코테 스터디

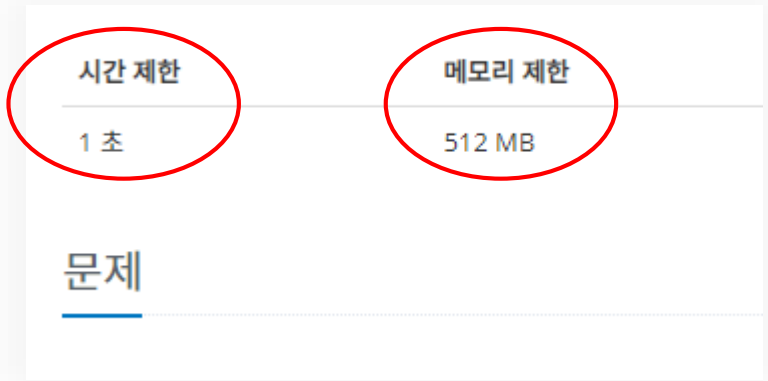
2주차 - 2024 / 04 / 30

알고리즘 성능 측정하기

시간, 공간복잡도 알아보기

> 효율적인 알고리즘

- PS 문제에는 시간 / 메모리 제한이 존재함



> 효율적인 알고리즘

- 연산 횟수가 많아질수록 걸리는 시간이 늘어남
- 저장하는 변수가 많을수록 사용하는 메모리가 늘어남

> 효율적인 알고리즘

- 연산 횟수가 많으면 **시간 제한을 초과**하게 됨
- 많은 변수를 저장하게 되면 **메모리 제한을 초과**하게 됨

> 효율적인 알고리즘

- **PS에서 시간 / 메모리 제한을 두는 이유?**

> 효율적인 알고리즘

- **PS에서 시간 / 메모리 제한을 두는 이유?**

→ 연산 횟수와 메모리 사용량이 적은 효율적인 코드를 요구하기 위해

> 시간 복잡도?

- 문제의 입력 크기와 해결하는 데 걸리는 시간의 관계를 나타내는 함수

> 시간 복잡도?

- 문제의 입력 크기와 해결하는 데 걸리는 시간의 관계를 나타내는 함수

  (보통 PS에서는 다른 요소들을 배제하고 계산)

> 시간 복잡도?

- 연산 횟수를 계산하면 프로그램 수행 시간을 유추할 수 있음
- 보통 최악의 경우를 고려함

> 공간 복잡도?

- 문제의 입력 크기와 해결하는 데 필요한 메모리의 관계를 나타내는 함수

> 공간 복잡도?

- 보통 코딩 테스트는 메모리 제한이 512~1024MB정도로 널널한 편

  (512MB = int형 변수 2^27개)

- 메모리 제한이 크게 작지 않는 이상, 보통은 신경 쓸 필요가 없음

> 시간 복잡도를 표기하는 방법

- Big-O : 최악의 경우
- Big-Ω : 최선의 경우
- Big-Θ : 평균의 경우

> 시간 복잡도를 표기하는 방법

- **Big-O : 최악의 경우**
- Big-Ω : 최선의 경우
- Big-Θ : 평균의 경우

> 시간 복잡도를 표기하는 방법

- **Big-O : 최악의 경우**

→ PS에서는 대부분 Big-O Notation을 사용함

> Big-O Notation

- $O\big(g(N)\big) = \{\, f(N) : N \geq N_0$인 N에 대해서 $0 \leq f(N) \leq c \times g(N)$을 만족하는 양의 실수 c와 N_0이 존재 $\}$

  참고

> Big-O Notation

- 문제의 입력 크기와 해결하는 데 걸리는 시간의 관계를 나타내는 함수

> Big-O Notation

- 문제의 입력 크기와 해결하는 데 걸리는 시간의 관계를 나타내는 함수

N

T(N)

> Big-O Notation

- **O(T(N))와 같이 표기함**
- **Ex)** $O(N^2),\ O(N\log N), O(N!)$

> Big-O Notation

- **T(N)을 표기할 때, 다항 함수는 최고차항의 차수만 고려함**

$$\mathrm{T}(N) = 3N^3 + 4N^2 + 2$$

$$= O(N^3)$$

> Big-O Notation

- **T(N)을 표기할 때, 지수 함수는 밑의 최댓값만 고려함**

$$\mathrm{T}(N) = 3^N + 2^N + 100$$
$$= O(3^N)$$

> Big-O Notation

- 이런 식으로 T(N)을 표기하는 이유?
  - N이 충분히 큰 경우(10억, 100억, ...) 에는

    최고차항 / 밑이 큰 N이 연산 횟수에 가장 크게 영향을 끼침

> Big-O Notation

$$T(N) = N^2 + 5N + 5$$

> Big-O Notation

$$T(N) = N^2 + 5N + 5$$

$$= O(N^2)$$

> Big-O Notation

$$T(N) = N^2 + 5N + 5$$

$$= O(N^2)$$

$N = 10{,}000$일 때,

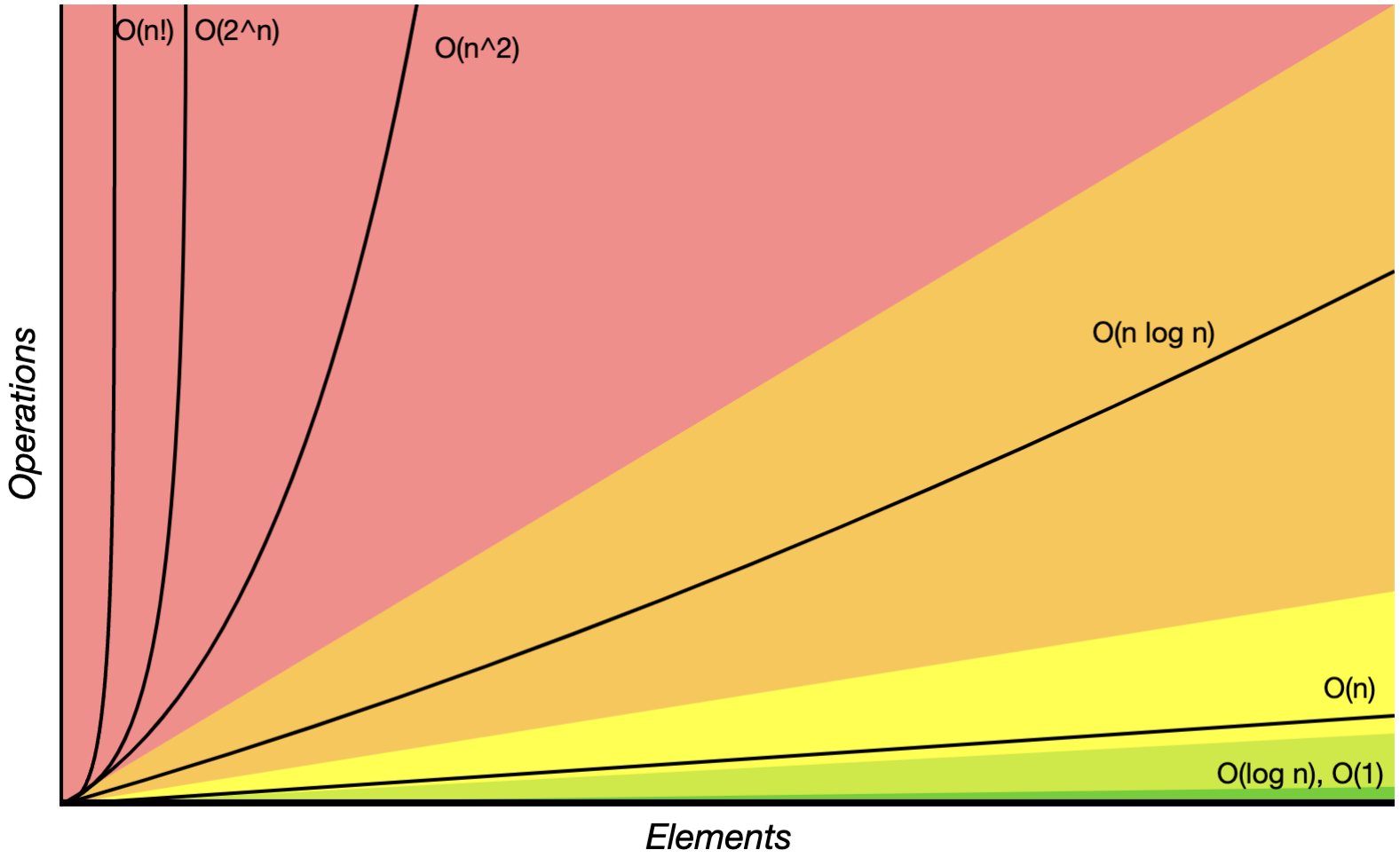
$$\begin{cases} T(N) = 10^8 + 5 \times 10^4 + 5 = 100{,}050{,}005 \approx 100{,}000{,}000 \\ O(N^2) \approx 100{,}000{,}000 \end{cases}$$

> Big-O Notation

- **대표적인 시간 복잡도의 연산 횟수 비교**

$$O(1) < O(\log N) < O(N) < O(N \log N) < O(N^2) < O(N^3) < O(2^N) < O(N!)$$

> Big-O Notation



질문 시간

편하게 질문해주세요

> 소스코드의 시간 복잡도 구하기

- BOJ 24262번 알고리즘 수업 - 알고리즘의 수행 시간 1
  - https://www.acmicpc.net/problem/24262

> 소스코드의 시간 복잡도 구하기

문제

오늘도 서준이는 알고리즘의 수행시간 수업 조교를 하고 있다. 아빠가 수업한 내용을 학생들이 잘 이해했는지 문제를 통해서 확인해보자.

입력의 크기 n이 주어지면 MenOfPassion 알고리즘 수행 시간을 예제 출력과 같은 방식으로 출력해보자.

MenOfPassion 알고리즘은 다음과 같다.

```
MenOfPassion(A[], n) {
    i = ⌊n / 2⌋;
    return A[i]; # 코드1
}
```

입력

첫째 줄에 입력의 크기 n($1 \le n \le 500,000$)이 주어진다.

출력

첫째 줄에 코드1 의 수행 횟수를 출력한다.

둘째 줄에 코드1의 수행 횟수를 다항식으로 나타내었을 때, 최고차항의 차수를 출력한다. 단, 다항식으로 나타낼 수 없거나 최고차항의 차수가 3보다 크면 4를 출력한다.

> 소스코드의 시간 복잡도 구하기

```
1 int ManOfPassion(int a[], int n)
2 {
3   int i = n / 2;
4   return a[i];
5 }
```

- 입력의 크기 N이 주어졌을 때,

  배열 A에서 n/2번째 인덱스를 반환하는 함수

> 소스코드의 시간 복잡도 구하기

```
1 int ManOfPassion(int a[], int n)
2 {
3   int i = n / 2;
4   return a[i];
5 }
```

- N을 2로 나누는 연산 1번
- $T(N) = 1 \rightarrow O(1)$

> 소스코드의 시간 복잡도 구하기

- BOJ 24264번 알고리즘 수업 - 알고리즘의 수행 시간 3
  - https://www.acmicpc.net/problem/24264

> 소스코드의 시간 복잡도 구하기

문제

오늘도 서준이는 알고리즘의 수행시간 수업 조교를 하고 있다. 아빠가 수업한 내용을 학생들이 잘 이해했는지 문제를 통해서 확인해보자.

입력의 크기 n이 주어지면 MenOfPassion 알고리즘 수행 시간을 예제 출력과 같은 방식으로 출력해보자.

MenOfPassion 알고리즘은 다음과 같다.

```
MenOfPassion(A[], n) {
    sum <- 0;
    for i <- 1 to n
        for j <- 1 to n
            sum <- sum + A[i] × A[j]; # 코드1
    return sum;
}
```

입력

첫째 줄에 입력의 크기 n($1 \le n \le 500,000$)이 주어진다.

출력

첫째 줄에 코드1 의 수행 횟수를 출력한다.

둘째 줄에 코드1의 수행 횟수를 다항식으로 나타내었을 때, 최고차항의 차수를 출력한다. 단, 다항식으로 나타낼 수 없거나 최고차항의 차수가 3보다 크면 4를 출력한다.

> 소스코드의 시간 복잡도 구하기

```
 1 int ManOfPassion(int a[], int n)
 2 {
 3   int sum = 0;
 4
 5   for (int i = 1; i <= n; i++)
 6   {
 7     for (int j = 1; j <= n; j++)
 8     {
 9       sum = sum + a[i] * a[j];
10     }
11   }
12
13   return sum;
14 }
```

- 입력의 크기 N이 주어졌을 때,
  배열 A의 가능한 모든 두 쌍의 곱의 합을
  반환하는 함수

> 소스코드의 시간 복잡도 구하기

```
 1 int ManOfPassion(int a[], int n)
 2 {
 3   int sum = 0;
 4
 5   for (int i = 1; i <= n; i++)
 6   {
 7     for (int j = 1; j <= n; j++)
 8     {
 9       sum = sum + a[i] * a[j];
10     }
11   }
12
13   return sum;
14 }
```

- 함수에서 i = 1일 때, j = 1...N이므로 sum에 더해주는 연산이 N번
- i = 1...N이면 총 연산은 N * N번
- $T(N) = N^2 \rightarrow O(N^2)$

> 시간 복잡도 이용하기

- 보통 1초에 1억 개 정도의 연산을 할 수 있음

> 시간 복잡도 이용하기

- 보통 1초에 1억 개 정도의 연산을 할 수 있음
- 이를 통해 시간 제한과 입력 크기를 보고 요구하는 시간 복잡도를 예측 가능

> 시간 복잡도 이용하기

- Ex) N = 300 시간 제한 : 1초

> 시간 복잡도 이용하기

- **Ex) N = 300　시간 제한 : 1초**

$$O(N^4) = 81 \times 10^8$$

$\therefore O(N^4)$보다 빠른 시간 복잡도를 요구함

> 시간 복잡도 이용하기

- Ex) N = 200,000　시간 제한 : 5초

> 시간 복잡도 이용하기

- **Ex) N = 200,000　시간 제한 : 5초**

$$O(N^2) = 4 \times 10^{10}$$

$\therefore O(N^2)$보다 빠른 시간 복잡도를 요구함

질문 시간

편하게 질문해주세요

무식하게 문제 풀어보기

브루트 포스 알고리즘 알아보기

> 브루트 포스 알고리즘

- 단순하게 모든 경우의 수를 다 구해보는 알고리즘

> 브루트 포스 알고리즘

- **Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수**

> 브루트 포스 알고리즘

- Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수

첫번째 눈 = 1 (1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6)

첫번째 눈 = 2 (2, 1) (2, 2) (2, 3) (2, 4) (2, 5) (2, 6)

첫번째 눈 = 3 (3, 1) (3, 2) (3, 3) (3, 4) (3, 5) (3, 6)

첫번째 눈 = 4 (4, 1) (4, 2) (4, 3) (4, 4) (4, 5) (4, 6)

첫번째 눈 = 5 (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (5, 6)

첫번째 눈 = 6 (6, 1) (6, 2) (6, 3) (6, 4) (6, 5) (6, 6)

> 브루트 포스 알고리즘

- Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수

첫번째 눈 = 1 (1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6)

첫번째 눈 = 2 (2, 1) (2, 2) (2, 3) (2, 4) (2, 5) (2, 6)

첫번째 눈 = 3 (3, 1) (3, 2) (3, 3) (3, 4) (3, 5) (3, 6)

첫번째 눈 = 4 (4, 1) (4, 2) (4, 3) (4, 4) (4, 5) (4, 6)

첫번째 눈 = 5 (5, 1) (5, 2) (5, 3) (5, 4) (5, 5) (5, 6)

첫번째 눈 = 6 (6, 1) (6, 2) (6, 3) (6, 4) (6, 5) (6, 6)

- Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수

첫번째 눈 = 1 (1, 1) (1, 2) (1, 3) (1, 4) (1, 5) (1, 6)

첫번째 눈 = 2 (2, 1) (2, 2) (2, 3) (2, 4) (2, 5) **(2, 6)**

첫번째 눈 = 3 (3, 1) (3, 2) (3, 3) (3, 4) **(3, 5)** (3, 6)

첫번째 눈 = 4 (4, 1) (4, 2) (4, 3) **(4, 4)** (4, 5) (4, 6)

첫번째 눈 = 5 (5, 1) (5, 2) **(5, 3)** (5, 4) (5, 5) (5, 6)

첫번째 눈 = 6 (6, 1) **(6, 2)** (6, 3) (6, 4) (6, 5) (6, 6)

→ 5가지

> 브루트 포스 알고리즘

- **Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수**

- 입력의 크기 N = 6 (주사위의 눈의 개수)

> 브루트 포스 알고리즘

- **Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수**

- 입력의 크기 N = 6 (주사위의 눈의 개수)
- 첫번째 눈이 1...N일 때, 각각 두번째 눈의 값은 1...N이므로 연산이 N * N번 발생

> 브루트 포스 알고리즘

- **Ex) 주사위를 두 번 던져서 눈의 합이 8이 되는 경우의 수**

- 입력의 크기 N = 6 (주사위의 눈의 개수)
- 첫번째 눈이 1...N일 때, 각각 두번째 눈의 값은 1...N이므로 연산이 N * N번 발생
- $T(N) = N^2 \rightarrow O(N^2)$

> 브루트 포스 알고리즘

- 시간 복잡도를 계산해봤을 때 브루트 포스 알고리즘이 시간 안에 수행된다면, 단순하게 모든 경우를 구해보는 것도 방법이 될 수 있음

> 브루트 포스로 문제 풀어보기

- BOJ 2309번 일곱 난쟁이
  - https://www.acmicpc.net/problem/2309

> 브루트 포스로 문제 풀어보기

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	128 MB	145598	60673	42477	41.774%

문제

왕비를 피해 일곱 난쟁이들과 함께 평화롭게 생활하고 있던 백설공주에게 위기가 찾아왔다. 일과를 마치고 돌아온 난쟁이가 일곱 명이 아닌 아홉 명이었던 것이다.

아홉 명의 난쟁이는 모두 자신이 "백설 공주와 일곱 난쟁이"의 주인공이라고 주장했다. 뛰어난 수학적 직관력을 가지고 있던 백설공주는, 다행스럽게도 일곱 난쟁이의 키의 합이 100이 됨을 기억해 냈다.

아홉 난쟁이의 키가 주어졌을 때, 백설공주를 도와 일곱 난쟁이를 찾는 프로그램을 작성하시오.

입력

아홉 개의 줄에 걸쳐 난쟁이들의 키가 주어진다. 주어지는 키는 100을 넘지 않는 자연수이며, 아홉 난쟁이의 키는 모두 다르며, 가능한 정답이 여러 가지인 경우에는 아무거나 출력한다.

출력

일곱 난쟁이의 키를 오름차순으로 출력한다. 일곱 난쟁이를 찾을 수 없는 경우는 없다.

> 브루트 포스로 문제 풀어보기

- 입력의 크기 N = 9 (난쟁이들의 수)
- 7명의 난쟁이들의 키의 합을 조사하는 모든 경우의 수 = N^7
- $T(N) = N^7 \rightarrow O(N^7)$

> 브루트 포스로 문제 풀어보기

- **N = 9, 시간 제한 : 2초**
- $O(N^7) = 9^7 \approx 5 \times 10^6$

  → 시간 제한 안에 코드가 작동함

> 브루트 포스로 문제 풀어보기

```
for (int a = 0; a < 9; a++) {
    for (int b = a + 1; b < 9; b++) {
        for (int c = b + 1; c < 9; c++) {
            for (int d = c + 1; d < 9; d++) {
                for (int e = d + 1; e < 9; e++) {
                    for (int f = e + 1; f < 9; f++) {
                        for (int g = f + 1; g < 9; g++) {
                            if (h[a] + h[b] + h[c] + h[d] + h[e] + h[f] + h[g] == 100)
{                               PrintAnswer();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

> 약간의 아이디어 적용하기

- 난쟁이 7명의 키의 합

  = 난쟁이 전체의 키의 합 – 난쟁이 2명의 키의 합

> 약간의 아이디어 적용하기

- 입력의 크기 N = 9 (난쟁이들의 수)
- 2명의 난쟁이들의 키의 합을 조사하는 모든 경우의 수 = N^2
- $T(N) = N^2 \rightarrow O(N^2)$

> 약간의 아이디어 적용하기

- **N = 9, 시간 제한 : 2초**
- $O(N^2) = 9^2 = 81$

  → 시간 제한 안에 코드가 작동함

> 약간의 아이디어 적용하기

```
1 for (int a = 0; a < 9; a++) {
2     for (int b = a + 1; b < 9; b++) {
3         if (sum - h[a] + h[b] == 100)
4         {
5             PrintAnswer();
6         }
7     }
8 }
```

> 브루트 포스로 안된다면

- 아이디어, 알고리즘을 이용해 시간 복잡도를 줄여야 함

> 연습 문제

- **백준 24313** 알고리즘 수업 – 점근적 표기 1
  Big-O Notation에 대해 이해할 수 있는 문제
- **백준 24262...24267** 알고리즘 수업 – 알고리즘의 수행 시간 1~6
  소스코드를 보고 시간 복잡도를 구해보는 문제
- **백준 2309** 일곱 난쟁이
  단순하게 찾아도 되고, 나머지를 찾아도 되는 문제

> 연습 문제

- **백준 4673** — **셀프 넘버**
  브루트 포스로 풀리는 수학 문제
- **백준 1051** — **숫자 정사각형**
  2차원 배열에서 브루트 포스를 해보는 문제
- **백준 2231** — **분해 합**
  특정 수의 가장 작은 생성자를 찾는 문제

> 추가로 읽어볼 만한 자료

- Big-O 쉽게 이해하기
- 알고리즘의 성능과 시간 복잡도

수고하셨습니다

질문 받습니다