

DIVIDE

CONQUER

WHAT?

Chia để trị là gì

WHAT?

Là một chiến lược thiết kế thuật toán:

- Top-down approach
- Được cài dựa trên đệ quy
- Chia - trị - hợp

Ý tưởng cơ bản

Gồm 3 phần:

- Chia: chia bài toán lớn thành nhiều bài toán con nhỏ hơn và đồng dạng và độc lập với nhau để giải quyết
- Trị: Giải quyết các bài toán con
- Hợp: Kết hợp kết quả của các bài toán con để có được lời giải cuối cùng cho bài toán lớn ban đầu

DECREASE AND CONQUER

WHY?

Điểm mạnh/Điểm yếu

WHEN?

Khi nào áp dụng chia để trị

WHERE?

Ứng dụng vào đâu

HOW?

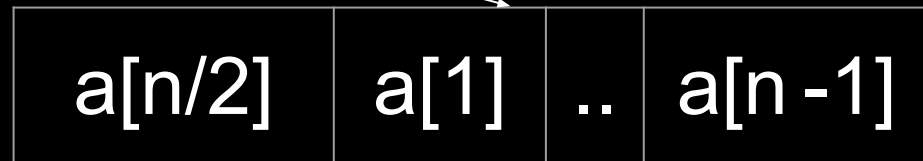
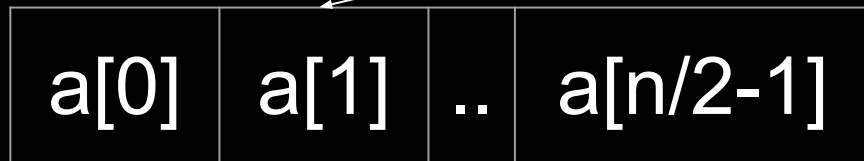
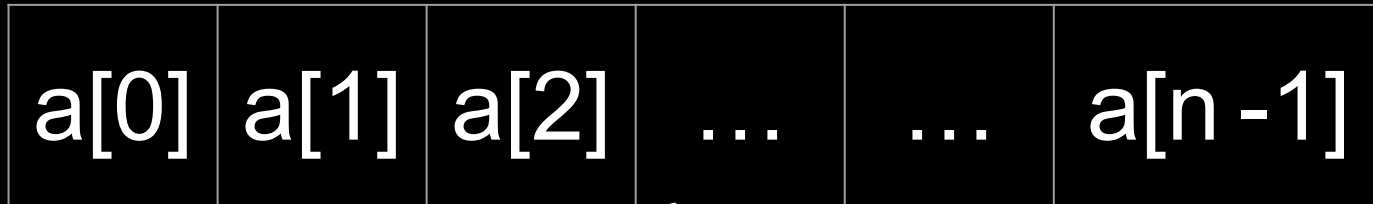
Cài đặt thế nào

```
DnC (P) :  
    if (P is small):  
        Solve(P)  
    else:  
        Divide P into n  
        smaller sub-problem p  
        Apply DnC(p_n)  
        Combine(DnC(p_n))
```

FRAMEWORK

Tính tổng n số

Tính tổng n số



Thời gian chạy

$$T(n) = a T(n/b) + f(n).$$

Với:

- $T(n)$ là thời gian chạy
- a là số vòng lặp đệ quy
- $T(n/b)$ là thời gian chạy mỗi vòng lặp đệ quy
- $f(n)$ là thời gian phân chia và hợp các bài toán con

Master Theorem

Masters Theorem

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

$$f(n) = \Theta(n^k \log^p n) \quad \text{Với } a \geq 1, b > 1$$

Masters Theorem

Case 1: if $\log_b a > k$ then $\theta(n^{\log_b a})$

Case 2: if $\log_b a = k$

if $p > -1$ $\theta(n^k \log^{p+1} n)$

if $p = -1$ $\theta(n^k \log \log n)$

if $p < -1$ $\theta(n^k)$

Case 3: if $\log_b a < k$

if $p \geq 0$ $\theta(n^k \log^p n)$

if $p < 0$ $O(n^k)$

Masters Theorem

$$T(n) = 4T(n/2) + n$$

$$T(n) = 4T(n/2) + n^2$$

$$T(n) = 4T(n/2) + n^3$$

Masters Theorem

$$T(n) = 4T(n/2) + n$$

$$a = 4$$

$$k = 1$$

$$\log_2(4) = 2 > k = 1$$

$$b = 2$$

$$p = 0$$

Case 1: $\Theta(n^2)$

Masters Theorem

$$T(n) = 4T(n/2) + n^2$$

$$a = 4$$

$$k = 2$$

$$\log_2(4) = k = 2$$

$$b = 2$$

$$p = 0$$

Case 2: $\Theta(n^2 \log n)$

Masters Theorem

$$T(n) = 4T(n/2) + n^3$$

$$a = 4$$

$$k = 3$$

$$\log_2(4) = 2 < k = 3$$

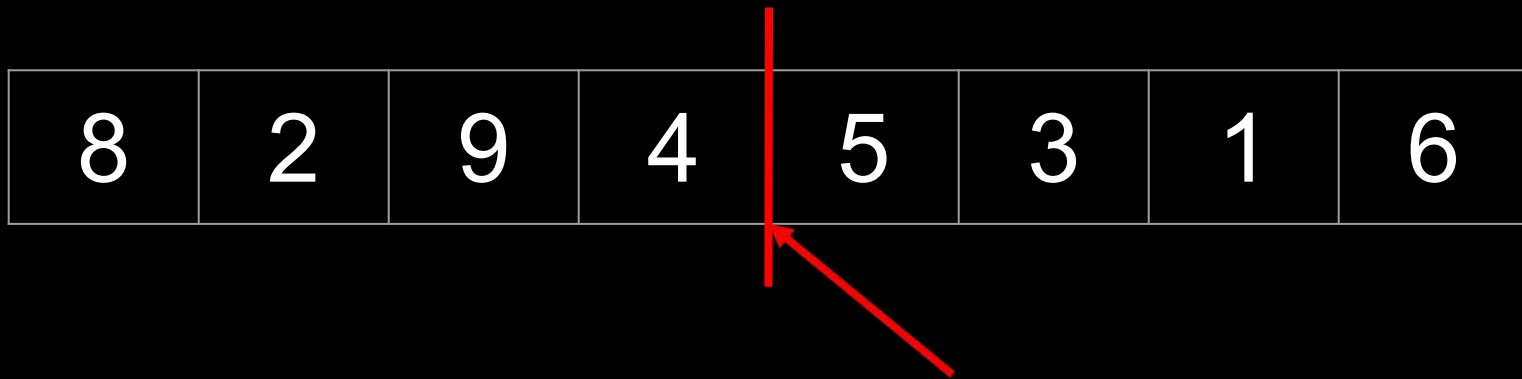
$$b = 2$$

$$p = 0$$

Case 3: $\Theta(n^3)$

Merge Sort

Merge Sort



- Chia mảng thành hai phần
- Sắp xếp từng phần theo cách đệ quy
- Hợp nhất hai mảng được sắp xếp nhỏ thành một mảng được sắp xếp.

Merge Sort

8 2 9 4 5 3 1 6

8 2 9 4

5 3 1 6

8 2

9 4

5 3

1 6

8

2

9

4

5

3

1

6

2 8

4 9

3 5

1 6

2 4 8 9

1 3 5 6

1 2 3 4 5 6 8 9

```
# Divide
```

```
def mergeSort(arr):  
    if len(arr) > 1:  
        mid = len(arr) // 2  
        L = arr[:mid]  
        R = arr[mid:]  
        mergeSort(L)  
        mergeSort(R)
```



```
#Sort and merge
i = j = k = 0
while i < len(L) and j
< len(R):
    if L[i] <= R[j]:
        arr[k] =
            L[i]
        i += 1
    else:
        arr[k] =
            R[j]
        j += 1
    k += 1
```

```
#Checking if any element was
left

    while i < len(L):
        arr[k] = L[i]
        i += 1
        k += 1

    while j < len(R):
        arr[k] = R[j]
        j += 1
        k += 1
```

Độ phức tạp

$$C(n) = 2C(n/2) + C_{\text{merge}}(n) \text{ for } n > 1, C(1) = 0$$

Theo Master Theorem:

$$C(n) = \theta(n \log n)$$

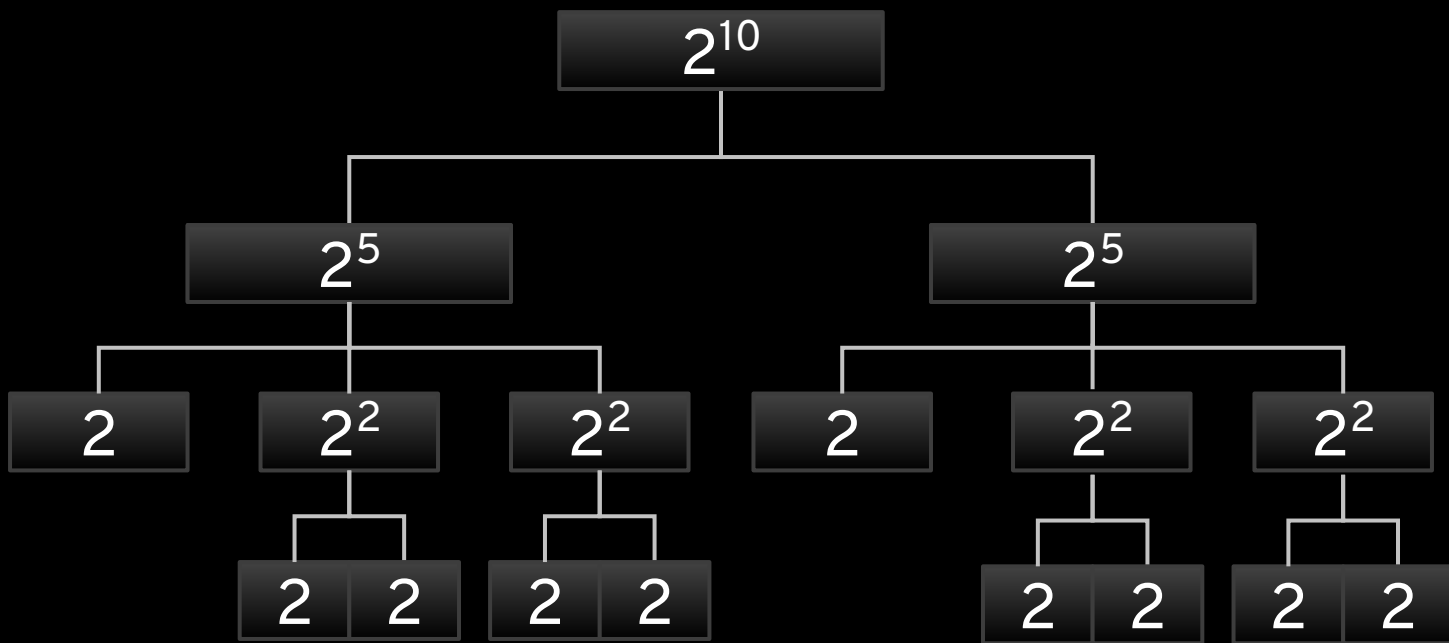
Tính $\text{pow}(x, n)$

Tính $\text{pow}(x, n)$

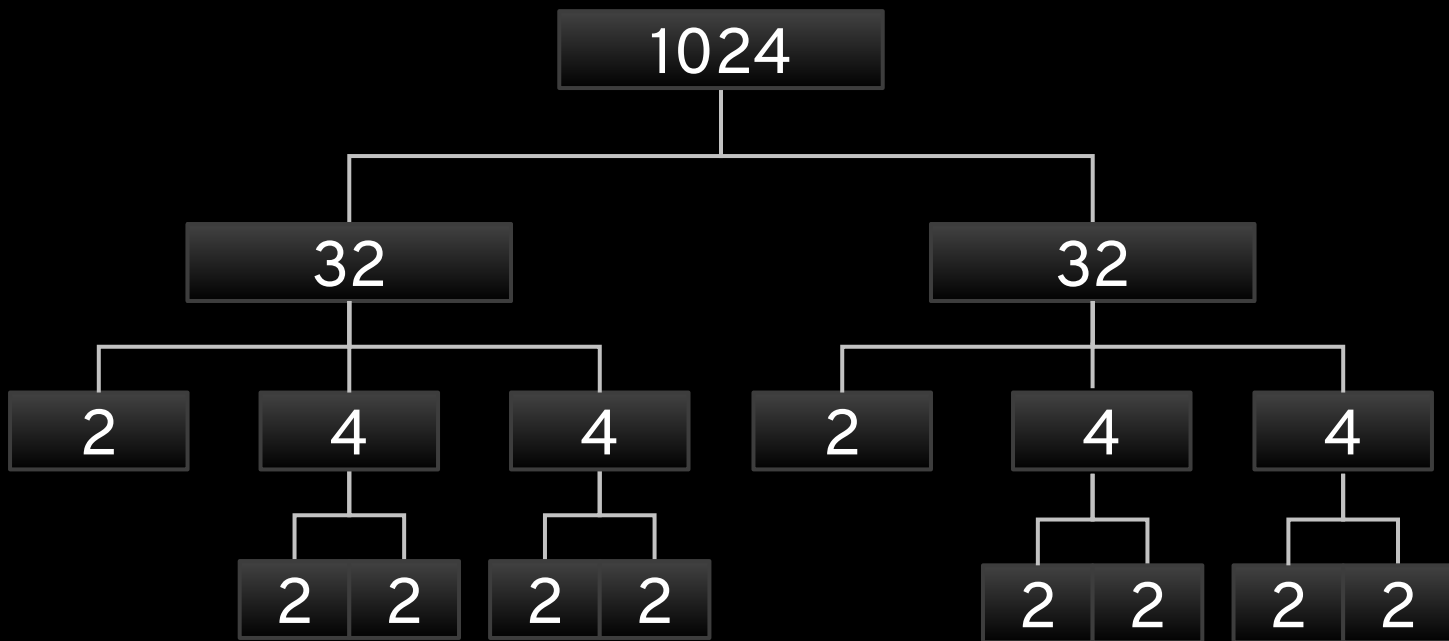
$$x^n$$

- Chia phép tính thành hai phần
- Tính giá trị từng phần theo cách đệ quy
- Hợp nhất các phần nhỏ đã được tính thành một phần mới.

Tính $\text{pow}(x, n)$



Tính $\text{pow}(x, n)$



Tính $\text{pow}(x, n)$

Vấn đề có thể được xác định đệ quy bởi:

- $\text{power}(x, n) = \text{power}(x, n / 2) * \text{power}(x, n / 2);$ // if n is even
- $\text{power}(x, n) = x * \text{power}(x, n / 2) * \text{power}(x, n / 2);$ // if n is odd

```
def power(x, y):  
    if y == 1:  
        return x  
  
    temp = power(x, int(y / 2))  
    if y % 2 == 0:  
        return temp * temp  
    else:  
        return x * temp * temp
```


Độ phức tạp

Theo Master Theorem:

$$C(n) = \theta(\log n)$$

Quick Sort

Quick Sort

1. First, it partitions an array into two parts,
2. Then, it sorts the parts independently,
3. Finally, it combines the sorted subsequences by a simple concatenation.

```
def partition(array, low, high):  
    pivot = array[high]  
    i = low - 1  
    for j in range(low, high):  
        if array[j] <= pivot:  
            i = i + 1  
            (array[i], array[j]) = (array[j], array[i])  
    (array[i + 1], array[high]) = (array[high], array[i + 1])  
    return i + 1
```

```
def quick_sort(array, low, high):  
    if low < high:  
        pi = partition(array, low, high)  
        quick_sort(array, low, pi - 1)  
        quick_sort(array, pi + 1, high)
```

Độ phức tạp

Trường hợp tốt nhất

$$C(n) = 2C(n/2) + n \quad \text{for } n > 1, C_{\text{best}}(1) = 0.$$

Cảm ơn!