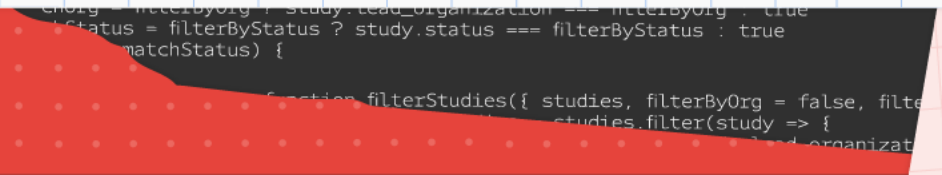
Google Developer Student Clubs
Soongsil University

# 인덱스를 모르고 쿼리를 짠다고?

인덱스의 장점과 단점,
그리고 대표적으로 사용하는 자료구조에 대해 알아보자

우수연
GDSC Soongsil Lead
dntndus9611@gmail.com



# 나는 누구?

Cloud

Infra

Backend

Design

AI/ML

Engineer

Devloper

Lead

Manager

Data

Google Developer Student Clubs
Soongsil University

# 주제를 정한 이유

| | | | | |
|---|---|---|---|---|
| 3 | 2021.11.13 | "우수연" | 헤르미온느식 시간관리법 | 우수연 |

# Agenda

- 인덱스란?
- 인덱스를 쓰는 이유
- 그럼 인덱스를 많이 쓰면 좋은 건가?
- 대표적인 자료구조 (해시테이블, B+트리)
- 한번 직접 테스트해보자

# Index

```
List<String> names = Lists.newArrayList();

names.add(“우수연”);
names.add(“고광서”);
names.add(“이하늘”);

String firstUser = names.get(0);
```

# 인덱스(Index) == 색인

# Index

## 목차



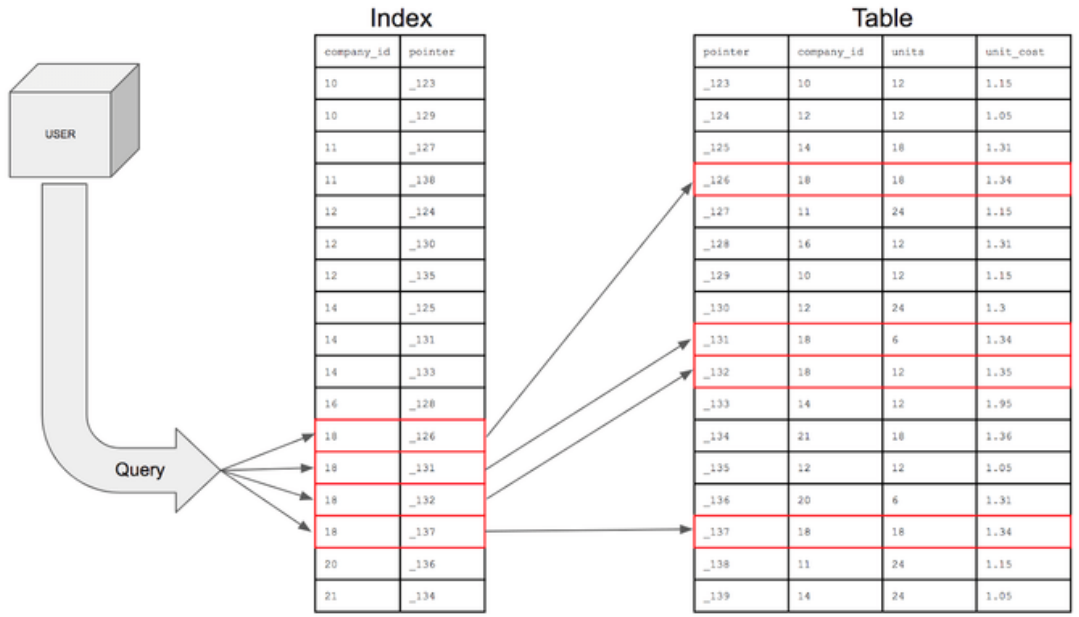
## 색인





참고 : 빅데이터를 지탱하는 기술 (니시다 케이스케 저)

# DB Index






출처 : https://chartio.com/learn/databases/how-does-indexing-work/

# DB Index 장점

검색 속도 및 성능 향상

# DB Index 단점

인덱스 관리

저장공간 필요

성능 저하

# DB Index 단점 – DML 성능저하

## INSERT

- Index 테이블 추가
- 다른 key 값들에 대한 split 작업 필요

## DELETE

- Index 테이블 삭제
- 저장공간 낭비

## UPDATE

- Index 테이블 삭제 후 추가

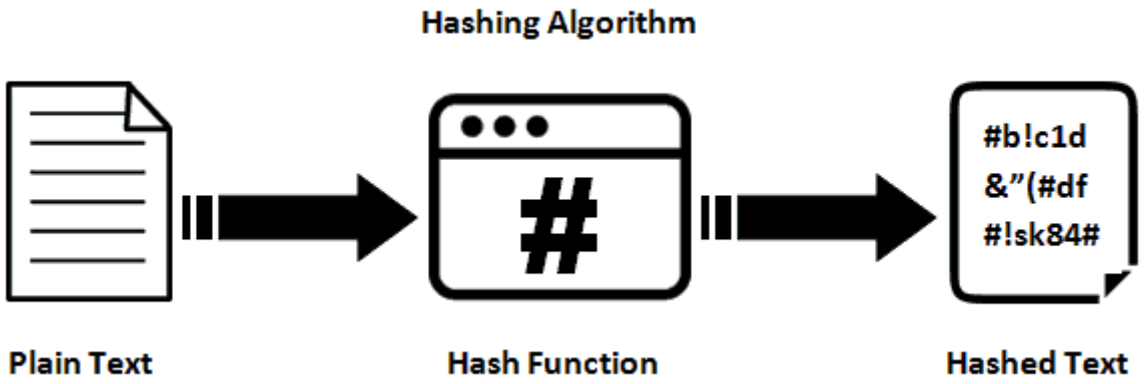
# 대표 자료구조

Hash Table

B+Tree

# Hash Table

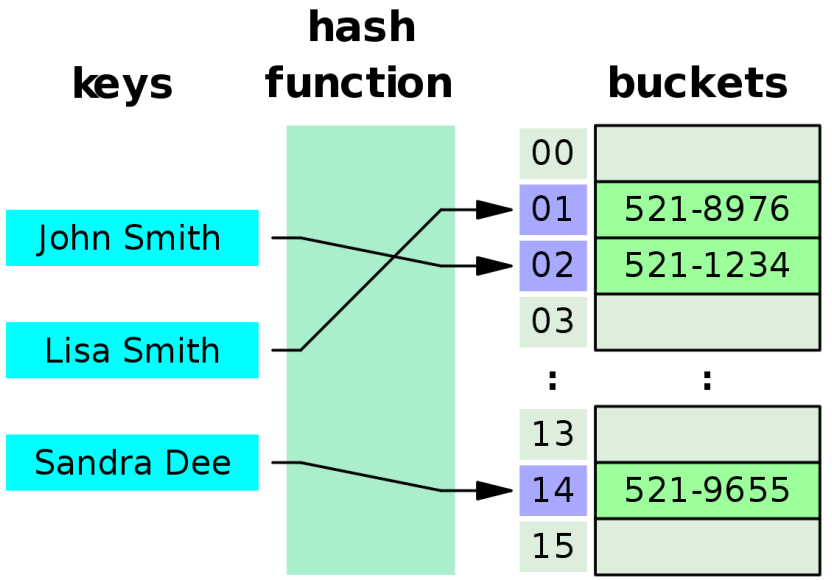
## 해싱(Hashing)이란?






출처 : https://networkencyclopedia.com/hashing-algorithm/

# Hash Table

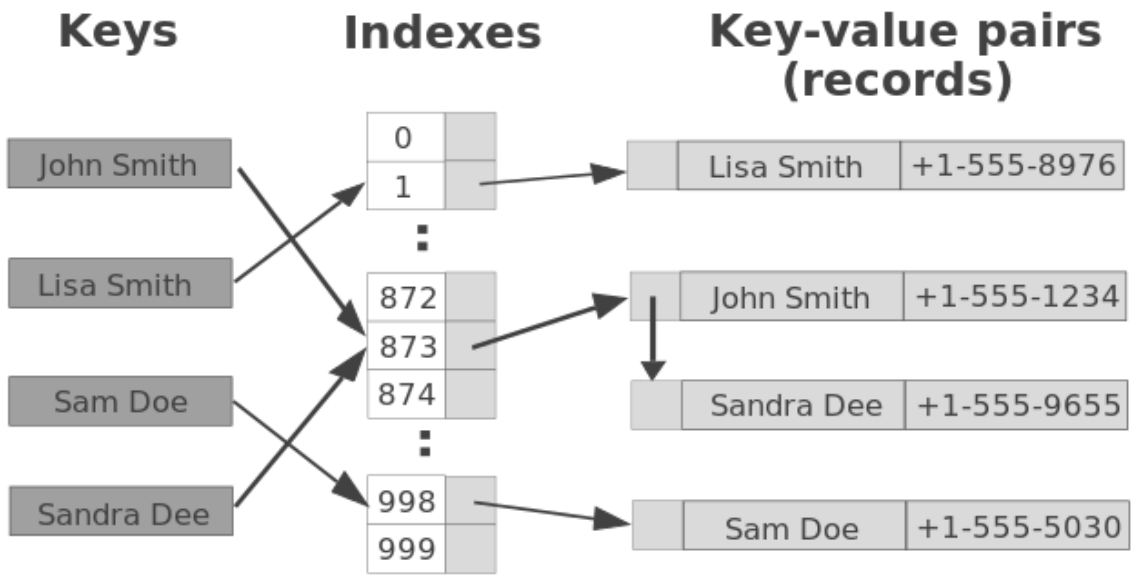



(key, value) = (컬럼의 값, 데이터의 위치)



출처 : https://en.wikipedia.org/wiki/Hash_table

# Hash Table 문제점 1





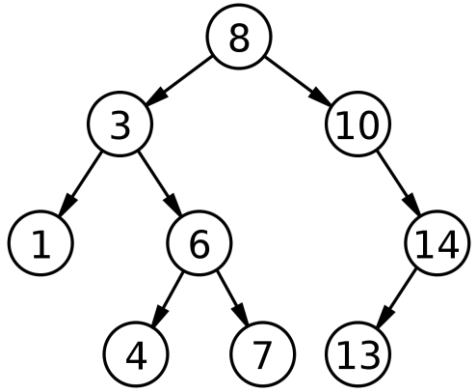

출처 : https://yourbasic.org/algorithms/hash-tables-explained/

# Hash Table 문제점 2

SELECT * FROM user WHERE name = '우수연'; O

SELECT * FROM user WHERE age > 25; X

# Binary Search Tree



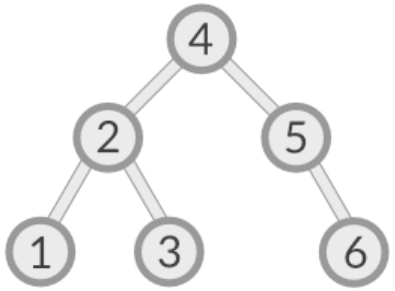

이진탐색

- 탐색 시간복잡도 O(log n) → 빠름

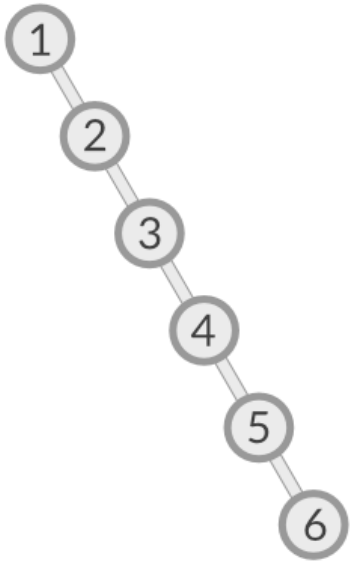
연결리스트

- 입력, 삭제 시간복잡도 O(1) → 빠름
- 탐색 시간복잡도 O(n) → 느림

# Balanced vs non-Balanced




O(log n)

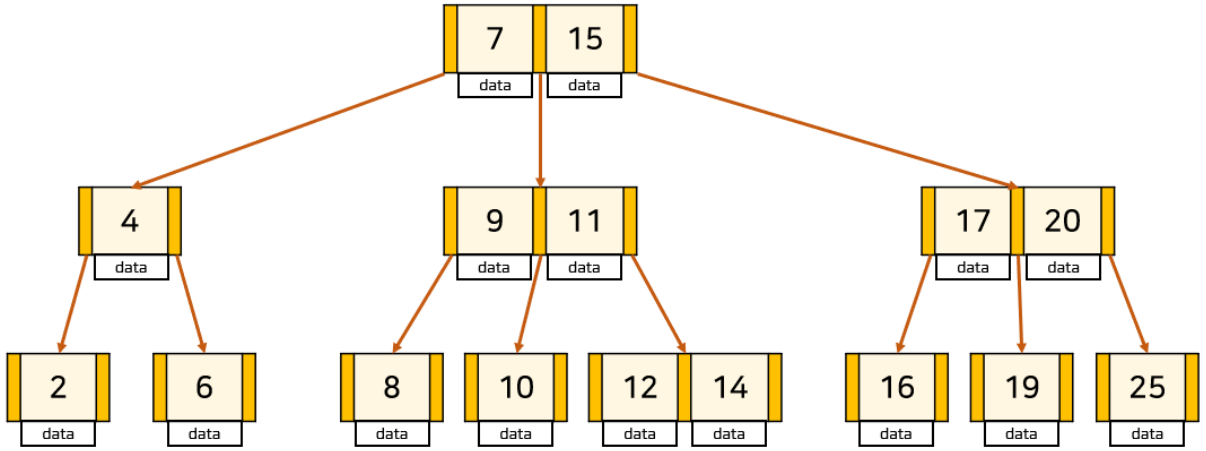



O(n)



출처 : https://stackoverflow.com/questions/59206128/balanced-vs-unbalanced-binary-tree-clarification-needed

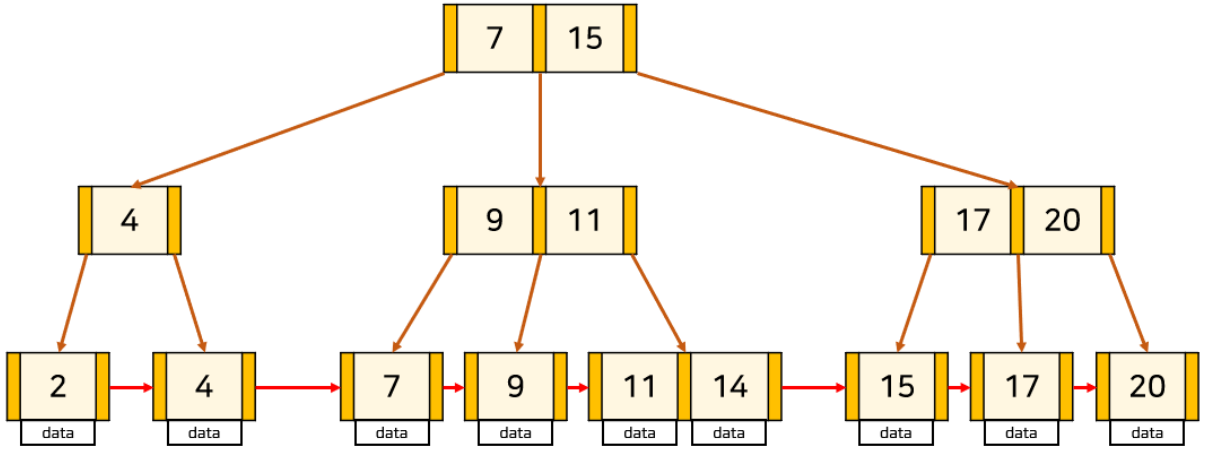
# B-Tree






출처 : https://rebro.kr/169

# B+Tree






출처 : https://rebro.kr/169

# Hash Table vs B+Tree

## Hash Table

O(1)

‘=’ 연산밖에 불가능

## B+Tree

O(log n)

# Example

member

| id (PK) | name | part | grade | age |
|---|---|---|---|---|
| 1 | 우수연 | core | 100 | 27 |
| 2 | 공소나 | web | 96 | 52 |
| 3 | 김민주 | web | 38 | 55 |
| 4 | 김태현 | web | 74 | 2 |
| 5 | 손익준 | ai | 81 | 39 |
| 6 | 신종원 | ai | 67 | 22 |
| 7 | 신홍석 | ai | 55 | 26 |
| 8 | 나상우 | server | 29 | 9 |
| 9 | 오진호 | server | 81 | 9 |
| 10 | 이용택 | server | 27 | 80 |

# Example

SELECT * FROM member WHERE name = '나상우';

member

| id (PK) | name | part | grade | age |
|---|---|---|---|---|
| 1 | 우수연 | core | 100 | 27 |
| 2 | 공소나 | web | 96 | 52 |
| 3 | 김민주 | web | 38 | 55 |
| 4 | 김태현 | web | 74 | 2 |
| 5 | 손익준 | ai | 81 | 39 |
| 6 | 신종원 | ai | 67 | 22 |
| 7 | 신홍석 | ai | 55 | 26 |
| 8 | 나상우 | server | 29 | 9 |
| 9 | 오진호 | server | 81 | 9 |
| 10 | 이용택 | server | 27 | 80 |

Full Scan

# Full Scan... OTL

SELECT * FROM member WHERE name = '나상우';

member

| id (PK) | name | part | grade | age |
|---|---|---|---|---|
| 1 | 우수연 | core | 100 | 27 |
| 2 | 공소나 | web | 96 | 52 |
| 3 | 김민주 | web | 38 | 55 |
| 4 | 김태현 | web | 74 | 2 |
| 5 | 손익준 | ai | 81 | 39 |
| 6 | 신종원 | ai | 67 | 22 |
| 7 | 신홍석 | ai | 55 | 26 |
| 8 | 나상우 | server | 29 | 9 |
| 9 | 오진호 | server | 81 | 9 |
| 10 | 이용택 | server | 27 | 80 |
| ... | ... | ... | ... | ... |
| 10000000 | 너구리 | server | 50 | 20 |

Full Scan

# Index 등장!

```
CREATE INDEX idx_name ON member(name);

SELECT * FROM member WHERE name = '나상우';
```

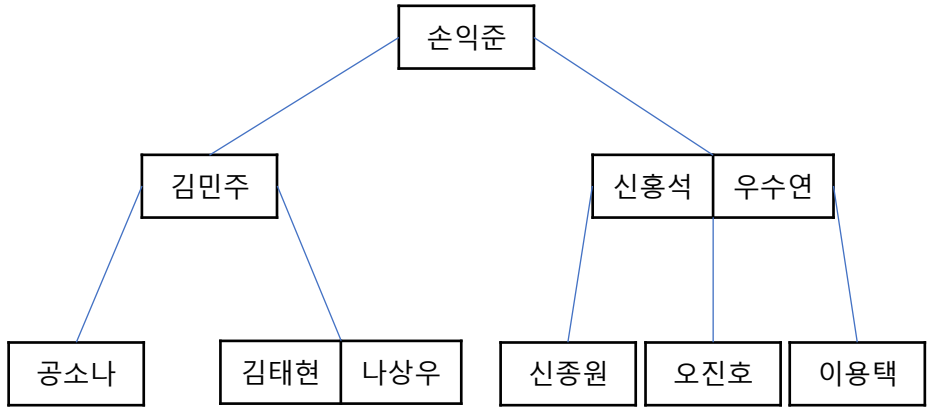
idx_name

member

| mid (PK) | name | part | grade | age |
|---|---|---|---|---|
| 1 | 우수연 | core | 100 | 27 |
| 2 | 공소나 | web | 96 | 52 |
| 3 | 김민주 | web | 38 | 55 |
| 4 | 김태현 | web | 74 | 2 |
| 5 | 손익준 | ai | 81 | 39 |
| 6 | 신종원 | ai | 67 | 22 |
| 7 | 신홍석 | ai | 55 | 26 |
| 8 | 나상우 | server | 29 | 9 |
| 9 | 오진호 | server | 81 | 9 |
| 10 | 이용택 | server | 27 | 80 |
| ... | ... | ... | ... | ... |
| 10000000 | 너구리 | server | 50 | 20 |

# Index 등장!

B-Tree (3차)

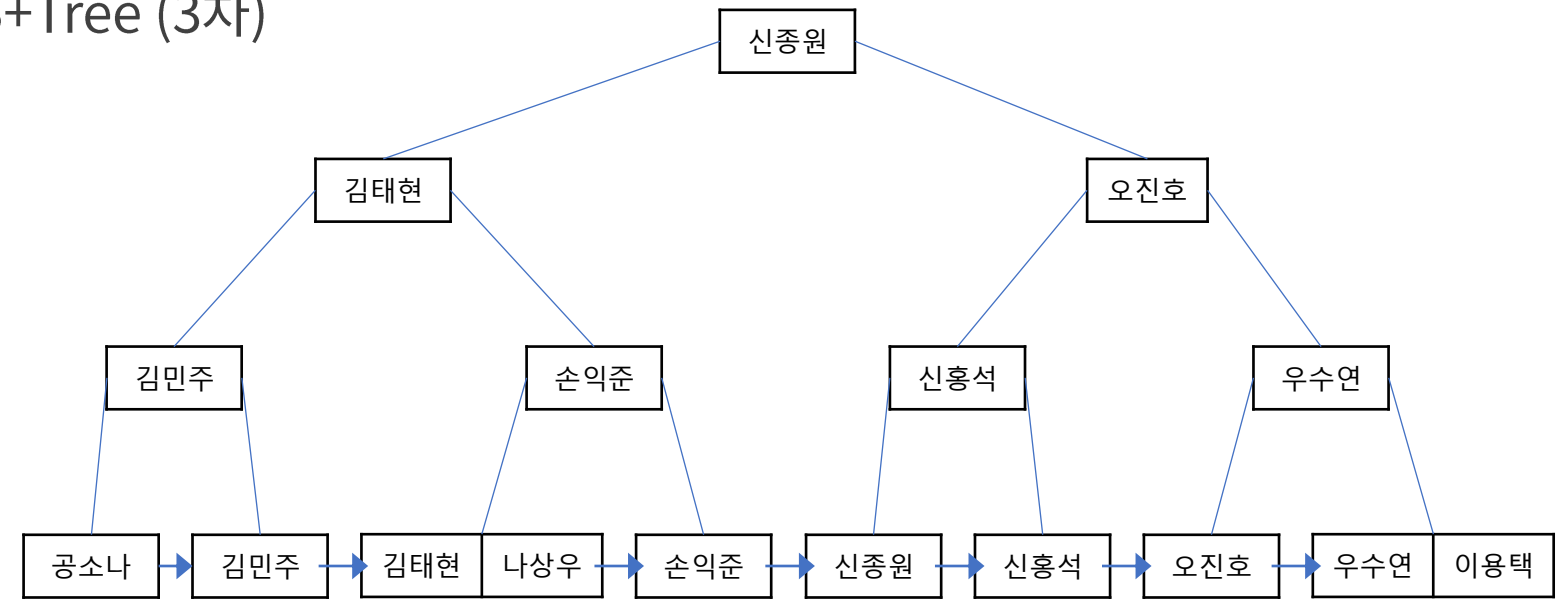



key

| name |
| --- |
| 우수연 |
| 공소나 |
| 김민주 |
| 김태현 |
| 손익준 |
| 신종원 |
| 신홍석 |
| 나상우 |
| 오진호 |
| 이용택 |
| ... |
| 너구리 |

# Index 등장!

B+Tree (3차)




key

| name |
| --- |
| 우수연 |
| 공소나 |
| 김민주 |
| 김태현 |
| 손익준 |
| 신종원 |
| 신홍석 |
| 나상우 |
| 오진호 |
| 이용택 |
| ... |
| 너구리 |

# 한번 직접 테스트해보자

```sql
# 테스트 데이터 17만건 준비
select count(*) from member;

# cache 사용 안함
show VARIABLES like 'have_query_cache';

# member 테이블의 인덱스 확인
show index from member;

# 인덱스 없이 테스트
select * from member where name = '나상우';    # 153 ms
explain select * from member where name = '나상우';

# name 인덱스 생성 후 테스트
create index idx_name on member(name);
select * from member where name = '나상우';    # 72 ms
explain select * from member where name = '나상우';
```

# 그럼 요건?

```
SELECT AVG(grade) FROM member
WHERE age BETWEEN 20 AND 40;
```

member

AVG(grade) 20 < age < 40

| mid (PK) | name | part | grade | age |
|---|---|---|---|---|
| 1 | 우수연 | core | 100 | 27 |
| 2 | 공소나 | web | 96 | 52 |
| 3 | 김민주 | web | 38 | 55 |
| 4 | 김태현 | web | 74 | 2 |
| 5 | 손익준 | ai | 81 | 39 |
| 6 | 신종원 | ai | 67 | 22 |
| 7 | 신홍석 | ai | 55 | 26 |
| 8 | 나상우 | server | 29 | 9 |
| 9 | 오진호 | server | 81 | 9 |
| 10 | 이용택 | server | 27 | 80 |
| ... | ... | ... | ... | ... |
| 10000000 | 너구리 | server | 50 | 20 |