1. 알고리즘이란?

알고리즘이란?

- 어떤 문제를 해결하기 위해 밟아 나가는 연속적인 단계

- 명확함: 각 단계가 명료하고 간결하며 모호하지 않다.
- 효율성: 각 동작이 문제 해결에 기여
- 유한함: 알고리즘이 유한한 단계를 거친 후 종료

- + 정확성: 입력이 같으면 같은 결과를 내야한다.

좋은 알고리즘 이란?

1. 가독성

2. 속도가 빠르다

3. 효율적

4. 재이용하기 쉽다.

```python
a = [i for i in range(10) if i % 2 == 0]
print(a)

b = []
for i in range(10):
    if i % 2 == 0:
        b.append(i)

print(b)
```

시간 복잡도 – 빅 O 표기법

- 알고리즘의 효율에서 가장 중요한 부분은 데이터 크기 n이 커질 때 알고리즘의 단계가 얼마만큼 증가하는가

- 빅 O 표기법 : n이 커짐에 따라 알고리즘의 시간 또는 공간의 요건이 얼마나 커지는지를 나타내는 수학적 표기법

시간 복잡도 – 상수 시간

- 알고리즘이 n의 크기에 관계없이 동일한 단계만 필요한 경우

- Ex) 서점에서 매일 첫번째 방문 고객에게 무료로 책을 선물하는 경우

```
customer = [1,2,3,4,5]
free_books = customer[0]
```

시간 복잡도 – 로그 시간

- 데이터의 로그에 비례하여 알고리즘의 단계가 늘어날 때
- Ex) 알고리즘의 탐색 범위를 1/2f로 줄여 나가는 이진탐색
- O(log n)으로 표기

시간 복잡도 - 선형시간

- 데이터의 크기가 커지는 만큼 같은 비율로 단계가 늘어나는 알고리즘

- Ex)

```
cnt = 0
for i in range(10):
    cnt += i
```

- O(n)으로 표기

시간 복잡도 – 선형 로그 시간

- 로그 시간으로 실행되는 알고리즘을 n번 반복하는 형태
- 로그 시간 복잡도와 선형 시간 복잡도를 곱한 만큼 커짐

- Ex) 데이터를 작은 부분으로 나누고, 독립적으로 처리하는 형태
- 병합 정렬 같은 정렬 알고리즘

시간 복잡도 – 2차 시간

- N의 제곱에 정비례하는 알고리즘, O(n ** 2)로 표기
- Ex)

```
cnt = 0
for i in range(10):
    for j in range(10):
        cnt += i
        cnt += j
```

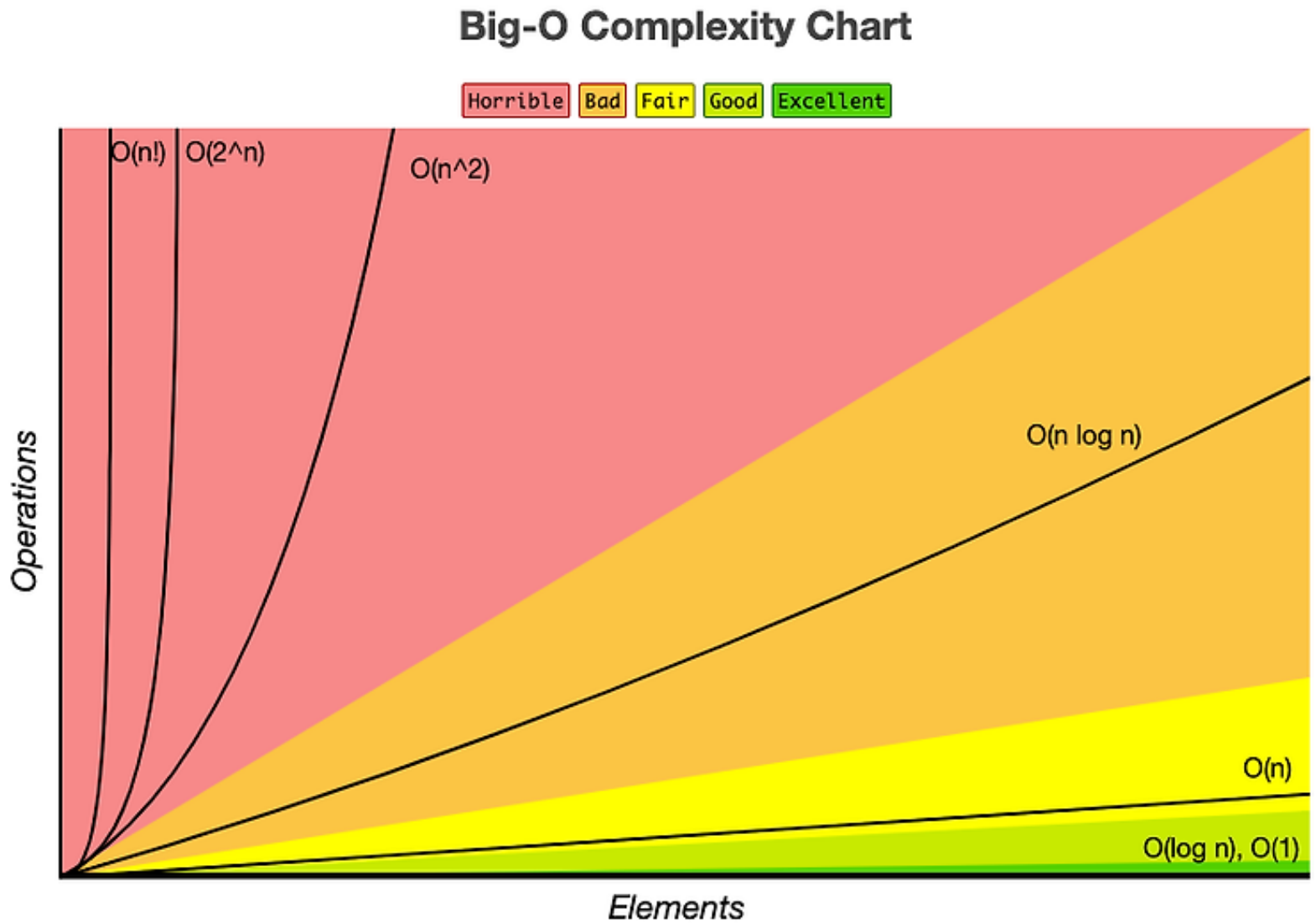
- 3차 시간 -> n의 세제곱에 정비례, O(n ** 3)으로 표기

시간 복잡도 – 지수 시간

- 상수 c를 n제곱 한 만큼 커지는 알고리즘, O(c**n)으로 표기
- 최악의 시간 복잡도로 꼽힘

- Ex)

```
pin = 931
n = len(pin)
for i in range(10**n):
    if i == pin:
        print(i)
```


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

공간 복잡도

- 알고리즘이 실행을 완료 할때까지 필요한 자원의 양
- **고정 공간, 데이터 구조 공간, 임시 공간의 메모리**를 얼마나 잘 사용 하는지를 나타냄

- **고정공간**: 프로그램 자체가 차지하는 메모리
- **자료구조 공간**: 데이터 세트 ex)탐색에 필요한 리스트를 저장하는데 필요한 메모리
- 임시 공간: 알고리즘에서 중간 처리를 위해 사용하는 메모리
  Ex) 임시로 리스트 사본을 만들 때 필요한 메모리

공간 복잡도 - 예시

- 상수 시간 ->

```
x = 1
n = 5
for i in range(1,n+1):
    x = x * i
print(x)
```

- 선형 시간 ->

```
x = 1
n = 5
a = []
for i in range(1,n+1):
    a.append(x)
    x = x * i
```