2022-1학기 스터디 #0

나정휘
https://justiceHui.github.io/

목차

강사 소개

- 소속
  - (21.03-) 숭실대학교 컴퓨터학부
    - (22.03-) 문제해결 소모임 SCCC 회장
    - (21.03-21.08) 기하계산연구실 학부연구생
- 하는 일
  - 알고리즘 5년 공부
  - BOJ 5등7000+문제 / CodeUp 2등1300+문제
  - 삼성전자 사내 알고리즘 교육 강사
  - SSAFY 알고리즘 특강 강사
  - 각종 대회 수상
  - 각종 고등학교/대학교 알고리즘 대회 운영
  - 각종 코딩 테스트 출제
  - 유명한(?) 알고리즘 문제 해결 블로그 운영justiceHui.github.io

스터디 소개

- 일시: 5월 첫째 주 ~ 6월 첫째 주 수요일/토요일 저녁 7시
- 장소: Zoom, 링크는 스터디 시작 전 공지
- 대상: PS(Problem Solving)를 처음 접해보는 사람
  - 문제를 풀어본 경험이 적은 사람
  - 알고리즘 공부를 했지만 기초적인 내용을 잘 응용하지 못하는 사람
- 내용: PS에서 가장 기초적인 개념과 실제 문제에 적용하는 방법

스터디 소개

- 주의사항
  - 스터디 참여도가 낮은 신입 부원은 다음 학기 소모임 등록이 제한될 수 있습니다.
  - 카메라 안 켜도 됩니다.
  - 속도가 너무 빠르거나 질문 있으면 채팅 남겨주세요.

- BOJ 관련 (acmicpc.net)
  - 가입 안 했으면 회원가입하세요
  - icpc.me/g/14636 가입 신청하세요
  - 연습 소스 코드 공개 설정 바꾸세요

질문?

ICPC

- 국제 대학생 프로그래밍 대회 (ICPC)
  - 3인 1팀
  - Regional Contest
    - 한국은 Asia Pacific 소속
    - Seoul, Yokohama, Hanoi, Taipei, Jakarta, …
  - World Final
    - Asia Pacific 지역은 리저널 별 상위 2~3개 대학 진출
- ICPC Seoul Regional
  - 인터넷 예선(보통 10월)
    - 3시간, 학교마다 진출 컷 다름
  - 본선(보통 11월)
    - 5시간, 상위 N팀 수상, 상위 M개 대학 WF 진출

UCPC

- 전국 대학생 프로그래밍 대회 동아리 연합 (https://ucpc.me)
- ICPC 대비 여름 대회 개최
    - 3인 1팀
    - 예선 (7/2)
        - 3시간, 상위 4~50팀 + @ 본선 진출
    - 본선 (7/23)
        - 5시간, 상위 N팀 수상

팀원 모집

- 슬랙 #2022-team-building 에서 팀원 구하면 됩니다.
  - 목표, 본인 실력, 원하는 팀원 실력, 작년 대회 성과, 기타 등등
  - 소개 올려놓고 기다리는 것보다 직접 DM 보내는 게 좋아요.
  - 팀원 다 모았으면 스레드에 모집 마감했다고 메시지 남겨주세요.

  - 예시
    - 목표: 참가에 의의를 둠, 본선 진출, 수상, ...
    - 본인 실력: 알고리즘 처음 해봄, BOJ/codeforces 아이디, 수상 실적, ...

개인 대회

- SCPC (삼성 대학생 프로그래밍 경진대회)
  - 1차 예선(7월, 24시간), 2차 예선(8월, 12시간), 본선(9/3, 4시간)
- Google Code Jam
  - 진행 중
  - 보통 4월에 시작
- Facebook Hacker Cup
  - 아직 시작 안 함
- 현대모비스 알고리즘 경진대회
  - 작년에 열렸는데 올해도 할까?
- 대회 소식 있으면 슬랙에 올릴 예정

질문?

solved.ac

- BOJ 문제에 알고리즘 태그와 난이도를 붙이는 커뮤니티 프로젝트
  - 문제를 해결한 유저가 직접 난이도와 사용한 알고리즘을 투표하는 시스템

  - 문제 난이도
    - Unrated / Bronze V, IV, III, II, I / Silver V, IV, III, II, I / Gold V, IV, III, II, I
    - Platinum V, IV, III, II, I / Diamond V, IV, III, II, I / Ruby V, IV, III, II, I

    - 코딩 테스트 합격을 가르는 문제는 대부분 S1~G3
      - 임의의 G3 문제를 30분 안에 해결할 수 있으면 전세계 상위 20% 정도 (Codeforces 1600+)
    - G1~P5 이하의 문제를 모두 해결하면 높은 확률로 ICPC Seoul Regional 진출
      - 임의의 G1 문제를 30분 안에 해결할 수 있으면 전세계 상위 3% 정도 (Codeforces 2100+)
    - P1 이하의 문제를 모두 해결하고 D5 이상 하나 풀면 ICPC Seoul Regional 수상 가능

solved.ac

- 어떤 문제를 풀어야 할까?
  - 클래스 문제집 (solved.ac/class)
    - 고인물들이 직접 엄선한 좋은 문제 (shiftpsh[solvedac 제작자], jh05013[BOJ 9등], koosaga[1등], jhnah917[5등])
    - 새로운 개념, 구현 연습, 풀이가 어려운 문제
      - 어떤 걸 공부해야 할지 모를 때 하나씩 풀어보면서 모르는 거 공부하면 좋음

  - 문제 검색
    - ex. 난이도 S2~G3인 동적 계획법 문제 중 내가 아직 안 푼 문제
      - *s2..g3 #dp -s@$me
    - ex. 난이도가 S5~S1인 정수론 문제 중 jhnah917이 풀고 내가 안 푼 문제
      - *s #number_theory s@jhnah917 –s@$me
    - ex. 난이도가 G5..G3인 문제 중 수학이 들어가지 않은 문제
      - *g5..g3 -#math

solved.ac

- 어떤 문제를 풀어야 할까?
  - 올림피아드 문제
    - 주로 여러 단계의 사고 과정을 거쳐야 하는 문제
    - USACO Bronze / Silver 추천
  - ICPC 기출 문제
    - 올림피아드보다 더 넓은 출제 범위인류가 아는 모든 지식
  - 코딩테스트 기출 문제
    - (삼성) 구현해야 할 것이 많은 문제
      - BOJ 문제집에 정리되어 있음
    - (카카오) 기초적인 개념을 응용하는 문제
      - Programmers에 정리되어 있음
      - 대부분 봄 스터디에서 다루는 범위 안에서 출제

질문?

2022-1학기 스터디 #1

나정휘
https://justiceHui.github.io/

목차

- Problem Solving이란?
- 시간 복잡도
- 정렬 알고리즘
- 이분 탐색

Problem Solving 이란?

Problem Solving 이란?

- 문제를 해결하는 것
  - 문제를 해결하는 방법을 찾는 것
  - 문제를 해결하는 절차를 세우는 것
  - 주어진 문제를 해결하는 알고리즘을 설계하는 것

- 우리가 풀어야 하는 문제들은…
  - 입력과 출력이 정의된 적당한 문제가 주어짐
  - 입력을 넣었을 때 올바른 결과를 출력하는 프로그램 작성
  - 시간 제한과 메모리 제한이 있음
  - (optional) 정해진 시간 동안 여러 문제를 빠르게 풀어야 함

Problem Solving?

- 풀어야 하는 문제들
  - 문제에 나와있는 내용을 그대로 구현 (ex. BOJ 5373 큐빙)
  - 조건을 만족하는 최댓값/최솟값 (ex. 1014 컨닝)
  - 주어진 연산을 효율적으로 처리 (ex. BOJ 2042 구간 합 구하기)
  - 주어진 작업을 하는데 필요한 최소/최대 비용 (BOJ 1011 Fly me to the Alpha Centauri)
  - 등등

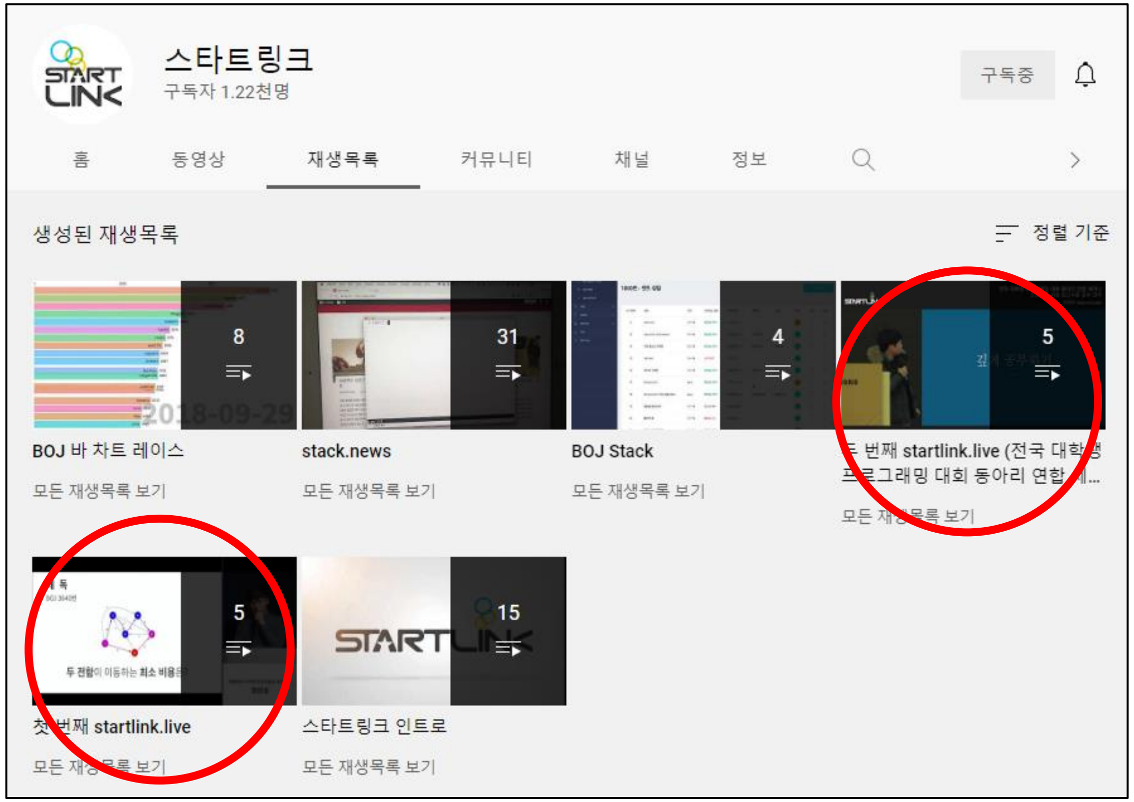
Problem Solving 이란?

- 문제를 해결하는 과정
  - 자연어로 된 문제를 잘 읽는다.
  - 수학적으로 잘 모델링한다.
  - 풀이를 생각한다.
  - 내가 생각한 풀이가 맞는지 검증한다.
  - 내가 생각한 풀이를 코드로 옮긴다.
  - 온라인 저지에 제출한다.
  - 왜 틀렸는지 고민한다.
  - (반복)

  - ex. BOJ 13900 순서쌍의 곱의 합

Problem Solving 이란?

- 의문점
  - 이걸 왜 배워야 하지?
  - 개발에 도움이 되나?



Problem Solving 이란?

- 어떻게 공부해야 할까?
  - 일단 5주 동안 스터디 꾸준히 참여하시면
  - 마지막에 앞으로 어떻게 공부해야 할지 알려 드릴게요

시간 복잡도

알고리즘의 성능을 판단하는 척도

- 정확성: 얼마나 정확한 답을 구할 수 있는가?
- 작업량: 얼마나 적은 연산을 필요로 하는가?
- 메모리 사용량: 얼마나 적은 공간을 사용하는가?
- 단순성: 알고리즘의 작동 과정이 얼마나 단순한가?
- 최적성: 더 이상 개선할 여지가 없을 만큼 최적화가 잘 되었나?

시간 복잡도

- 문제를 해결하는데 걸리는 시간과 입력 크기의 관계를 나타내는 함수
  - 연산이 많아질수록 오래 걸림
  - PS에서는 네트워크 통신, CPU 점유 등은 신경 안 씀
  - 연산을 몇 번 수행하는지 확인하면 수행 시간을 대략적으로 유추할 수 있음
  - 최악의 경우가 중요함

- 문제를 해결하는데 필수적인 기본 연산
- 기본 연산의 수행 횟수

시간 복잡도 – 예시

- 길이가 N인 배열에서 값이 K인 원소가 존재하는지 확인
  - 기본 연산: 배열의 한 원소와 K의 값을 비교
  - 기본 연산의 수행 횟수: 최대 N회
  - 시간 복잡도: $T(N) = N$

시간 복잡도 – 예시

- 길이 N인 배열에서 최댓값 찾기
  - 기본 연산: 배열의 두 원소의 값 비교
  - 기본 연산의 수행 횟수: 최대 N-1회
  - 시간 복잡도 T(N) = N-1

질문?

함수에서 가장 중요한 정보

- $f(x) = \ x^2 + 5x, g(x) = 27x$
  - x = 1: f(x) = 6, g(x) = 27
  - x = 10: f(x) = 150, g(x) = 270
  - x = 100: f(x) = 10500, g(x) = 2700
  - x = 1000: f(x) = 1005000, g(x) = 27000

- x가 적당히 큰 수면 f(x) >g(x)를 만족함

- x > x_0이면 항상 f(x) > g(x)를 만족함

함수에서 가장 중요한 정보

- 다항 함수: 최고차항의 차수가 중요
  - ex) $f(x) = x^2 + 5x, g(x) = 27x$
  - x > 22이면 항상 f(x) > g(x)
- 지수 함수는 밑수의 최댓값이 중요
  - $f(x) = 3^x + x - 10, g(x) = 2^x + x^3 + 5x$
  - x > 4.6이면 항상 f(x) > g(x)
- 최고차항의 차수 or 밑수의 최댓값이 큰 함수가 더 크다.

Big-O Notation

- $f(x) \in O\big(g(x)\big)$
  - 어떤 실수 x_0과 양의 실수 c가 있어서
  - $x > x_0$을 만족하는 모든 x에 대해
  - $f(x) \leq c\ g(x)$를 만족한다.
- 어떤 실수 x_0보다 큰 모든 x
  - x가 한 없이 커지면 항상 부등호가 성립
- 어떤 양의 실수 c
  - 최고차항의 계수 무시
  - 최고차항의 차수와 지수 항의 밑수가 중요함

Big-O Notation

- $f(x) \in O\left(g(x)\right)$
  - $f(x)$가 아무리 빨리 증가해도
  - 최대 $\mathrm{g}(x)$에 비례하는 수준으로 증가한다
  - $f(x)$의 상계를 나타냄

Big-O Notation

- $f(x) = 5x, g(x) = \ x^2$
  - $x_0 = 5, c = 1$이면 $5x \ \leq 1 \times x^2$: $5x \ \in O(x^2)$

- $f(x) = \ 2^x + 10x + 5, g(x) = \ 2^x$
  - $x_0 = 7, c = 2$이면 $2^x + 10x + 5 \ \leq 2 \times 2^x$: $2^x + 10x + 5 \ \in O(2^x)$

질문?

Big-Omega Notation

- Big-O Notation과 반대로 함수의 하계를 나타냄
- $f(x) \in \ \Omega(g(x))$
  - 어떤 실수 x_0과 양의 실수 c가 있어서
  - $x > x_0$을 만족하는 모든 x에 대해
  - $f(x) \geq c\ g(x)$를 만족한다.

Big-Theta Notation

- 상계와 하계의 교집합(Tight Bound)
- $f(x) \in \Theta\left(g(x)\right) \Leftrightarrow f(x) \in O\left(g(x)\right) \land f(x) \in \Omega(g(x))$

극한을 이용한 표현

- $f(x) \in O\big(g(x)\big) \iff \lim_{n\to\infty} \frac{f(x)}{g(x)} = c$로 수렴
- $f(x) \in \Omega\big(g(x)\big) \iff \lim_{n\to\infty} \frac{f(x)}{g(x)} > 0$
- $f(x) \in \Theta\big(g(x)\big) \iff \lim_{n\to\infty} \frac{f(x)}{g(x)} = c$로 수렴 (단, $c > 0$)

질문?

정렬

정렬이란?

- 주어진 데이터를 일정한 순서대로 나열하는 것
  - 오름차순 정렬: a < b이면 a가 b보다 앞에 오도록 나열
  - 내림차순 정렬: a > b이면 a가 b보다 앞에 오도록 나열
  - 위상 정렬: a → b 간선이 있으면 a가 b보다 앞에 오도록 나열

버블 정렬

- 서로 인접한 두 원소를 비교해서 정렬하는 알고리즘
  - 5 3 4 2 1 3 2 1 4 5
  - 3 5 4 2 1 2 3 1 4 5
  - 3 4 5 2 1 2 1 3 4 5
  - 3 4 2 5 1 2 1 3 4 5
  - 3 4 2 1 5 2 1 3 4 5

  - 3 4 2 1 5 2 1 3 4 5
  - 3 4 2 1 5 1 2 3 4 5
  - 3 2 4 1 5 1 2 3 4 5
  - 3 2 1 4 5 1 2 3 4 5
  - 3 2 1 4 5 1 2 3 4 5

버블 정렬

- 배열을 한 번 순회할 때
  - 비교 N-1번
  - 교환 N-1번
  - i번째 순회에서 i번째로 큰 값이 N-i+1번째로 이동
    - N-1번만 순회하면 정렬 끝

- 시간 복잡도
  - 기본 연산: 비교, 교환
  - 각각 $(N-1)^2$번 수행
  - 시간 복잡도: $O(N^2)$

질문?

선택 정렬

- 가장 작은 원소를 맨 앞에 있는 원소와 교환하면서 정렬하는 알고리즘
  - 5 3 4 1 2
  - 1 3 4 5 2
  - 1 2 4 5 3
  - 1 2 3 5 4
  - 1 2 3 4 5

선택 정렬

- i번째 단계
  - 비교 N-i-1번
  - 교환 1번
  - i번째로 작은 값이 i번째로 이동

- 시간 복잡도
  - 기본 연산: 비교, 교환
  - 비교 N(N-1)/2번, 교환 N-1번
  - 시간 복잡도: $O(N^2)$

다른 정렬 알고리즘

- 삽입 정렬 – $O(N^2)$
- 합병 정렬 – $O(N \log N)$
- 퀵 정렬 – 평균 $O(N \log N)$, 최악 $O(N^2)$
- 힙 정렬 – $O(N \log N)$

비교 기반 정렬

- 두 원소를 비교/교환해서 정렬하는 알고리즘
  - ex. 오름차순 정렬
    - i < j and A[j] < A[i] 인 (i, j)가 없어야 함
    - i < j and A[j] < A[i]이면 A[i]와 A[j] 교환

  - bool Compare(T a, T b)
    - a가 b보다 반드시 앞에 나와야 하면 true, 아니면 false
    - i < j and Compare(A[j], A[i]) = true 인 (i, j)가 없어야 함
    - i < j and Compare(A[i], A[j]) = false 이면 A[i]와 A[j] 교환

    - 오름차순 정렬: Compare(a, b) = a < b

질문?

비교 함수

- bool Compare(T a, T b)
  - a가 b보다 반드시 앞에 나와야 하면 true
  - 그렇지 않으면 false
  - Strict Weak Ordering을 만족해야 함
  - i < j && Compare(A[j], A[i]) == true인 i, j가 존재하지 않으면 됨
  - i < j && Compare(A[i], A[j]) == false면 A[i]와 A[j]를 교환

- 오름차순: Compare(a, b): a < b 　　　 a ≤ b 아님
- 내림차순: Compare(a, b): a > b 　　　 a ≥ b 아님

Strict Weak Ordering

- Strict Weak Ordering
  - 비반사성(irreflexivity)
    - 모든 x에 대해 R(x, x)는 거짓
  - 비대칭성(asymmetry)
    - 모든 x,y에 대해 R(x, y)이 참이면 R(y, x)는 거짓
  - 추이성(transitivity)
    - 모든 x,y,z에 대해 R(x, y), R(y, z)가 참이면 R(x, z)는 참
  - 비비교성의 추이성(transitivity of incomparability)
    - 모든 x,y,z에 대해 R(x, y), R(y, x), R(y, z), R(z, y)가 거짓이면 R(x, z), R(z, x)는 거짓

Strict Weak Ordering

- 비교성(comparability)
  - 비교를 할 수 있다
  - (정렬에서) 두 원소의 순서를 정할 수 있다
- 비비교성(incomparability)
  - 비교를 할 수 없다
  - (정렬에서) 두 원소의 순서를 정할 수 없다 = 동등하다
- ex) 오름차순 정렬
  - 값이 같은 두 원소는 순서를 정할 수 없음
  - 값이 다른 두 원소는 순서를 정할 수 있음

Strict Weak Ordering

- 비반사성(irreflexivity)
  - 모든 x에 대해 R(x, x)는 거짓
  - 값이 같은 두 원소는 비교가 불가능하다
  - Compare(x, x)는 두 x 중 반드시 먼저 와야 하는 것을 결정할 수 없음

- 오름차순의 비교 함수로 a ≤ b를 사용할 수 없는 이유

Strict Weak Ordering

- 비대칭성(asymmetry)
  - 모든 x,y에 대해 R(x, y)이 참이면 R(y, x)는 거짓
  - 두 개가 모두 참이면 두 원소의 순서를 정할 수 없음
  - R(x, y), R(y, x) 모두 거짓이 되어야 함

- 사이클이 있는 그래프에서 위상 정렬이 불가능한 이유

Strict Weak Ordering

- 추이성(transitivity)
  - 모든 x,y,z에 대해 R(x, y), R(y, z)가 참이면 R(x, z)는 참
  - 삼단 논법
- 비비교성의 추이성(transitivity of incomparability)
  - 모든 x,y,z에 대해 R(x, y), R(y, x), R(y, z), R(z, y)가 거짓이면 R(x, z), R(z, x)는 거짓
  - x y가 동등하고(비교가 불가능하고), y z가 동등하면 x z도 동등해야 함

질문?

std::sort

- sort(first, last): 시작 주소, 끝 주소
  - 기본적으로 오름차순 정렬
  - 구조체/클래스의 경우 < 연산이 정의되어 있어야 함
- sort(first, last, comp): 시작 주소, 끝 주소, 비교 함수
- O(N log N)

```
#include <stdio.h>
#include <algorithm>

bool Compare(int a, int b){
    return a > b;
}

int main(){
    int arr[5] = {5, 3, 1, 4, 2};
    std::sort(arr, arr+5); // 정렬할 범위의 첫 주소, 마지막 주소의 한 칸 뒤
    for(int i=0; i<5; i++) printf("%d ", arr[i]);
    printf("\n");

    std::sort(arr, arr+5, Compare);
    for(int i=0; i<5; i++) printf("%d ", arr[i]);
}
```

std::stable_sort

- 동등한(비교 불가능한) 원소들의 순서는 어떻게 결정할까?
  - stable sort: 기존 순서 유지
  - unstable sort: 마음대로
- std::sort는 unstable sort
- stable sort를 사용하고 싶다면 std::stable_sort 사용

```
#include <stdio.h>
#include <algorithm>

struct Data{
    int x, y;
};

bool Compare(Data a, Data b){
    return a.x < b.x;
}

int main(){
    Data arr[3] = { {1, 3}, {1, 2}, {0, 1} };
    std::stable_sort(arr, arr+3, Compare);
    for(int i=0; i<3; i++)
        printf("%d %d\n", arr[i].x, arr[i].y);
}
```

질문?

이진 탐색

업/다운 게임

- 철수는 1 이상 N 이하인 자연수 X를 하나 선택한다.
- 영희는 철수가 생각한 X를 맞춰야 한다.
- 영희가 어떤 수 Y를 말하면, 철수는 아래와 같은 대답을 한다.
  - X > Y라면: Up
  - X < Y라면: Down
  - X = Y라면: Accept

- 영희의 질문 횟수를 최소화 시키자.

영희의 전략

- 정답은 [1, N] 사이에 있다.
  - 중간 지점 M_1 = floor((1+N)/2)을 선택한다.
    - 정답이 M_1보다 작다면 구간을 [1, M_1-1]로 조절한다.
    - 정답이 M_1보다 크다면 구간을 [M_1+1, N]으로 조절한다.
    - [S_2, E_2]라고 하자
- 정답은 [S_2, E_2] 사이에 있다.
  - 중간 지점 M_2 = floor((S_2+E_2)/2)를 선택한다.
    - 정답이 M_2보다 작다면 구간을 [S_2, M_2-1]로 조절한다.
    - 정답이 M_2보다 크다면 구간을 [M_2+1, E_2]로 조절한다.
- 반복한다.

영희의 전략

- 정답이 존재할 수 있는 구간이 매번 절반으로 감소한다.
  - 최악의 경우에도 $\log_2 N + 1$번의 질문으로 답을 구할 수 있다.

- 이게 최적 전략일까?
  - YES
  - 상대자 논증을 이용해 증명 가능

이분 탐색

- 정렬된 배열 A가 주어진다. K가 A의 몇 번째 원소인지 구해라.
  - A = {1, 2, 3, 4, 5, 6, 7}, K = 3
  - A = {1, 2, 3, 4, 5, 6, 7}, K = 3
  - A = {1, 2, 3, 4, 5, 6, 7}, K = 3
- 배열의 길이를 N이라고 하면 $O(\log N)$에 해결할 수 있다.
  - 기본 연산: 배열의 원소와 어떤 수를 비교하는 것
  - 기본 연산의 수행 횟수: 최악의 경우 $\log_2 N + 1$번

질문?

과제

- 필수
  - 2751 수 정렬하기 2
  - 11650 좌표 정렬하기
  - 1950 수 찾기
  - 2417 정수 제곱근
- 심화
  - 1654 랜선 자르기
  - 2470 두 용액