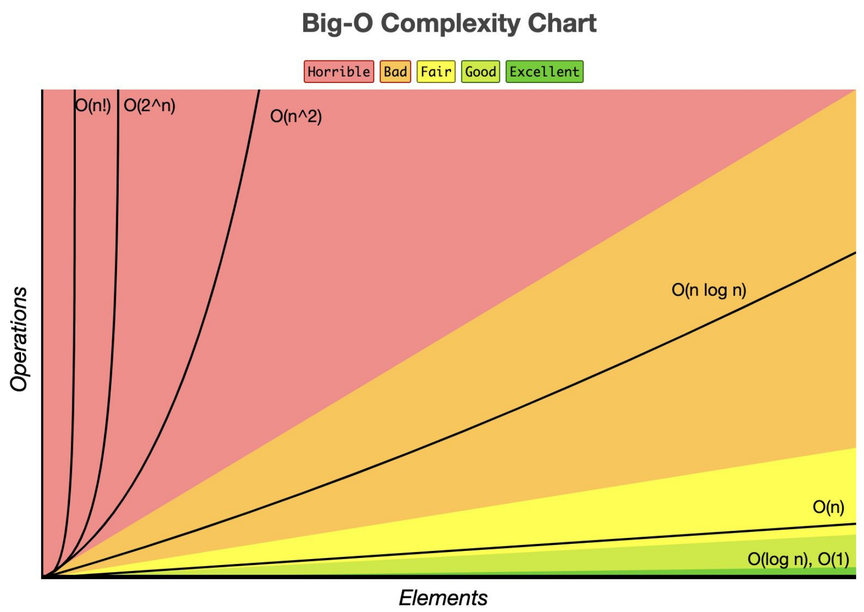
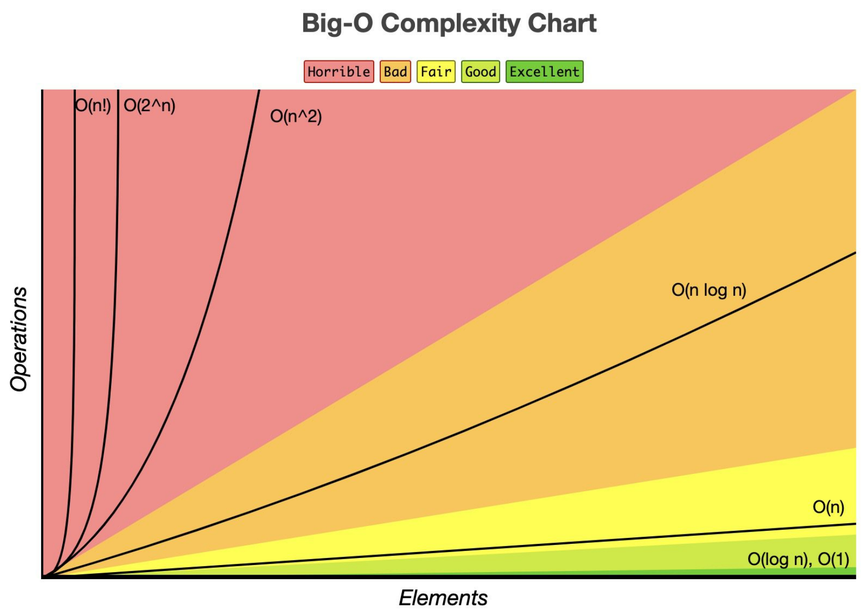


What is BigO ?

- Big O biểu diễn tốc độ tăng trưởng của thời gian chạy hoặc không gian bộ nhớ tối đa theo kích thước đầu vào của thuật toán. Ví dụ, O(n) cho thấy thời gian chạy tăng tuyến tính với kích thước đầu vào, trong khi $O(n^2)$ cho thấy sự tăng trưởng theo cấp số nhân.

- Big O hữu ích trong việc so sánh hiệu năng giữa các thuật toán khác nhau. Khi chọn thuật toán, nhà phát triển thường ưu tiên những thuật toán có Big O thấp, vì chúng thường hiệu quả hơn với dữ liệu lớn.



What is BigO ?

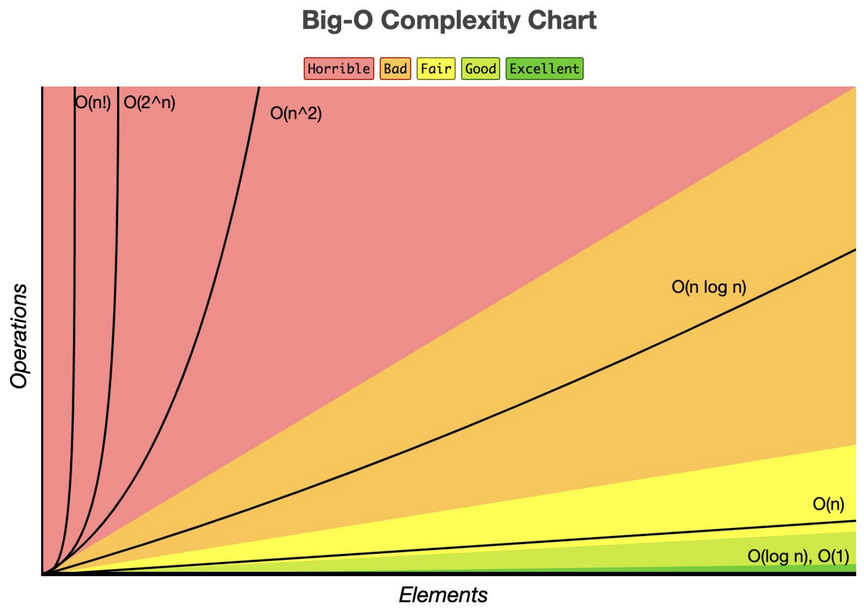
- O(1)
- O(logN)
- O(N)
- O(N + C)
- $O(N^2)$
- $O(N^3)$
-
-
-

N : Input size → Số lượng phần tử array, …

C : Constant → If hay ham gì đó …

Log ở trong thuật toán là log cơ số 2

TechX



What is BigO ?

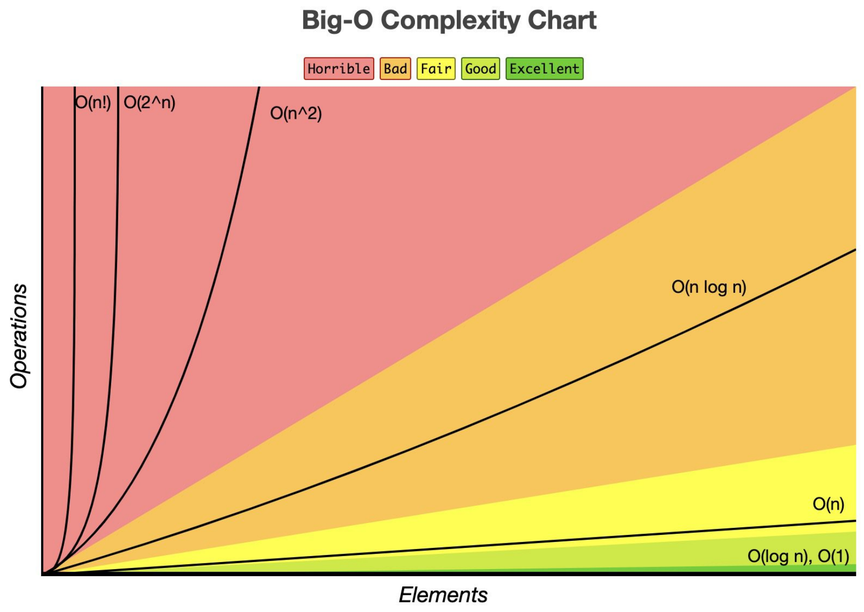
- O(CN)
- O(C+N)

→

- O(N)

- Với N luôn tiến tới ∞

TechX



What is BigO ?

- F(N) : Function

$$F(N) : N^4 + N^3 + N(\text{Log}(N))^2 + N! + 10$$

→ Trường hợp xấu nhất : $O(N^4)$

Độ phức tạp O(C) ?

```
a = 0

While ( a < 100000 ) :

    a = a + 1
```

→ Độ phức tạp là O(1) bởi vì số lần lặp vẫn là cố định
(và không phụ thuộc vào yếu tố khác)

Độ phức tạp O(n) ?

```
a = 0
While ( a < n ) :
    a = a + 1
```

→ F(N) : 1 + 2n

→ Độ phức tạp là O(n)

Độ phức tạp O(n) ?

```
a = 0
While ( a < n ) :
    a = a + 100
```

→ F(N) : 1 + 2n / 100

→ Độ phức tạp là O(n)

Bài tập

Hãy tạo hàm tính S = 1 + 2 + 3 + 4 + … + N . Cho biết độ phức tạp của hàm đó

```
def Sum  (n) :
    total = 0
    For x in Range (0, n) :
        total += x
    return total
```

→ F(N) : 1 + 2n

→ Độ phức tạp là O(N)

Bài tập

Hãy tạo hàm tính S = 1 + 2 + 3 + 4 + … + N . Cho biết độ phức tạp của hàm đó

```
def Sum  (n) :

    return n*( n+1) / 2
```

→ F(N) : 1

→ Độ phức tạp là O(1)

Độ phức tạp $O(N)^2$?

```
a = 0
b = 0
For i range ( 0,n) :
    For j range ( 0,n) :
        a++
        b++
```

→ F(N) : $2 + N + 3n^2$

→ Độ phức tạp là $O(N^2)$

Độ phức tạp $O(N)^2$?

For x in Range (0, n) : ———— n

For a in Range (0, x) : ———— n*(n+1) / 2

#Câu lệnh thực hiện ———— n*(n+1) / 2

→ F(N) : N * 2*(N +1) / 2 + N

→ Độ phức tạp là $O(N^2)$

Bài tập

For x in Range (0, n) : ———— n

For a in Range (x, n) : ———— n*(n+1) / 2

#Câu lệnh thực hiện ———— n*(n+1) / 2

→ F(N) : N * 2*(N +1) / 2 + N

→ Độ phức tạp là $O(N^2)$

Độ phức tạp O(logN) ?

```
a = 1

While ( a < n) :

    a = a * 2
```

→ F(N) :log(n)

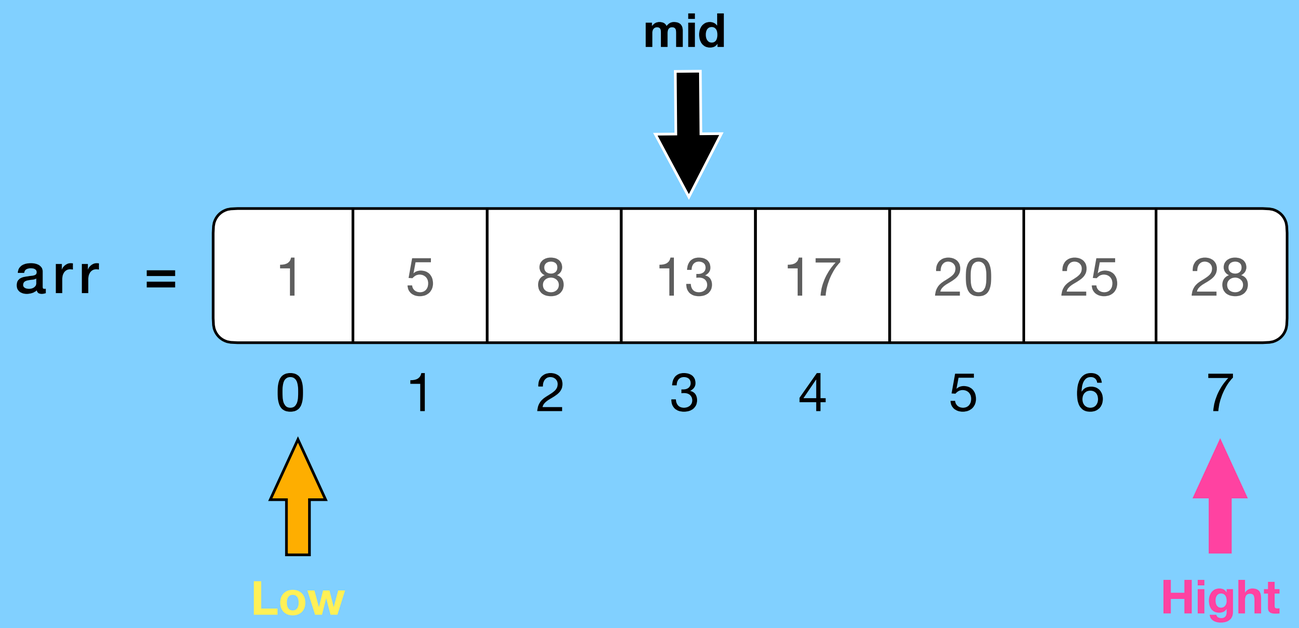
→ Độ phức tạp là O(log(n))

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else
            high = mid - 1
    return -1


binary_search(arr, 25):
```

1

X = 25

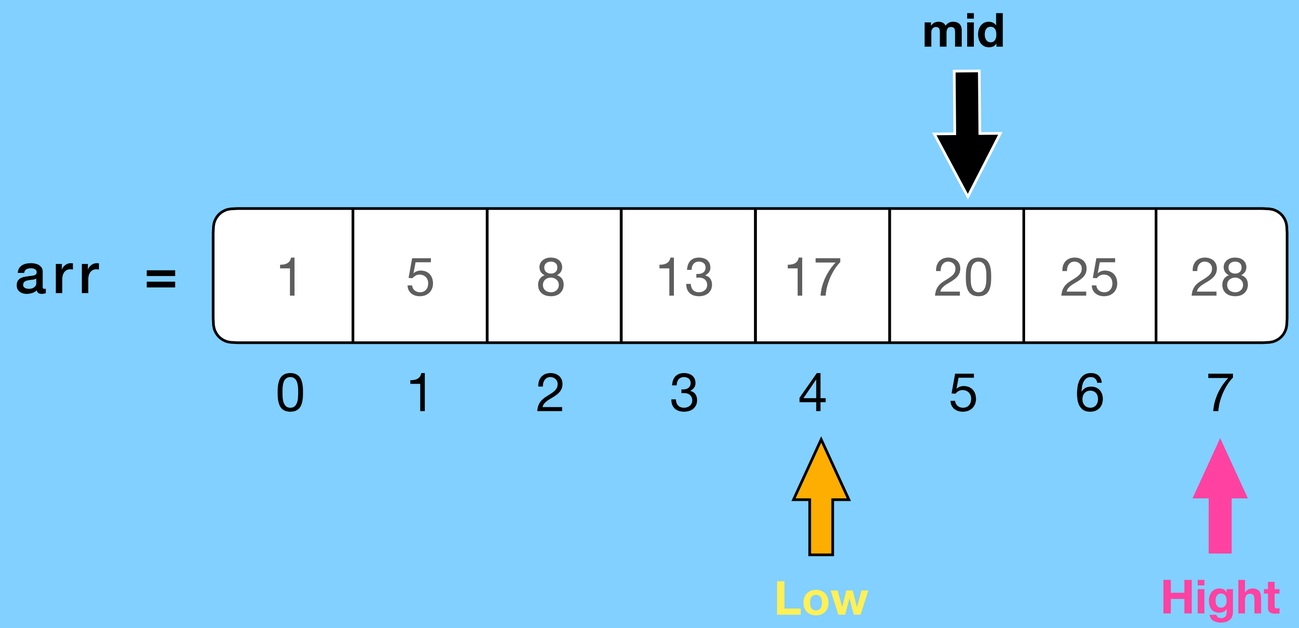


```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else
            high = mid - 1
    return -1

binary_search(arr, 25):
```

2

X = 25

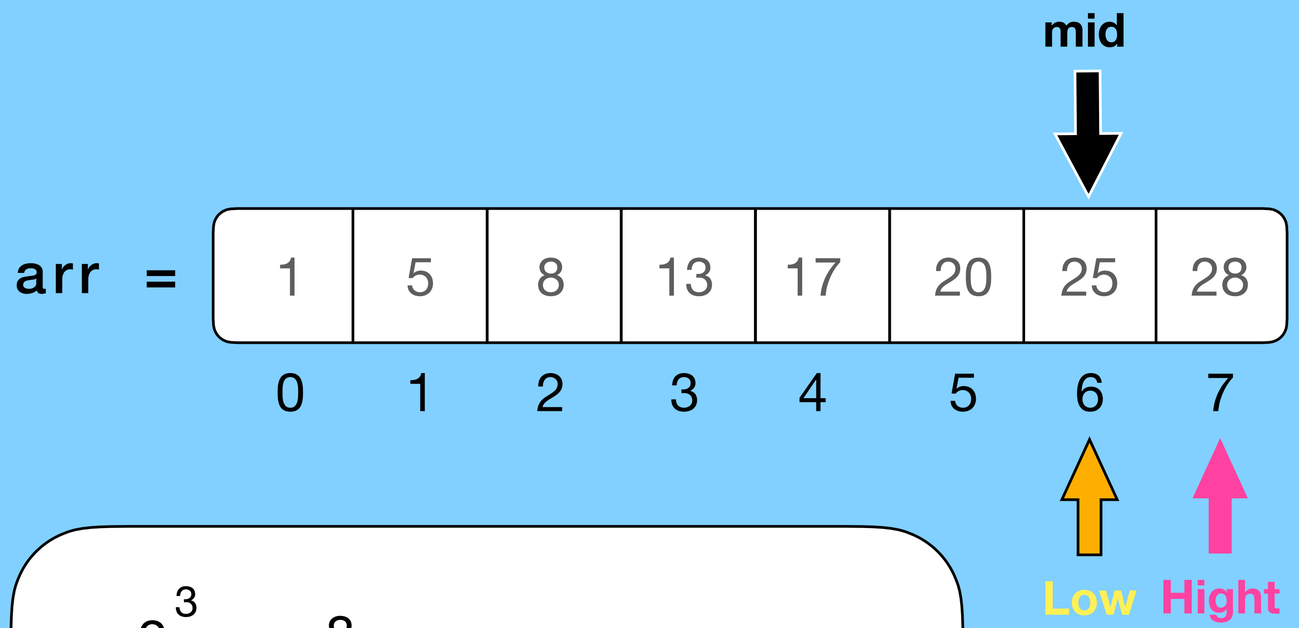


```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            low = mid + 1
        else
            high = mid - 1
    return -1

binary_search(arr, 25):
```

3

X = 25



$2^3 = 8$

Log8 = 3

Độ phức tạp : O(log(n))