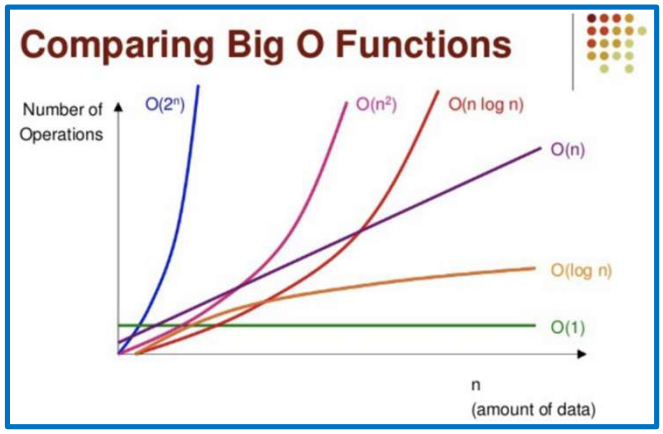
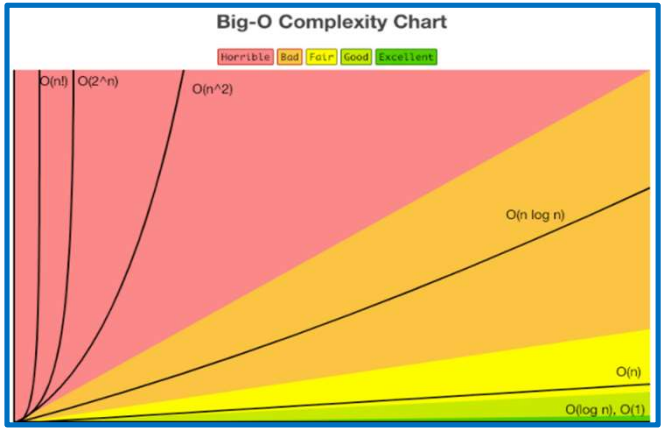

JAVA
자 료 구 조
Big O and Time Complexity

0. Big O and Time Complexity





점근적 표기법(Asymptotic Notation)

- 최상의 경우 : 오메가 표기법 (Big-Ω Notation)
- 평균의 경우 : 세타 표기법 (Big-θ Notation)
- 최악의 경우 : **빅오 표기법 (Big-O Notation)**

시간 복잡도(Time Complexity)

어떤 알고리즘을 수행하는 데 걸리는 시간을 설명하는 계산 복잡도를 의미하며, 계산 복잡도를 표기하는 대표적인 방법

다시말해, 알고리즘의 성능의 지표

0. Big O and Time Complexity

O(1) : 상수

아래 예제 처럼 입력에 관계없이 복잡도는 동일하게 유지된다.

```
def hello_world():
        print("hello, world!")
```

`O(1) - 상수 시간 : 문제를 해결하는데 오직 한 단계만 처리.`

- Stack 에서의 PUSH, POP : O (1)

O(log n) O(n log n)

주로 입력 크기에 따라 처리 시간이 증가하는 정렬알고리즘에서 많이 사용된다.
다음은 이진검색의 예이다.

```
def binary_search(li, item, first=0, last=None):
        if not last:
                last = len(li)

        midpoint = (last - first) / 2 + first

        if li[midpoint] == item:
                return midpoint

        elif li[midpoint] > item:
                return binary_search(li, item, first, midpoint)

        else:
                return binary_search(li, item, midpoint, last)
```

`O(log n) - 로그 시간 : 문제를 해결하는데 필요한 단계들이 연산마다 특정 요인에 의해 줄어듬.`

- 이진 트리 : O (log n)

`O(n log n) : 문제를 해결하기 위한 단계의 수가 N*(log2N) 번만큼의 수행시간을 가진다. (선형로그형)`

- 퀵 정렬, 병합 정렬, 힙 정렬 : O (n*log n)

0. Big O and Time Complexity

O(N) : 선형

입력이 증가하면 처리 시간또는 메모리 사용이 선형적으로 증가한다.

```
def print_each(li):
    for item in li:
        print(item)
```

`O(n)` – 직선적 시간 : 문제를 해결하기 위한 단계의 수와 입력값 `n`이 `1:1` 관계를 가짐.

- 하나의 루프를 사용하여 단일 요소 집합을 반복 하는 경우(for문) : O (n)
- 컬렉션의 절반 이상 을 반복 하는 경우 : O (n / 2) -> O (n)
- 두 개의 다른 루프를 사용하여 두 개의 개별 콜렉션을 반복 할 경우 : O (n + m) -> O (n)

O(N^2) : Square

반복문이 두번 있는 케이스

```
def print_each_n_times(li):
    for n in li:
        for m in li:
            print(n,m)
```

`O(n^2)` – 2차 시간 : 문제를 해결하기 위한 단계의 수는 입력값 `n`의 제곱.

- 두 개의 중첩 루프를 사용하여 두 개의 다른 콜렉션을 반복 할 경우(이중 for문) : O (n * m) -> O (n^2)
- 두 개의 중첩 루프를 사용하여 단일 컬렉션을 반복하는 경우 : O (n^2)

※ 피보나치 수열: $O(2^n)$

0. Big O and Time Complexity

Complexity	1	10	100
O(1)	1	1	1
O(log N)	0	2	5
O(N)	1	10	100
O(N log N)	0	20	461
O(N^2)	1	100	10000
O(2^N)	1	1024	1267650600228229401496703205376
O(N!)	1	3628800	화면에 표현할 수 없음!

Sorting Algorithms	공간 복잡도	시간 복잡도		
	최악	최선	평균	최악
Bubble Sort	O(1)	O(n)	$O(n^2)$	$O(n^2)$
Heapsort	O(1)	O(n log n)	O(n log n)	O(n log n)
Insertion Sort	O(1)	O(n)	$O(n^2)$	$O(n^2)$
Mergesort	O(n)	O(n log n)	O(n log n)	O(n log n)
Quicksort	O(log n)	O(n log n)	O(n log n)	$O(n^2)$
Selection Sort	O(1)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Shell Sort	O(1)	O(n)	$O(n \log n^2)$	$O(n \log n^2)$
Smooth Sort	O(1)	O(n)	O(n log n)	O(n log n)

Data Structures	Average Case			Worst Case		
	Search	Insert	Delete	Search	Insert	Delete
Array	O(n)	N/A	N/A	O(n)	N/A	N/A
Sorted Array	O(log n)	O(n)	O(n)	O(log n)	O(n)	O(n)
Linked List	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Doubly Linked List	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Stack	O(n)	O(1)	O(1)	O(n)	O(1)	O(1)
Hash table	O(1)	O(1)	O(1)	O(n)	O(n)	O(n)
Binary Search Tree	O(log n)	O(log n)	O(log n)	O(n)	O(n)	O(n)
B-Tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Red-Black tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
AVL Tree	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)