

코테스터디

Tetz Todo List

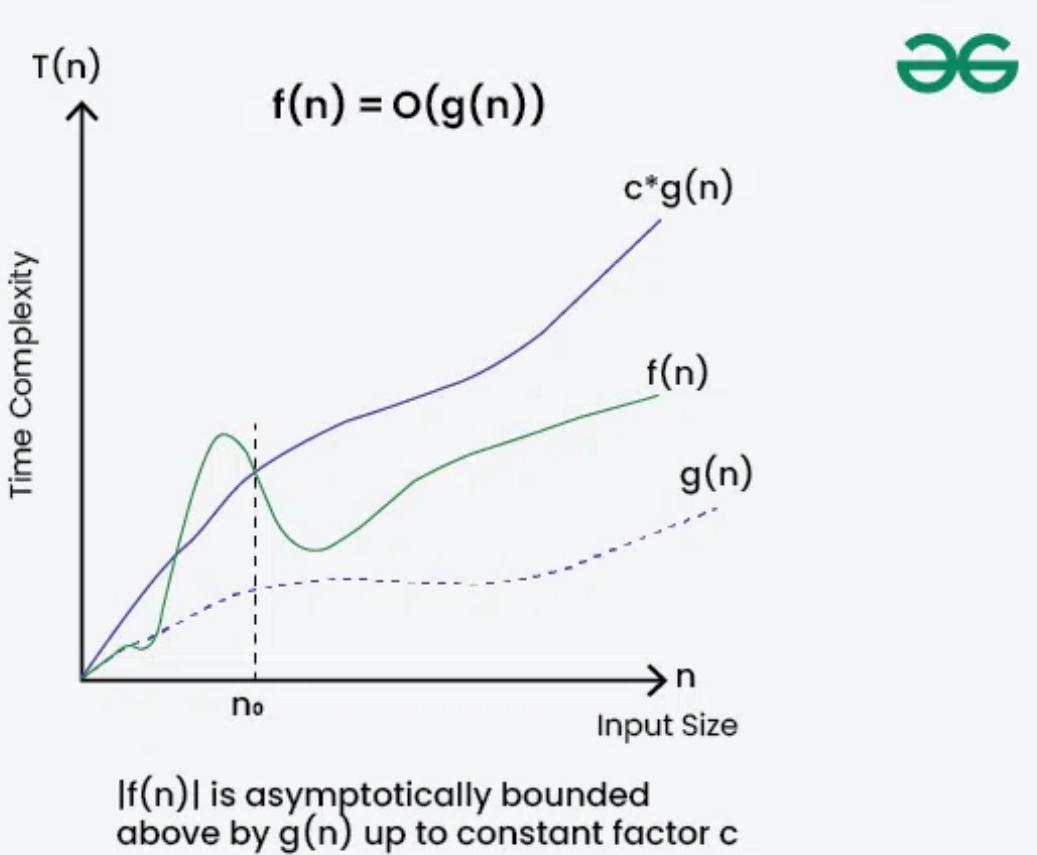
 추가

- 투두리스트 만들기 삭제
- DOM 마스터하기 삭제



좋은 알고리즘의
기준은 무엇일까?

Big O Analysis



복잡도

빅 오 표기법

Big O 표기법

빅오 표기법은는 알고리즘을 수학적으로 보여주는 표기법입니다.
빅오 표기법은 실제 알고리즘의 작동시간을 측정하기 위한 것이
아닌, 데이터의 추가에 따른 처리시간의 증가율을 통해 알고리즘의
성능을 예측하는것이 목표로 둡니다.
 $O(f(n))$ 과 같은 방식으로 표기합니다.

Big O 외의 표기법은?

총 3가지의 표기법이 존재합니다.

Big-O(빅-오) - 최악의 상황

Big- Ω (빅-오메가) - 최선의 상황

Big- Θ (빅-세타) - 평균적인 상황

위 표기법은 시간복잡도를 각각 최악, 최선, 중간(평균)의 경우에 대해서 나타내는 방법입니다.

복잡도

알고리즘의 성능과 효율성을 나타내는 척도로, 각 알고리즘이 주어진 특정 크기의 입력(n)을 기준으로 수행시간(연산) 혹은 사용공간이 얼마나 되는지 객관적으로 비교할 수 있는 기준을 제시합니다.

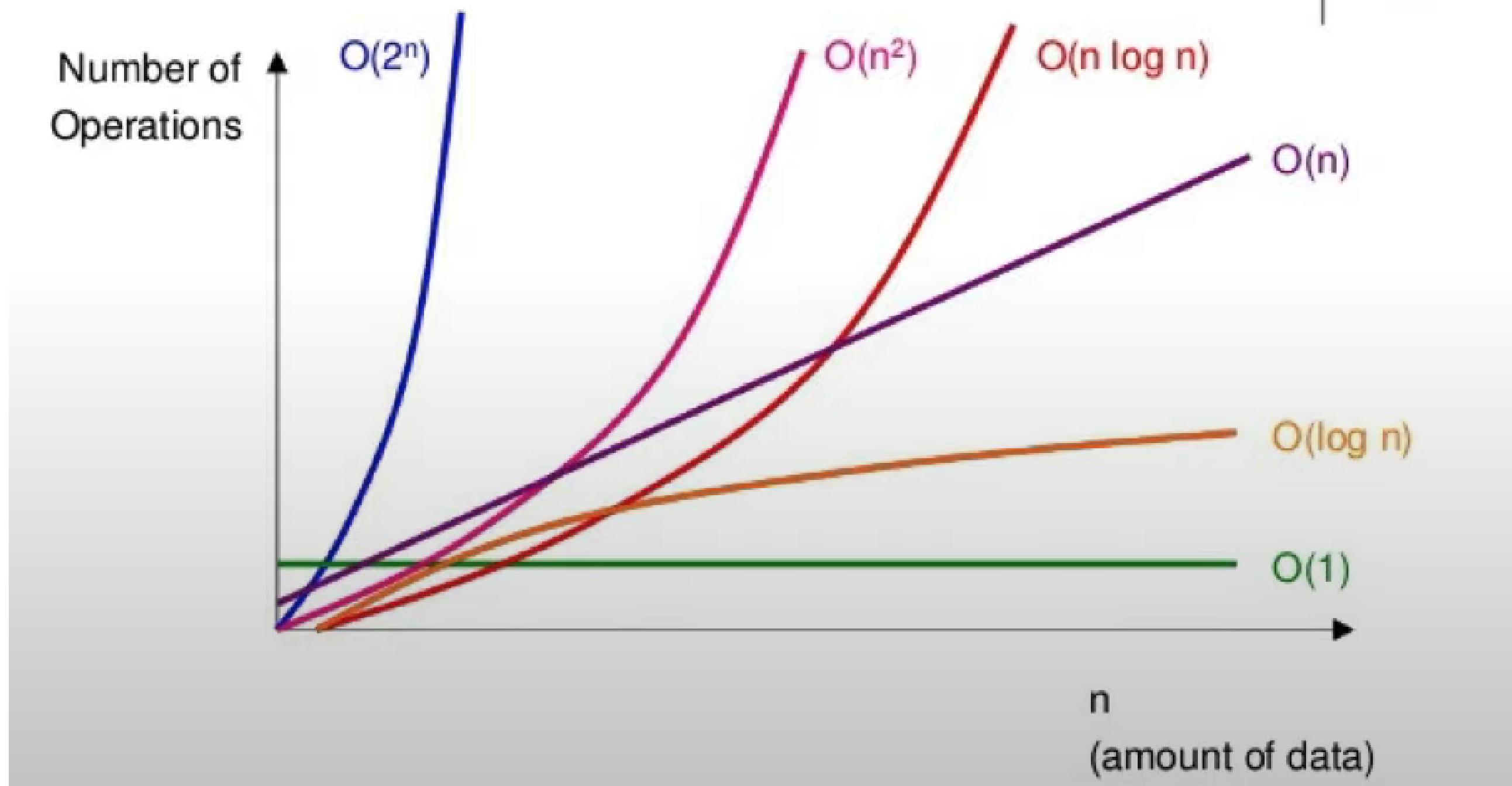
-->알고리즘이 수행을 시작하여 결과가 도출될 때까지 실행에 걸리는 시간이 짧고 연산하는 컴퓨터 내의 메모리와 같은 자원을 덜 사용하는 것이 효율적이라고 할 수 있습니다

왜 Big O 표기법을 사용할까?

빅오 표기법은 최악의 경우를 고려하므로,
프로그램이 실행되는 과정에서 소요되는 최악의 시간까지 고려할
수 있기 때문입니다. "최소한 특정 시간 이상이 걸린다" 혹은 "이
정도 시간이 걸린다"를 고려하는 것보다 "이 정도 시간까지 걸릴
수 있다"를 고려해야 그에 맞는 대응이 가능하기 때문입니다.

Big-O 그래프

Comparing Big O Functions



복잡도

네이비즘 서버시간 인터파크 티켓팅 예스24 티켓팅 멜론 티켓팅 커뮤니티 티켓팅 포도알 티켓팅 연습게임 채팅방 네이비즘 리포트

Are you ready? :) Fight! [즐겨찾기 추가](#) [트윗하기](#)

* 필기시험 면제 패키지*
청소년지도사 자격증 취득 **68%** 할인

어서와... 쓸데 없지만 아무나 다 알려준다는 표준시간부터 보여줄께 ...

2019년 08월 19일 10시 21분 38초

네이비즘에서 정확한 표준시간 확인도 좋지만 [고려대학교 수강신청 서버시간](#)을 확인하러 가볼까요?

정각알람듣기 1분전에도 2분전에도 3분전에도 빨간색 싫어 날짜 제거 쓸데 없는 밀리초 보기 [채팅방 입장](#)

빅데이터 기반의 AI로 취약부분만 빠르게 보완하는 고효율 학습! [AI 확인학습 PASS](#)

[공지] 네이비즘과 네이버시계의 차이점 자세히 알아보기

시간복잡도



공간복잡도

공간복잡도

공간 복잡도란 작성한 프로그램이 얼마나 많은 메모리를 차지하는지 분석하는 방법입니다.

최근 컴퓨터 성능의 비약적인 발전으로 중요성이 예전보다 많이 떨어지고 있지만, 데이터를 많이 다루는 빅데이터 분야에서는 여전히 중요한 지표로 다뤄지고 있습니다.

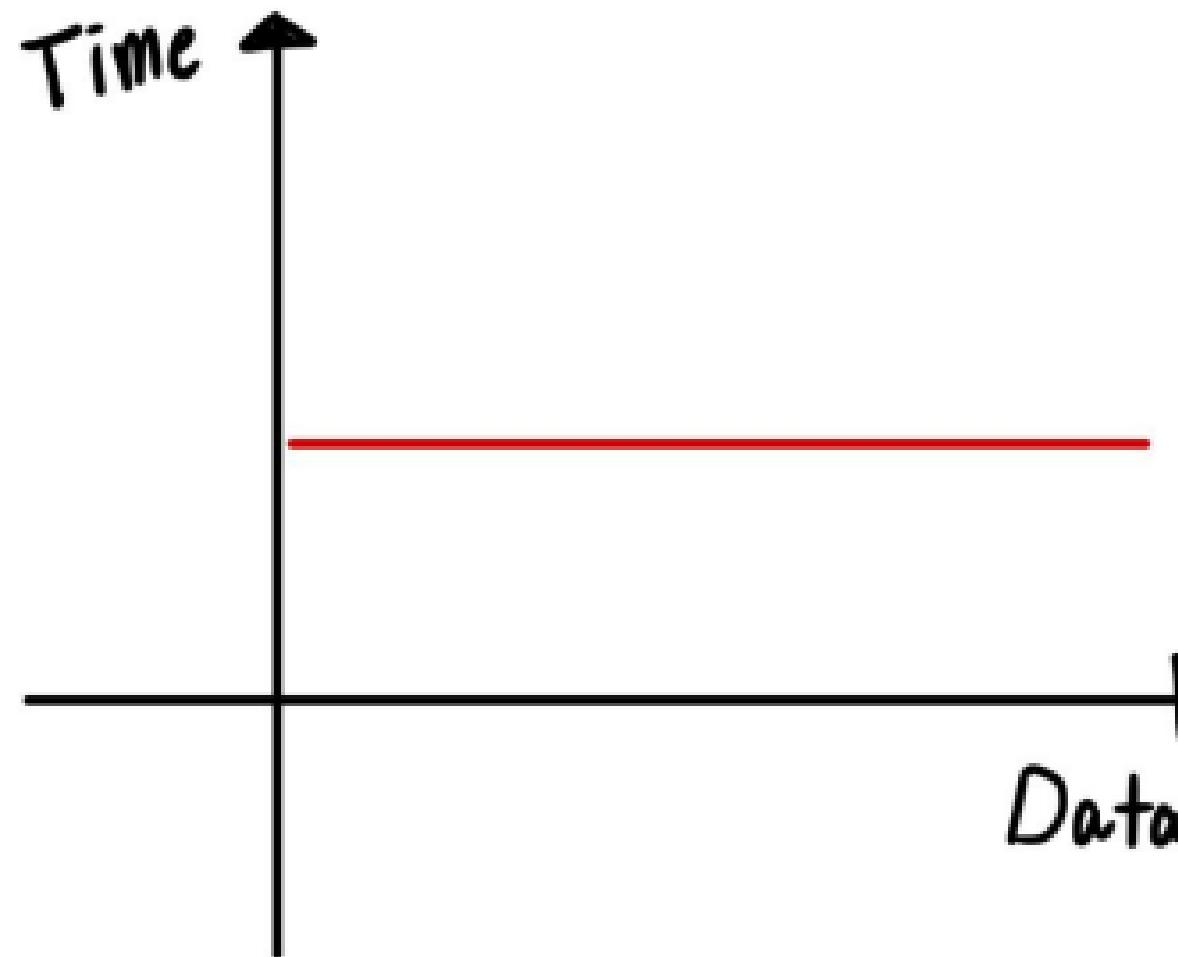
시간복잡도

특정 크기의 입력을 기준으로 할 때 필요한 연산의 횟수를 의미합니다.

시간 복잡도는 주로 빅 오 표기법을 통해서 나타냅니다.

$O(1)$ Constant Time (상수)

입력 데이터 크기에 상관없이 언제나 일정한 시간이 걸리는 알고리즘입니다.

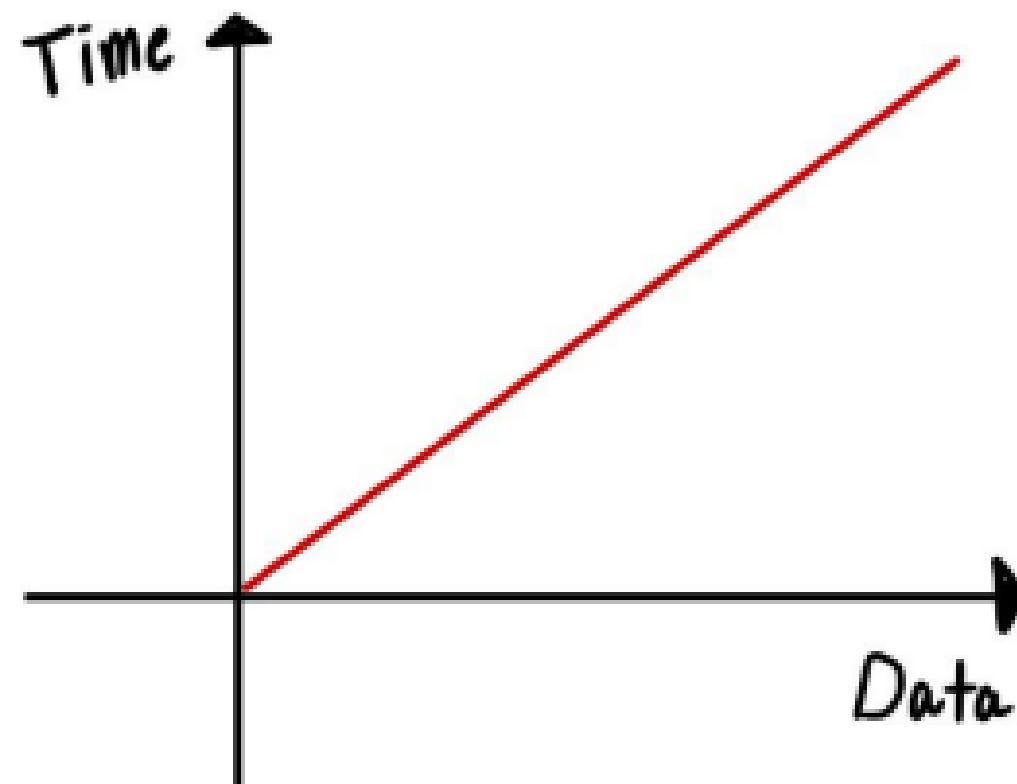


```
public static void main(String[] args) throws IOException {  
    int A = 1;  
    System.out.println(A);  
}
```

```
public static void main(String[] args) throws IOException {  
    String []arr = {"김구수", "이재용"};  
    System.out.println(arr[0]);  
}
```

$O(n)$ Linear Time (선형)

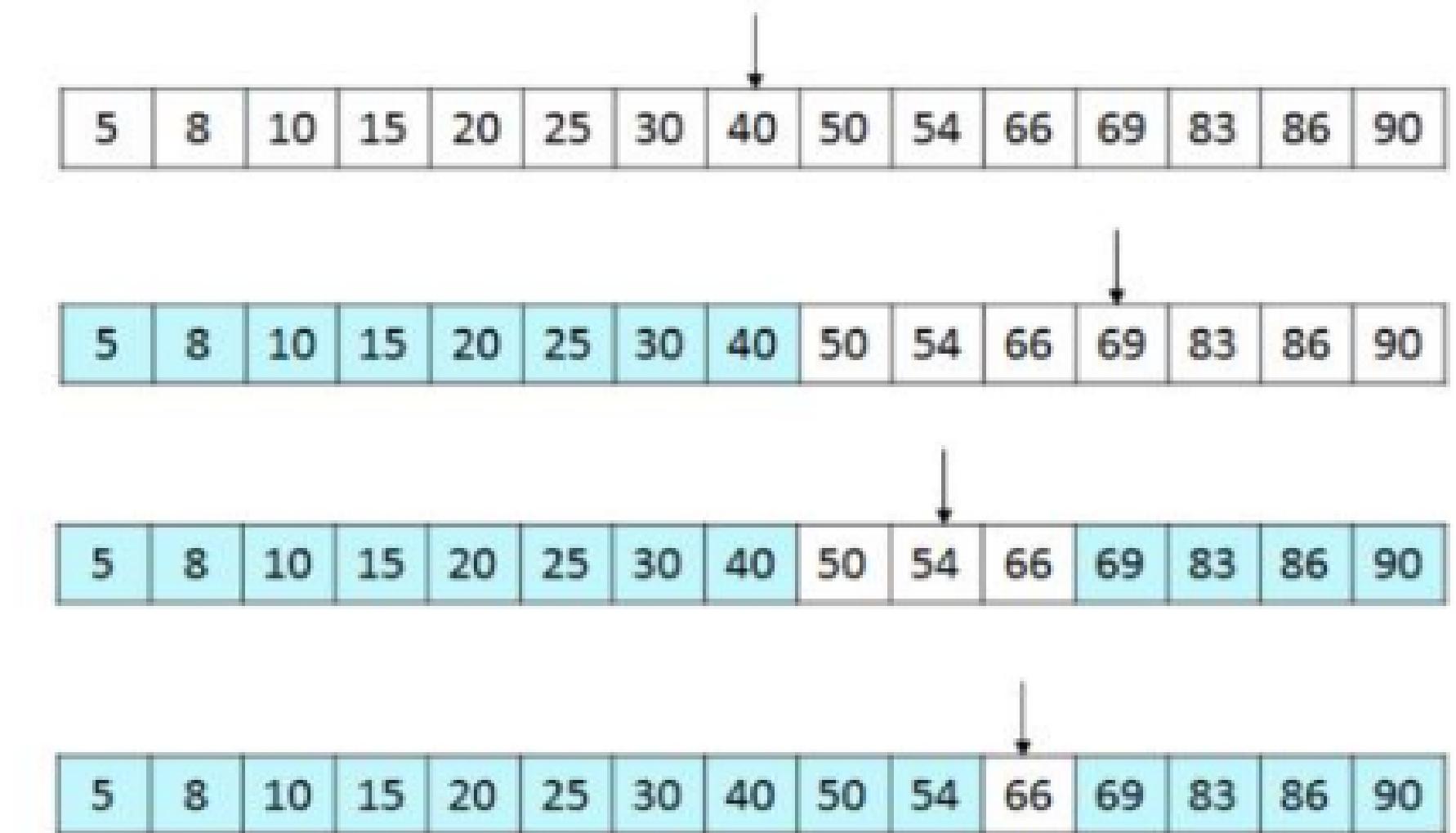
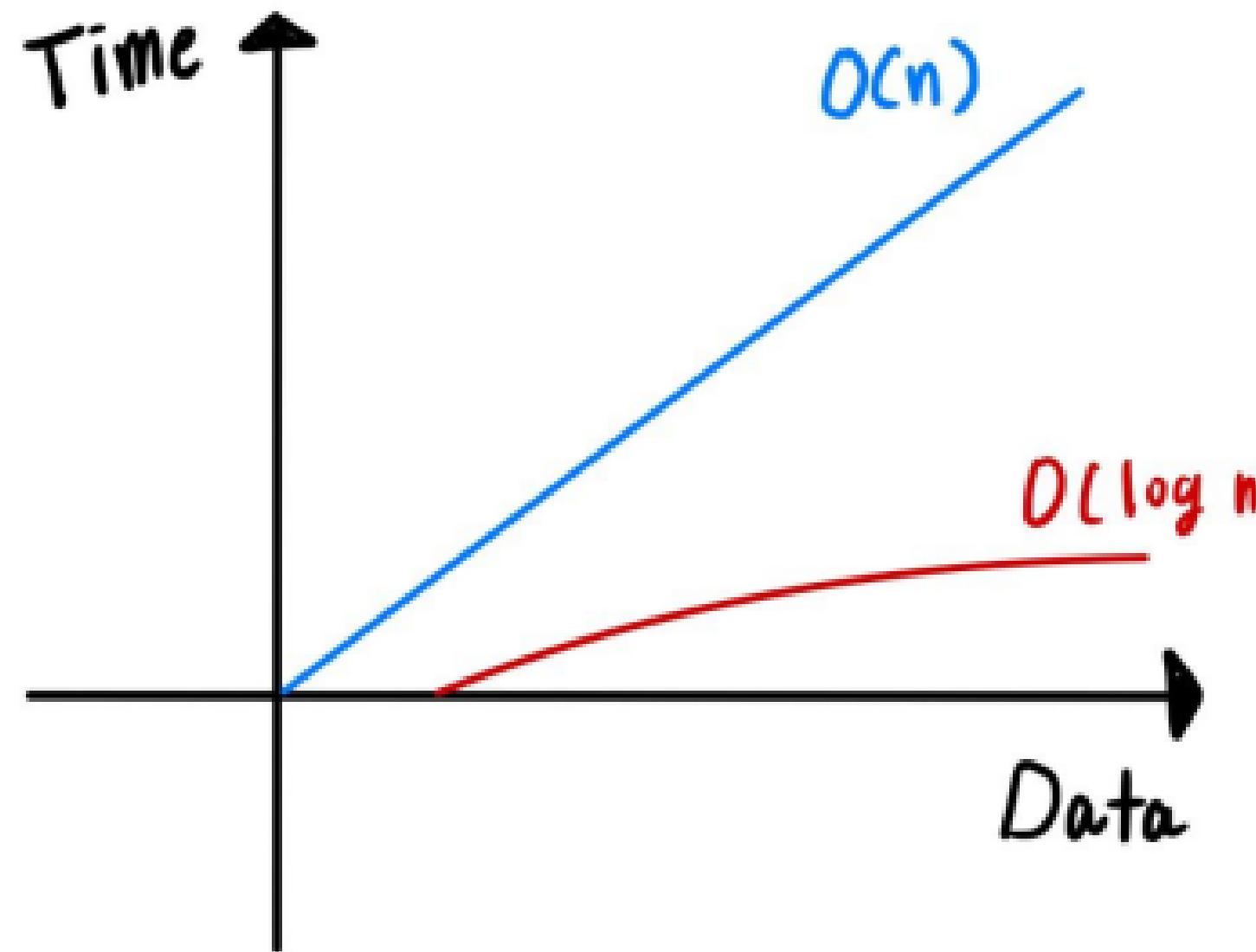
입력 데이터의 크기에 비례해서 처리 시간이 걸리는 알고리즘입니다.
 n 이 1번 늘어날 때마다 처리시간이 1 증가하여 선형적으로 증가합니다.
(n 크기만큼 처리시간이 증가)



```
public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int N = Integer.parseInt(br.readLine());
    int sum = 0;
    for (int i=0; i<N; i++){
        sum += i;
    }
    System.out.println(sum);
}
```

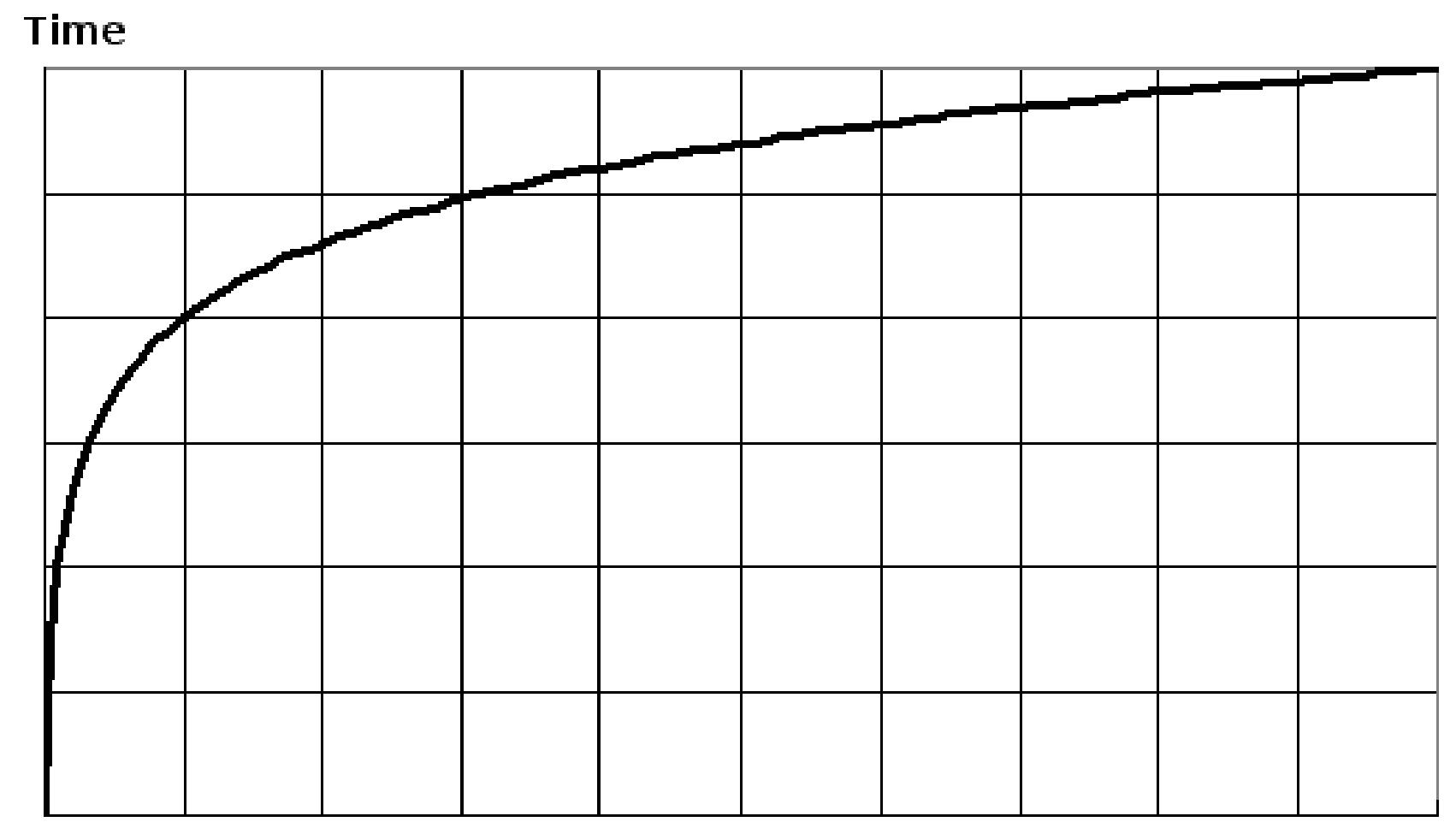
$O(\log n)$

$O(\log n)$ 의 대표적인 알고리즘은 이진 검색(binary search)입니다.



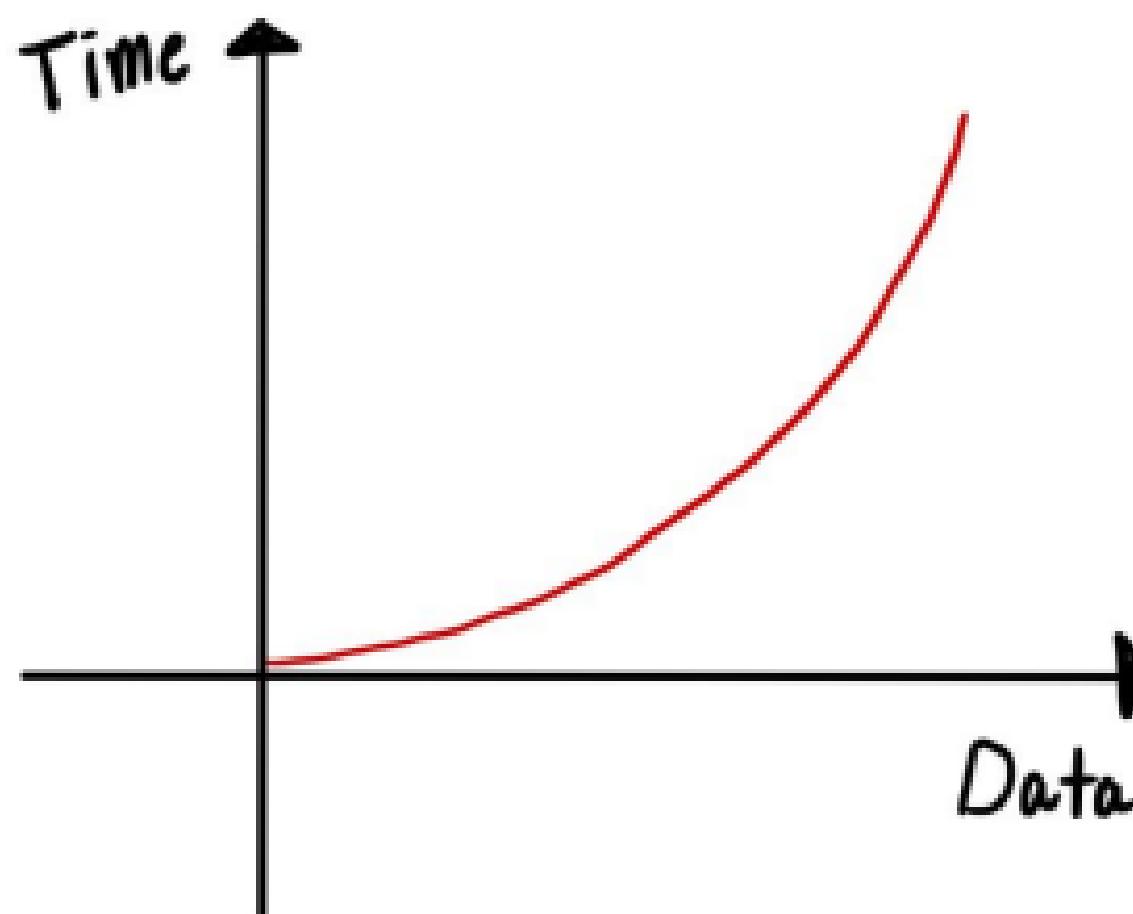
$O(n \log n)$

선형 로그 시간 복잡도를 의미합니다. 입력 크기에 따라 연산의 횟수가 증가하지만, 이 증가는 로그 함수의 성질에 따라 상대적으로 느리게 증가함을 말합니다.



$O(n^2)$ Quadratic Time

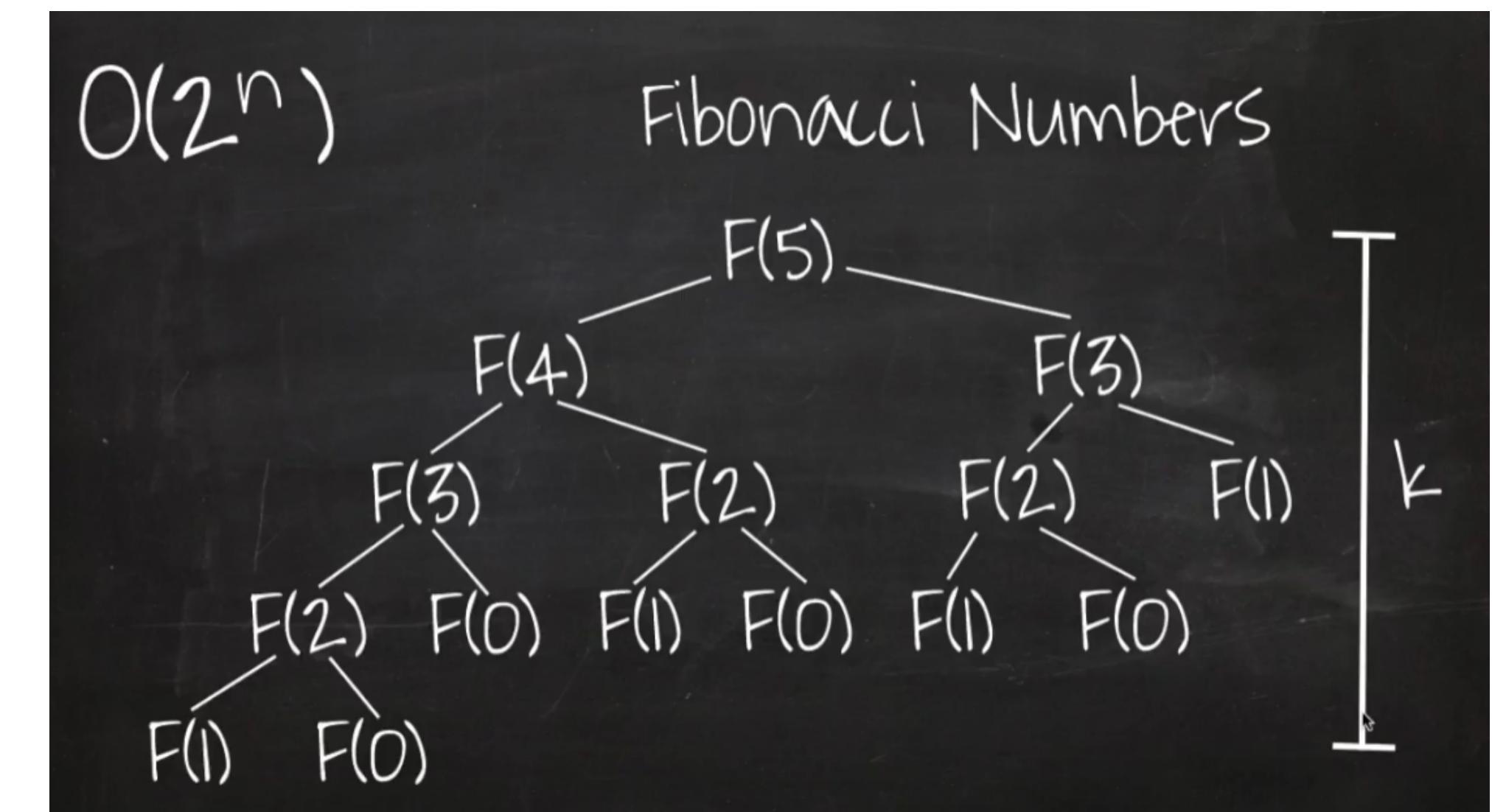
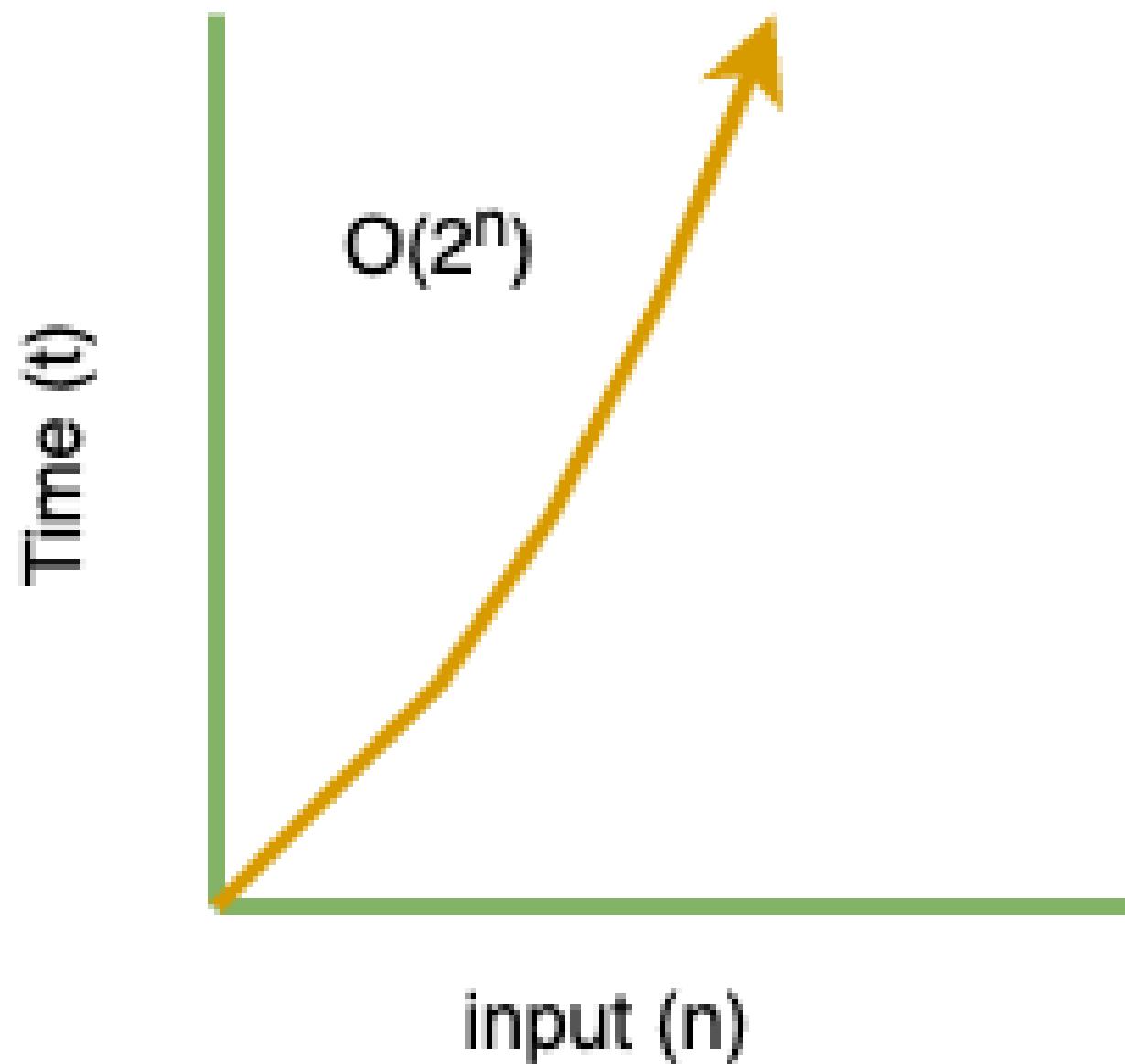
입력 데이터 n 만큼 반복하는데, 그 안에서 n 만큼 또 반복할 때의 표기 방법입니다.
데이터가 적을 때는 문제 없지만 많아질수록 수직상승합니다.



```
public static void main(String[] args) throws IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    int N = Integer.parseInt(br.readLine());
    int sum = 0;
    for (int i=0; i<N; i++){
        for(int j=0; j<N; j++){
            sum+=1;
        }
    }
    System.out.println(sum);
```

$O(2^n)$

n번 반복할 수록 2의 n제곱만큼 시간이 증가함을 의미합니다.
대표적인 예시로는 재귀로 구현한 피보나치 수열이 있습니다.



다음 코드의 시간복잡도는?

```
for(int i=0; i<N; i++){
    System.out.println(i);
}

for(int i=0; i<N; i++){
    System.out.println(i);
}
```

정답 : O(N)

```
for(int i=0; i<N; i++){
    System.out.println(i);
}

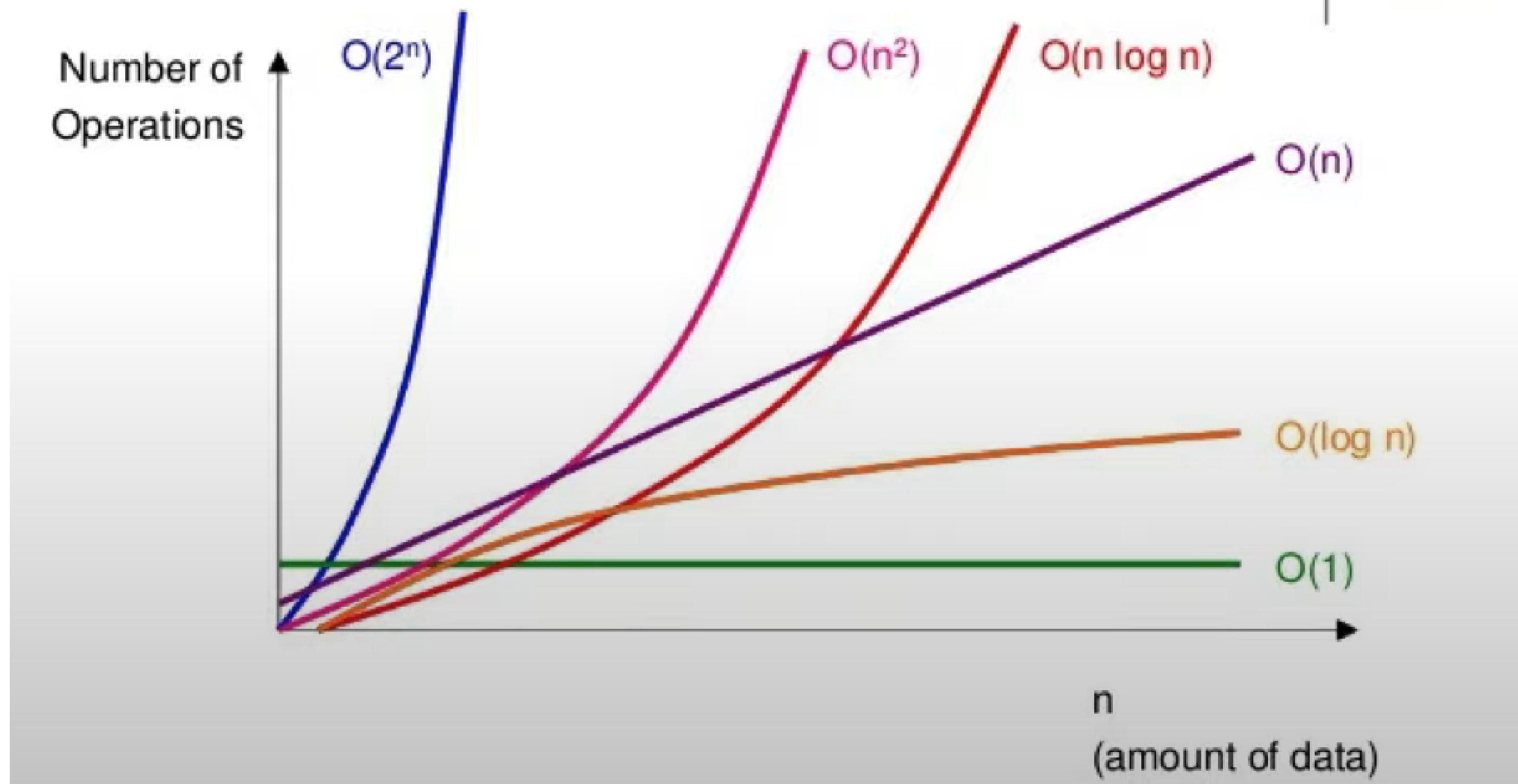
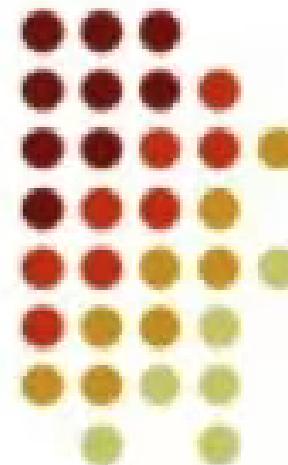
for(int i=0; i<N; i++){
    System.out.println(i);
}
```

왜 $O(2N)$ 이 아니라 $O(N)$ 인가?

데이터의 추가에 따른 처리시간의 증가율을 통해 알고리즘의 성능을 예측하는것이 목표로 둡니다.

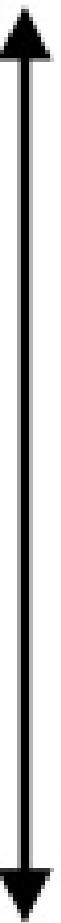
하지만 2와 같은 상수 같은 경우는 증가하는 정도가 정해져 있기 때문에, Big-O의 영향을 주는 요소로서 보지 않는 것입니다.

Comparing Big O Functions



Big-O: functions ranking

BETTER



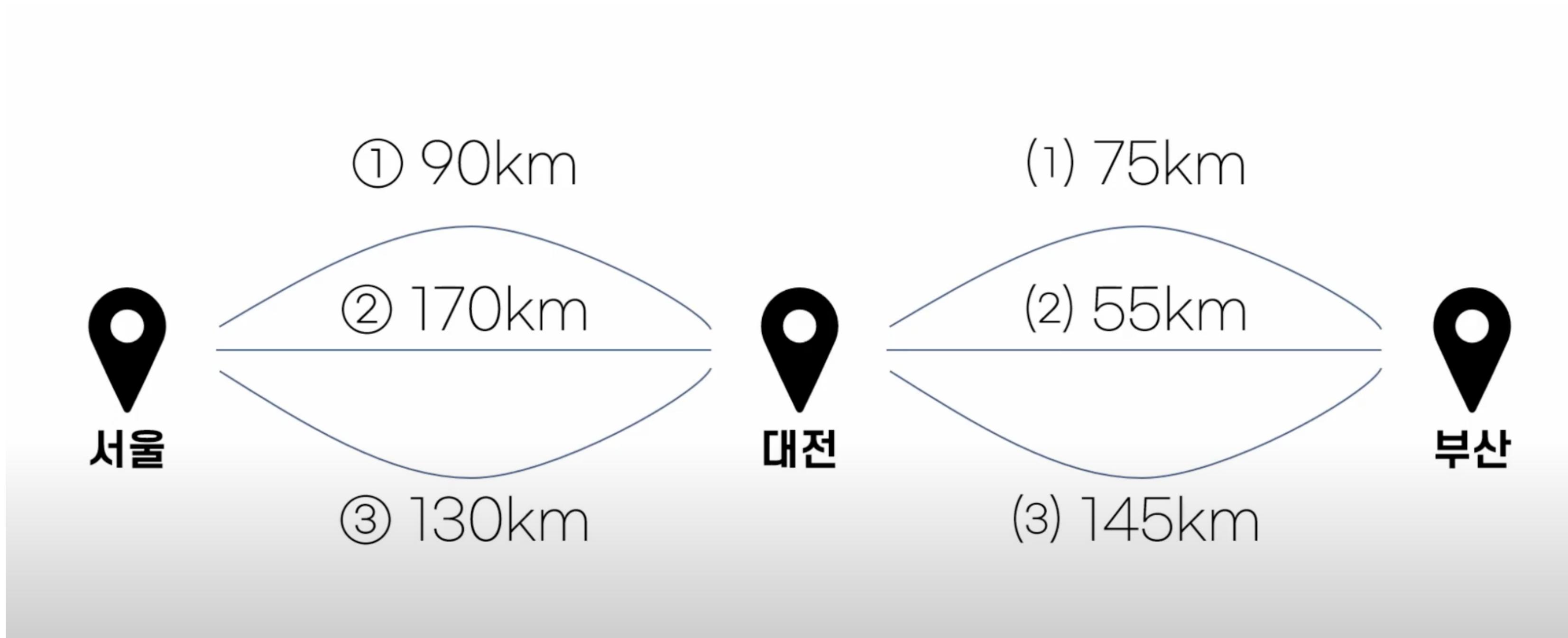
WORSE

- $O(1)$ constant time
- $O(\log n)$ log time
- $O(n)$ linear time
- $O(n \log n)$ log linear time
- $O(n^2)$ quadratic time
- $O(n^3)$ cubic time
- $O(2^n)$ exponential time

그리드 알고리즘



서울에서 부산까지 최소 거리로 가는 방법은?



그리디 알고리즘 Greedy Algorithm

그리디 알고리즘은 '탐욕적 알고리즘'이라고도 불리며, 주어진 문제를 단순하고 탐욕적인 방식으로 푸는 알고리즘입니다.
여기서 '탐욕적'이라는 말은 '현재 상황에서 가장 좋은 것만을 선택하는 방법'을 의미합니다.

그리디 알고리즘 언제 적용하면 될까?

1) 탐욕스러운 선택 조건

앞의 선택이 이후의 선택에 영향을 주지 않아야 합니다.

2) 최적 부분 구조 조건

전체 문제에 대한 최적해가 부분 문제에 대해서도 최적해야 합니다.

그리디 알고리즘의 풀이방법

1. 해의 후보 중에서 하나를 선택합니다.
2. 선택한 후보가 문제의 제약 조건에 부합하는지 확인합니다.
3. 부합한다면, 선택한 후보를 정답에 추가하고, 문제의 제약 조건을 수정합니다
4. 1-3의 과정을 반복합니다.

그리드 알고리즘의 예시문제

<https://www.acmicpc.net/problem/5585>

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	45296	29669	25287	65.449%

문제

타로는 자주 JOI잡화점에서 물건을 산다. JOI잡화점에는 잔돈으로 500엔, 100엔, 50엔, 10엔, 5엔, 1엔이 충분히 있고, 언제나 거스름돈 개수가 가장 적게 잔돈을 준다. 타로가 JOI잡화점에서 물건을 사고 카운터에서 1000엔 지폐를 한장 냈을 때, 받을 잔돈에 포함된 잔돈의 개수를 구하는 프로그램을 작성하시오.

입력

입력은 한줄로 이루어져있고, 타로가 지불할 돈(1 이상 1000미만의 정수) 1개가 쓰여져있다.

출력

제출할 출력 파일은 1행으로만 되어 있다. 잔돈에 포함된 매수를 출력하시오.

예제 입력 1 복사

```
380
```

예제 출력 1 복사

```
4
```

풀이방법 1

```
public static void main(String[] args) throws IOException {
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int cost = Integer.parseInt(bf.readLine());
    int remain = 1000-cost; //남은돈
    int coins = 0; //개수
    while(remain>0){
        if(remain >= 500){
            coins += remain/500;
            remain %= 500;
        }
        else if(remain >= 100){
            coins += remain/100;
            remain %= 100;
        }
        else if(remain >= 50){
            coins += remain/50;
            remain %= 50;
        }
        else if(remain >= 5){
            coins += remain/5;
            remain %= 5;
        }
        else if(remain >= 1){
            coins += remain;
            remain %= 1;
        }
    }
    System.out.println(coins);
}
```

풀이방법 2

```
public static void main(String[] args) throws IOException {
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int cost = Integer.parseInt(bf.readLine());
    int remain = 1000-cost; //남은돈
    int coins = 0; //개수
    int []arr = {500,100,50,10,5,1};
    for(int i=0; i<6; i+){
        coins += remain/arr[i];
        remain %= arr[i];
    }
    System.out.println(coins);
}
```

시간복잡도는 몇일까요?

```
public static void main(String[] args) throws IOException {
    BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
    int cost = Integer.parseInt(bf.readLine());
    int remain = 1000-cost; //남은돈
    int coins = 0; //개수
    int []arr = {500,100,50,10,5,1};
    for(int i=0; i<6; i++){
        coins += remain/arr[i];
        remain %= arr[i];
    }
    System.out.println(coins);
}
```

그리디 알고리즘의 장점과 단점

<장점>

1. 간단하며 직관적인 방법 : 구현이 간단하고 이해하기 쉬운 방법입니다.
2. 빠른 계산 시간 : 일반적으로 계산 시간이 짧아서 대용량 데이터에도 적용 가능합니다.

<단점>

최적해가 보장되지 않음 : 문제에 따라 부분적으로 최적해일 수 있지만, 전체적으로는 최적해가 아닐 수 있다.

끌났다 — !!

