

# 알튜비튜

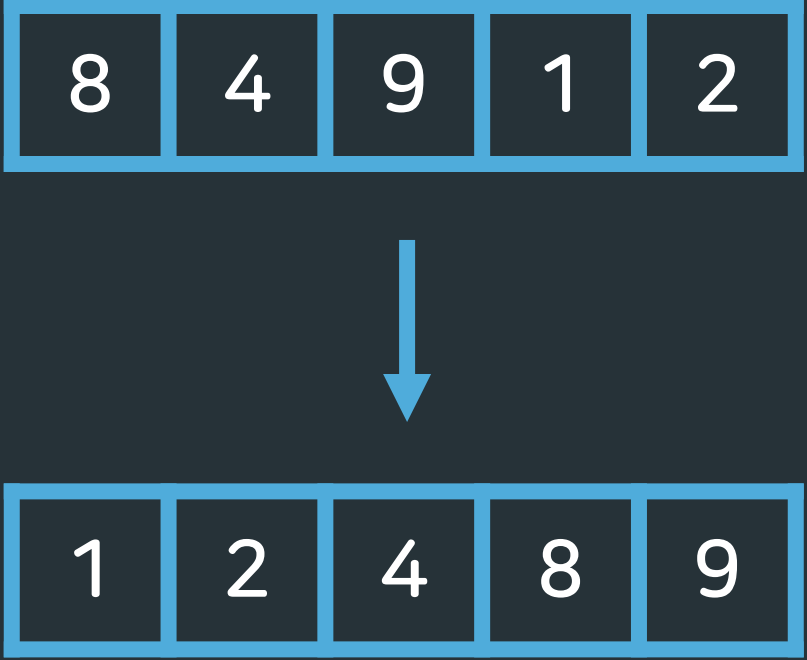
## 정렬, 맵, 셋

배열의 원소를 정렬하는 방법에는 여러가지가 있습니다.

그 중에서 시간 복잡도  $O(n^2)$ 의 버블 정렬과  $O(n \log n)$ 의 합병 정렬을 알아본 뒤, STL의 sort 알고리즘에 대해 배웁니다.

STL에서 제공하는 associative container인 set과 map에 대해 알아봅니다.

데이터를 선형으로 저장하는 sequence container (ex. vector)와 달리 연관된 key-value 쌍을 저장합니다.



$O(n^2)$

Insertion sort  
Selection sort  
Bubble sort

$O(n \log n)$

Quick sort  
Merge sort  
Heap sort

$O(n^2)$

Insertion sort  
Selection sort  
Bubble sort

$O(n \log n)$

Quick sort  
Merge sort  
Heap sort

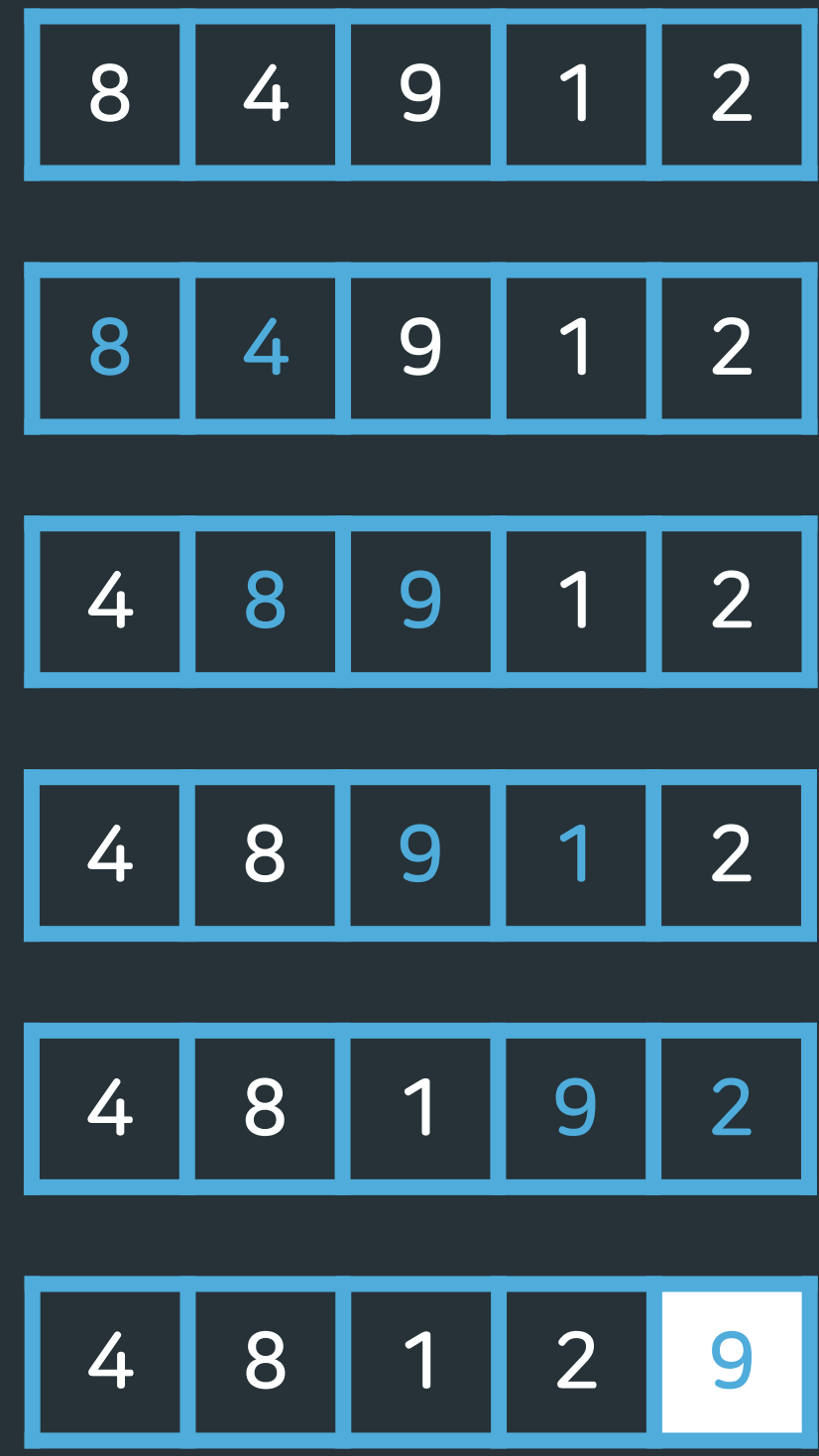
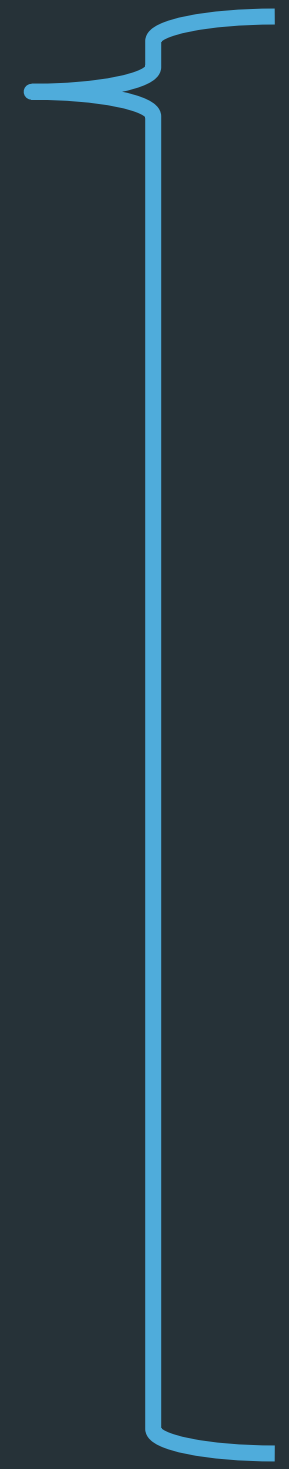
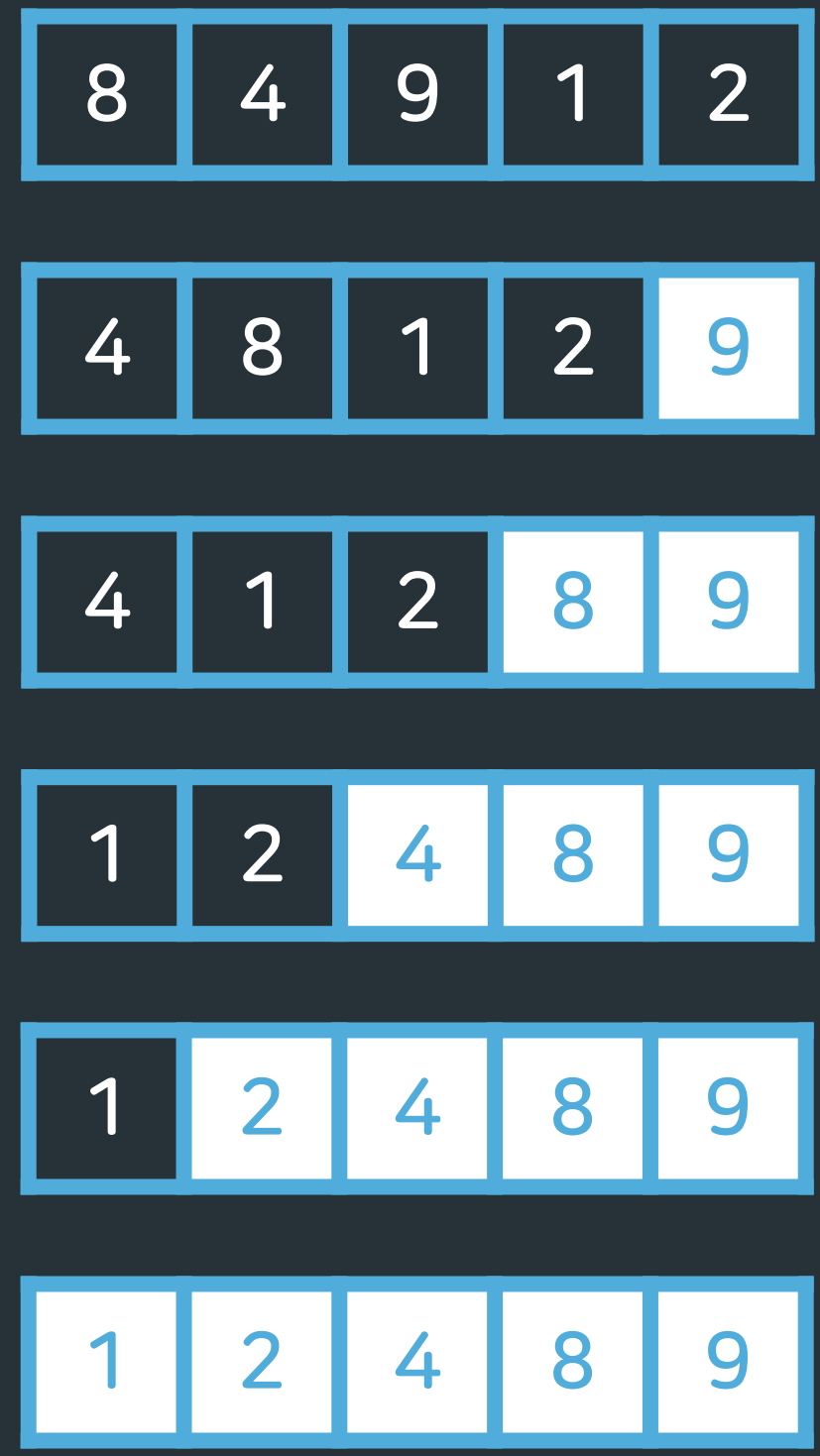
오름차순 정렬이라고 가정하고 설명합니다!

## Bubble sort

- 인접한 두 원소를 비교
- (왼쪽 원소) > (오른쪽 원소) 라면 swap!
- 가장 큰 원소부터 오른쪽에 정렬됨
- 데이터가 하나씩 정렬되면서 비교에서 제외

# 버블 정렬

(왼쪽 원소) > (오른쪽 원소) 라면 swap!



## /<> 2750번 : 수 정렬하기 - Bronze 1

### 문제

- N개의 수를 오름차순 정렬

### 제한 사항

- N의 범위는  $1 \leq N \leq 1,000$
- 각각의 수 k는  $-1,000 \leq k \leq 1,000$ 이며 중복되지 않음

## 예제 입력1

```
5
5 2 3 4 1
```

## 예제 입력2

```
5
2 1 3 4 5
```

## 예제 출력1

```
1 2 3 4 5
```

## 예제 출력2

```
1 2 3 4 5
```



## /<> 2750번 : 수 정렬하기 - Bronze 1

### 문제

- N개의 수를 오름차순 정렬

### 제한 사항

- N의 범위는  $1 \leq N \leq 1,000$
- 각각의 수 k는  $-1,000 \leq k \leq 1,000$ 이며 중복되지 않음

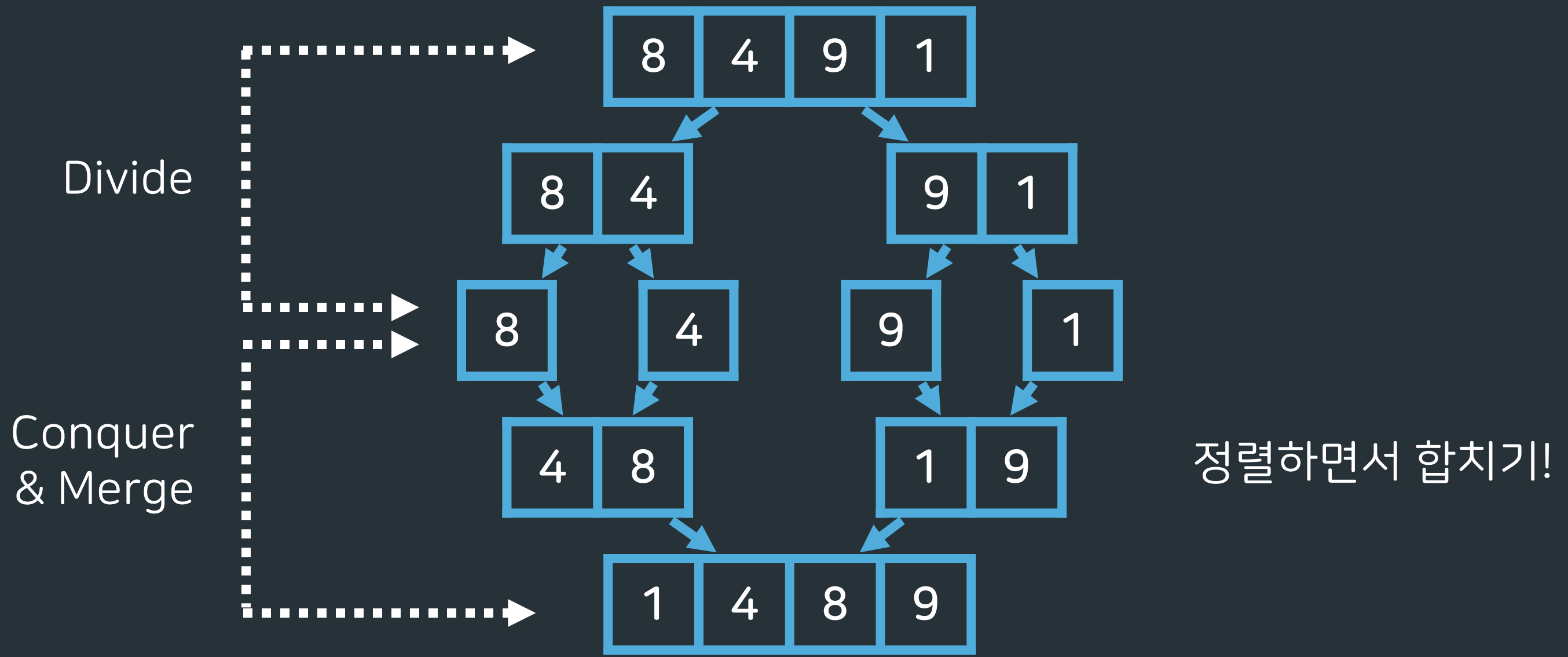
→ N의 범위가 최대 1,000이기 때문에  $O(n^2)$ 의 알고리즘이라도 시간초과가 발생하지 않음!

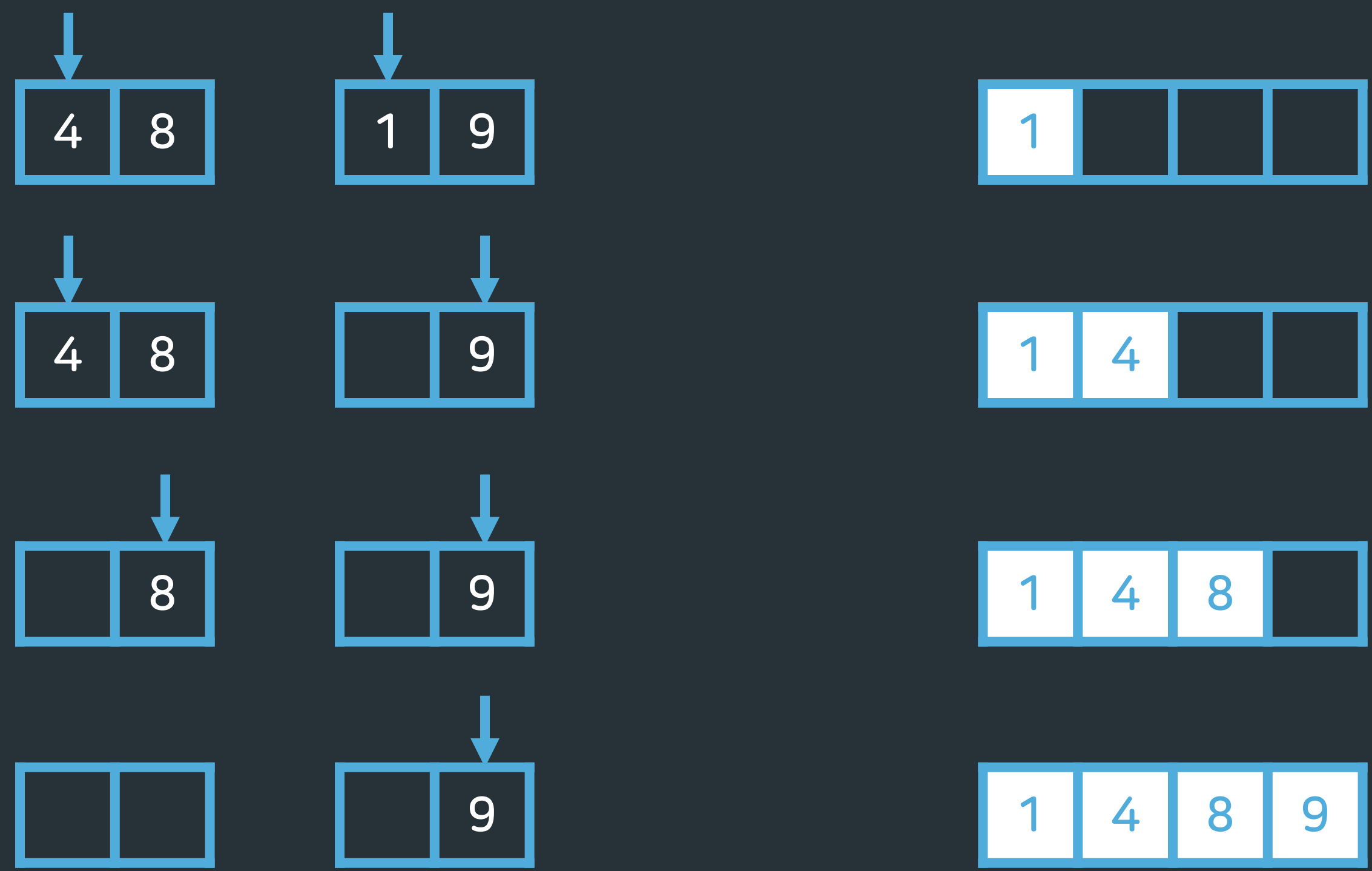
## Merge sort

- 분할 정복(Divide and Conquer) 방식으로 설계된 알고리즘
- 하나의 배열을 정확히 반으로 나눔 (Divide)
- 나뉜 배열들을 정렬 (Conquer)
- 다시 하나의 배열로 합치기 (Merge)

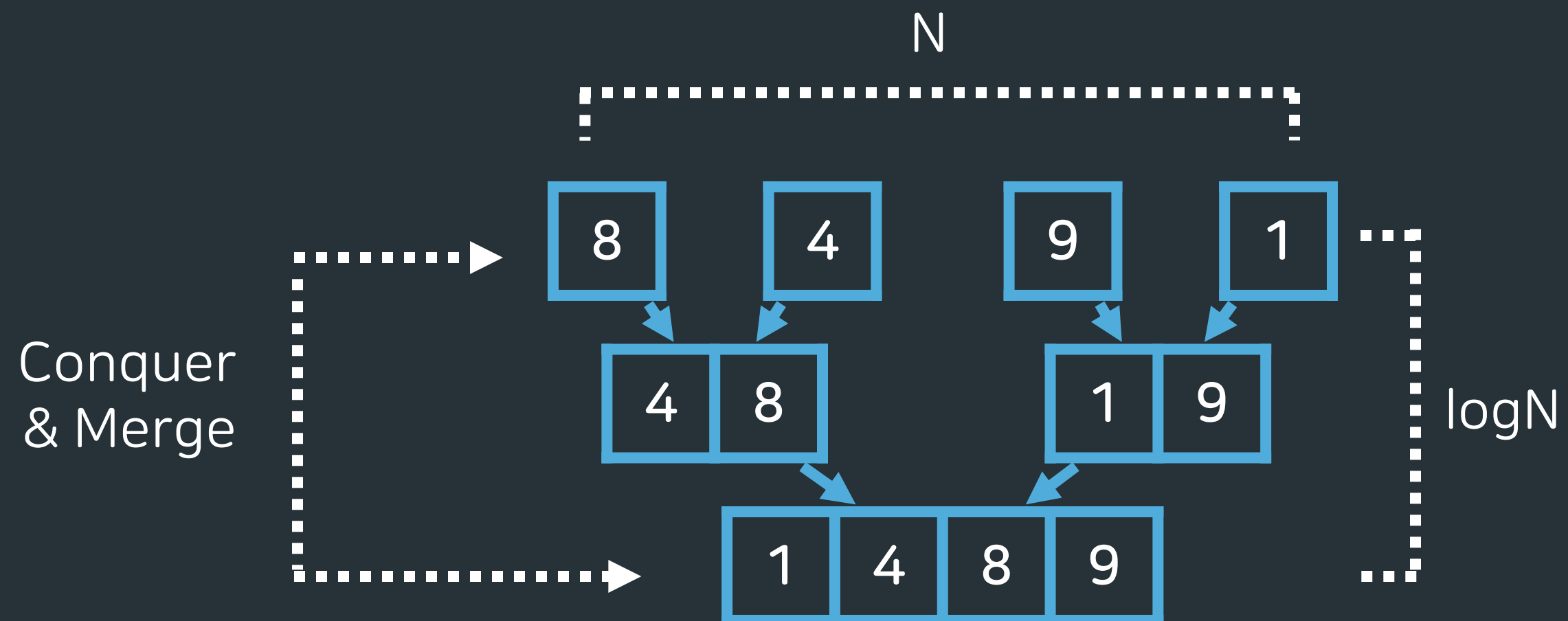
## 분할 정복

- 한 번에 해결할 수 없는 문제를 작은 문제로 분할하여 해결하는 알고리즘
- 주로 재귀 함수로 구현
- 크게 3 단계로 이루어짐
  1. Divide: 문제 분할
  2. Conquer: 쪼개진 작은 문제 해결
  3. Combine: 해결된 작은 문제들을 다시 합침





- 시간복잡도  $O(n \log n)$



## /<> 2751번 : 수 정렬하기 2 – Silver 5

### 문제

- N개의 수를 오름차순 정렬

### 제한 사항

- N의 범위는  $1 \leq N \leq 1,000,000$
- 각각의 수 k는  $-1,000,000 \leq k \leq 1,000,000$ 이며 중복되지 않음

## 예제 입력1

```
5
5 2 3 4 1
```

## 예제 입력2

```
5
2 1 3 4 5
```

## 예제 출력1

```
1 2 3 4 5
```

## 예제 출력2

```
1 2 3 4 5
```



## /<> 2751번 : 수 정렬하기 2 – Silver 5

### 문제


- N개의 수를 오름차순 정렬

### 제한 사항

- N의 범위는  $1 \leq N \leq 1,000,000$
- 각각의 수 k는  $-1,000,000 \leq k \leq 1,000,000$ 이며 중복되지 않음

→ N의 범위가 최대 1,000,000이기 때문에  $O(n^2)$ 의 알고리즘이라면 시간초과!

# 세상에 정렬할 일이 얼마나 많은데...



Search:

Reference <algorithm> sort

Not logged in

C++  
Information  
Tutorials  
Reference  
Articles  
Forum

Reference  
+ C library:  
+ Containers:  
+ Input/Output:  
+ Multi-threading:  
- Other:  
  <algorithm>  
  <bitset>  
  <chrono>  
  <codecvt>  
  <complex>  
  <exception>  
  <functional>  
  <initializer\_list>  
  <iterator>  
  <limits>  
  <locale>  
  <memory>  
  <new>  
  <numeric>  
  <random>  
  <ratio>  
  <regex>  
  <stdexcept>  
  <string>  
  <system\_error>  
  <tuple>  
  <typeindex>  
  <typeinfo>

You were redirected to [cplusplus.com/sort](http://cplusplus.com/sort) || See search results for: "sort"

function template  
**std::sort** <algorithm>

default (1)

custom (2)

```
template <class RandomAccessIterator>
void sort (RandomAccessIterator first, RandomAccessIterator last);

template <class RandomAccessIterator, class Compare>
void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

**Sort elements in range**  
Sorts the elements in the range `[first,last)` into ascending order.  
  
The elements are compared using `operator<` for the first version, and `comp` for the second.  
  
Equivalent elements are not guaranteed to keep their original relative order (see `stable_sort`).

**Parameters**  
**first, last**  
Random-access iterators to the initial and final positions of the sequence to be sorted. The range used is `[first,last)`, which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.  
RandomAccessIterator shall point to a type for which `swap` is properly defined and which is both *move-constructible* and *move-assignable*.  
**comp**  
Binary function that accepts two elements in the range as arguments, and returns a value convertible to `bool`. The value returned indicates whether the element passed as first argument is considered to go before the second in the specific *strict weak ordering* it defines. The function shall not modify any of its arguments. This can either be a function pointer or a function object.

**Return value**  
none

## /<> 10825번 : 국영수 - Silver 4

### 문제

- 도현이네 반 학생 N명의 이름과 국어, 영어, 수학 점수가 주어진다.
- 다음의 조건으로 학생들을 정렬하자.
  1. 국어 점수가 감소하는 순서
  2. 국어 점수가 같다면 영어 점수가 증가하는 순서
  3. 국어 점수와 영어 점수가 같다면 수학 점수가 감소하는 순서
  4. 모든 점수가 같으면 이름이 사전 순으로 증가하는 순서

### 제한 사항

- N의 범위는  $1 \leq N \leq 100,000$
- 점수의 범위는  $1 \leq \text{score} \leq 100$
- 이름은 알파벳 대소문자로 이루어진 10자리 이하의 문자열

예제 입력

```
12
Junkyu 50 60 100
Sangkeun 80 60 50
Sunyoung 80 70 100
Soong 50 60 90
Haebin 50 60 100
Kangsoo 60 80 100
Donghyuk 80 60 100
Sei 70 70 70
Wonseob 70 70 90
Sanghyun 70 70 80
nsj 80 80 80
Taewhan 50 60 90
```

예제 출력

```
Donghyuk
Sangkeun
Sunyoung
nsj
Wonseob
Sanghyun
Sei
Kangsoo
Haebin
Junkyu
Soong
Taewhan
```

## Hint

1. 구조체... 기억나시나요?
2. 분명히 아까 쓴 sort 함수는 인자(parameter)가 2개였는데?

```
std::sort<algorithm>  
  
default (1)  template <class RandomAccessIterator>  
              void sort (RandomAccessIterator first, RandomAccessIterator last);  
  
custom (2)   template <class RandomAccessIterator, class Compare>  
              void sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);
```

이건 뭘까요??

## std::sort

- 인자로 배열의 처음 시작 위치와, 끝 위치를 보내줌
- default 값은 오름차순 정렬
- 내림차순 정렬은 세 번째 인자에 `greater<>()` 을 넣어서
- 세 번째 인자에 비교함수(`cmp`)를 넣어서 원하는 조건대로 정렬할 수 있음!
- 비교함수가 `false`를 리턴할 경우 `swap`하는 것임을 주의!

## 정리

- 정렬 알고리즘은 종류가 많다. (Insertion, Selection, Bubble, Merge, Quick, ...)
- 근데 그냥 구현하지 말고 `sort` 함수 쓰자!
- `default` 값은 오름차순 정렬, 내림차순 정렬은 `greater<>()`, 그 밖의 정렬은 `comp` 정의하기.
- `comp` 정의할 때는 헛갈리지 말기! `sort`는 `comp`가 `false`를 반환해야 `swap`됨! (sort는...?)
- 정렬 알고리즘은 그리디 문제에 쓰이는 경우가 많아요!

## 이것도 알아보세요!

- 정렬 알고리즘 중엔 시간 복잡도가  $O(n)$ 인 계수 정렬(Counting sort)이 있어요.
  1. 어떻게 겨우  $O(n)$ 만에 정렬을 할 수 있을까요?
  2. 우리 그럼 왜 계수 정렬을 쓰지 않고  $O(n \log n)$ 의 정렬 알고리즘을 사용하는 걸까요?
- 정렬 알고리즘은 `stable sort`와 `unstable sort`로 나눌 수 있어요. 이건 어떤 개념일까요?
- 자료형이 `pair<int, int>`인 배열을 `comp`없이 정렬하면 어떻게 될까요?



이런 문제가 있다고 해봅시다.



“배열 [1, 6, 2, 1, 9, 8]에서 중복된 수를 제거한 뒤, 오름차순 정렬한 결과는?”



# 벡터를 사용한다면?



```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main(){
    vector<int> arr = {1, 6, 2, 1, 9, 8};
    vector<int> result;
    for (int i = 0; i < arr.size(); i++) {
        // 존재하지 않는 원소라면 -> result에 넣기(중복 방지)
        if (find(result.begin(), result.end(), arr[i]) == result.end()) {
            result.push_back(arr[i]);
        }
    }
    sort(result.begin(), result.end()); //오름차순 정렬
    return 0;
}
```

시간 복잡도면에서도 효율적이지 않고, 코드도 길다.

## Set

- key 라고 불리는 원소(value)의 집합
- key 값을 정렬된 상태로 저장
- key 값을 중복 없이 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log N)$
- 랜덤한 인덱스의 데이터에 접근 불가

# 셋으로 다시 구현해봅시다!

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
    vector<int> arr = {1, 6, 2, 1, 9, 8};
    set<int> result;

    for (int i = 0; i < arr.size(); i++) {
        result.insert(arr[i]);
    }
}
```

← set에서 삽입 = insert()

→ 1 2 6 8 9



# 랜덤한 인덱스에 접근 불가?

```
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main() {
    vector<int> v;
    set<int> s;

    v.push_back(2);
    v.push_back(1);
    s.insert(2);
    s.insert(1);

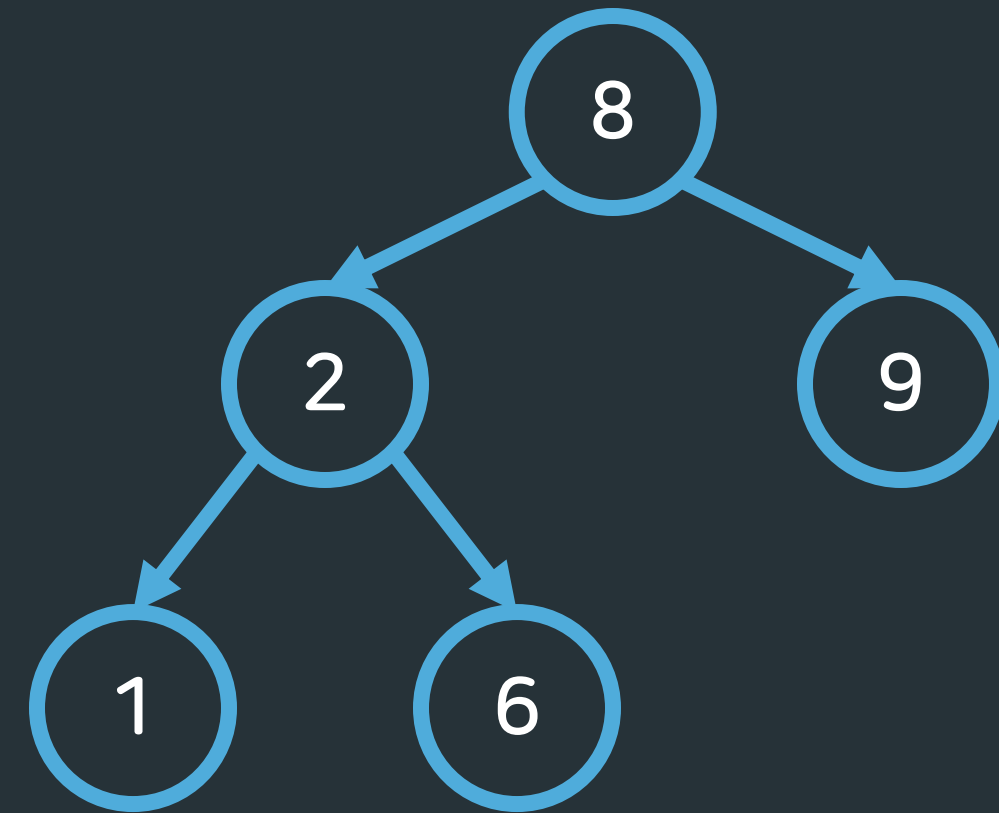
    int a = v[0];
    int b = s[0];
}
```

← 가능  
← 불가능

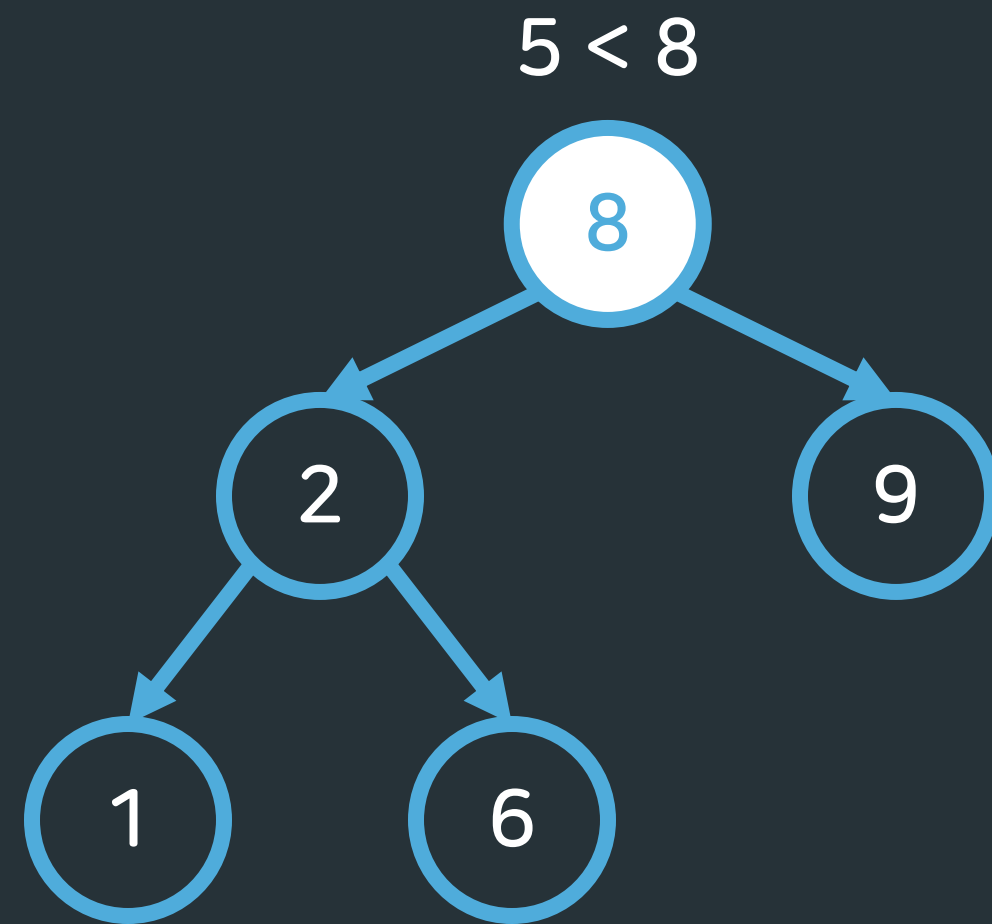
## BST (Binary Search Tree)

- 하나의 parent(root)에 최대 2개의 child가 있음
- 부모의 왼쪽 서브 트리 값들은 모두 부모 노드보다 작음
- 부모의 오른쪽 서브 트리 값들은 모두 부모 노드보다 큼

\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree(균형 이진 트리)를 사용

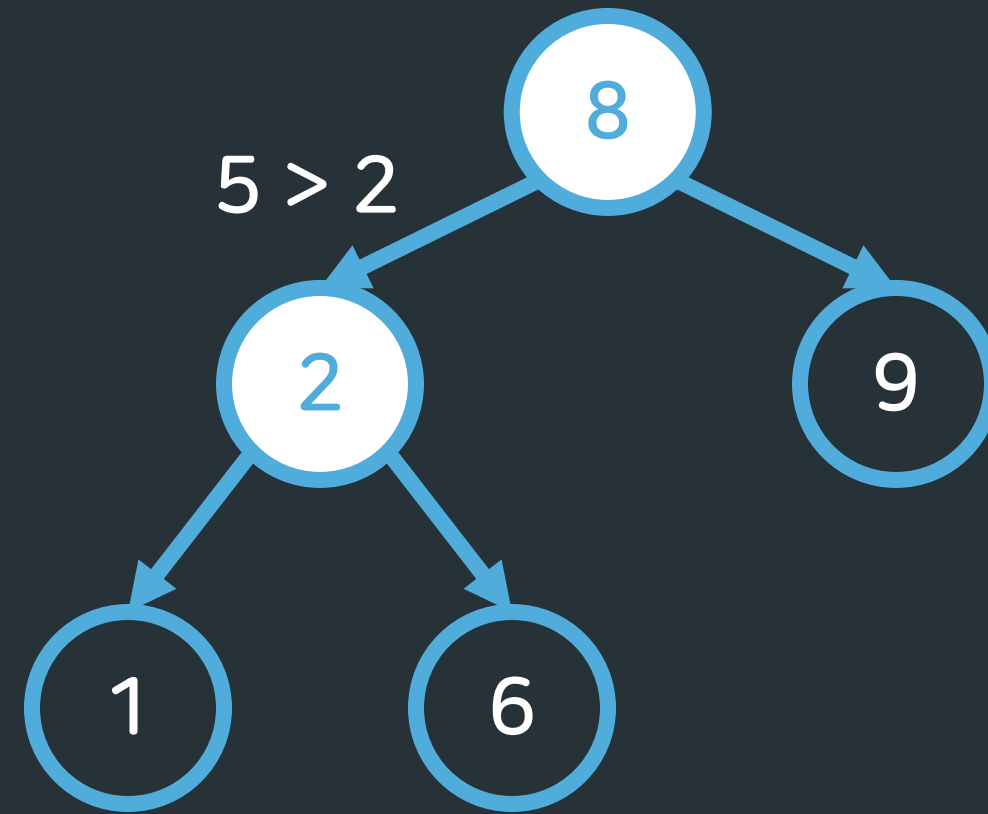


# C++ 셋에 데이터가 들어가는 과정



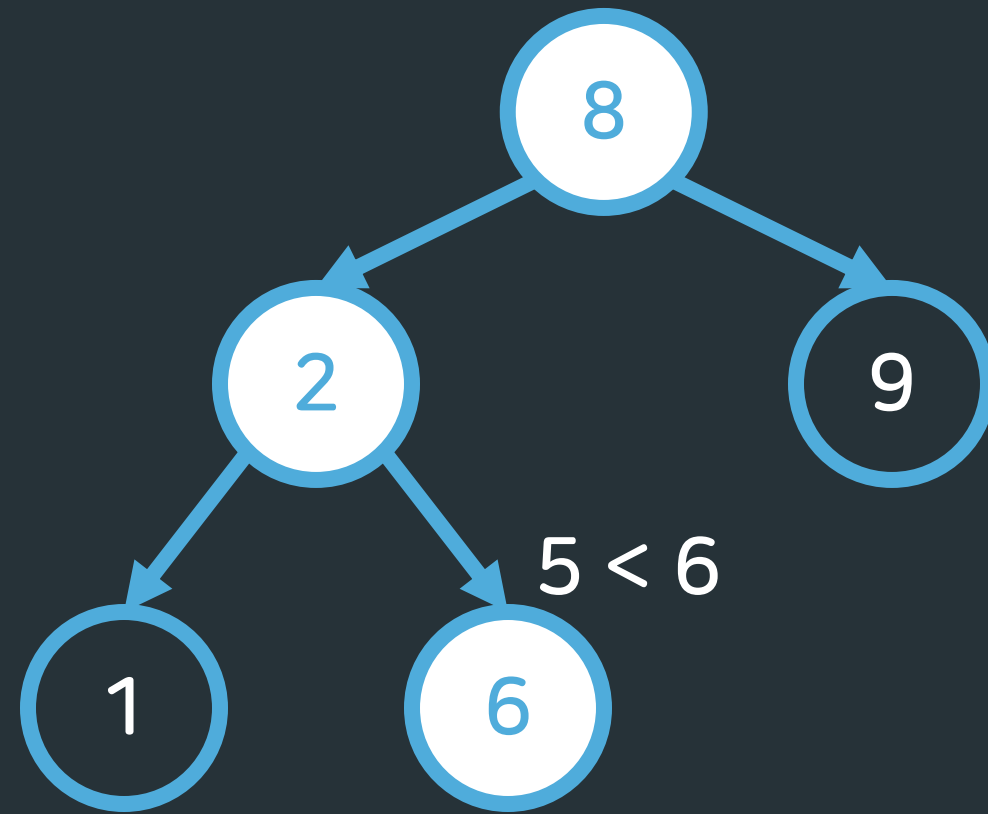
현재 들어오는 key = 5

# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5

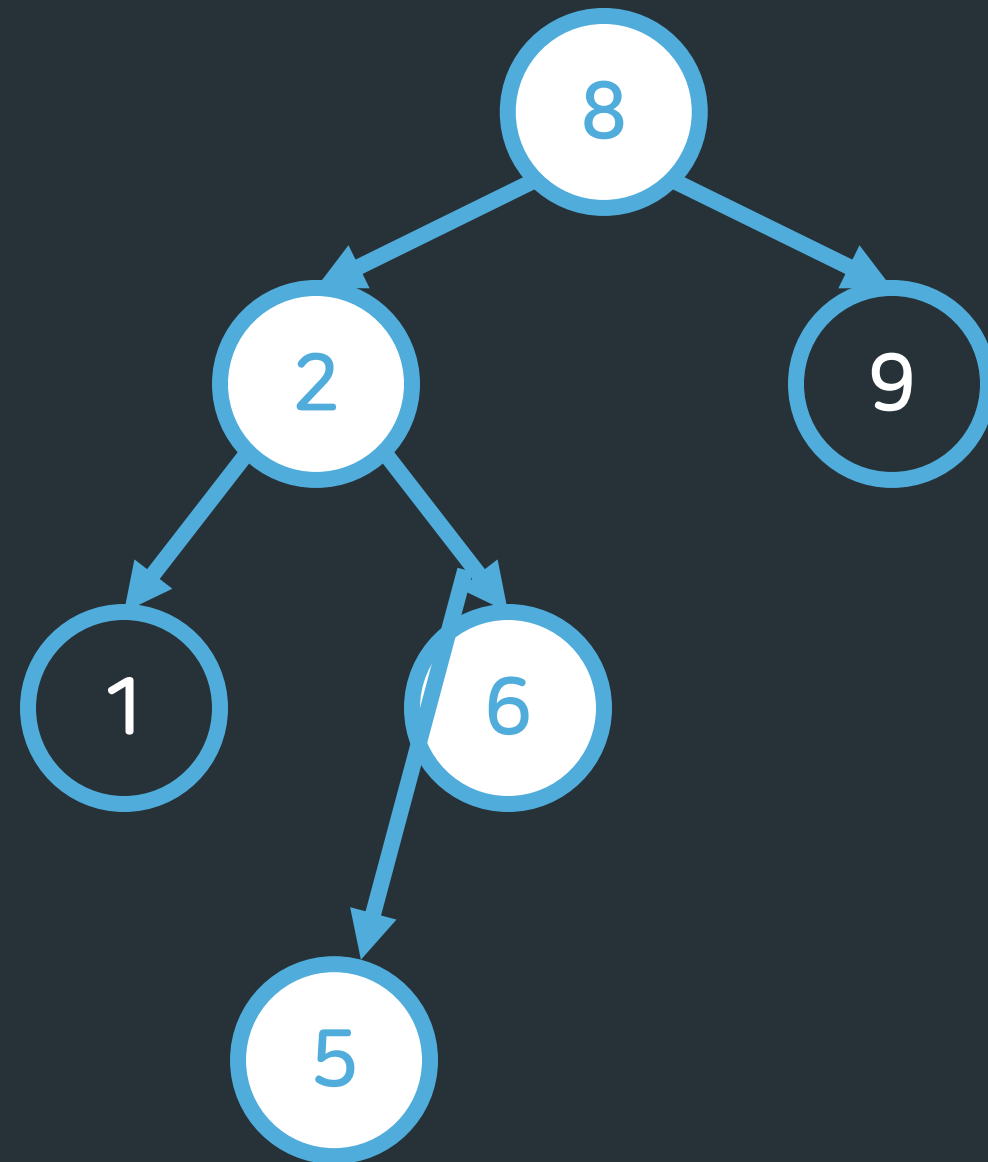
# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5



# C++ 셋에 데이터가 들어가는 과정



현재 들어오는 key = 5

## 반복자 (iterator)

- 포인터와 비슷한 개념
- 컨테이너에 보관된 원소에 접근할 때 사용
- "container<자료형>::iterator" 로 사용 가능
- begin(): 순차열의 시작
- end(): 순차열의 끝 (실제 원소를 가르키는 게 아니라 마지막 원소의 다음을 가리킴)
- 임의 접근 반복자(vector, deque)를 제외하고는 사칙연산 불가능

```
#include <iostream>
#include <set>

using namespace std;

int main() {
    set<int> s;

    s.insert(2);
    s.insert(1);

    set<int>::iterator iter; ← 반복자 선언
    for (iter = s.begin(); iter != s.end(); iter++) { ← 순회
        cout << *iter << ' '; ← 포인터로 접근
    }
}
```

# 낯설어하실 것 같아서 벡터로도 준비했어요

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    vector<int> v;

    v.push_back(2);
    v.push_back(1);

    vector<int>::iterator iter; ← 반복자 선언
    for (iter = v.begin(); iter != v.end(); iter++) { ← 순회
        cout << *iter << ' '; ← 포인터로 접근
    }
}
```

# C++은 생각보다 똑똑해요

```
#include <iostream>
#include <set>
```

```
using namespace std;
```

```
int main() {
    set<int> s;
```

```
    s.insert(2);
    s.insert(1);
```

```
    for (auto iter = s.begin(); iter != s.end(); iter++) {
        cout << *iter << ' ';
    }
```

← 자동으로 반복  
자 선언과 동시에  
순회

```
    for (auto iter:s) {
        cout << iter << ' ';
    }
```

← 자동으로 반복자 선언과 동시에 순회  
(조금 더 향상된 버전)

```
}
```

## /<> 7785번 : 회사에 있는 사람 - Silver 5

### 문제

- 어떤 사람이 회사에 들어왔는지(enter), 나갔는지(leave)가 기록된 시스템 로그가 주어짐
- 현재 회사에 있는 모든 사람을 사전의 역순으로 출력

### 제한 사항

- N(로그의 출입 기록 수)의 범위는  $2 \leq N \leq 1,000,000$
- 회사에는 동명이인이 없고, 대소문자를 구별
- 사람들의 이름은 5글자 이하의 문자열

## /<> 7785번 : 회사에 있는 사람 - Silver 5

### 문제

- 어떤 사람이 회사에 들어왔는지(enter), 나갔는지(leave)가 기록된 시스템 로그가 주어짐
- 현재 회사에 있는 모든 사람을 사전의 역순으로 출력

### 제한 사항

- N(로그의 출입 기록 수)의 범위는  $2 \leq N \leq 1,000,000$
  - 회사에는 동명이인이 없고, 대소문자를 구별 → 중복 x
  - 사람들의 이름은 5글자 이하의 문자열
- => enter이면 set에 추가(insert), leave면 set에서 제거(erase)  
=> 마지막에 set에 있는 모든 사람을 출력

## 예제 입력1

```
4
Baha enter
Askar enter
Baha leave
Artem enter
```

## 예제 출력1

```
Askar
Artem
```



이런 문제가 있다고 해봅시다.



“학생의 이름과 해당 학생의 수학 성적이 주어진다.  
학생의 이름이 입력되면 해당 학생의 수학 성적을 구하라.”

# 구조체와 벡터를 사용한다면?



```
#include <iostream>
#include <vector>

using namespace std;

struct info {
    string name;
    int math_score;
};

int main() {
    vector<info> student;

    student.push_back({"lee", 42});
    student.push_back({"lim", 100});
    student.push_back({"bae", 50});

    string target = "bae";
    for (int i = 0; i < student.size(); i++) {
        if (student[i].name == target) {
            cout << student[i].math_score;
        }
    }
}
```

cf) vector 컨테이너에 지금 구현한 것처럼  
검색역할을 해주는 함수가 있어요!  $O(n)$ 으  
로 동일한 시간복잡도를 가집니다.

# 구조체와 벡터를 사용한다면?



```
#include <iostream>
#include <vector>

using namespace std;

struct info {
    string name;
    int math_score;
};

int main() {
    vector<info> student;

    student.push_back({"lee", 42});
    student.push_back({"lim", 100});
    student.push_back({"bae", 50});

    string target = "bae";
    for (int i = 0; i < student.size(); i++) {
        if (student[i].name == target) {
            cout << student[i].math_score;
        }
    }
}
```

학생 1명을 찾는데  $O(n)$ 의 시간 복잡도... 만약 찾아야할 학생이 천만명이라면?

## Map

- 다양한 자료형의 데이터를 **key-value** 쌍으로 저장
- key 값을 정렬된 상태로 저장
- key 값을 중복 없이 저장
- 검색, 삽입, 삭제에서의 시간 복잡도는  $O(\log N)$
- 랜덤한 인덱스의 데이터에 접근 불가

# 맵으로 다시 구현해봅시다!

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> student;

    student["lee"] = 42; ← key 값을 인덱스처럼 접근해서 key-value 삽입 가능
    student["lim"] = 100;
    student["bae"] = 50;

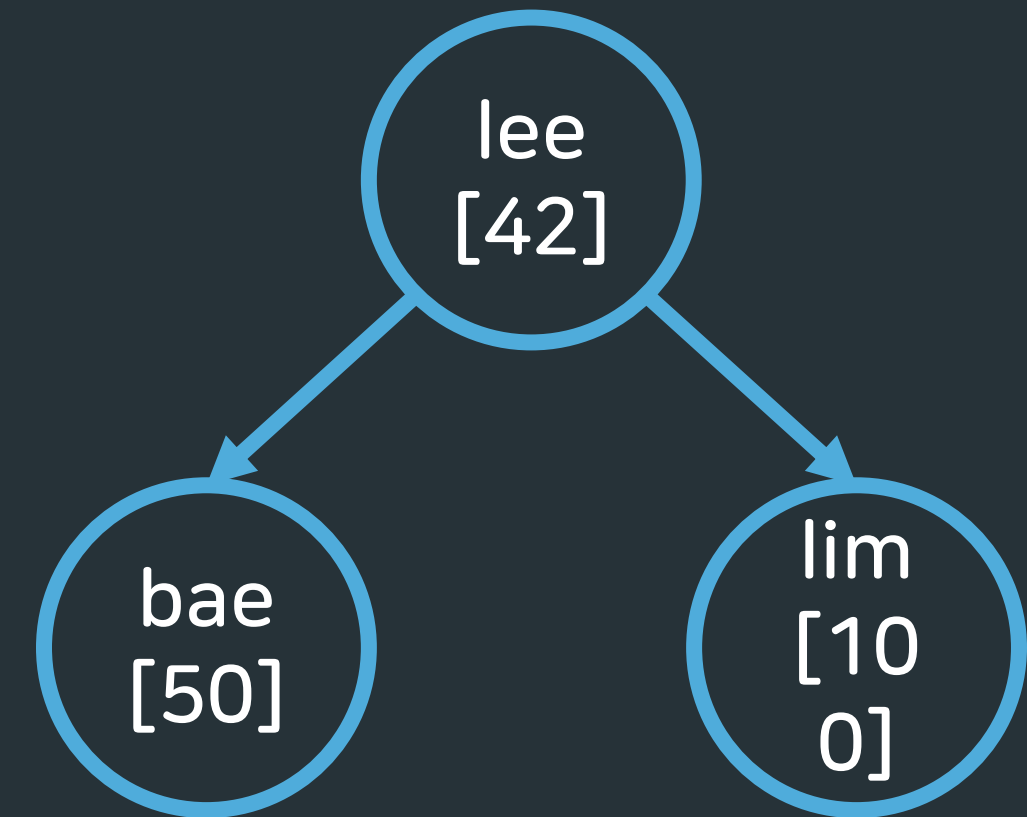
    string target = "bae";
    cout << student[target]; ← key 값을 인덱스처럼 사용해서 value에 접근 가능
}
```



## BST (Binary Search Tree)

- 하나의 parent(root)에 최대 2개의 child가 있음
- 부모의 왼쪽 서브 트리 값들은 모두 부모 노드보다 작음
- 부모의 오른쪽 서브 트리 값들은 모두 부모 노드보다 큼

\* 사실 정확히 말하면 여기서 발전된 형태인 red-black tree(균형 이진 트리)를 사용



## /<> 1620번 : 나는야 포켓몬 마스터 이다솜 - Silver 4

### 문제

- 포켓몬의 이름(string)이 입력되면 해당 포켓몬의 번호를 출력
- 포켓몬의 번호(int)가 입력되면 해당 포켓몬의 이름을 출력

### 제한 사항

- 도감에 수록되어 있는 포켓몬의 수의 범위는  $1 \leq N \leq 100,000$
- 맞춰야 하는 문제의 개수의 범위는  $1 \leq M \leq 100,000$
- 포켓몬의 이름은 첫 글자가 대문자이며 길이가 20이하인 영어 문자열

예제 입력1

26 5	Caterpie	Spearow
Bulbasaur	Metapod	Fearow
Ivysaur	Butterfree	Ekans
Venusaur	Weedle	Arbok
Charmander	Kakuna	Pikachu
Charmeleon	Beedrill	Raichu
Charizard	Pidgey	25
Squirtle	Pidgeotto	Raichu
Wartortle	Pidgeot	3
Blastoise	Rattata	Pidgey
	Raticate	Kakuna

예제 출력1

Pikachu
26
Venusaur
16
14



## /<> 2002번 : 추월 - Silver 1

### 문제

- 차의 목록(string)이 터널에 들어간/나온 순서대로 주어진다.
- 터널 내부에서 반드시 추월했을 차가 몇 대인지 출력

### 제한 사항

- 차의 대수의 범위는  $1 \leq N \leq 1,000$
- 차량 번호는 영어 대문자와 숫자로 이루어진 중복 없는 6 ~ 8글자의 문자열

예제 입력1

4  
ZG431SN  
ZG5080K  
ST123D  
ZG206A  
ZG206A  
ZG431SN  
ZG5080K  
ST123D

예제 출력1

1

예제 입력2

5  
ZG5080K  
PU305A  
RI604B  
ZG206A  
ZG232ZF  
PU305A  
ZG232ZF  
ZG206A  
ZG5080K  
RI604B

예제 출력2

3

예제 입력3

5  
ZG206A  
PU234Q  
OS945CK  
ZG431SN  
ZG5962J  
ZG5962J  
OS945CK  
ZG206A  
PU234Q  
ZG431SN

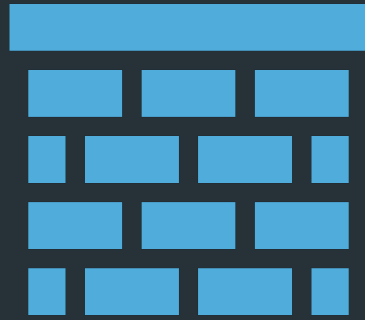
예제 출력3

2

## Hint

1. 각각의 차가 들어간 순서를 숫자로 나타내면 보기 쉽지 않을까요?  
?
2. 'A'차와 'B'차가 있을 때, A가 B를 추월했음을 어떻게 알까요?

# 문제를 간단하게 바꿔 볼게요



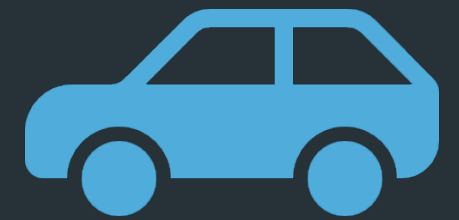
ZG431SN



ZG5080K

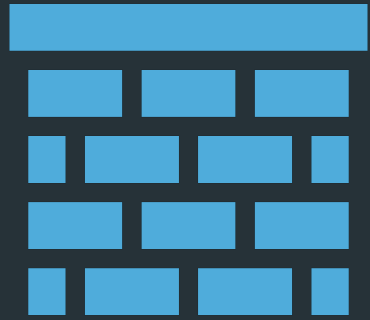


ST123D

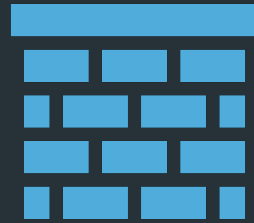


ZG206A

# 문제를 간단하게 바꿔 볼게요



# 문제를 간단하게 바꿔 볼게요



ZG431SN



ZG5080K



ST123D



ZG206A



ZG206A



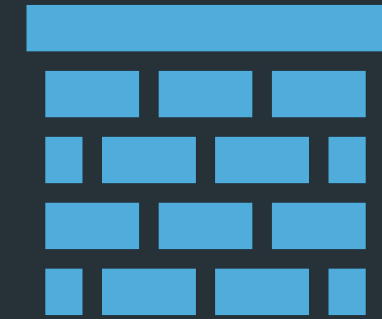
ZG431SN



ZG5080K



ST123D



# 문제를 간단하게 바꿔 볼게요



4



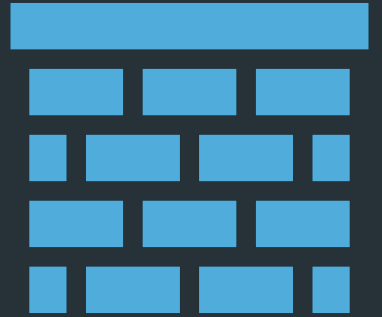
1



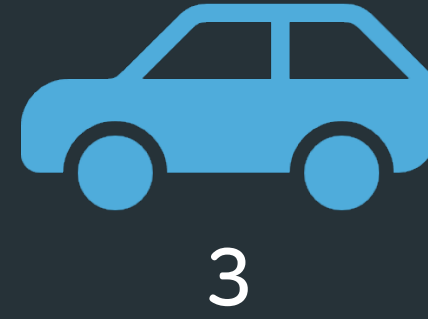
2



3

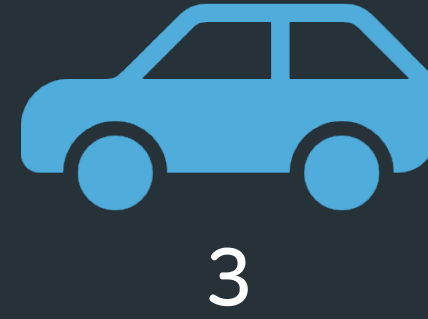
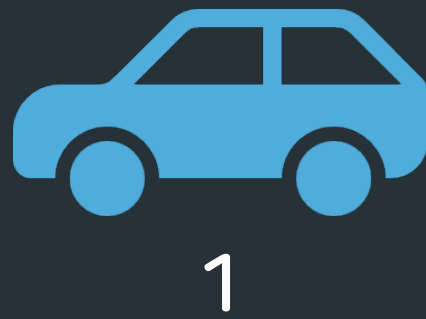


# 아마 터널 안에서는...

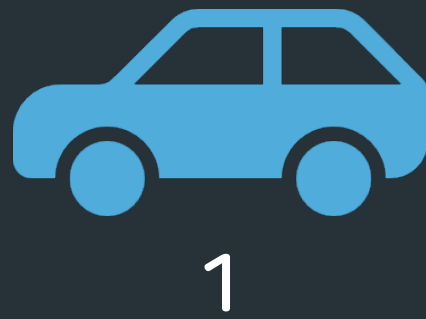




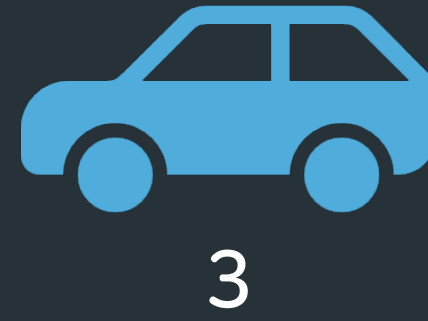
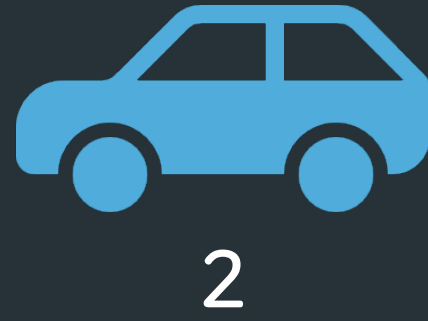
# 아마 터널 안에서는...



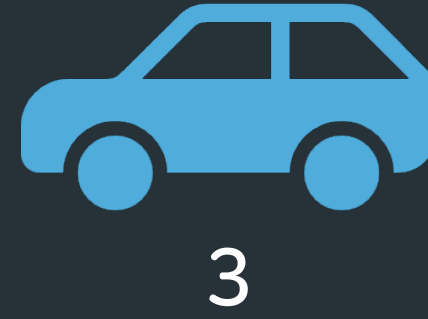
# 아마 터널 안에서는...



# 아마 터널 안에서는...



# 아마 터널 안에서는...



A보다 터널에서 늦게 나온 차 중에서  
A보다 인덱스가 작은 차가 하나라도 있다면  
먼저 들어왔는데 나올때 A보다 뒤에 있다는 것이므로  
A는 터널안에서 추월을 했다!

## 정리

- 연관 컨테이너(Set, Map)은 검색에 최적화된 자료구조
- 내부 구조는 BST에서 발전된 형태인 Red-Black Tree
- 따라서 C++의 Set, Map은 검색, 삽입, 삭제에서 시간복잡도  $O(\log N)$
- C++은 기본적으로 key값을 중복없이 정렬된 상태로 저장하지만, 정렬 없이 중복저장 하는 방법도 있음
- Set과 Map에 저장된 데이터를 순회하기 위해서는 반복자 (iterator)를 사용해야 함

## 이것도 알아보세요!

- BST와 Red-Black Tree의 **차이**는 뭡까요?
- BST에서 데이터를 **삭제**하기 위해선 어떻게 해야 할까요?
- 다음 코드의 **실행 결과**는?

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<string, int> m;
    int a = m["no_key"];
    cout << a;
}
```

1. 컴파일 에러
2. 런타임 에러
3. 오류 없음 (그렇다면 출력 결과는?)

## 필수

- /<> 19636번 : 요요 시뮬레이션 - Silver 5
- /<> 1431번 : 시리얼 번호 - Silver 3
- /<> 11478번 : 서로 다른 부분 문자열의 개수 - Silver 3

## 도전

- /<> 1946번 : 신입사원 - Silver 1
- /<> 9375번 : 패션왕 신해빈 - Silver 3



과제제출 마감 ~ 2월 21일 금요일 18:59

추가제출 마감 ~ 2월 23일 일요일 23:59