

**DIVIDE, DECREASE,
TRANSFORM AND
CONQUER**

LET'S START

NỘI DUNG

- LÝ THUYẾT
- ỨNG DỤNG VÀO CÁC BÀI TOÁN THỰC TẾ
- TRÒ CHƠI

**GOALS
PAINS
VIEW
HABIT** → **THEORY**

TOPICS

1

Divide and conquer

2

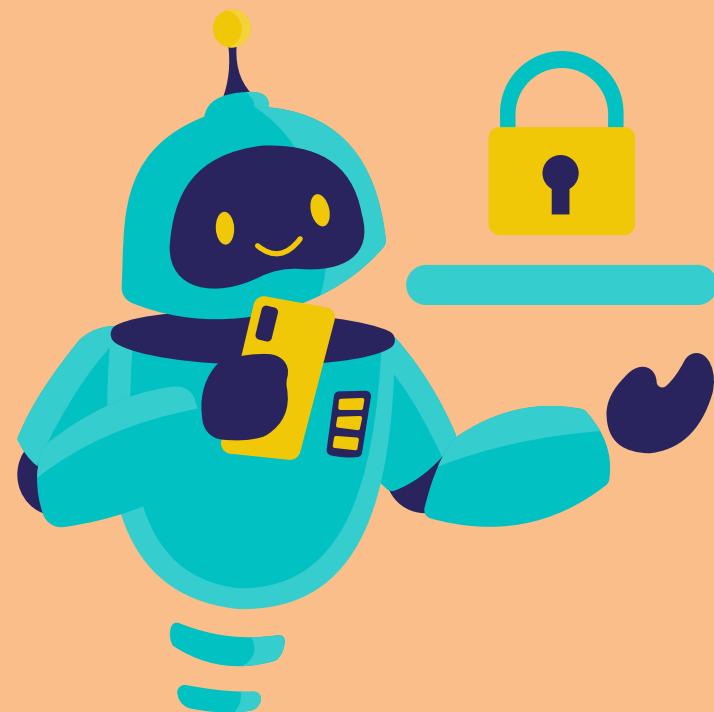
Decrease and conquer

3

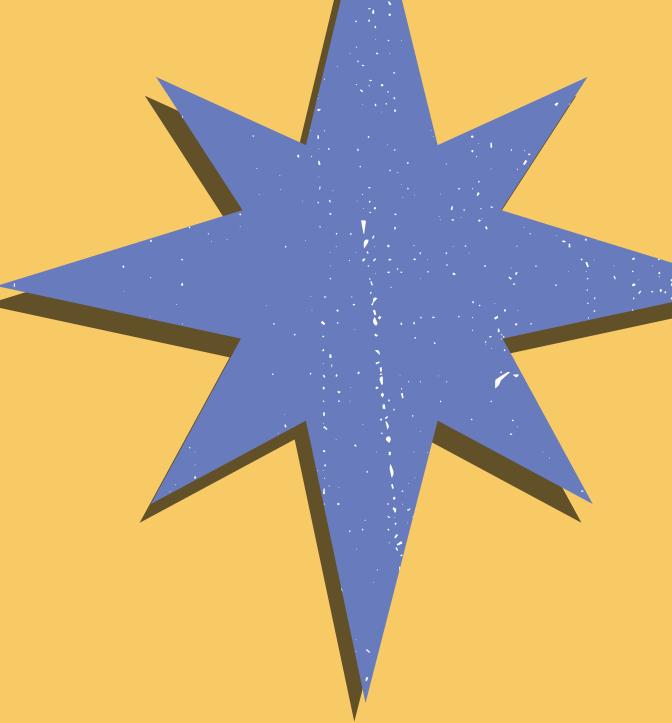
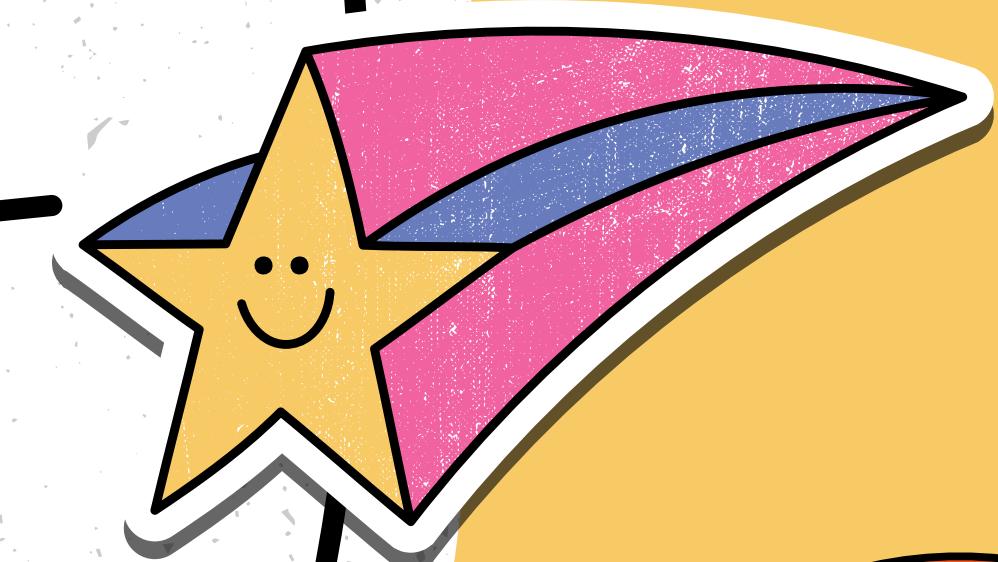
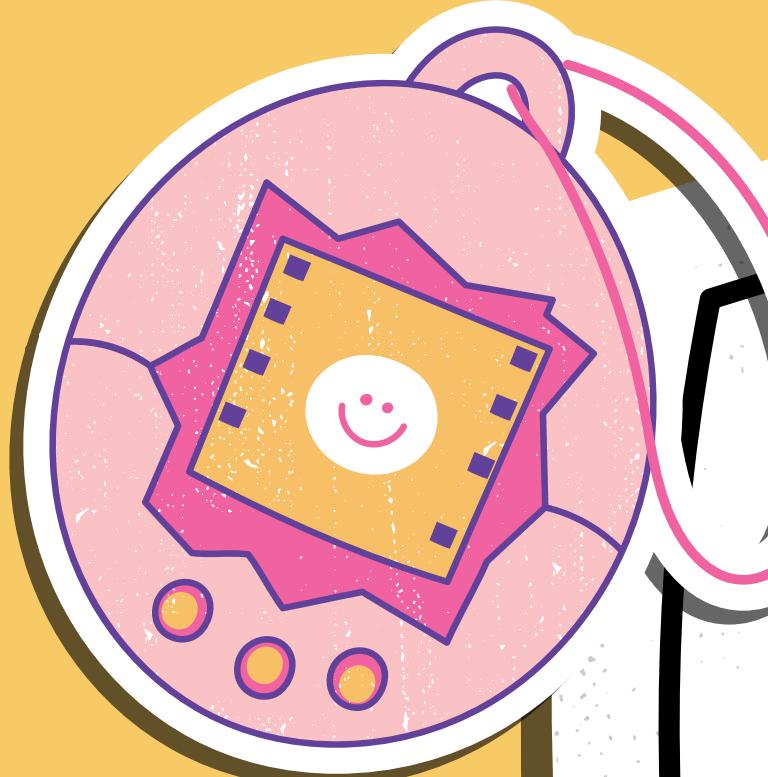
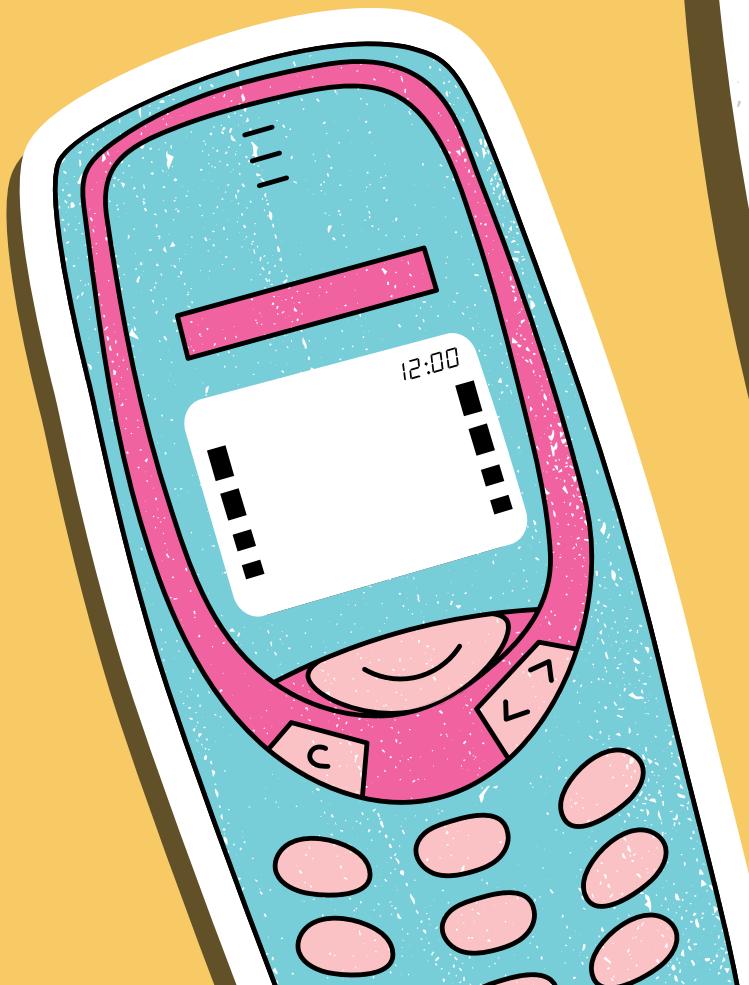
Transform and conquer

KHÁI NIỆM CHIA ĐỀ TRỊ

- GIẢI QUYẾT VẤN ĐỀ LIÊN QUAN ĐẾN VIỆC CHIA NHỎ MỘT VẤN ĐỀ PHỨC TẠP THÀNH CÁC PHẦN NHỎ HƠN.
- KẾT HỢP CÁC GIẢI PHÁP ĐỂ GIẢI QUYẾT VẤN ĐỀ BAN ĐẦU.



CÁC CÔNG ĐOẠN

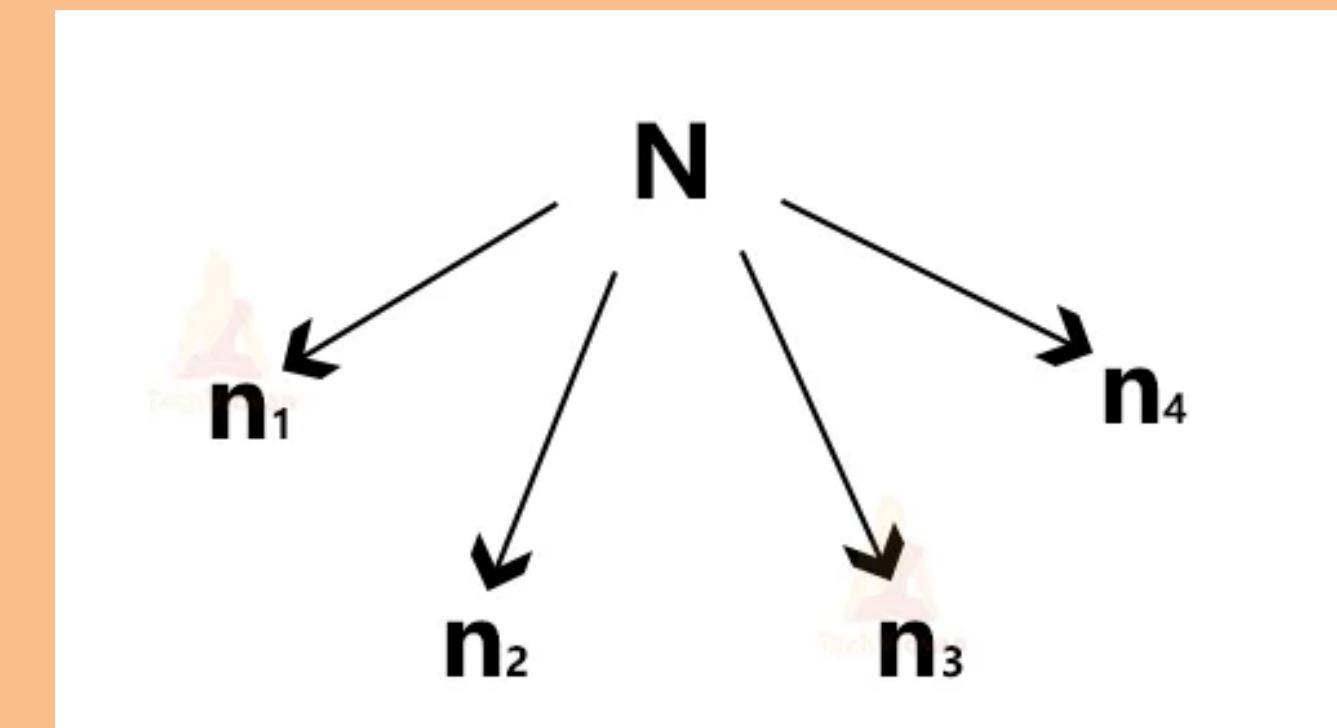


1. CHIA:

- CHIA NHỎ VẤN ĐỀ BAN ĐẦU THÀNH CÁC VẤN ĐỀ NHỎ HƠN.
- MỖI VẤN ĐỀ NHỎ PHẢI ĐẠI DIỆN CHO MỘT PHẦN CỦA VẤN ĐỀ TỔNG THỂ.
- MỤC TIÊU LÀ CHIA BÀI TOÁN CHO ĐẾN KHI TRỞ THÀNH BÀI TOÁN CƠ SỞ.



shutterstock.com · 2197389279



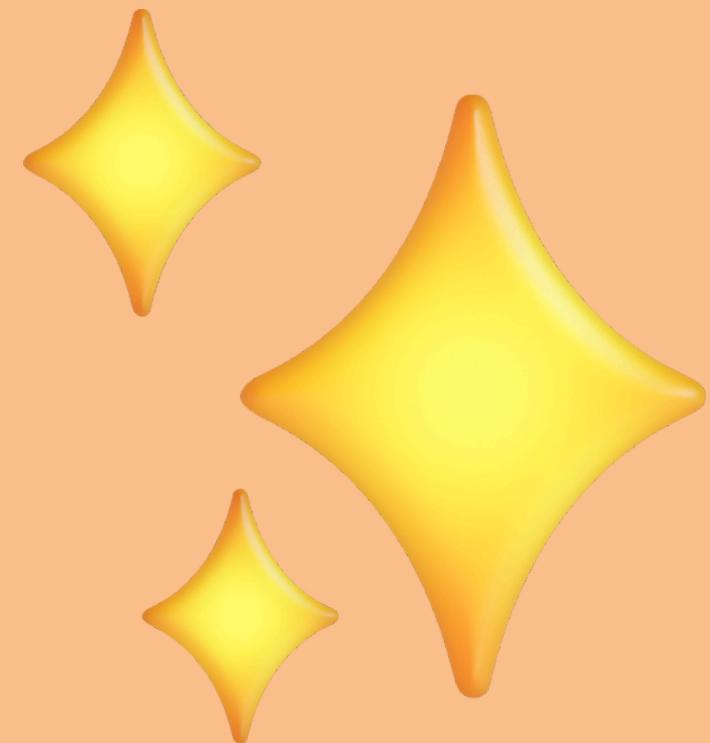
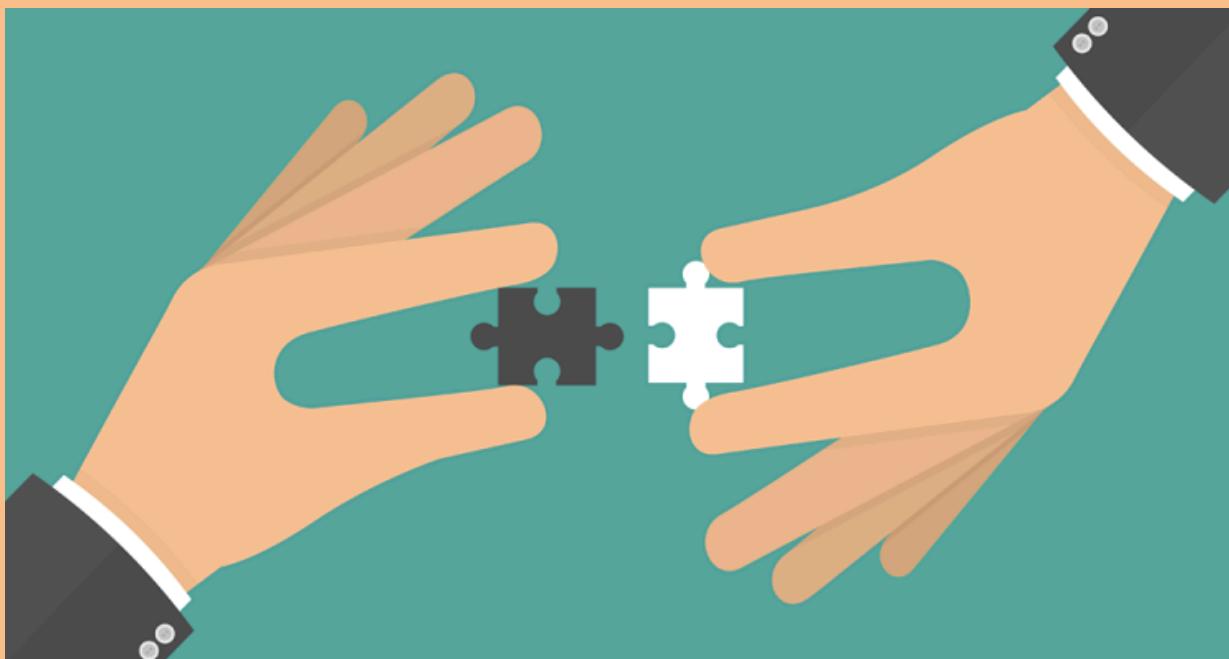
2. TRỊ

- GIẢI QUYẾT TỪNG BÀI TOÁN NHỎ RIÊNG LẺ.
- NẾU MỘT BÀI TOÁN CON ĐỦ NHỎ (THƯỜNG ĐƯỢC GỌI LÀ "TRƯỜNG HỢP CƠ SỞ"), CHÚNG TA SẼ GIẢI QUYẾT TRỰC TIẾP MÀ KHÔNG CẦN ĐỆ QUY THÊM.

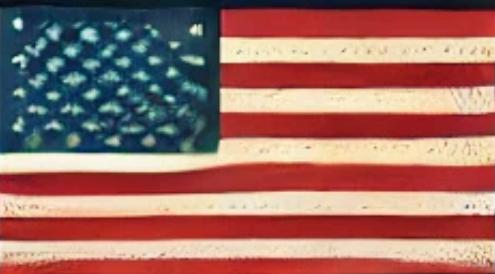


3. HỢP NHẤT

- KẾT HỢP CÁC BÀI TOÁN CON ĐỂ CÓ ĐƯỢC GIẢI PHÁP CUỐI CÙNG CHO TOÀN BỘ BÀI TOÁN.
- SAU KHI GIẢI QUYẾT XONG CÁC BÀI TOÁN CON NHỎ HƠN, CHÚNG TA KẾT HỢP ĐỆ QUY CÁC GIẢI PHÁP CỦA CHÚNG ĐỂ CÓ ĐƯỢC GIẢI PHÁP CHO BÀI TOÁN LỚN HƠN.



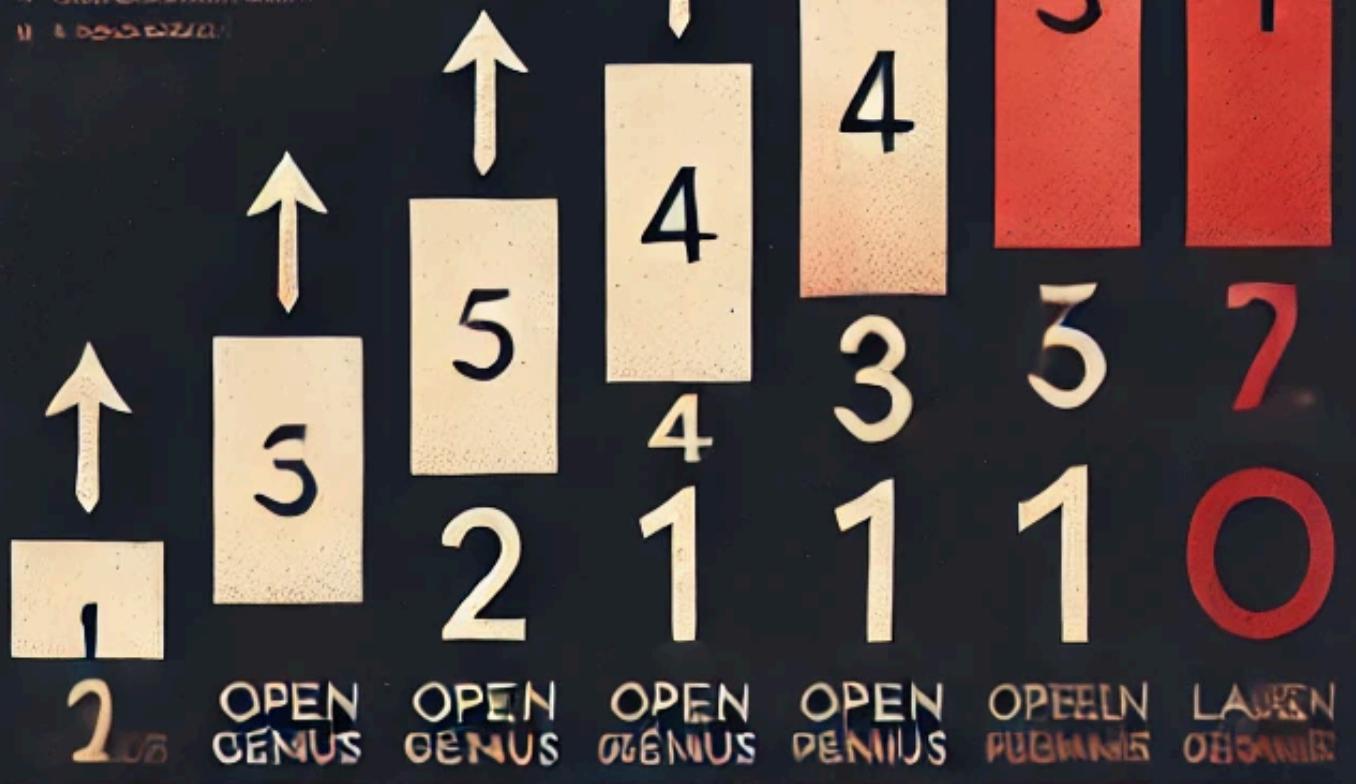
DECREASE & CONQUER ALGORIÆMS



ALGORATIMS THAT
MAKE YOU
NATIONAL PROGRAMMER

OPEN GENUS

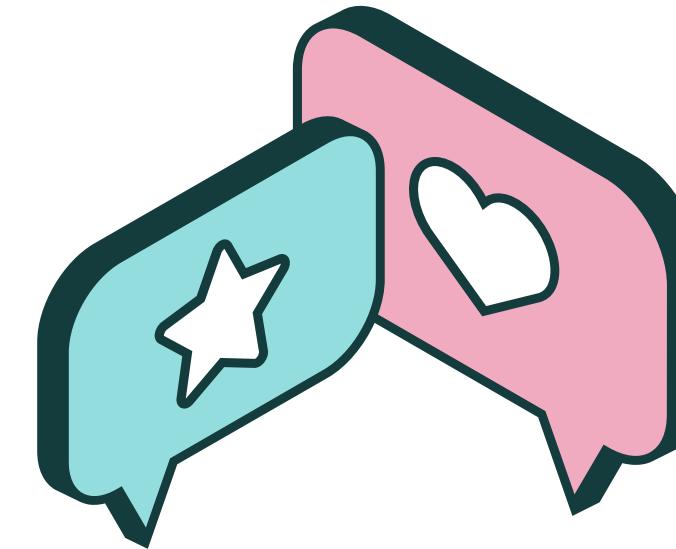
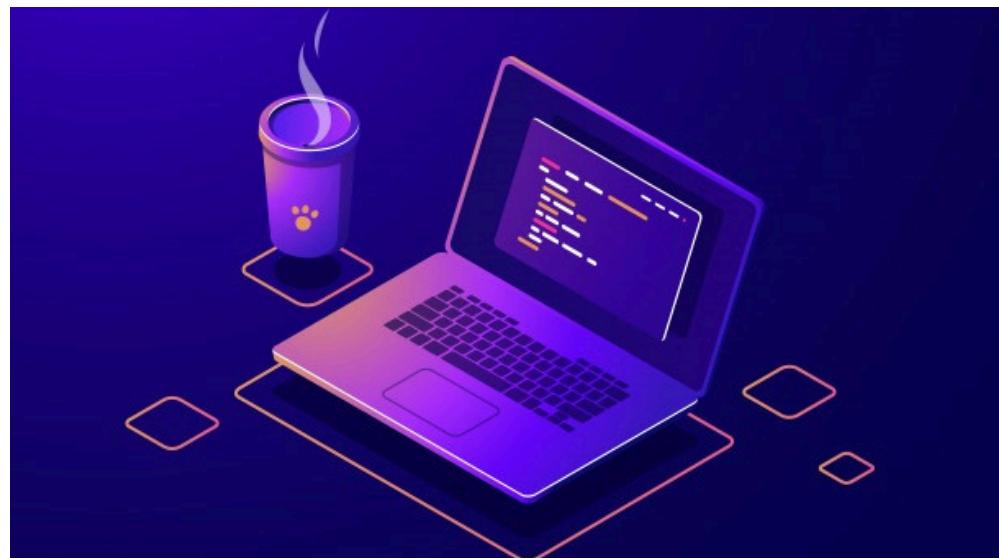
- 1. [Merge Sort](#)
- 2. [Quicksort](#)
- 3. [BFS](#)
- 4. [DFS](#)



OPEN GENUS

KHÁI NIỆM:

Decrease and conquer là một kỹ thuật được sử dụng để giải quyết các vấn đề bằng cách giảm kích thước của dữ liệu đầu vào ở mỗi bước của quá trình giải quyết và *không yêu cầu hợp nhất* kết quả.



CÔNG ĐOẠN CỦA GIẢM VÀ TRỊ

Giảm

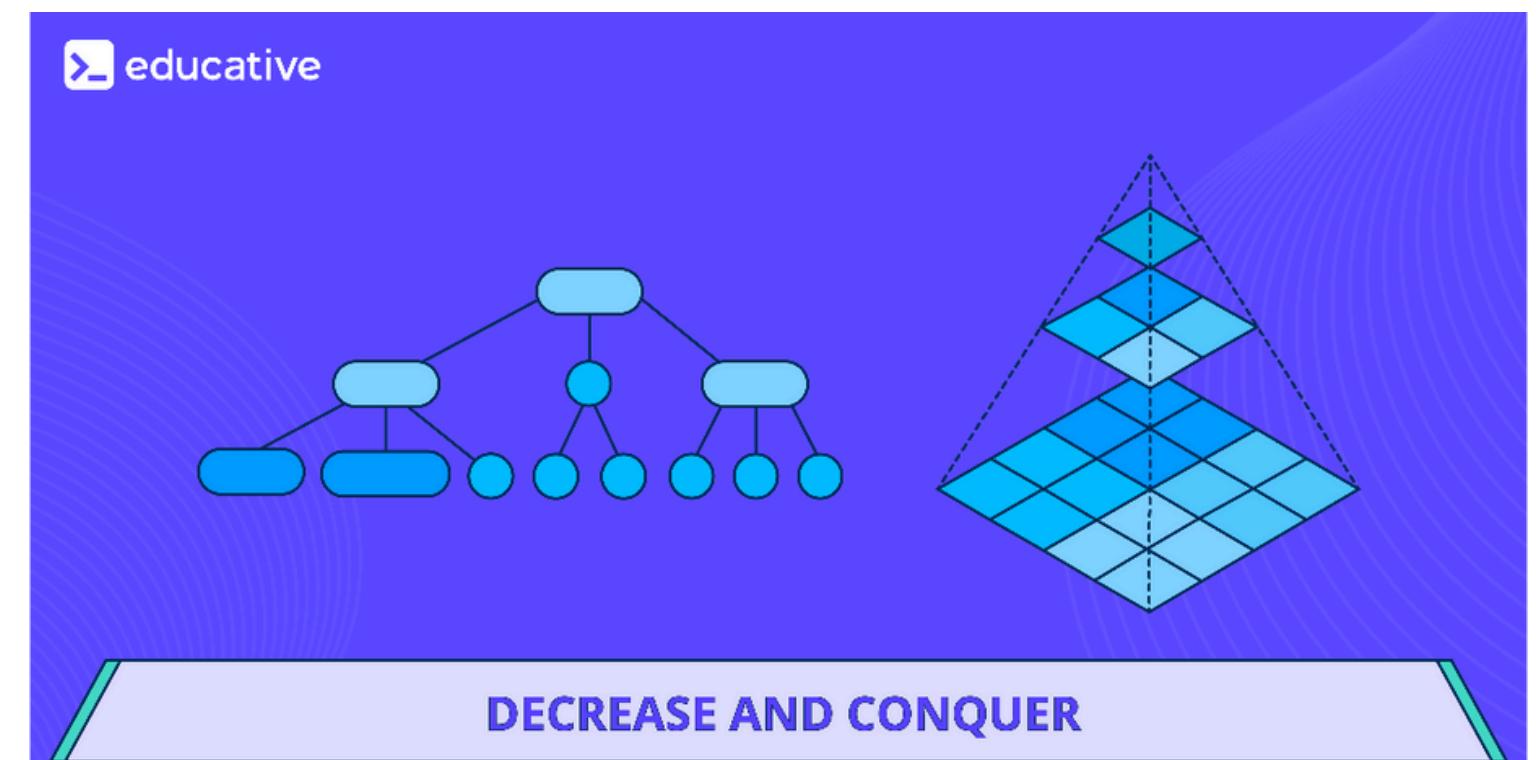
Thu nhỏ trường hợp bài toán thành trường hợp nhỏ hơn của cùng một bài toán và mở rộng giải pháp.

Trị

Giải quyết các trường hợp nhỏ hơn của bài toán.

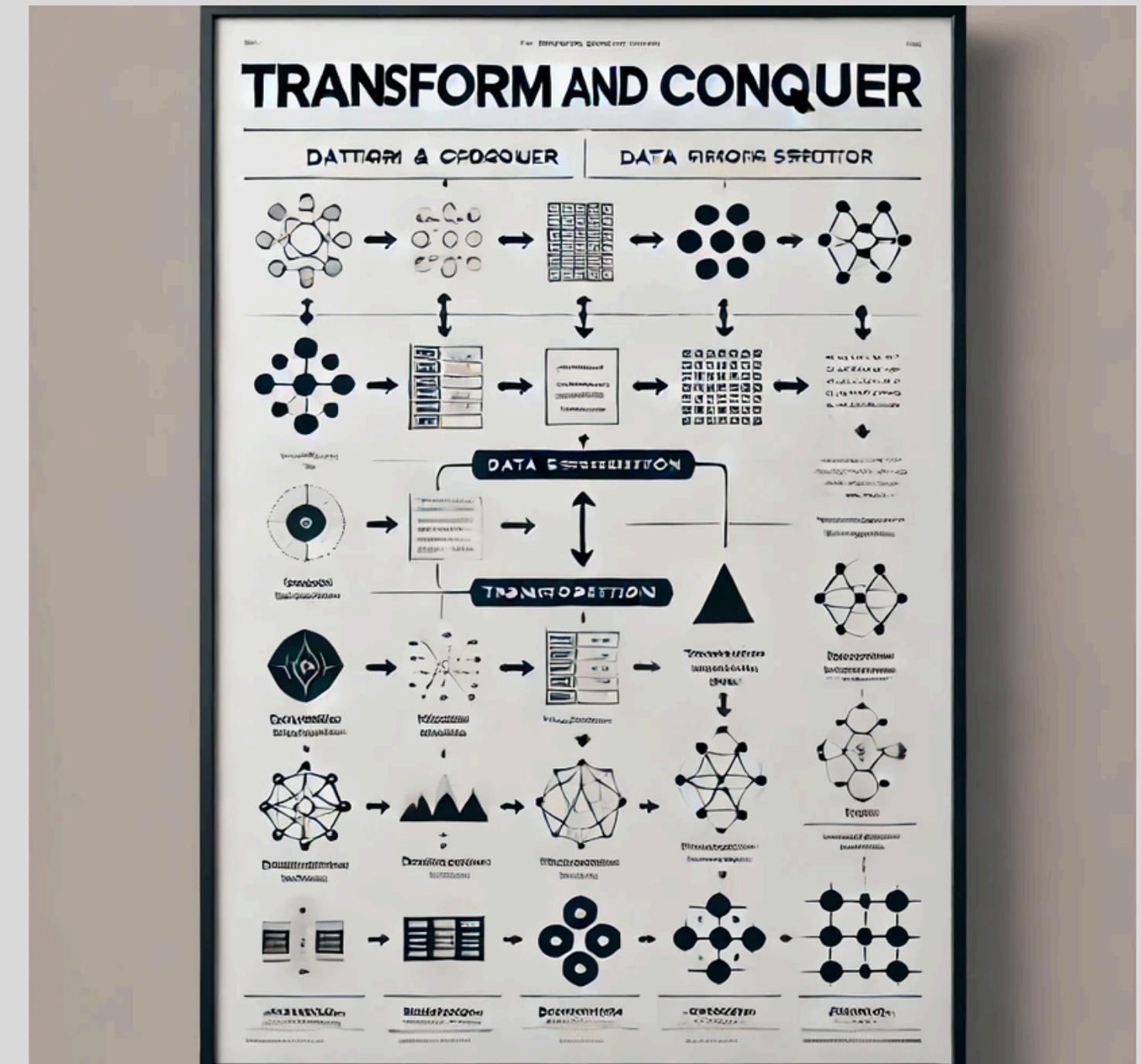
Mở rộng

Sử dụng kết quả của bài toán con để tạo lời giải cho bài toán gốc.



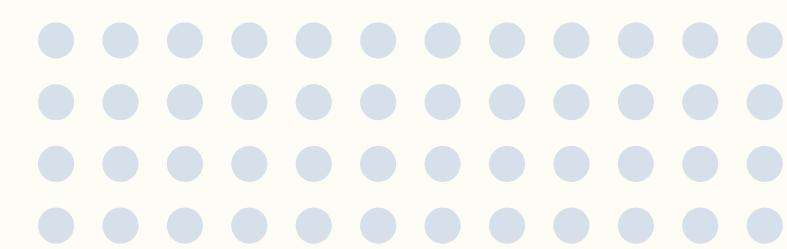


TRANSFORM AND CONQUER

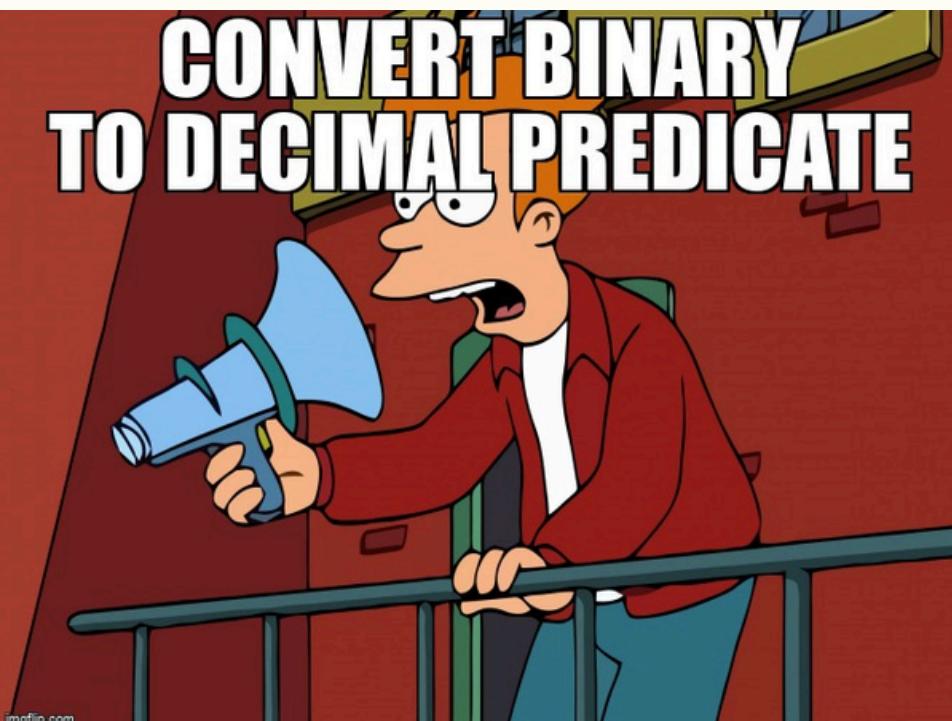


Khái niệm và công đoạn

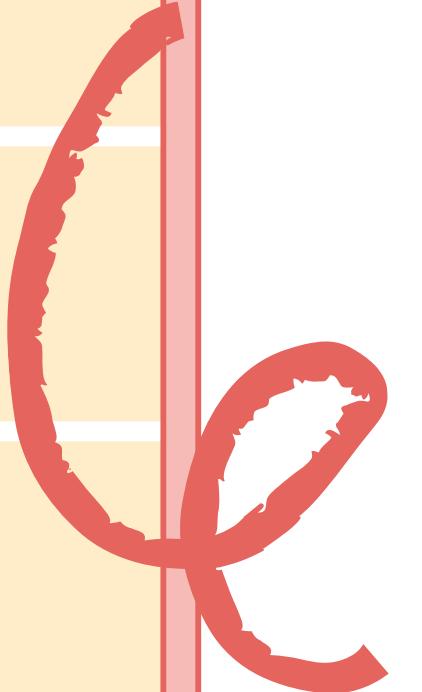
Transform and conquer giải quyết vấn đề phức tạp bằng cách biến đổi bài toán thành dạng đơn giản hơn.



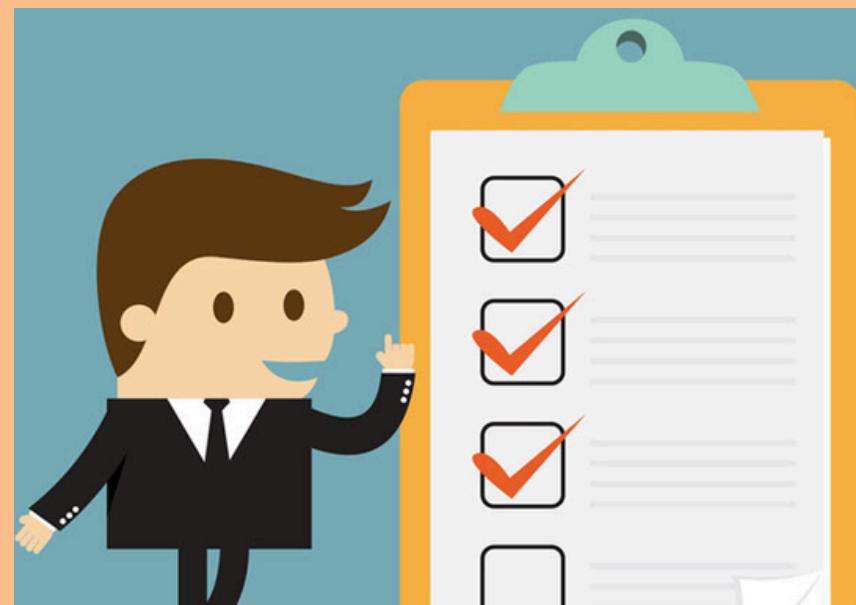
Biến đổi: Chuyển bài toán hoặc dữ liệu thành dạng dễ xử lý hơn như thay đổi cách biểu diễn dữ liệu, chuyển bài toán về một dạng bài toán khác dễ giải hơn.



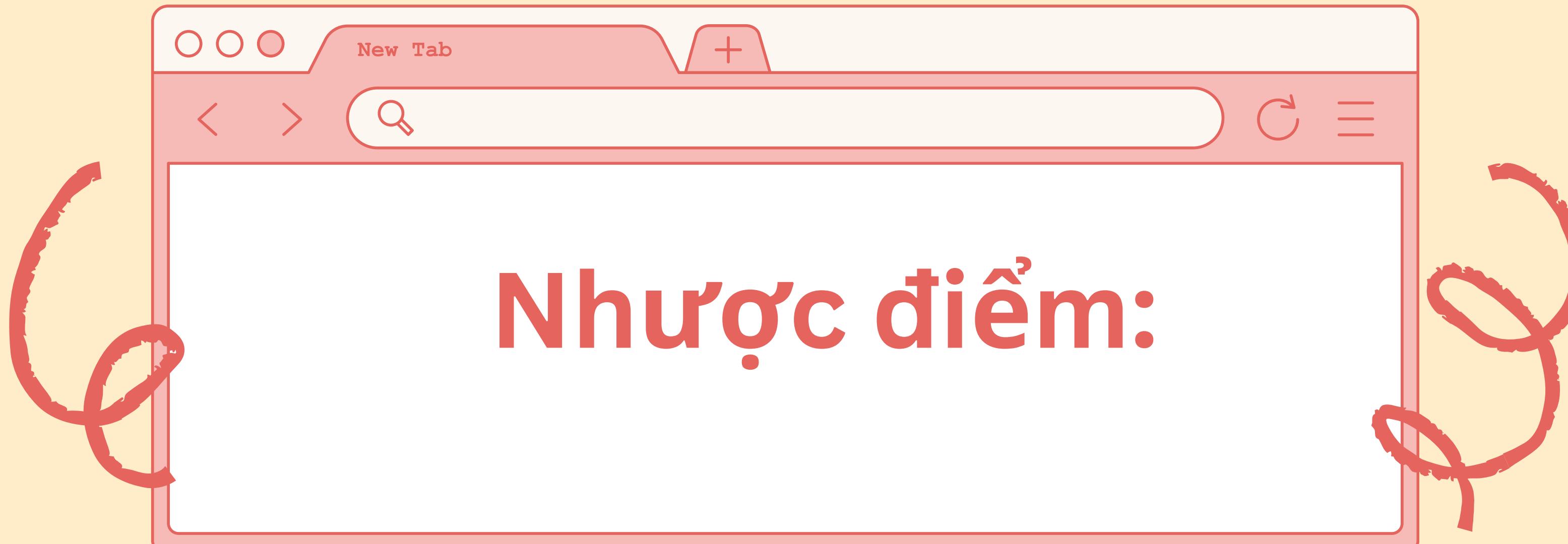
Trị: Áp dụng thuật toán để giải quyết bài toán sau khi đã biến đổi như đệ quy, quy hoạch động hoặc các thuật toán đã được tối ưu hóa khác.



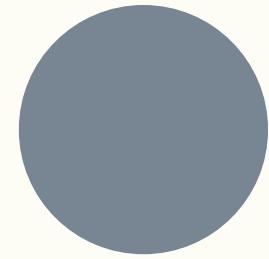
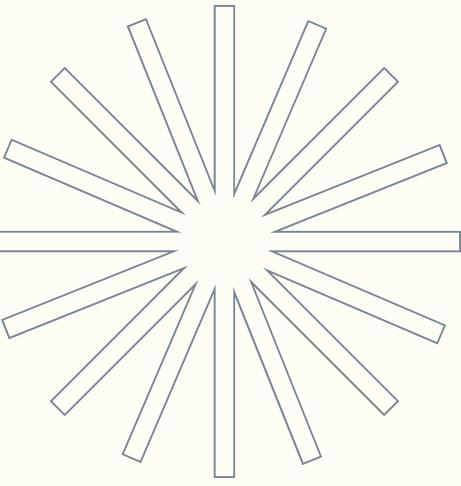
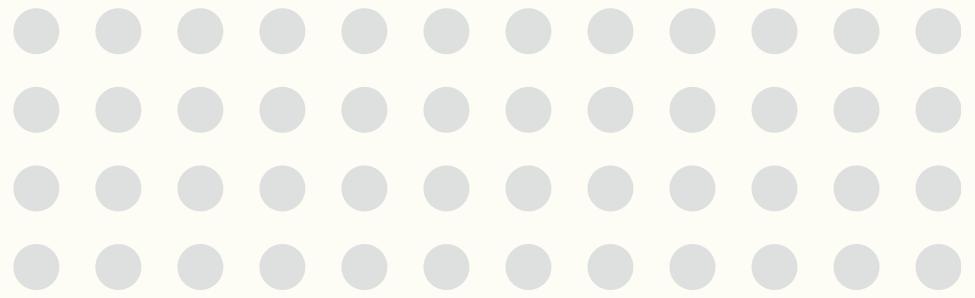
Ưu điểm:



- **Giải quyết các vấn đề khó:** Nó đòi hỏi một cách chia nhỏ vấn đề thành các vấn đề con đơn giản hơn và giải quyết tất cả chúng như một trường hợp riêng lẻ.
- **Dễ cài đặt và bảo trì :** thường sử dụng đệ quy, giúp dễ viết mã và dễ bảo trì.
- **Truy cập bộ nhớ:** Các thuật toán này tự nhiên sử dụng bộ nhớ đệm hiệu quả.



Nhược điểm:



Chi phí chung: Quá trình chia vấn đề thành các bài toán con và sau đó kết hợp các giải pháp có thể đòi hỏi thêm thời gian và nguồn lực.

Độ phức tạp: Chia một vấn đề thành các bài toán con nhỏ hơn phụ thuộc lẫn nhau có thể làm tăng ĐPT của giải pháp tổng thể.

Khó khăn khi triển khai: Một số vấn đề khó chia thành các bài toán con nhỏ hơn và khó để cài đặt.



"LET'S DIVIDE AND CONQUER!"



TÌM KIẾM NHỊ PHÂN

Thuật toán:

1. Chọn 1 điểm mid là trung điểm của dãy số. (Điểm mid là điểm giữa hai điểm l và r, có giá trị là $(l + r) / 2$).
2. Kiểm tra xem điểm mid có bằng với giá trị nhỏ hơn điểm mid không. Nếu không, quay lại bước 1.
3. **Giới hạn lại khi nào**: Nếu tìm được điểm mid, dừng chương trình. Nếu không, quay lại bước 1.
4. Nếu tìm được điểm mid, dừng chương trình. Nếu không, quay lại bước 1.

Decrease and conquer

CODE

```
function binary_search(list, target):
    left = 0
    right = length(list) - 1
    while left <= right:
        mid = (left + right) // 2
        if list[mid] == target:
            return mid
        elif list[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

VẬY TẠI SAO TA
LẠI CHỌN MID LÀ
ĐIỂM Ở GIỮA CỦA
MẢNG?



Trả lời:

Bởi vì mỗi lần độ dài của phạm vi tìm kiếm giảm đi một nửa nên độ phức tạp tối ưu sẽ đạt được $O(\log(n))$.



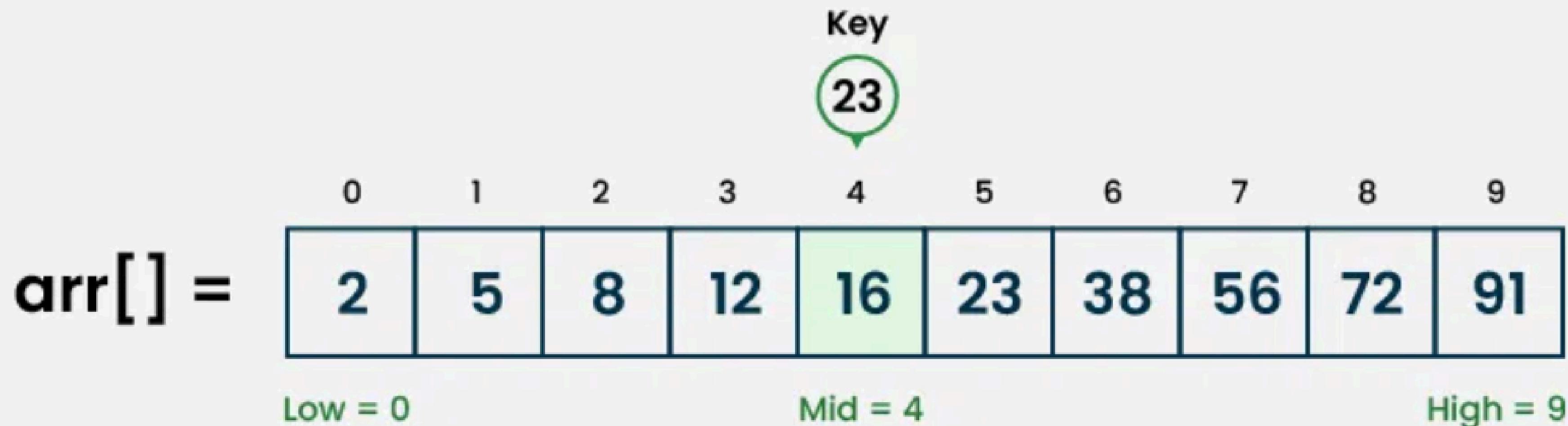
Initially

Find Key = 23 using Binary Search

	0	1	2	3	4	5	6	7	8	9
arr[] =	2	5	8	12	16	23	38	56	72	91

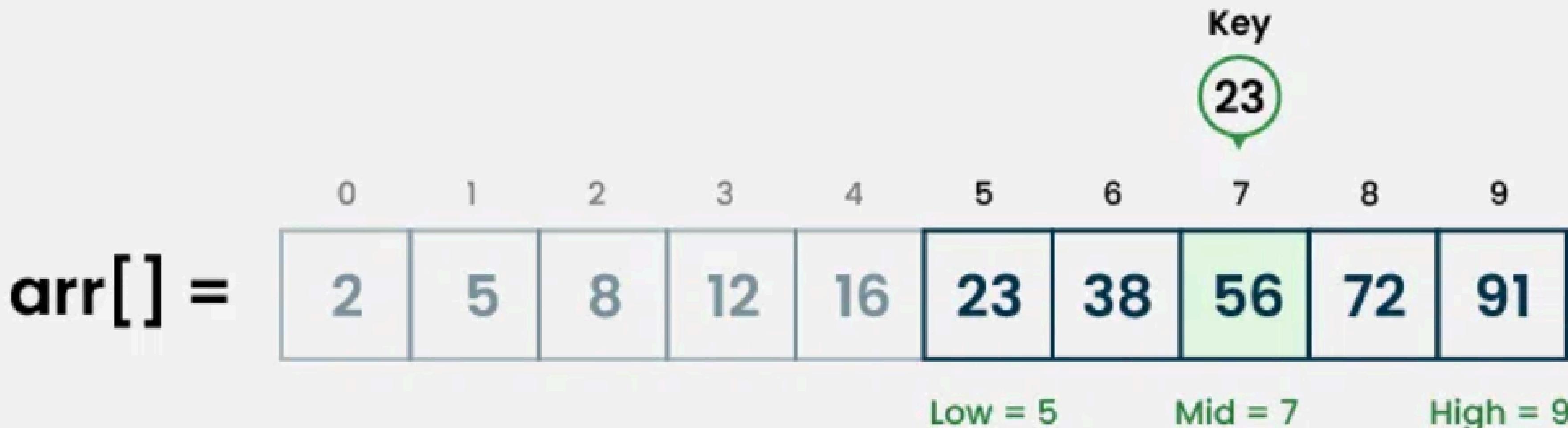
Step 1

Key (i.e., 23) is greater than current mid element (i.e., 16). The search space moves to the right.



Step 2

Key is less than the current mid 56.
The search space moves to the left.



Step 3

If the key matches the value of the mid element, the element is found and stop search.



BÀI TOÁN TÍNH LŨY THỪA VỚI SỐ MŨ LỚN:

Function Power(base, exponent):

```
if exponent == 0:
```

```
    return 1
```

```
if exponent == 1:
```

```
    return base
```

```
result = Power(base, exponent // 2)
```

```
if exponent is even:
```

```
    return result * result
```

```
else:
```

```
    return result * result * base
```

ỨNG DỤNG VỚI CÁC THUẬT TOÁN SORT:

QUICK SORT:

Hãy cho biết thuật toán này có sử dụng các kiến thức vừa học không và ở đâu??

```
quicksort (array){  
    if (array.length > 1){  
        choose a pivot;  
        while (there are items left in array){  
            if (item < pivot)  
                put item into subarray1;  
            else  
                put item into subarray2;  
        }  
        quicksort(subarray1);  
        quicksort(subarray2);  
    }  
}
```

ỨNG DỤNG VỚI CÁC THUẬT TOÁN SORT:

QUICK SORT:

Giải thuật trên sử dụng decrease and conquer.

ĐPT: $O(n \log(n))$.

```
quicksort(array){  
    if (array.length > 1){  
        choose a pivot;  
        while (there are items left in array){  
            if (item < pivot)  
                put item into subarray1;  
            else  
                put item into subarray2;  
        }  
        quicksort(subarray1);  
        quicksort(subarray2);  
    }  
}
```

ỨNG DỤNG VỚI CÁC THUẬT TOÁN SORT:

MERGE SORT:

```
MERGE-SORT( $A, p, r$ )
```

```
1   if  $p < r$ 
2        $q = \lfloor (p + r)/2 \rfloor$ 
3       MERGE-SORT( $A, p, q$ )
4       MERGE-SORT( $A, q + 1, r$ )
5       MERGE( $A, p, q, r$ )
```

Giải thuật trên sử dụng divide and conquer.

Được biết thuật toán sau chạy với độ phức tạp $O(n\log n)$. Điều đó thực sự tuyệt vời.

Nhưng nhờ kĩ thuật gì mà đạt được như vậy?



Liệu có thể sắp xếp lại bằng phương pháp Transform and conquer không?



Chuyển cách biểu diễn dãy số thành cấu trúc cây đỏ đen (Hay nói cách khác là dùng multiset).



```
Function sortArrayWithMultiset(array):
    // Step 1: Khởi tạo multiset để chứa các phần tử của dãy
    Initialize multiset

    // Step 2: Chèn từng phần tử của dãy vào multiset
    For each element in array:
        Insert element into multiset

    // Step 3: Truy xuất các phần tử từ multiset theo thứ tự đã sắp xếp
    Initialize sorted_array as an empty list
    For each element in multiset:
        Append element to sorted_array

    Return sorted_array
```

**SAU KHI HỌC XONG, CÁC BẠN
CÓ RÚT RA ĐƯỢC FOMAT CHUNG
CHO CÁC BÀI THUỘC ĐẠNG NÀY
KHÔNG?**

Điểm chung:

- Ta cần tìm ra được cách chia nhỏ bài toán tối ưu. (Đối với Divde, Decrease and conquer thường chia thành 2 phần bằng nhau, còn Transform and conquer thường chuyển sang 1 dạng biểu diễn khác tương đương).
- Các bài toán đã chia nhỏ phải có thuật toán giải được trong thời gian tối ưu.

Điểm riêng:

- **Divde and conquer:** Cần tìm ra được cách kết hợp tối ưu giữa các khối con sau khi đã xử lí.
- **Transform and conquer:** Cần tìm được 1 dạng tương đương với bài toán ban đầu nhưng dễ sử lí hơn.



Q & A

LUCKY LUNCH

LET'S PLAY!

• LUẬT CHƠI:

- Ban đầu chúng ta sẽ có random ra 1 người bất kì và được làm người chơi chính.
- Người chơi chính sẽ trả lời 5 câu hỏi, mỗi câu hỏi gồm 4 đáp án.
- **Tranh lượt:** Khi người chơi chính trả lời sai, những người còn lại trong lại sẽ có quyền trả lời, nếu trả lời đúng sẽ thế chỗ người chơi chính.
- **Người chơi chính sau khi lượt thứ 5 kết thúc** sẽ nhận được 1 phần cơm trưa với giá 30k.



Are you
ready?

Câu 1:

Trong chiến lược Decrease and Conquer, bước nào sau đây là KHÔNG đúng?

a. Giảm kích thước của bài toán xuống một phần nhỏ hơn

b. Mở rộng giải pháp của trường hợp nhỏ hơn để giải quyết bài toán ban đầu

c. Kết hợp hai phần kết quả của bài toán

d. Chinh phục vấn đề bằng cách giải trường hợp nhỏ hơn của bài toán

Answer

Trong chiến lược Decrease and Conquer, bước nào sau đây là KHÔNG đúng?

- a. Giảm kích thước của bài toán xuống một phần nhỏ hơn
- b. Mở rộng giải pháp của trường hợp nhỏ hơn để giải quyết bài toán ban đầu
- c. Kết hợp hai phần kết quả của bài toán
- d. Chinh phục vấn đề bằng cách giải trường hợp nhỏ hơn của bài toán

Câu 2:

Để tính $x^0 + x^1 + x^2 + x^3 + x^4 + \dots + x^n$,
với $x < 10^{18}$.

Ta sẽ cần ứng dụng những kĩ thuật gì?

a. Transform and conquer

b. Divide and conquer

c. Decrease and conquer.
Divide and conquer

d. Transform and conquer.
Divide and conquer

• Answer •

Để tính $x^0 + x^1 + x^2 + x^3 + x^4 + \dots + x^n$,
với $x < 1e18$.

Ta sẽ cần ứng dụng những kĩ thuật gì?

a. Transform and conquer

b. Divide and conquer

c. Decrease and conquer.
Divide and conquer

d. Transform and conquer,
Divide and conquer

Câu 3:

Những thuật toán nào KHÔNG có ứng dụng kĩ thuật
vừa được học?

a. Segment tree

b. Tìm kiếm nhị phân

c. RMQ

d. Prefix-sum

Answer

Những thuật toán nào KHÔNG có Ứng dụng kĩ thuật
vừa được học?

a. Segment tree

b. Tìm kiếm nhị phân

c. RMQ

d. Prefix-sum

Câu 4:

Điều nào sau đây là một nhược điểm của chiến lược Transform and Conquer?

- a. Không thể giải quyết các vấn đề phức tạp.
- b. Tốn chi phí cao cho việc biến đổi dữ liệu.
- c. Chỉ áp dụng cho các vấn đề nhỏ.
- d. Khi không cần giảm kích thước của bài toán.

Answer

Điều nào sau đây là một nhược điểm của chiến lược Transform and Conquer?

- a. Không thể giải quyết các vấn đề phức tạp.
- b. Tốn chi phí cao cho việc biến đổi dữ liệu.
- c. Chỉ áp dụng cho các vấn đề nhỏ.
- d. Khi không cần giảm kích thước của bài toán.

• Câu 5:

Trong thuật toán tìm đường đi ngắn nhất floyd.
việc áp dụng kĩ thuật divide and conquer được thể
hiện ở điểm nào?

- a. Chia đường đi từ a
đến b thành đường đi
từ a đến k và từ k
đến b. rồi kết hợp lại.
- b. A và b đều đúng.
- c. Lấy đỉnh có trọng số
nhỏ nhất rồi thực hiện cập
nhật.
- d. Duyệt qua từng các
cạnh rồi cập nhập
trọng số của các đỉnh.

Answer

Trong thuật toán tìm đường đi ngắn nhất floyd.
việc áp dụng kĩ thuật divide and conquer được thể
hiện ở điểm nào?

- a. Chia đường đi từ a
đến b thành đường đi
từ a đến k và từ k đến
b. rồi kết hợp lại.
- b. A và b đều đúng.
- c. Lấy đỉnh có trọng số
nhỏ nhất rồi thực hiện cập
nhật.
- d. Duyệt qua hết tất
cả các cạnh rồi cập
nhập trọng số của các
đỉnh.



Thanks for
playing!

THE END