Trường Đại học Công nghệ Thông tin
Đại học Quốc Gia TP.HCM

PHƯƠNG Pháp Thiết kế THUẬT Toán

CHIA ĐỂ TRỊ

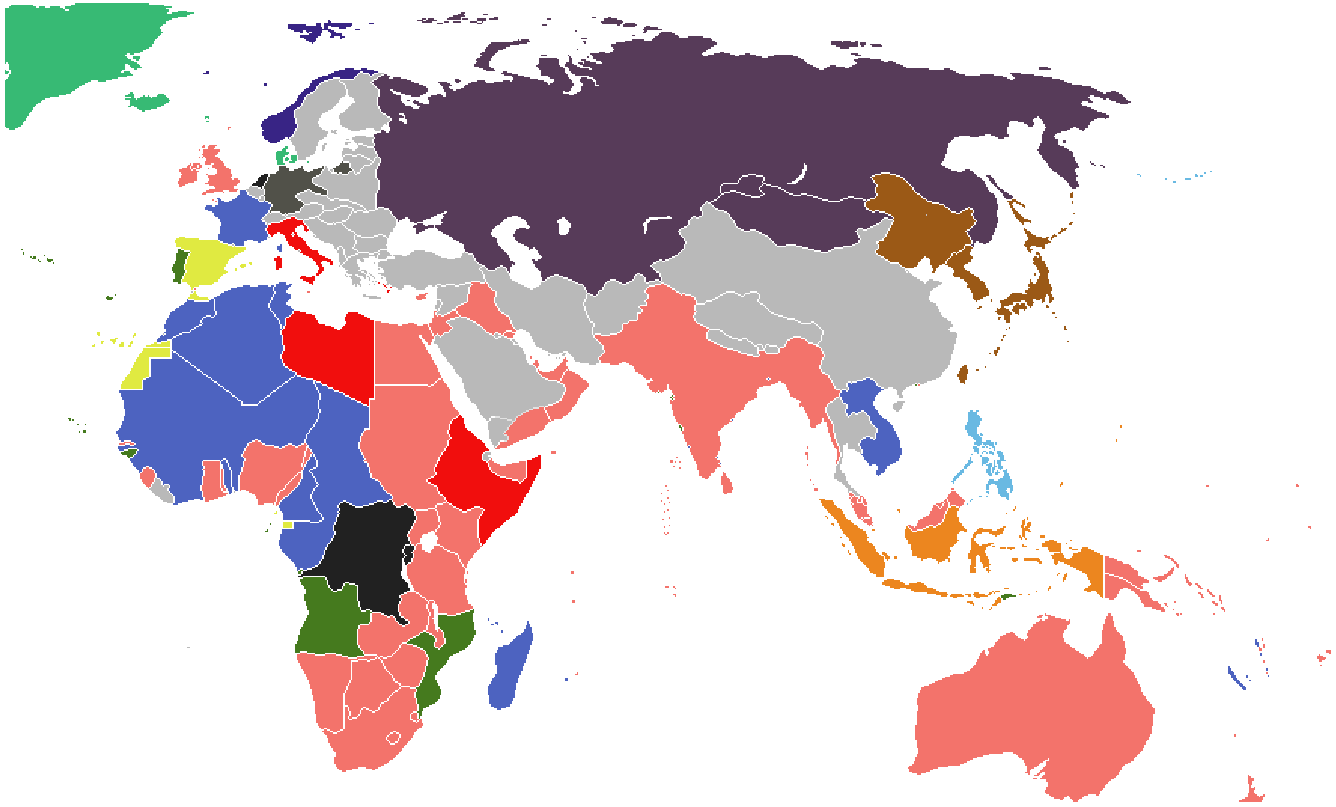
Lê Đoàn Phúc Minh
Nguyễn Duy Đạt

Nội dung

- Kỹ thuật chia để trị
- Các ứng dụng của phương pháp thiết kế thuật toán chia để trị
- Một số vấn đề khác

Kĩ thuật chia để trị

Các bạn hiểu Chia Để Trị là gì?



Khái niệm

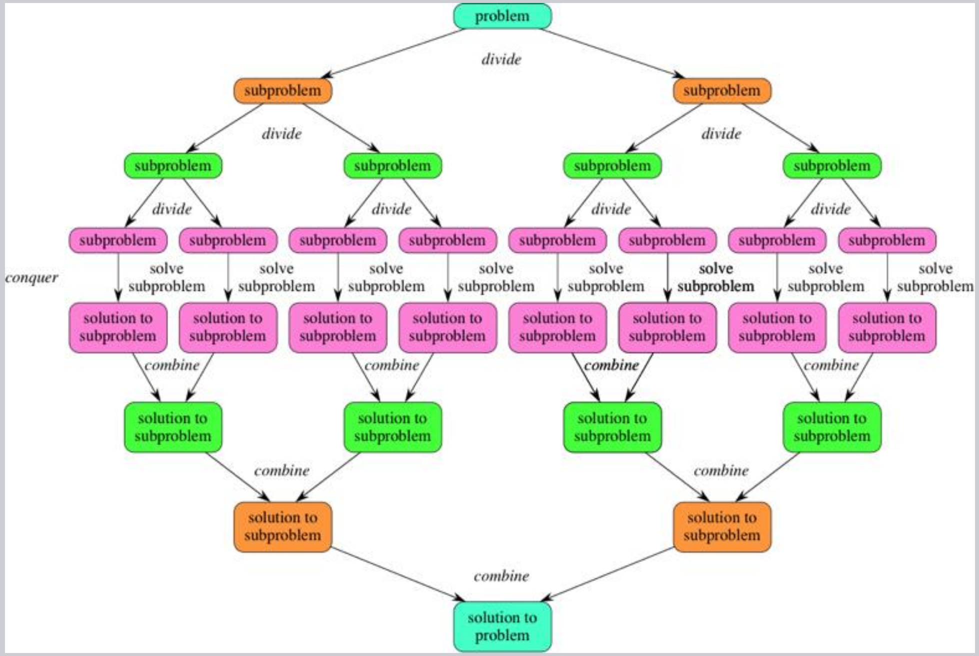
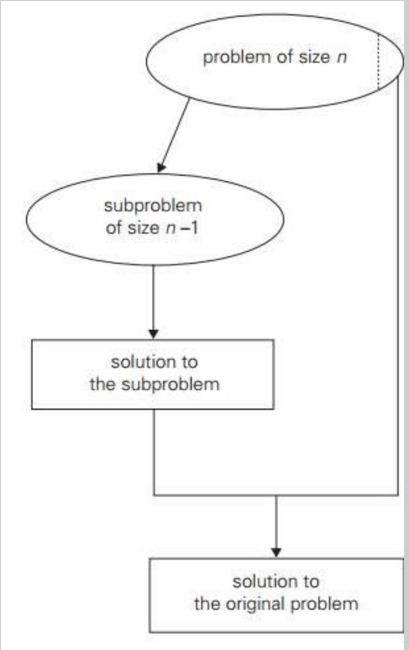
CHIA ĐỂ TRỊ

Chia (Divide): Chia bài toán ra thành các bài toán nhỏ hơn (subproblems). Về cơ bản thì những bài toán nhỏ này giống với bài toán ban đầu.

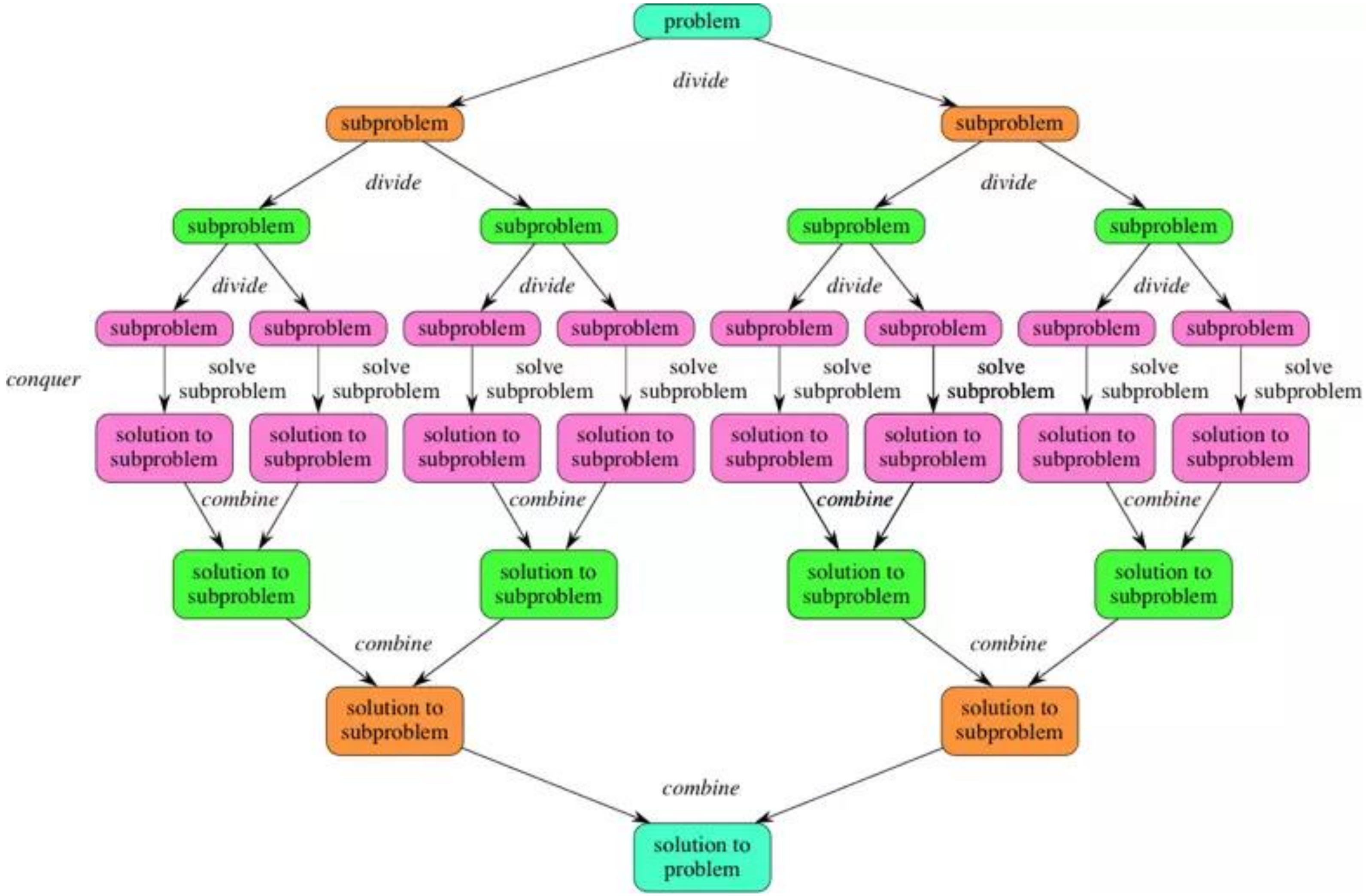
Trị (Conquer): Tiếp tục chia nhỏ đến khi giải trực tiếp được bằng hàm hoặc thư viện có sẵn, còn không thì tiếp tục chia nhỏ hơn nữa thành những bài toán con nhỏ hơn nữa.

Kết hợp (Combine): Kết hợp(sắp xếp, cộng lại, nhân lại,...) các kết quả từ bài toán con nhỏ nhất, để ra lời giải cho các bài toán con (subproblems), và cứ thế cuối cùng ra được lời giải cho bài toán ban đầu.

Nhánh của chia để trị

Chia để trị (Nhánh chính)	Giảm để trị (Nhánh phụ)
	
Chia bài toán ban đầu thành nhiều bài toán con nhỏ hơn Ví dụ: Merge Sort, Quick Sort	Đưa bài toán ban đầu thành bài toán nhỏ hơn Ví dụ: Tìm kiếm nhị phân, tính lũy thừa lớn

Hình ảnh miêu tả giải thuật chia để trị



Khi nào có thể sử dụng được chia để trị

- Khi bài toán có thể chia được thành các bài toán con với kích thước dữ liệu nhỏ hơn
- Khi các bài toán con có thể được giải quyết đệ quy (đôi khi việc đệ quy có thể không cần thiết)
- Khi kết quả của các bài toán con có thể được gộp lại để xử lý bài toán mẹ có chứa chúng
- Khi dữ liệu của các bài toán con sau khi chia ra từ bài toán mẹ hoàn toàn độc lập với nhau (nói cách khác là không overlap lên nhau)

Template (có sử dụng đệ quy)

```
DAC(a)
    // Nếu bài toán con có thư viện hoặc có hàm sẵn thì giải thẳng luôn! Còn không thì phải chia nhỏ ra!
    if (small(a))
        return solution(a)
    else
        // Chia bài toán thành n bài toán con (n là bao nhiêu tùy vào thuật toán)
        part = divide(a)

        // Xử lí bài toán con
        // Tùy vào số lượng bài toán con mà số lượng bài toán phải trị có thể nhiều hơn
        part[0] = DAC(part[0])
        part[1] = DAC(part[1])
        //...

        // Kết hợp kết quả từ bài toán con để suy ra kết quả bài toán trước đó
        result = combine(part)

    return result
```

Lưu ý: Có thể code thuật toán sử dụng phương pháp thiết kế thuật toán này mà không cần đến đệ quy

Master Theorem (Nhắc lại)

$$T(n) = aT(n/b) + f(n),$$

+ T(n) là độ phức tạp của bài toán
+ a là số bài toán sau mỗi lần chia để trị
+ b là số lần giảm đi của độ phức tạp sau mỗi lần chia
+ a >= 1, b > 1
+ n là lũy thừa của b
+ f(n) là hàm tính thời gian phân chia
 n thành n/b và gộp các kết quả lại

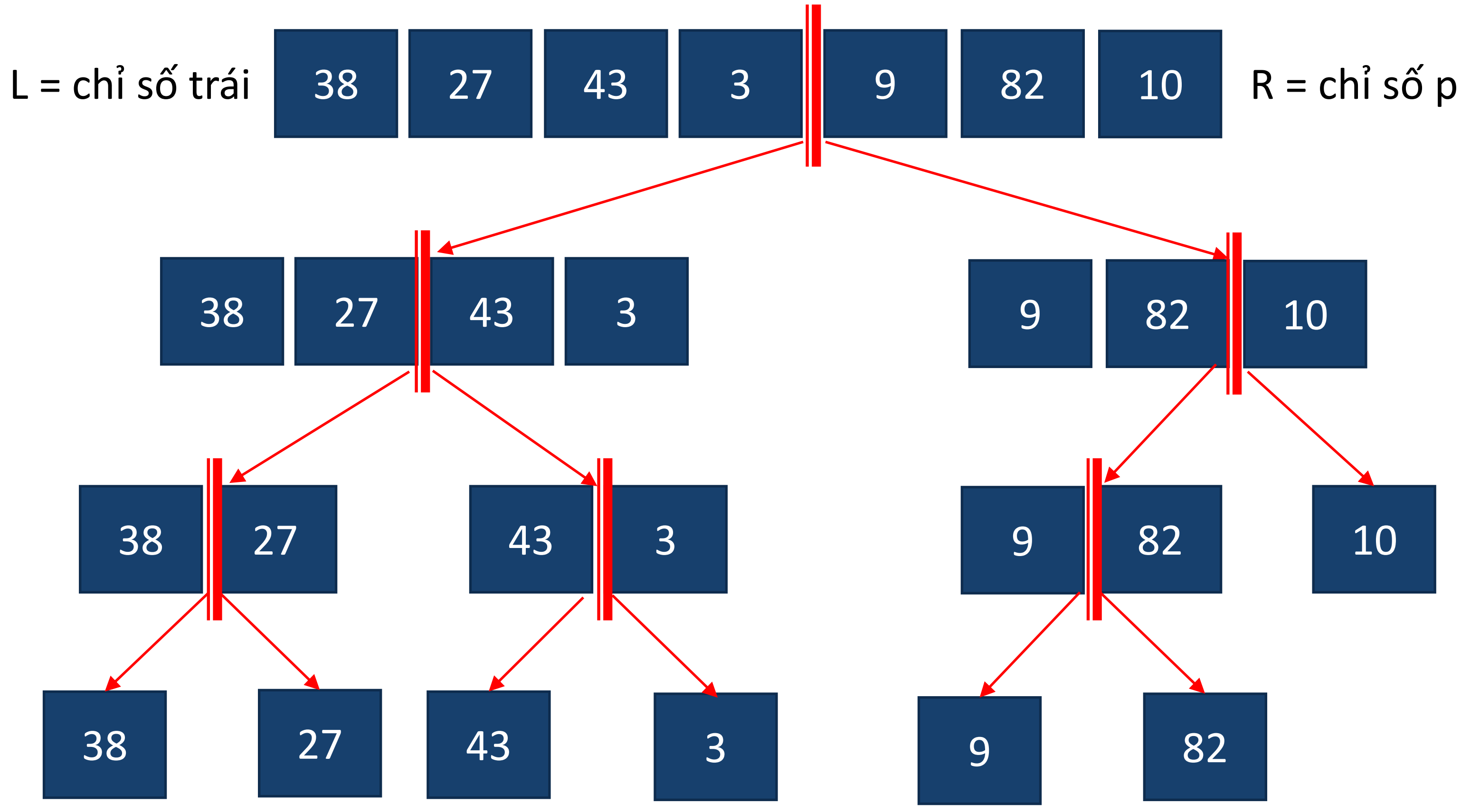
+ Nếu $f(n) \in O(n^d)$, d >= 0 -> $T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$

Các ứng dụng của phương pháp thiết kế thuật toán chia để trị

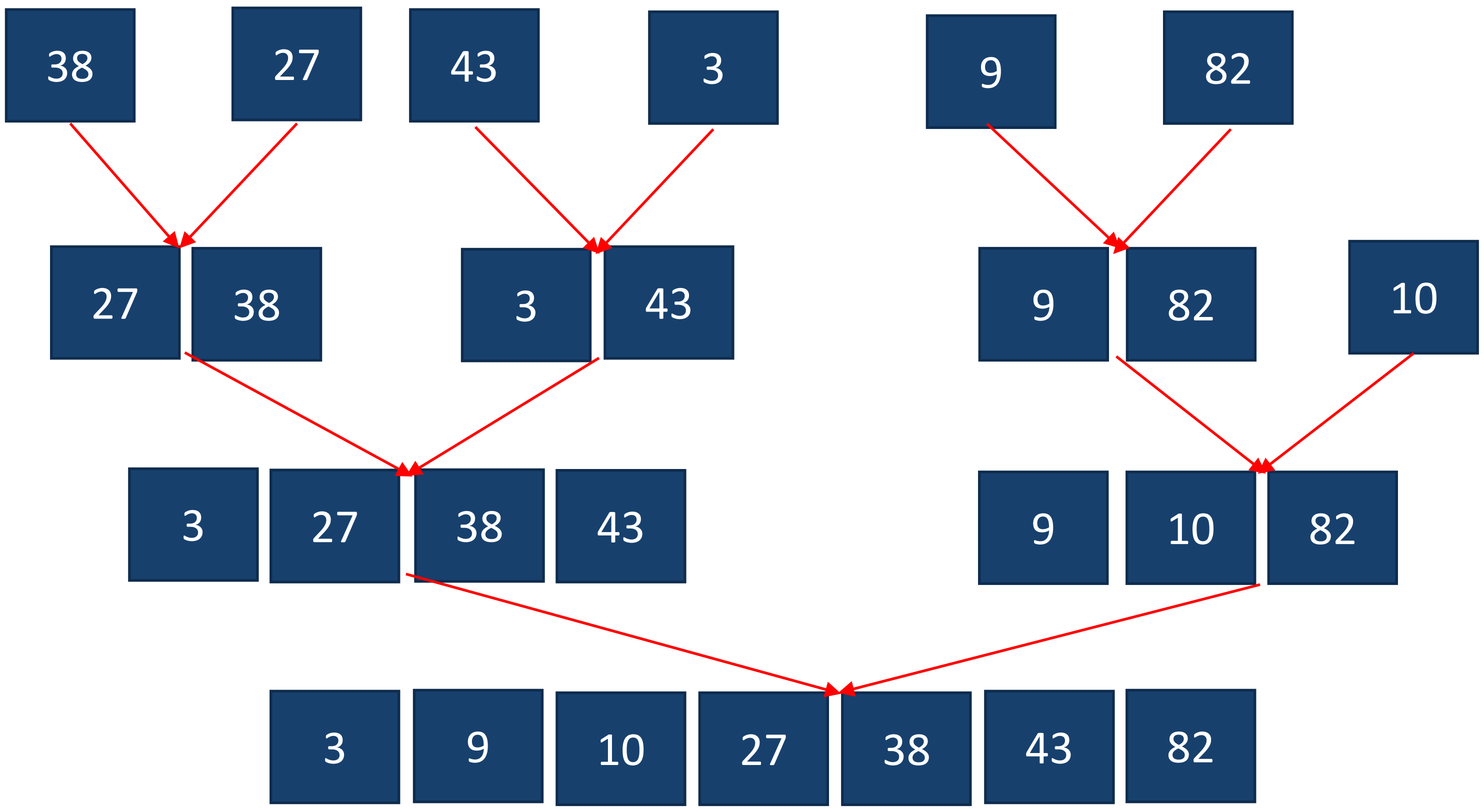
Mục lục

+ Chia : Chia đôi mảng
+ Trị: Sử dụng đệ quy sắp xếp 2 mảng con
+ Gộp: Gộp 2 mảng con với thời gian tuyến tính

Các bạn hãy cho biết dãy sau hoạt động theo cơ chế Merge Sort như thế nào ?



Các bạn hãy cho biết dãy sau hoạt động theo cơ chế Merge Sort như thế nào ?



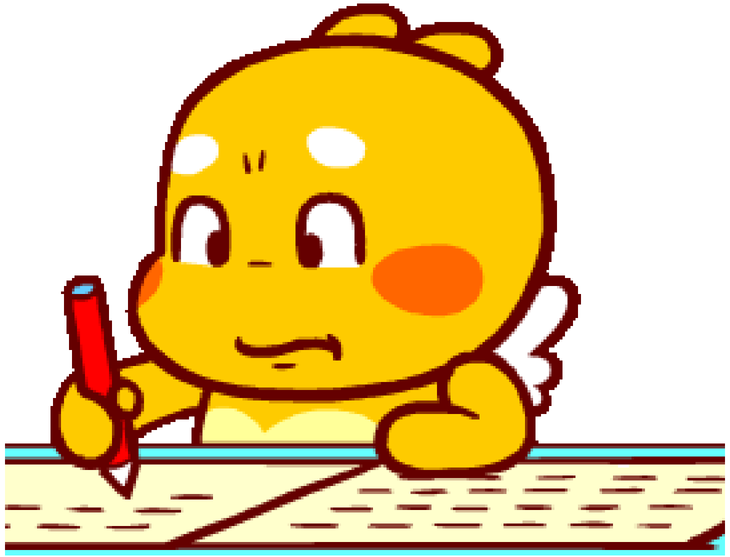
Dựa vào Master Theorem, hãy thử tính độ phức tạp của mergeSort!

Master Theorem

$$T(n) = aT(n/b) + f(n),$$

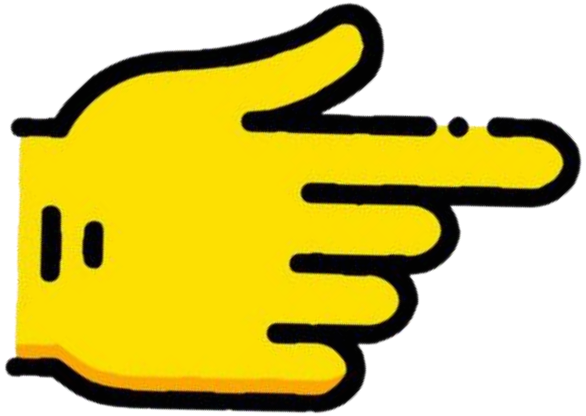
+ T(n) là độ phức tạp của bài toán
+ a là số bài toán sau mỗi lần chia để trị
+ b là số lần giảm đi của độ phức tạp sau mỗi lần chia
+ a >= 1, b > 1
+ n là lũy thừa của b
+ f(n) là hàm tính thời gian phân chia n thành n/b và gộp các kết quả lại

+ Nếu $f(n) \in O(n^d)$, d >= 0 ->
$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

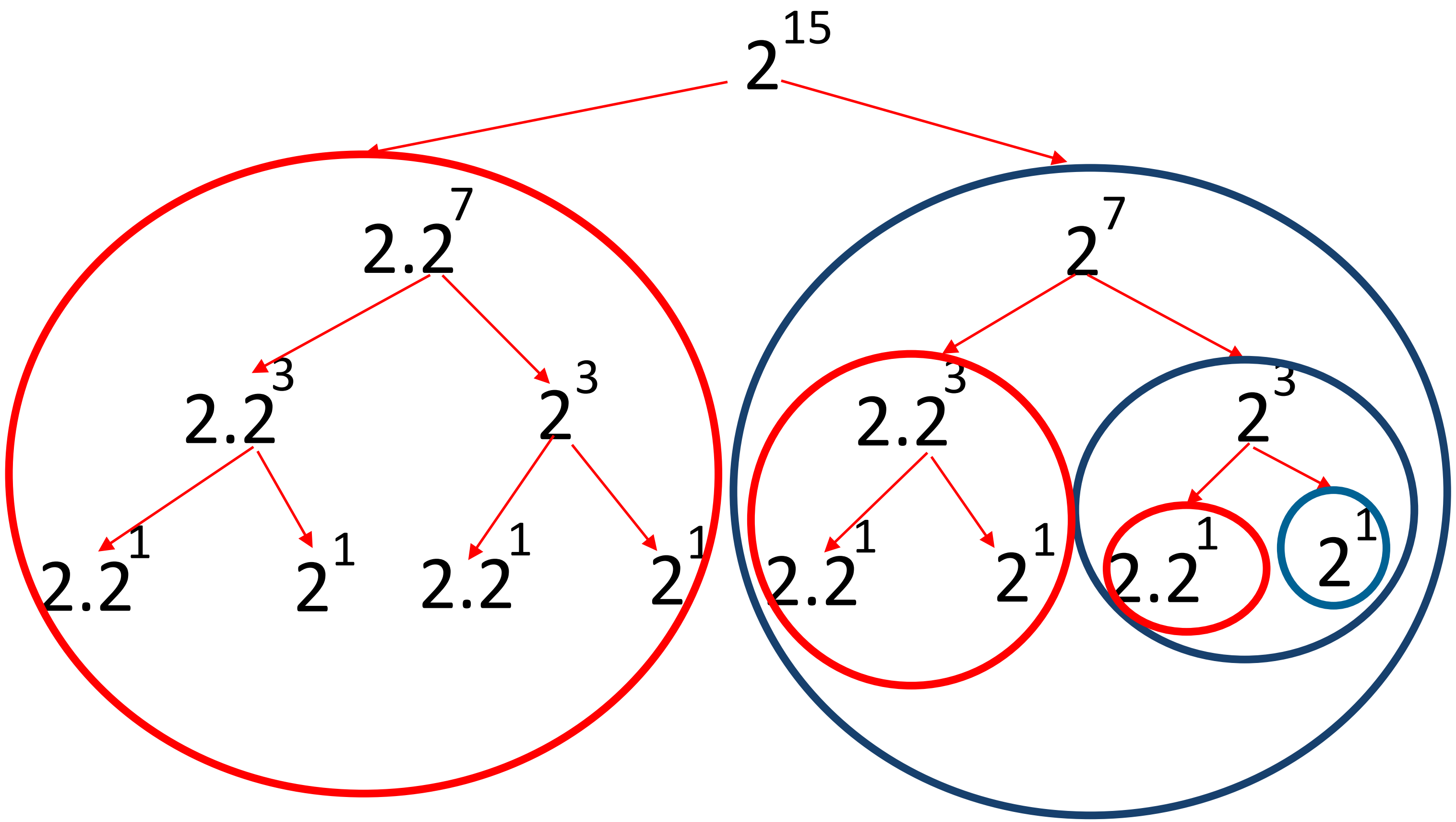


Tính 2^9

Các bạn hãy thử viết chương trình tính lũy thừa theo cách đơn giản nhất (sử dụng for hoặc while)



Tính 2^{15}



+ Lũy thừa cần tính sẽ được chia làm 2 nhánh
+ Việc đi tính sẽ chỉ thực hiện trên 1 nhánh con và nhánh còn lại sử dụng kết quả lại
(**Mô tả như trên hình:** vòng tròn đỏ sẽ sử dụng lại kết quả từ vòng tròn xanh)

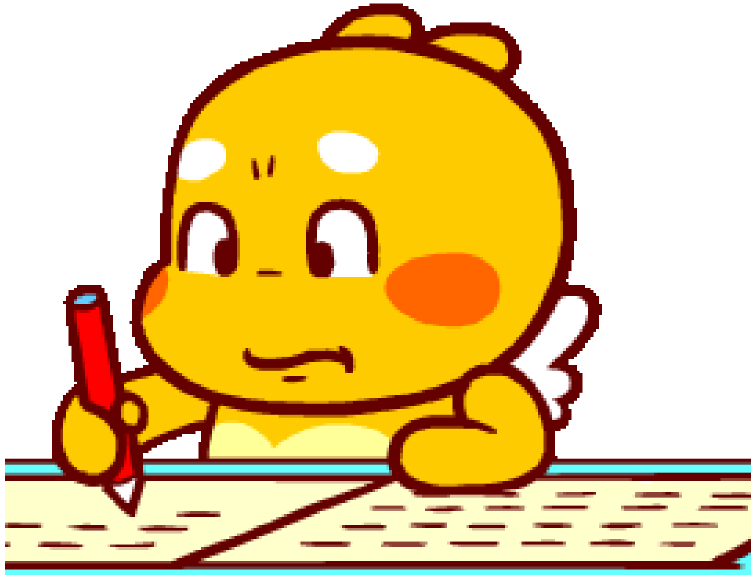
Dựa vào Master Theorem, hãy thử tính độ phức tạp của tính lũy thừa!

Master Theorem

$$T(n) = aT(n/b) + f(n),$$

+ T(n) là độ phức tạp của bài toán
+ a là số bài toán sau mỗi lần chia để trị
+ b là số lần giảm đi của độ phức tạp sau mỗi lần chia
+ a >= 1, b > 1
+ n là lũy thừa của b
+ f(n) là hàm tính thời gian phân chia n thành n/b và gộp các kết quả lại

+ Nếu $f(n) \in O(n^d)$, d >= 0 -> $$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$



$\downarrow^1_9$ x^y x*y A B

Các bạn hãy thử viết chương trình tính lũy thừa a^x (% MOD = 1e9+7) bằng chia để trị (a và x rất lớn)

Bài toán: Cho 2 số tự nhiên, hãy cho biết tích của 2 số đó

Phương pháp ngây thơ:

```
        12345
*        6789
-------------
       111105
+     98760
     86415
    74070
-------------
= 83810205 (Kết quả)
```

Độ phức tạp thuật toán: O(n*m)

Kỹ thuật nhân nhanh Karatsuba:

Đầu tiên, ta có x, y là số có lần lượt n_1, n_2 chữ số. $\forall m \in \mathrm{N}$ và $n_1, n_2 > 0$, ta viết lại hai số đã cho thành:

$$x = x_1 10^m + x_0 \text{ và } y = y_1 10^m + y_0$$

$$(x_0, y_0 < 10^m)$$

Gọi $z_2 = x_1 y_1$, $z_1 = x_1 y_0 + x_0 y_1$ và $z_0 = x_0 y_0$.

Lúc này, ta có:

$$x*y = (x_1 10^m + x_0)(y_1 10^m + y_0) = z_2 10^{2m} + z_1 10^m + z_0$$

Có thể viết lại z_1 thành $z_1 = (x_1 + x_0)(y_1 + y_0) – z_2 – z_0$.

Ứng dụng của kỹ thuật chia để trị và thuật toán đầy đủ:

$xy = ?$

$xy = z_2 10^{2m} + z_1 10^m + z_0$

$z_2 = z_{2'} 10^{2m} + z_{1'} 10^m + z_{0'}$

$z_1 = x_1' y_1' - z_2 - z_0$

$z_0 = z_{2'} 10^{2m} + z_{1'} 10^m + z_{0'}$

...

$x_1' y_1' = z_{2'} 10^{2m} + z_{1'} 10^m + z_{0'}$

...

...

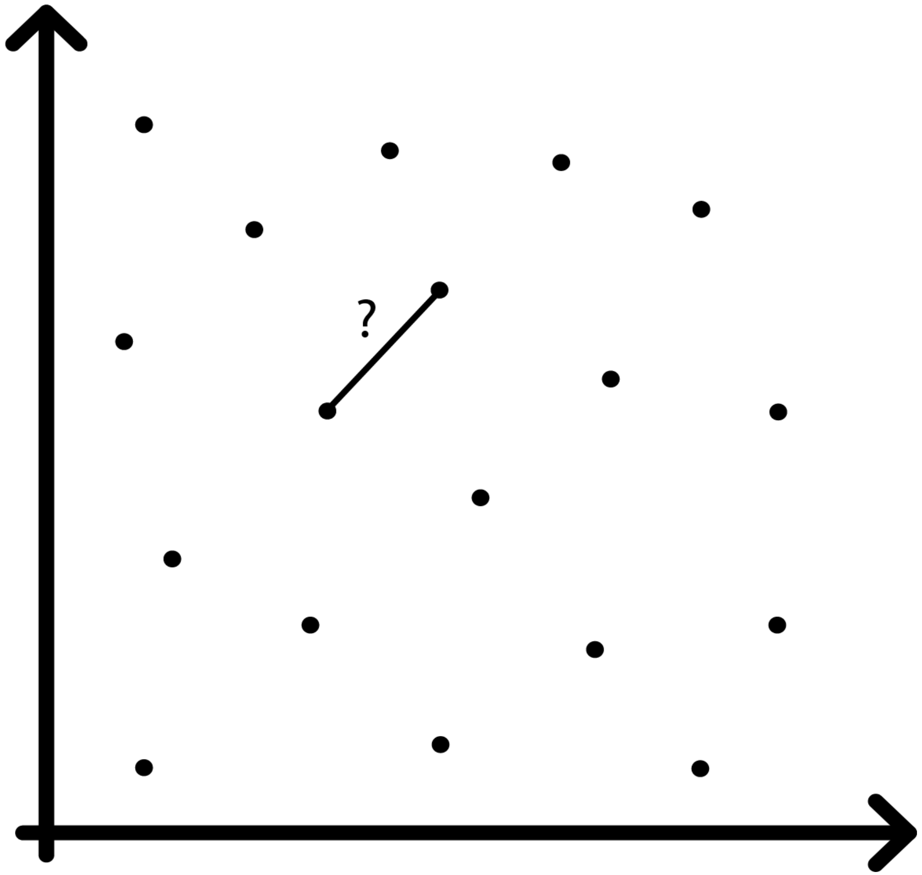
Độ phức tạp thuật toán:
$T(n) = 3T(n/2) + O(n)$

Lưu ý: x1'y1' = $(x_1 + x_0)(y_1 + y_0)$

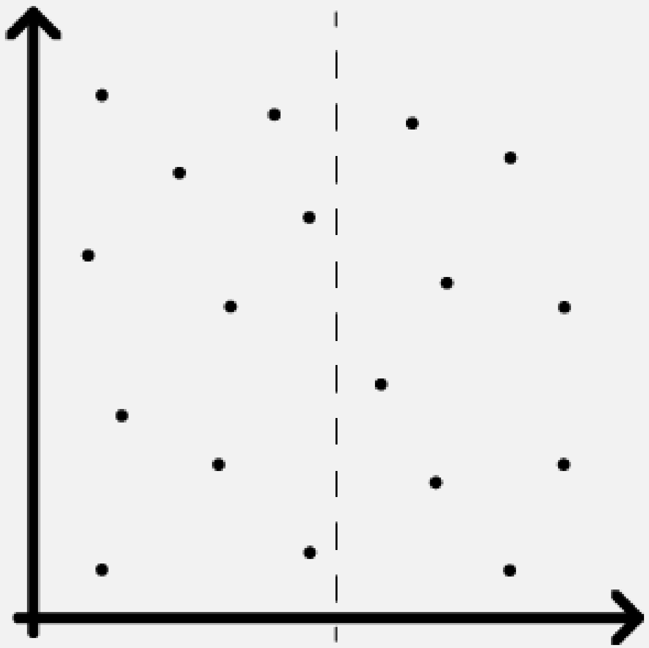
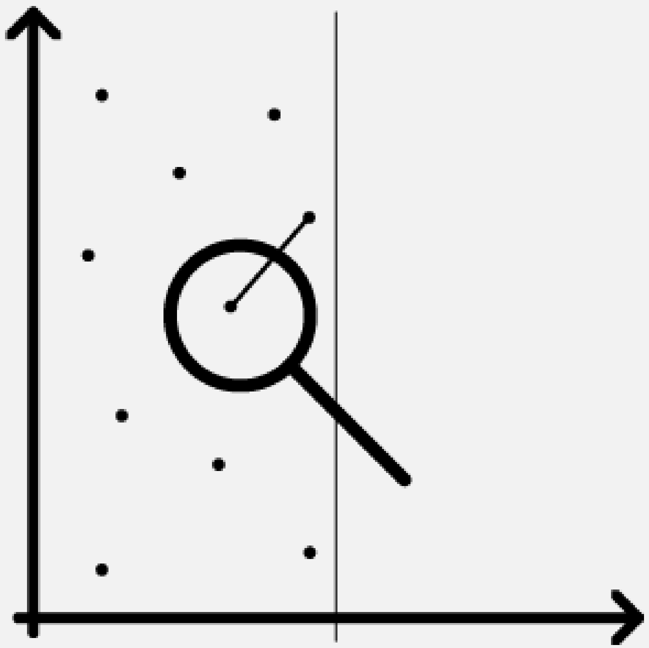
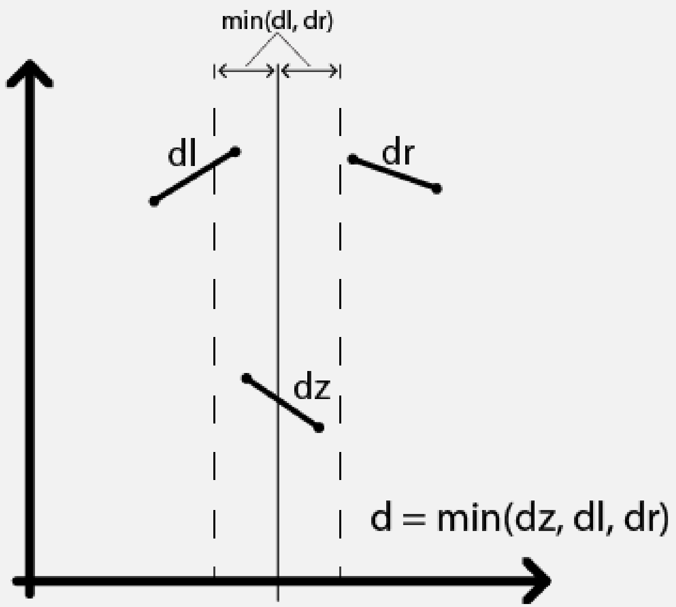
Theo các bạn, với phép tính 1234567890 * 987654321 và m = 3, thuật toán này sẽ phải thực hiện công đoạn chia và công đoạn trị bao nhiêu lần ?

Bài toán:

Cho một mặt phẳng gồm N điểm, hãy tìm cặp điểm gần nhau nhất và khoảng cách của chúng.

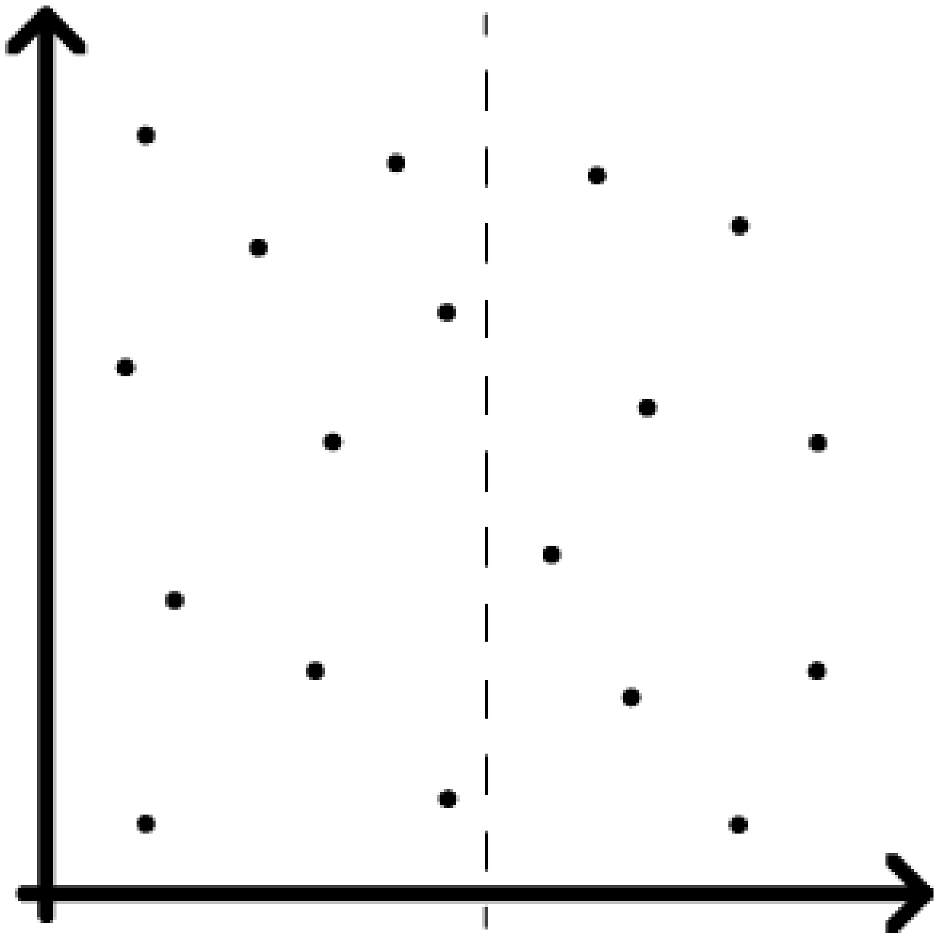


Tổng quan thuật toán sử dụng phương pháp chia để trị

Chia	Trị	Gộp
		

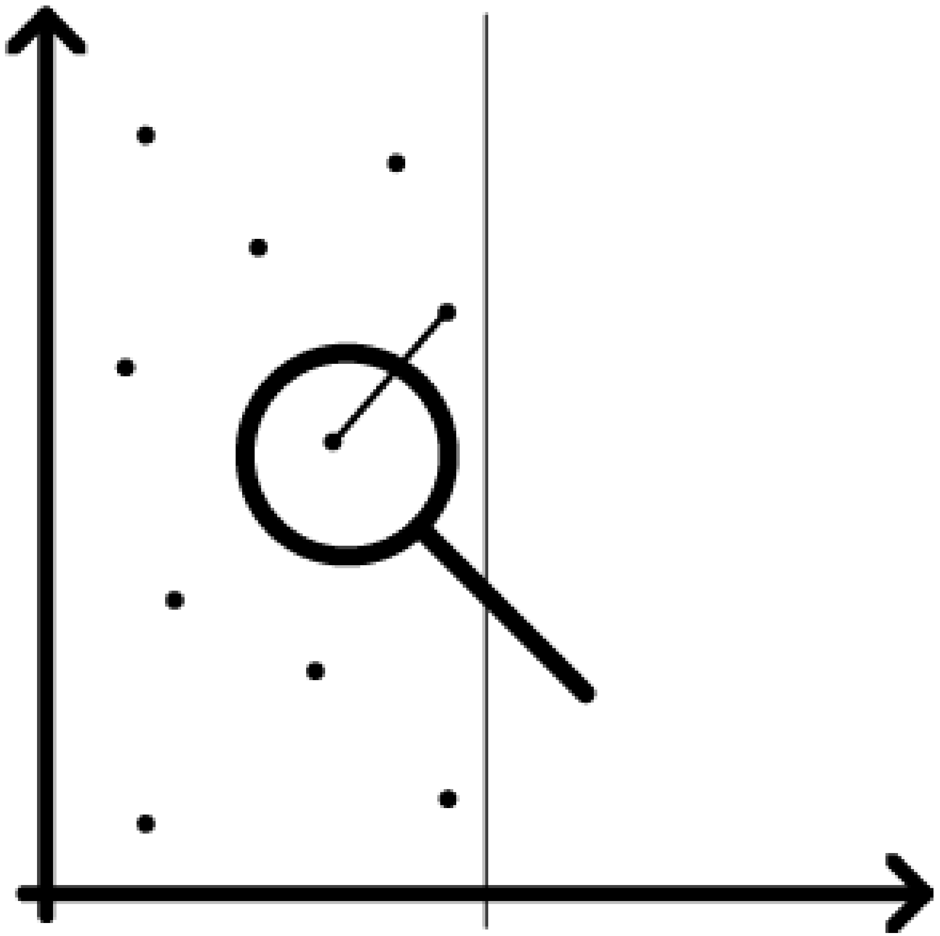
Chia:

Chọn điểm giữa nhất trong tập hợp các điểm và chia tập này ra thành 2 tập con.



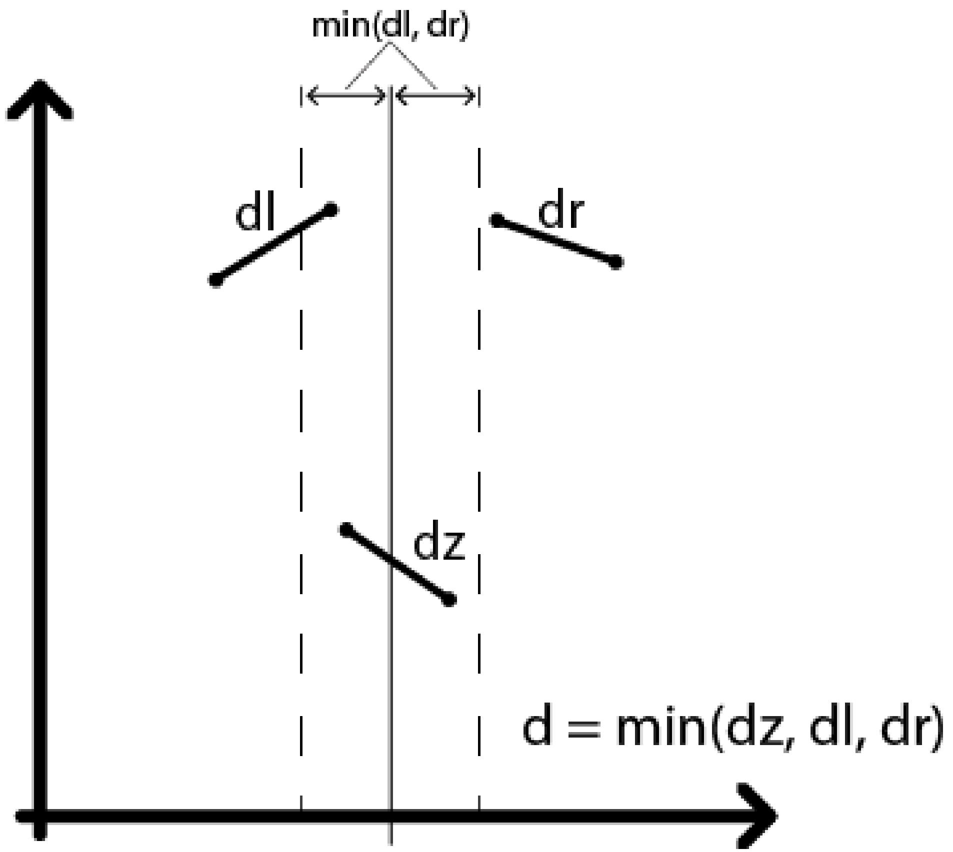
Trị:

Khi chỉ còn dưới 3 điểm trong tập hợp các điểm, duyệt hết qua các cặp đỉnh để lấy khoảng cách giữa 2 đỉnh nhỏ nhất cho giai đoạn gộp.

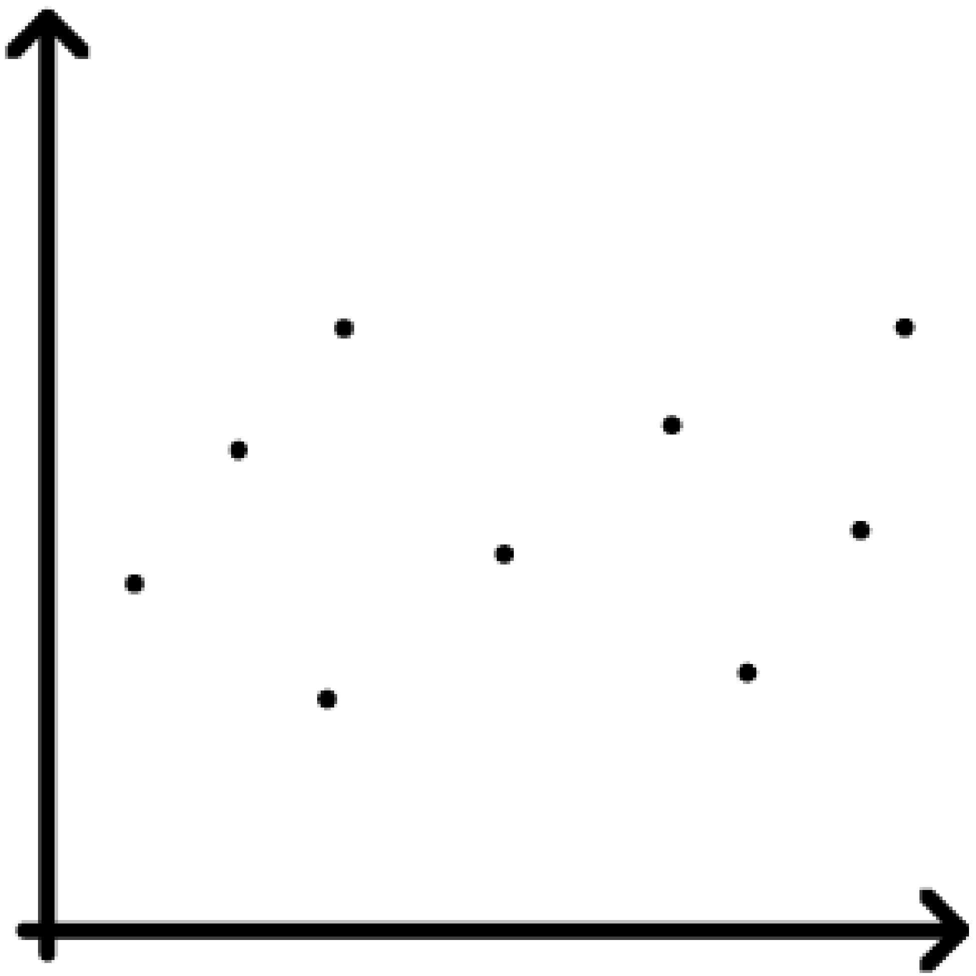


Gộp:

Tìm và duyệt khoảng cách nhỏ nhất của các cặp điểm trong số các điểm có tọa độ x nằm trong khoảng [mid – min(dl, dr); mid + min(dl, dr)] và so sánh nó với kết quả đã so sánh ở bước trước đó.

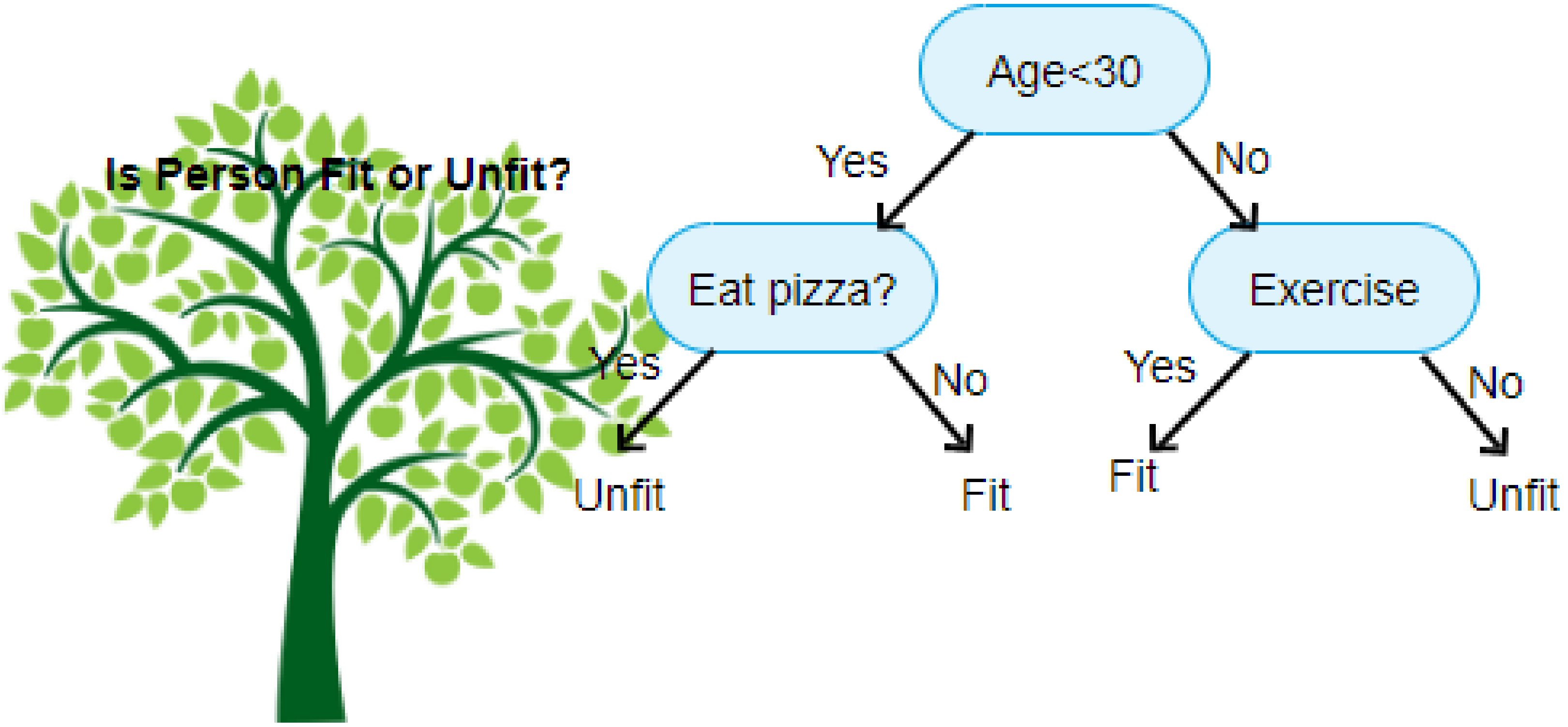


Với các điểm đã cho dưới đây, thuật toán sẽ thực hiện thao tác chia và thao tác trị bao nhiêu lần ?



Một số ứng dụng thực tế khác

Cây quyết định (*decision tree*)

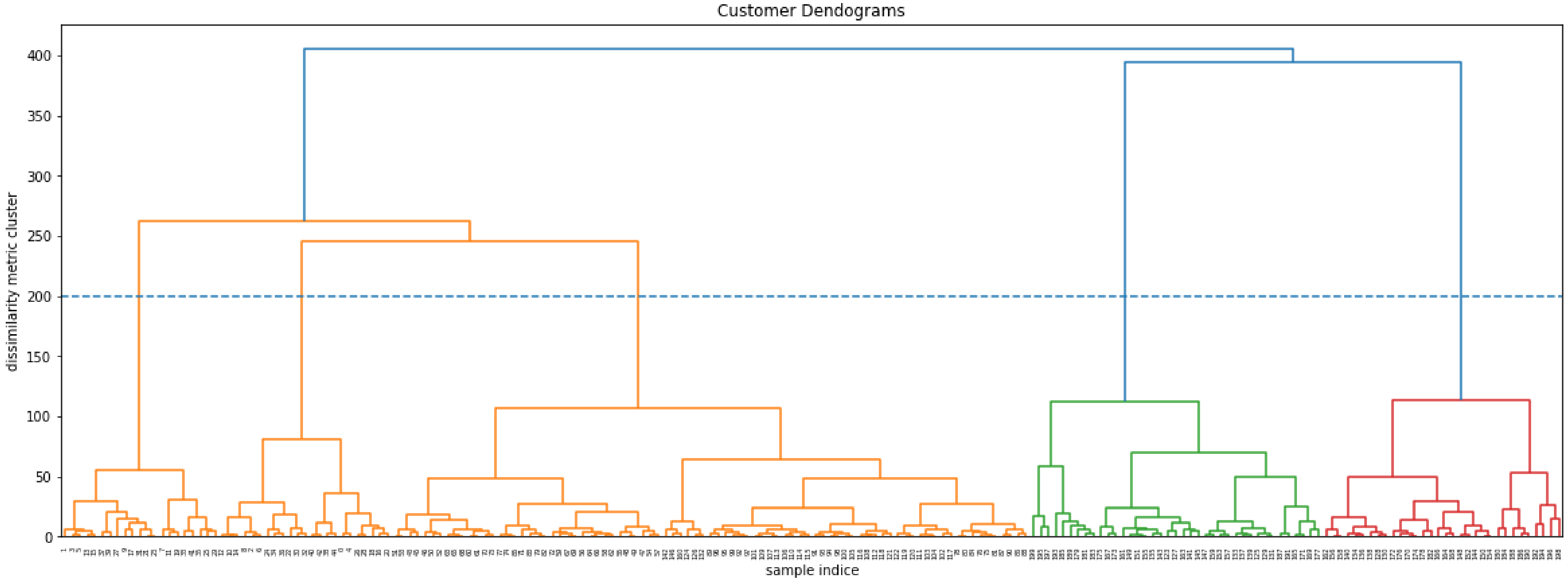


Một số ứng dụng thực tế khác

Hierarchical Clustering (*phân cụm phân cấp*)

Chiến lược hợp nhất (agglomerative)

Chiến lược phân chia (divisive)



Một số vấn đề khác

Ưu điểm và nhược điểm:

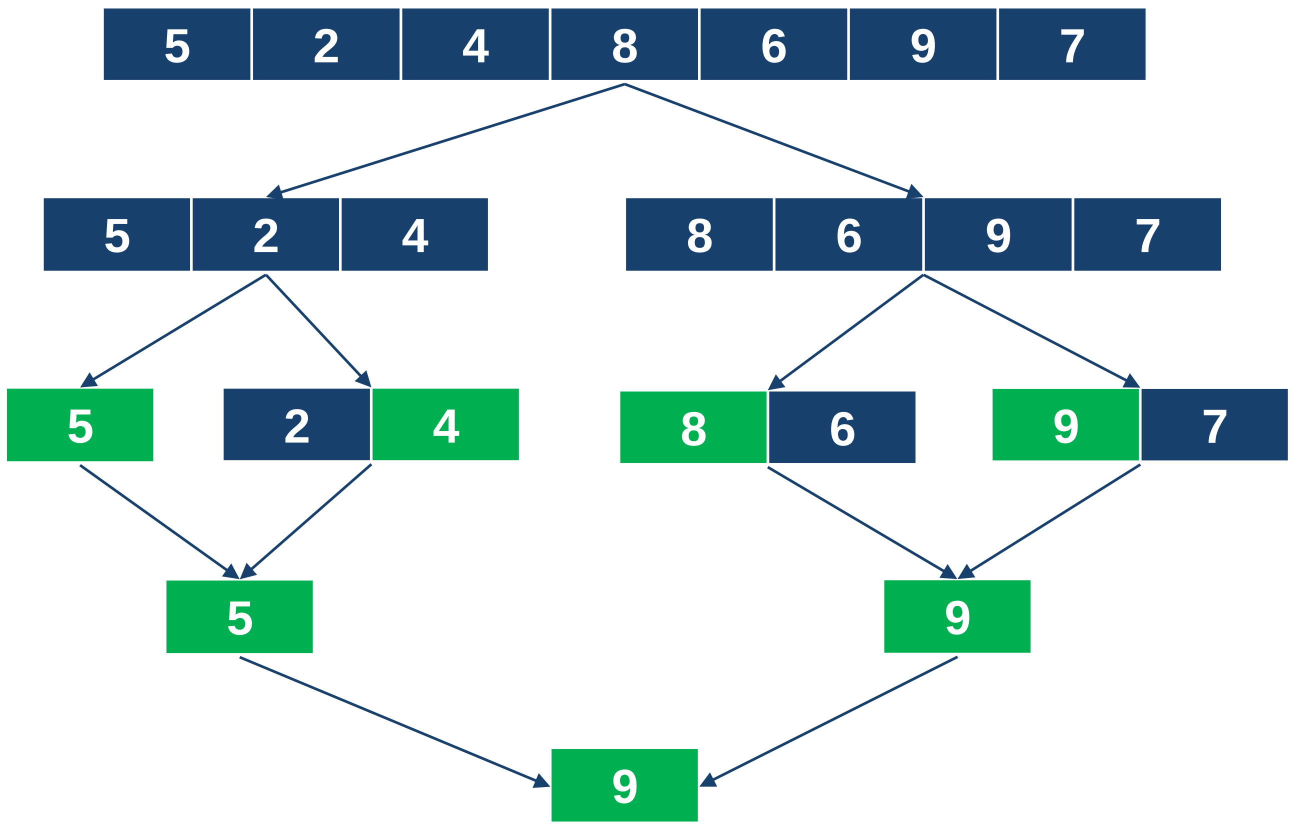
Ưu điểm	Nhược điểm
• Cho phép chúng ta dễ dàng giải quyết những bài toán khó xử lý và nhìn nhận. • Giúp giảm thiểu thời gian và chi phí thực thi đi gấp nhiều lần. • Sử dụng hiệu quả bộ nhớ đệm CPU. • Dễ dàng áp dụng trong việc lập trình song song[1]	• Đệ quy có thể trở thành rào cản hiệu năng. (Nếu có cài đặt) • Kết quả của những vấn đề đã giải quyết không thể được lưu lại cho những lần yêu cầu tiếp theo. • Việc lựa chọn kích thước cho bài toán con để trị quyết định khá nhiều đến hiệu năng thuật toán

[1]: Lecture_Parallelism_DC.pdf (berkeley.edu)

Ví dụ về bài toán không ứng dụng được:

Tìm Min/Max trong dãy số

Nguyên nhân: Phương pháp chia để trị cho ra thuật toán không tốt hơn phương pháp duyệt



Ví dụ về bài toán không ứng dụng được:

Bài toán xếp lịch

Nguyên nhân: Do bản chất các bài toán con phải tham chiếu nhau

Customer #	Start	End
1	7:30 AM	8:30 AM
2	8:00 AM	10:00 AM
3	9:00 AM	10:30 AM
4	11:00 AM	12:00 PM
5	11:30 AM	1:00 PM

Tài liệu tham khảo

- Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2014
- Divide and Conquer Algorithm | Introduction - Geeksforgeeks.org
- Divide and Conquer Introduction - javatpoint.com
- Wikipedia:
  - Divide and Conquer
  - Karatsuba Algorithm