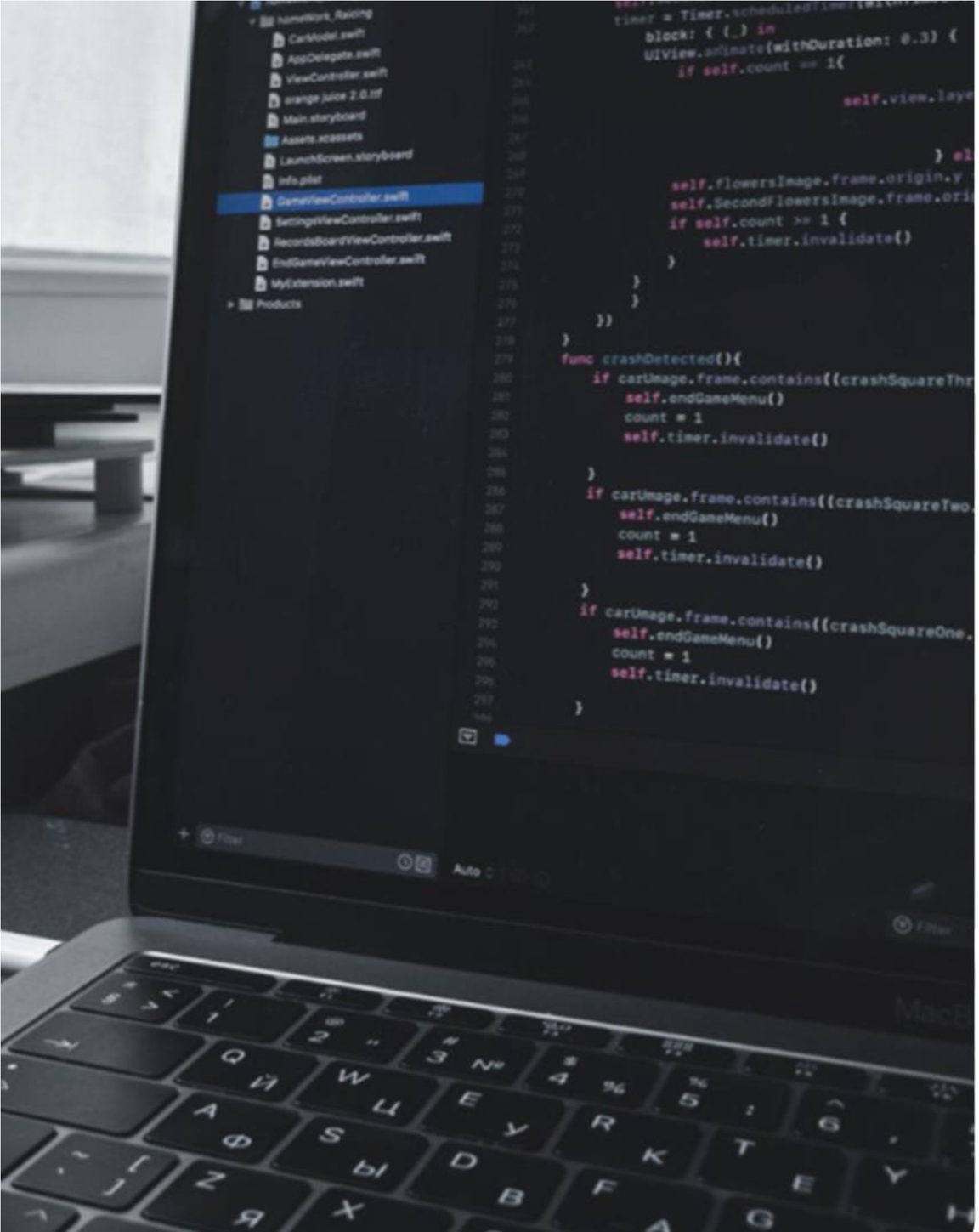


2022 1학기 수학 발표

지수, 로그와 시간 복잡도

20620 최재혁

목차

01. 시간 복잡도는 무엇인가?

02. Big-O 표기법

03. 그래프와 코드를 통한 Big-O표기법 확인

어떤 문제를 해결하기 위한 **효율적 절차**

어떤 방법이 **더 빠르게 해답을 구하고**
메모리 공간을 적게 차지할 지 계산하는 절차

시간 초과

제출 번호	아이디	문제	결과
41844323	gghh3017	5 2751	맞았습니다!!
41421399	gghh3017	5 2751	시간 초과
41421358	gghh3017	5 2751	시간 초과

시간 제한	메모리 제한
2 초	256 MB

문제

N개의 수가 주어졌을 때, 이를 오름차순으로 정렬하는 프로그램을 작성하시오.

입력

첫째 줄에 수의 개수 N(1 ≤ N ≤ 1,000,000)이 주어진다. 둘째 줄부터 N개의 줄에는 수가 주어진다. 이 수는 절댓값이 1,000,000보다 작거나 같은 정수이다는다.

출력

첫째 줄부터 N개의 줄에 오름차순으로 정렬한 결과를 한 줄에 하나씩 출력한다.

시간 복잡도

입력 값의 변화에 따라 연산을 실행할 때,
연산 횟수에 비해 시간이 얼마만큼 걸리는가?

↓

효율적 알고리즘
입력값이 커짐에 따라
증가하는 시간의 비율을 최소화

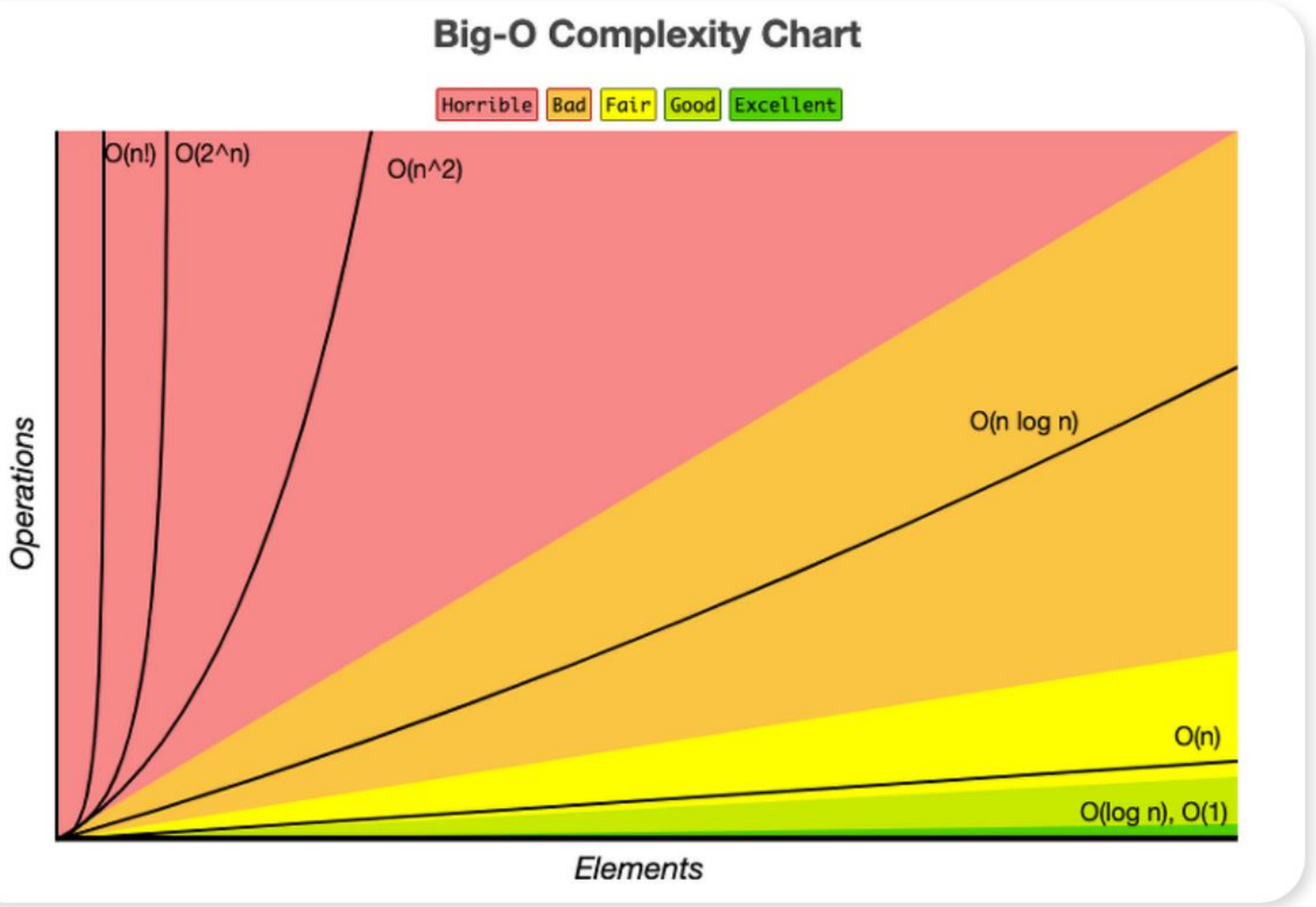
시간 복잡도
표기 방법

Big-O
최악의 경우

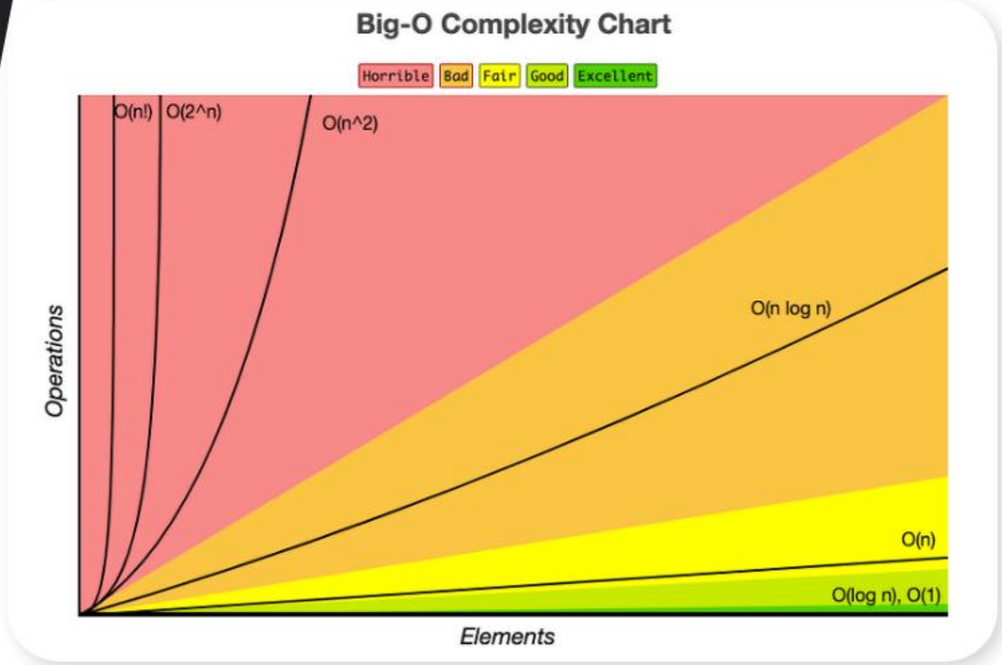
Big-Ω
최선의 경우

Big-θ
중간(평균)

Big-O 표기법


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

Big-O 표기법



좋음(Better)

나쁨(Worse)

순위	명칭
$O(1)$	상수 시간(Constant time)
$O(logN)$	로그 시간(Log time)
$O(N)$	선형 시간
$O(NlogN)$	로그 선형 시간
$O(N^2)$	이차 시간
$O(N^3)$	삼차 시간
$O(2^n)$	지수 시간

Big-O 표기법

```
void Insertion(int n) {
    int key;
    for (int i = 1; i <= n - 1; i++) {
        key = a[i];
        int j;
        for (j = i - 1; j >= 0; j--) {
            if (a[j] >key){
                a[j + 1] = a[j];
            }
            else{
                break;
            }
        }
        a[j + 1] = key;
    }
}
```

외부 for문 : 연산 (n-1)회

교환연산이 최대 n-1회

내부 for문 : 최대 i회

비교 연산은 최대

(n-1)+(n-2)...+1 = n(n-1)/2회

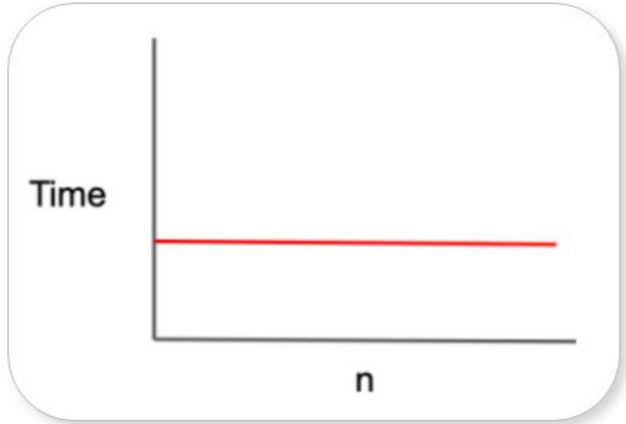
교환연산이 최대 n(n-1)/2회

교환연산이 최대 n-1회

$$1 * (n-1) + 2 * (n(n-1)/2) + 1 * (n-1) = n\%2 + n-2 = O(n^2)$$

O(1) :상수

입력 값의 크기에 관계없이
일정한 실행 시간을 가지는 알고리즘

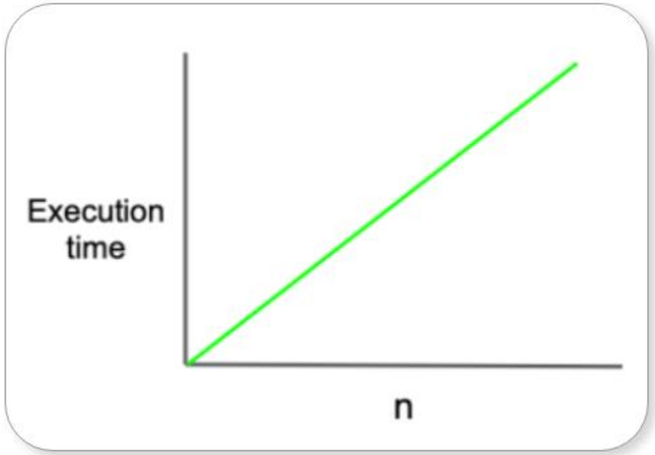



```c
#include <stdio.h>

int main() {
    printf("Hello, World!");
}
```

위 코드에서 입력이 없고, Hello, World!라는 출력만 있음
따라서, 입력 값에 관계 없이 일정한 시간을 가지게 됨

O(n) :선형

입력 값이 증가함에 따라 시간 또한 같은 비율로 증가

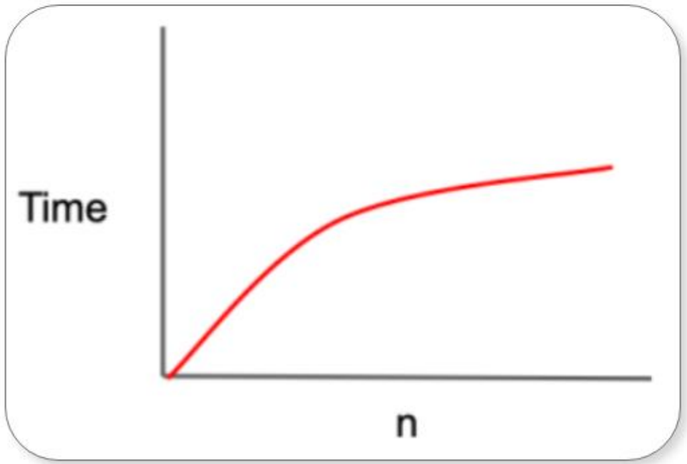



```c
#include <stdio.h>

int main() {
    for(int i = 1; i<=5; i++){
        printf("Hello, World! #%d\n", i);
    }
}
```

위 코드에서, for(반복문)으로, 1부터 5까지 1씩 증가하면서
반복하므로, 같은 비율로 입력에 따른 출력 시간이 증가한다.

O(log n) :로그

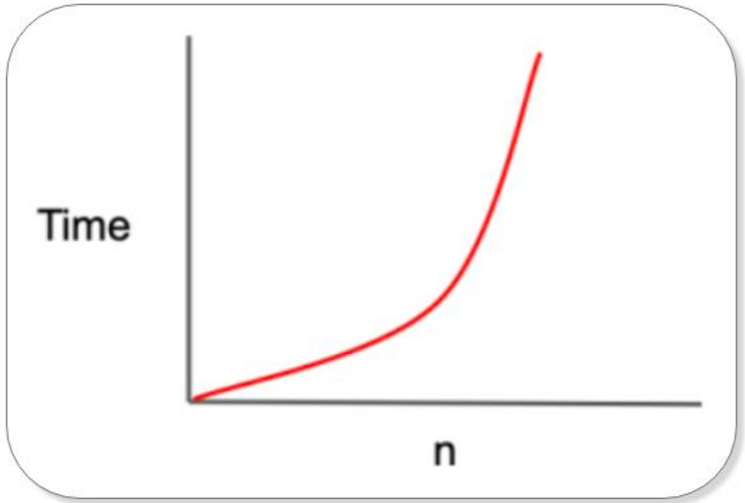
UP & Down 게임과 같이
범위를 줄여가면서
정답을 찾는 알고리즘 등에
사용된다




```c
#include <stdio.h>
#include <string.h>

void quickSort(int arr[], int L, int R) {
      int left = L, right = R;
      int pivot = arr[(L + R) / 2];
      int temp;
      do
      {
        while (arr[left] < pivot){
            left++;
        }

        while (arr[right] > pivot){
            right--;
        }

        if (left<= right) {
            temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            left++;
            right--;
        }
      } while (left<= right);

    if (L < right)
        quickSort(arr, L, right);

    if (left < R)
        quickSort(arr, left, R);
}
```

O(n2) : 이차

입력 값이 증가함에 따라 시간이 n의 제곱수의
비율로 증가하는 것을 의미함

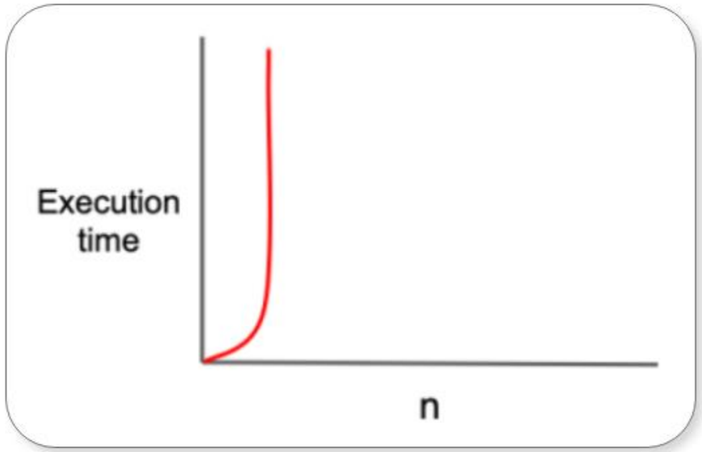



```c
#include <stdio.h>

int main() {
    for(int i = 0; i<5; i++){
        for(int j = 0; j<5-i; j++){
            if(a[j] > a[j+1]){
                int tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
        }
    }
}
```

n = 5라 하였을 때, 바깥쪽 반복문은 5번 반복하고,
안쪽 반복문은 최대 5번 반복하므로 O(n2)의 복잡도를 가짐

O(2n) : 지수

기하급수적 복잡도라고 부르며, Big-O 표기법 중
가장 느린 시간 복잡도를 가진다.




```
int fibonacci(int n) {
  if (n <= 1) {
    return 1;
  }
  return fibonacci(n - 1) + fibonacci(n - 2);
}
```

n을 40정도로만 두어도,
굉장히 오랜 시간이 소요됨을 알 수 있다.