ALOHA

#1주차멘토링

다양한 문제해결 기법

언제 어떤 기법을 사용해야 할까?

Brute-force (완전탐색법)

Greedy (탐욕법)

Divide & Conquer (분할정복법)

시간 복잡도와 효율성 추정

시간복잡도 (time complexity)

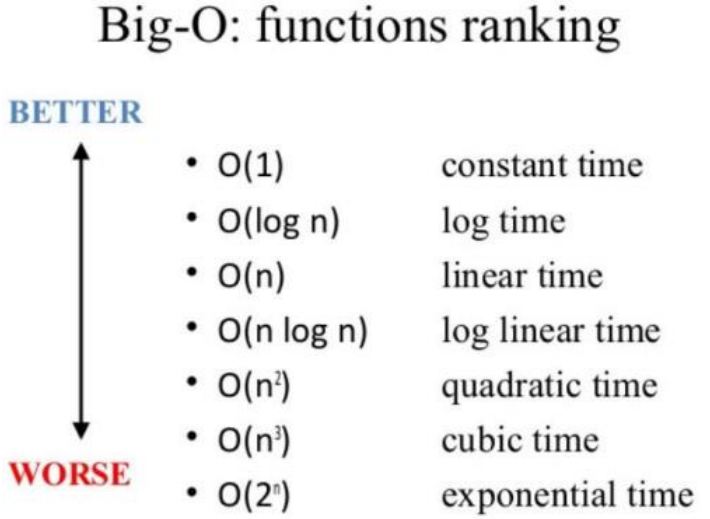
입력에 대해 알고리즘이 얼마만큼의 시간을 사용할지 근사적으로 표현한 것

표기: $O(f(n))$

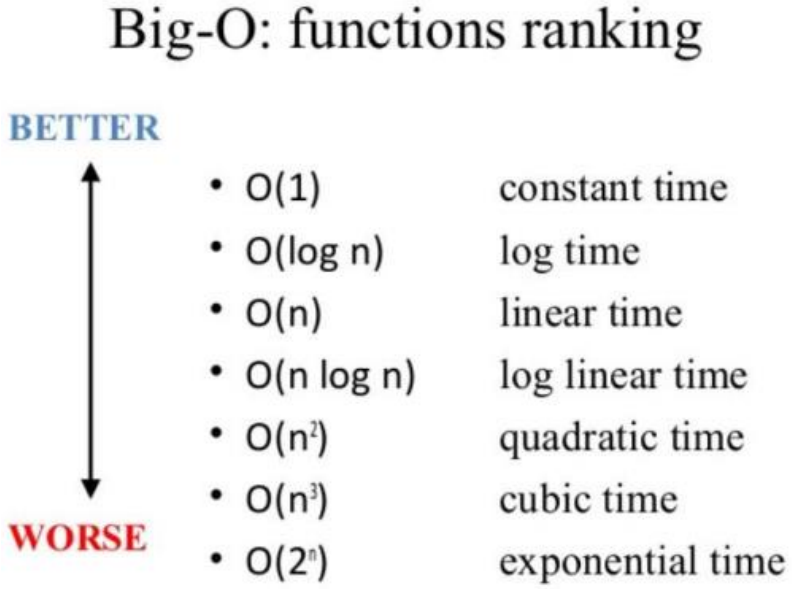
n: 입력의 크기

f: 대략적인 연산 횟수

(가장 복잡도가 큰 항 외에는 무시 / 계수, 상수 무시)



시간 복잡도와 효율성 추정



- **반복문** $O(nm)$ **vs** $O(n^2)$

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        ... // O(1)의 단순한 연산
    }
}
```

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n + 3; j++) {
        ... // O(1)의 단순한 연산
    }
}

for (int i = 0; i < n; i++) {
    ... // O(1)의 단순한 연산
}
```

- **재귀함수** $O(n)$ **vs** $O(2^n)$

```
void recursive(int n)
{
    if (n == 1) return;
    recursive(n - 1);
}
```

```
void recursive(int n)
{
    if (n == 1) return;
    recursive(n - 1);
    recursive(n - 1);
}
```

시간 복잡도와 효율성 추정

일반적으로 컴퓨터는 1초에 수 억 번 (10^8~10^9)의 연산을 수행할 수 있다.

따라서 문제의 시간 제한과 입력의 크기를 확인한 후

적절한 효율성의 알고리즘을 선택하는 것이 중요하다!

시간 제한	메모리 제한
1 초	128 MB

시간 제한이 빡빡하다면 복잡하더라도 효율적인 알고리즘을.
시간 제한이 널널하다면 효율적이진 않더라도 간단한 알고리즘을.

입력의 크기	추정 시간 복잡도
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$, $O(n)$
n 이 클 때	$O(1)$, $O(\log n)$

예제1 – Brute-force vs Greedy

최대 부분 배열 합 구하기

배열에서 연속해 있는 값들을 택해 그 합을 최대로 만드는 문제

이런 배열이 있다면 부분 배열 [2,4,-3,5,2]의 합 10이 최대 값이다.

예제1 – Brute-force vs Greedy

```cpp
int best = 0;
for (int a = 0; a < n; a++) {
    for (int b = a; b < n; b++) {
        int sum = 0;
        for (int k = a; k <= b; k++) {
            sum += arr[k];
        }
        best = max(best,sum);
    }
}
std::cout << best << "\n";
```

Brute-force

시간 복잡도: $O(n^3)$

```cpp
int best = 0;
for (int a = 0; a < n; a++) {
    int sum = 0;
    for (int b = a; b < n; b++) {
        sum += arr[b];
        best = max(best,sum)
    }
}
std::cout << best << "\n";
```

최적화된 Brute-force

시간 복잡도: $O(n^2)$

```cpp
int best = 0, sum = 0;
for (int k = 0; k < n; k++) {
    sum = max(arr[k], sum + arr[k]);
    best = max(best, sum);
}
std::cout << best << "\n";
```

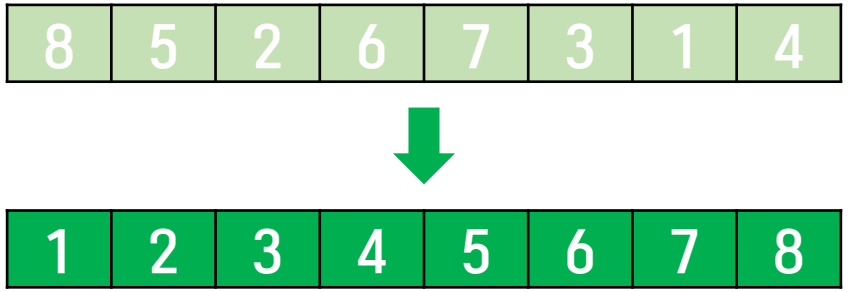
Greedy

시간 복잡도: $O(n)$

예제2 – Brute-force vs Greedy vs 분할정복

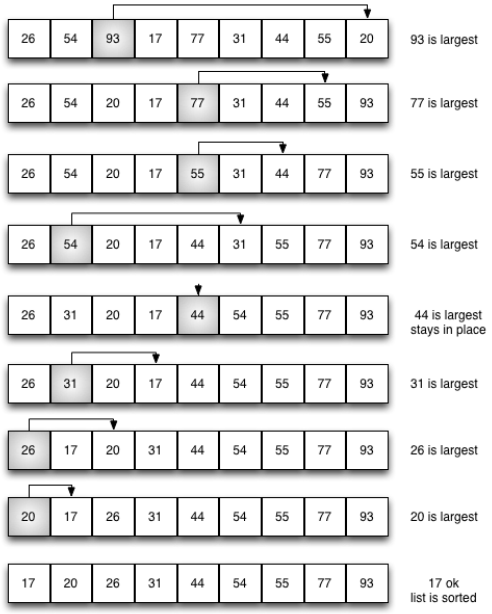
정렬 알고리즘

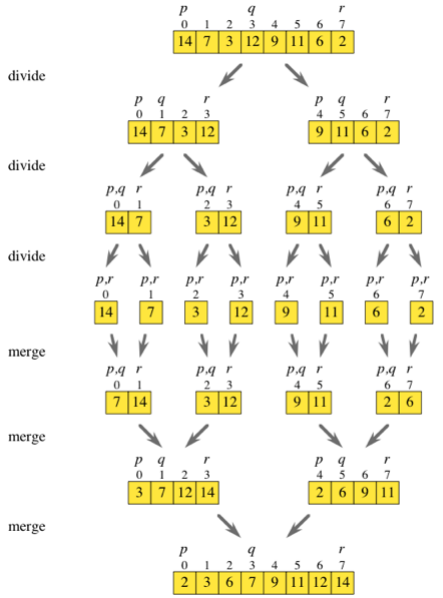
배열의 값을 오름차순 등으로 정렬하는 알고리즘



예제2 – Brute-force vs Greedy vs 분할정복

모든 가능한 순서에 대해 (n!)
정렬되었는지 판별 (n-1)





Brute-force

시간 복잡도: $O(n * n!)$

선택 정렬 (Greedy)

시간 복잡도: $O(n^2)$

병합 정렬 (분할정복)

시간 복잡도: $O(n \log n)$

다음 장은 퀴즈입니다

QUIZ

1. 다음 중 탐욕법을 이용한 경우로 가장 올바른 것은?

① 어떤 수의 집합에서 세 수의 합의 최댓값을 구하기 위해 모든 세 수의 조합을 확인해본다.

② 8자리 와이파이 비번을 뚫기 위해 8자리의 모든 문자 조합을 대입해본다.

③ 배열을 오름차순으로 정렬하기 위해 merge sort를 이용한다.

④ 수강신청을 할 때 가장 경쟁률이 높은 과목부터 신청한다.

QUIZ

2. 다음은 분할정복법을 이용하여 효율적인 거듭제곱 함수를 구현한 것이다.

n에 대하여 다음 코드의 시간 복잡도로 가장 알맞은 것은?

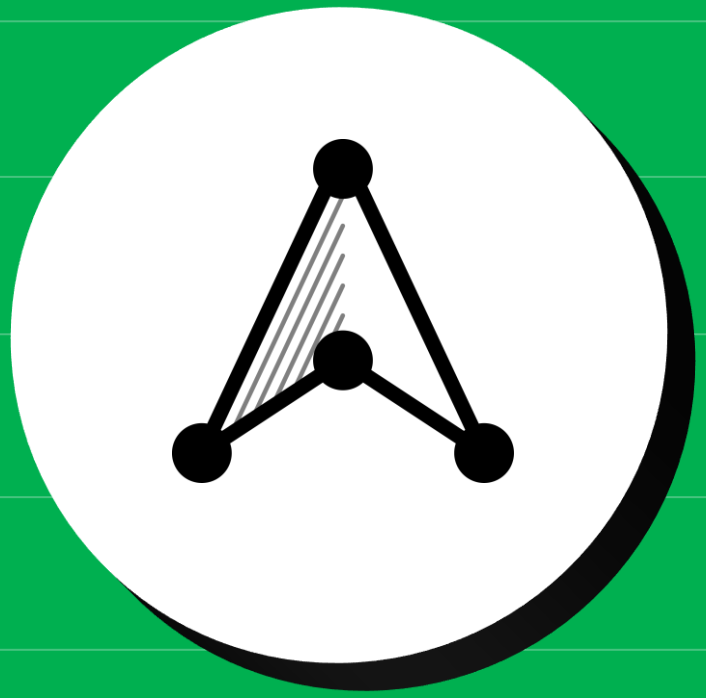
```
typedef long long ll;
ll pow(ll x, ll n)
{
    ll ans = 1;
    while (n > 0) {
        if (n % 2) ans *= x; // a가 홀수면 x곱함
        x = x * x;
        n /= 2;
    }
    return ans;
}
```

① $O(n)$ ② $O(n^2)$ ③ $O(\log n)$ ④ $O(\mathrm{n}\log n)$ ⑤ $O(1)$

퀴즈 정답

QUIZ 정답

1. ④
2. ③



다음 시간에 만나요~