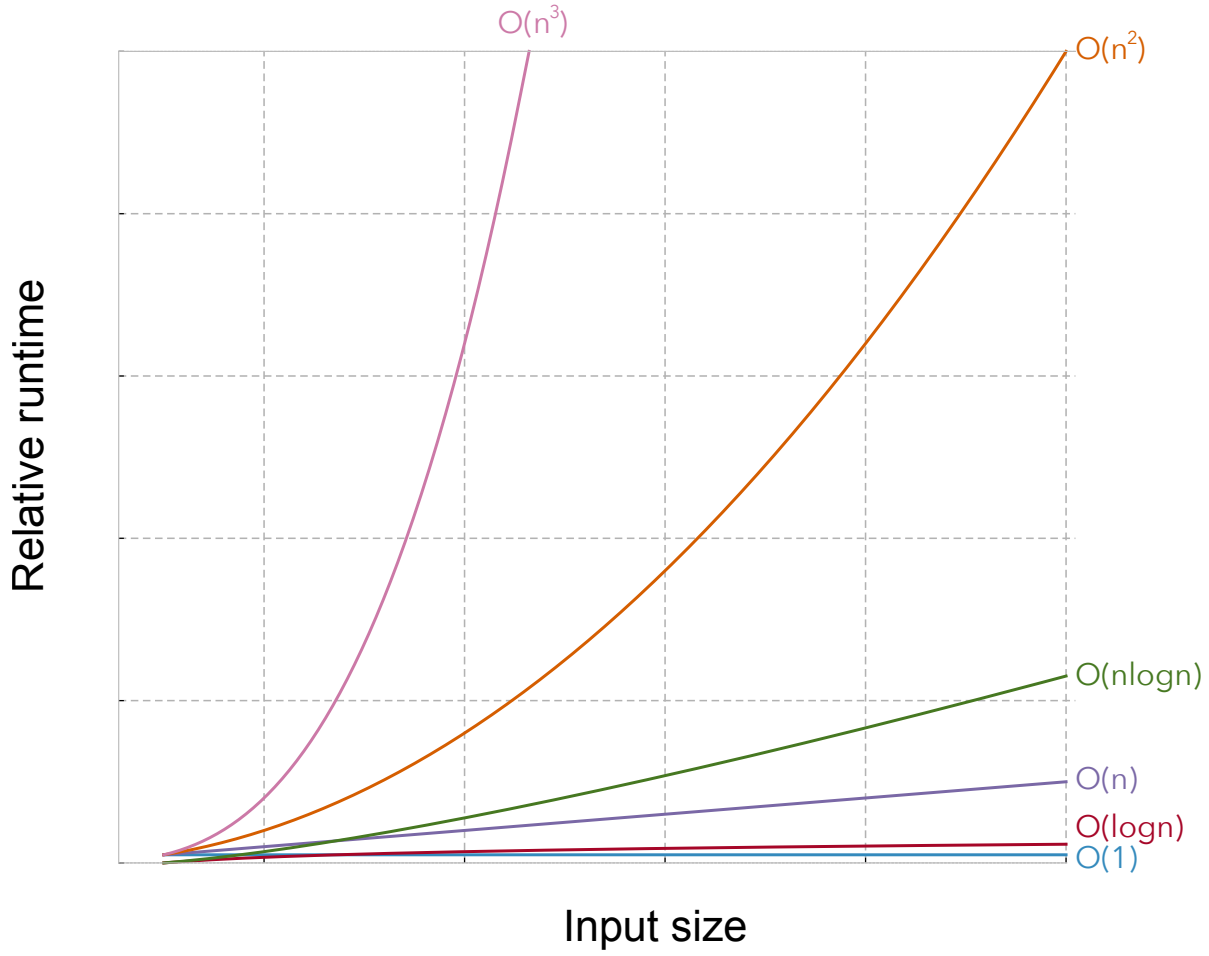
IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

$O(n^2)$, $O(n^3)$ – polynomial complexity

Thời gian thực thi chương trình tỉ lệ bậc 2 hoặc bậc 3 với kích thước dữ liệu đầu vào nhân với logarit của nó.

Ví dụ: Khởi tạo giá trị ban đầu cho mảng 2 chiều

```
for(i = 0; i < hang; ++i)
    for(j = 0; j < cot; ++j)
    {
        mult[i][j]=0;
    }
```



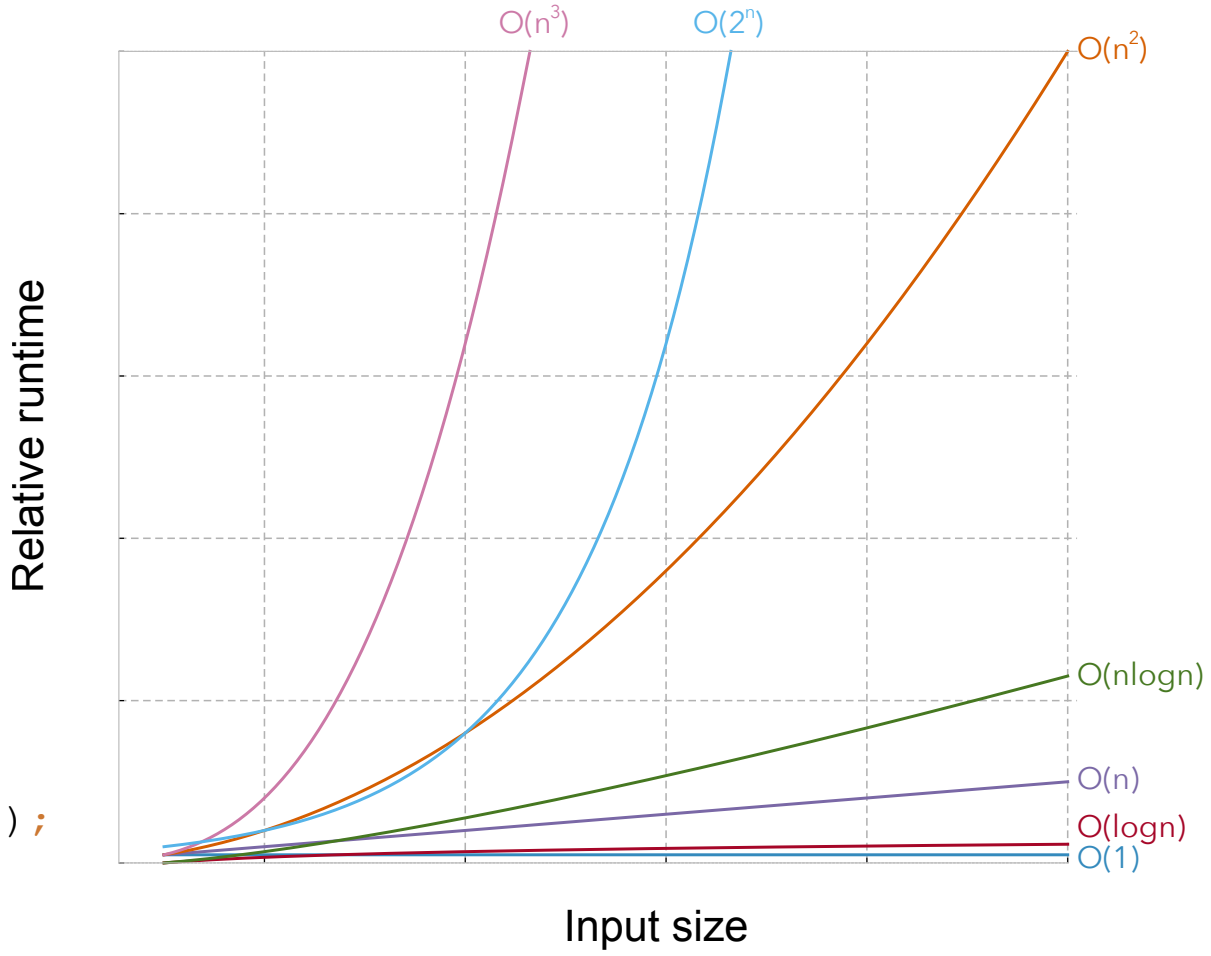
IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

$O(2^n)$

Các thuật toán có thời gian chạy O (2^n) thường là thuật toán đệ quy giải quyết một bài toán kích thước N bằng cách giải một cách đệ quy hai bài toán nhỏ hơn với kích thước N-1.

Ví dụ: Tìm dãy số Fibonacci

```
int fibonacci(int num)
{
    if (num <= 1)
        return num;
    return fibonacci(num - 2) + fibonacci(num - 1);
}
```

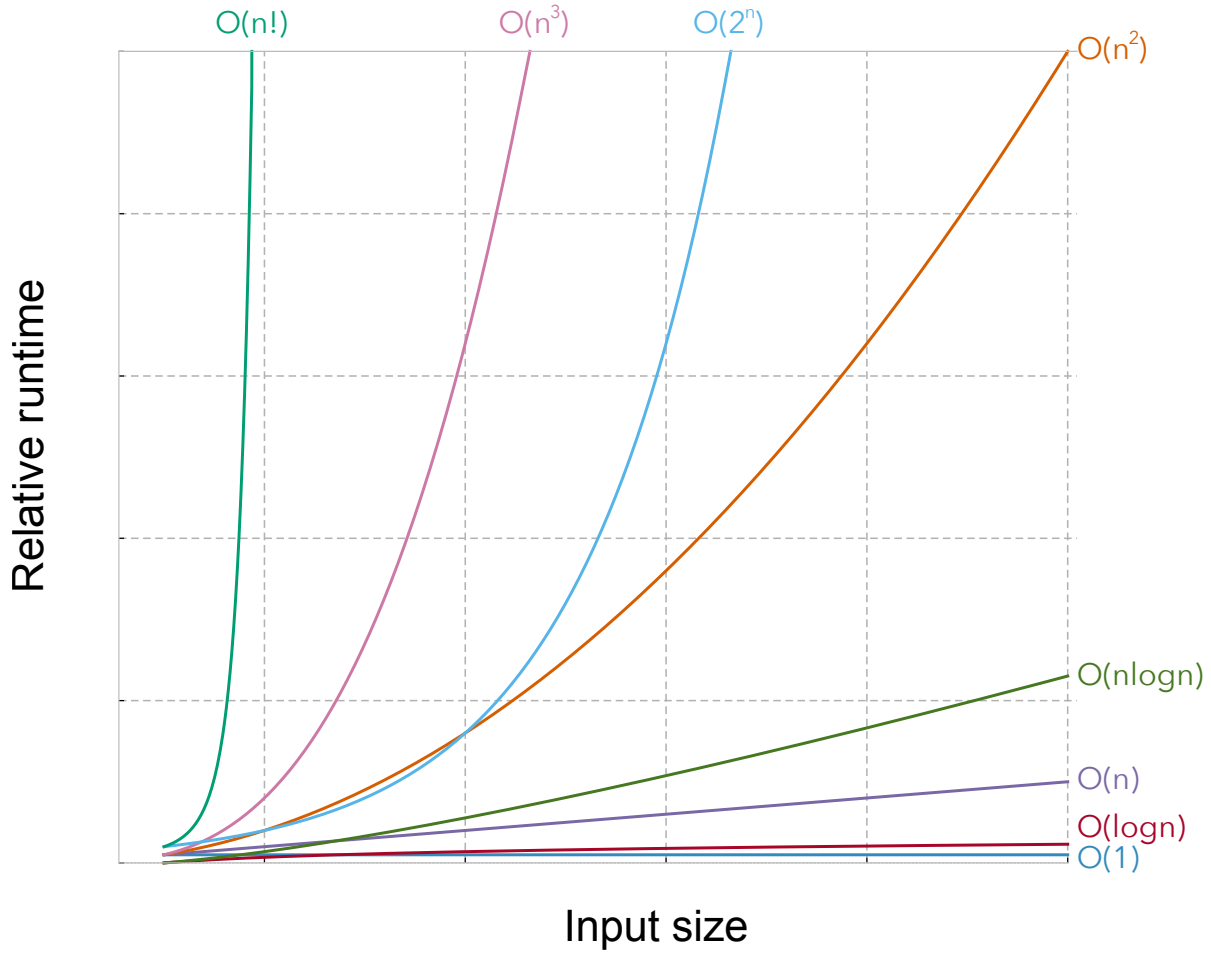


IV. ĐÁNH GIÁ ĐỘ PHỨC TẠP GIẢI THUẬT

O(n!)

Các thuật toán có thời gian giai thừa có thời gian thực thi dựa trên giai thừa của kích thước dữ liệu đầu vào. Thời gian tăng đáng kể ngay cả đối với dữ liệu có kích thước nhỏ

Ví dụ: Tạo ra hoán vị có thể có của tập hợp có n phần tử



V. CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Sắp xếp

1. Selection sort
2. Insertion sort
3. Bubble sort
4. Heap sort
5. Quick sort
6. Merge sort
7. Counting sort
8. Bucket sort

Tìm kiếm

1. Tìm kiếm tuyến tính
2. Tìm kiếm nhị phân

Cấu trúc dữ liệu

1. Danh sách liên kết
2. Ngăn xếp và hàng đợi
3. Cây
4. Bảng băm

VI. BÀI TẬP

Bài 1: Xác định mối quan hệ sau là đúng hay sai kèm giải thích.

$f(n)$	$g(n)$	Kết quả
$\log n^2$	$\log n + 5$	$f(n) = O(g(n))$
$\sqrt{n}$	$\log n^2$	$f(n) = O(g(n))$
$n \log n + n$	$\log n$	$f(n) = O(g(n))$
10	$\log 10$	$f(n) = O(g(n))$
2^n	$10n^2$	$f(n) = O(g(n))$
2^n	3^n	$f(n) = O(g(n))$

VI. BÀI TẬP

Bài 2: Giả sử chúng ta có 4 giải thuật với thời gian thực thi như bên dưới. Hãy cho biết mỗi thuật toán sẽ thay đổi như thế nào khi (a) tăng gấp đôi kích thước dữ liệu đầu vào hoặc (b) tăng kích thước dữ liệu đầu vào lên 1.

- n^3
- $100n^2$
- $n \log n$
- 2^n

VI. BÀI TẬP

Bài 3: Thuật toán sau tìm k số nguyên nhỏ nhất trong một mảng số nguyên.

Tìm độ phức tạp của thuật toán.

```
int selectkth(int a[], int k, int n) {
    int i, j, mini, tmp;
    for (i = 0; i < k; i++) {
        mini = i;
        for (j = i+1; j < n; j++)
            if (a[j]<a[mini])
                mini = j;
        tmp = a[i];
        a[i] = a[mini];
        a[mini] = tmp;
    }
    return a[k-1];
}
```

VI. BÀI TẬP

Bài 4: Tìm độ phức tạp của thuật toán cộng, nhân và chuyển vị ma trận n×n

```
// Cộng ma trận
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        a[i][j] = b[i][j] + c[i][j];
// Nhân ma trận
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = a[i][j] = 0; k < n; k++)
            a[i][j] += b[i][k] * c[k][j];
// Chuyển vị ma trận
for (int i = 0; i < n - 1; i++)
    for (int j = i+1; j < n; j++) {
        tmp = a[i][j];
        a[i][j] = a[j][i];
        a[j][i] = tmp;
    }
```

VI. BÀI TẬP

Bài 5: Liệt kê các thời gian thực thi sau theo thứ tự từ nhanh nhất tới lâu nhất

- n
- $n - n^3 + 7n^5$
- $n^2 + \lg n$
- n^3
- 2^n
- $\lg n$
- n^2
- $(\lg n)^2$
- $n \lg n$
- $\sqrt{n}$
- 2^{n-1}
- $n!$
- $\ln n$
- e^n
- $\lg \lg n$
- $n^{1+\varepsilon}$ với $0 < \varepsilon < 1$

THANK YOU FOR YOUR ATTENTIONS