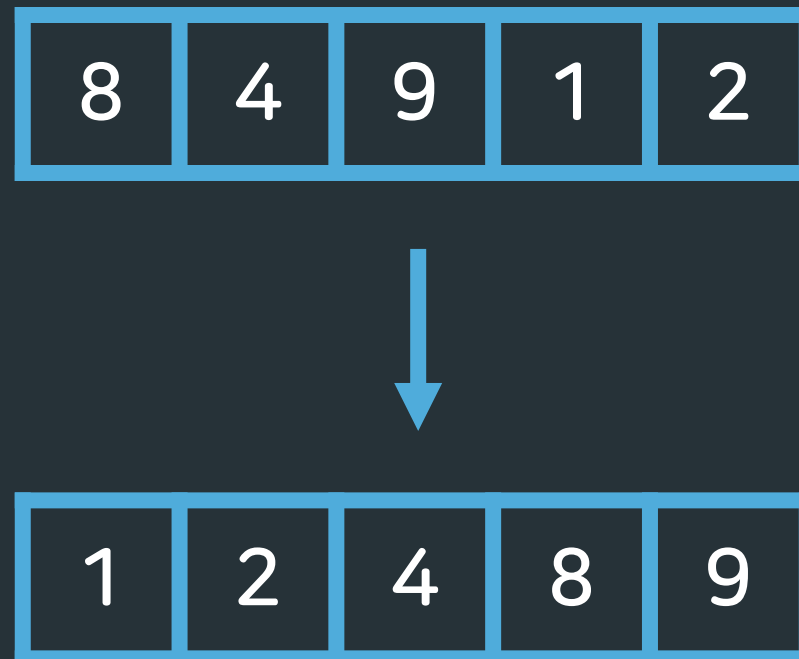


# 알튜비튜

## 정렬

배열의 원소를 정렬하는 방법에는 여러가지가 있습니다.  
오늘은 그 중에서 시간 복잡도  $O(n^2)$ 의 버블 정렬과  $O(n \log n)$ 의 합병 정렬을 알아본 뒤,  
STL의 sort 알고리즘에 대해 배웁니다.



# 대표적인 정렬 알고리즘



$O(n^2)$

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

$O(n \log n)$

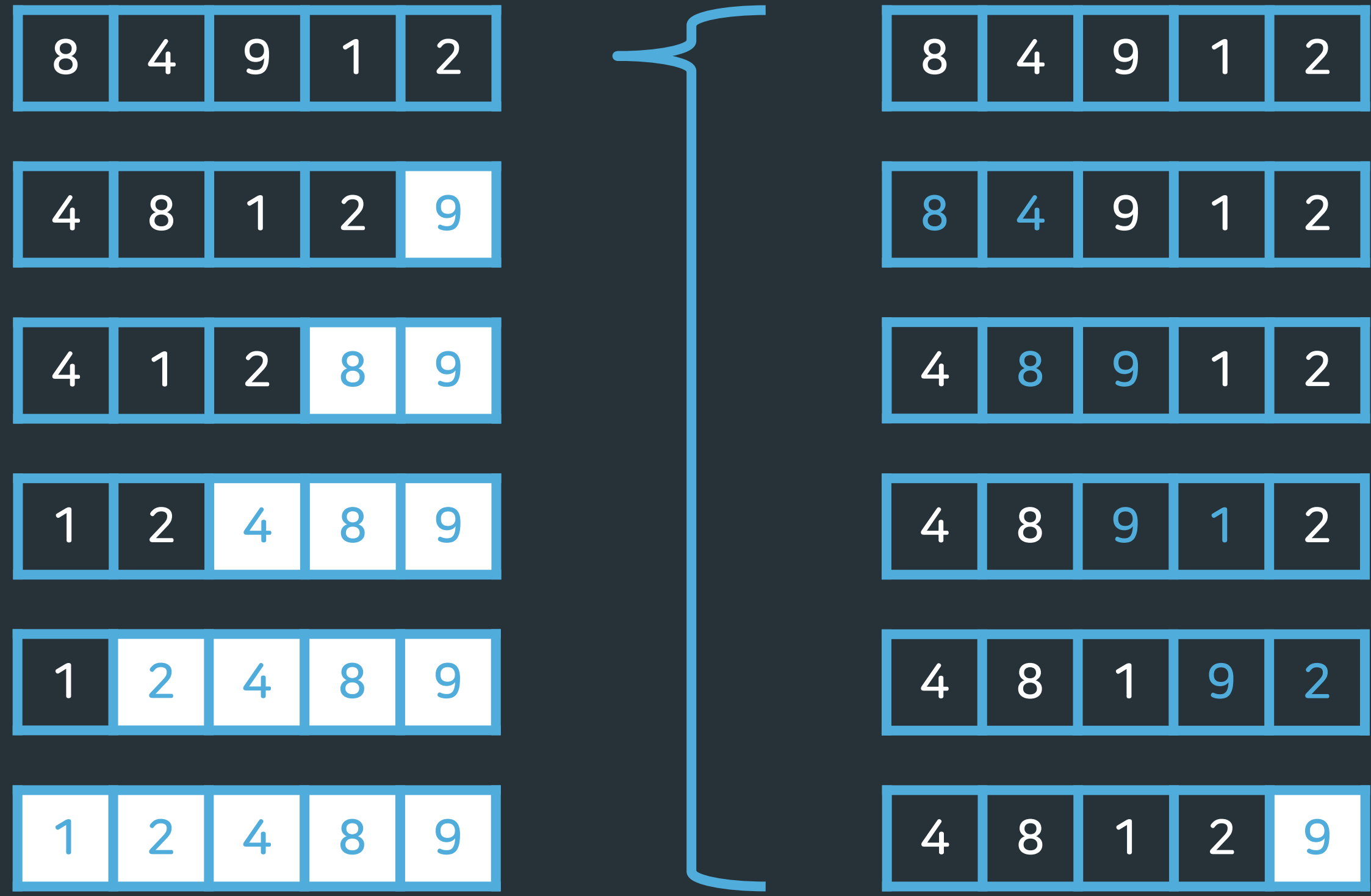
Quick sort  
Merge sort  
Heap sort

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



## /<> 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

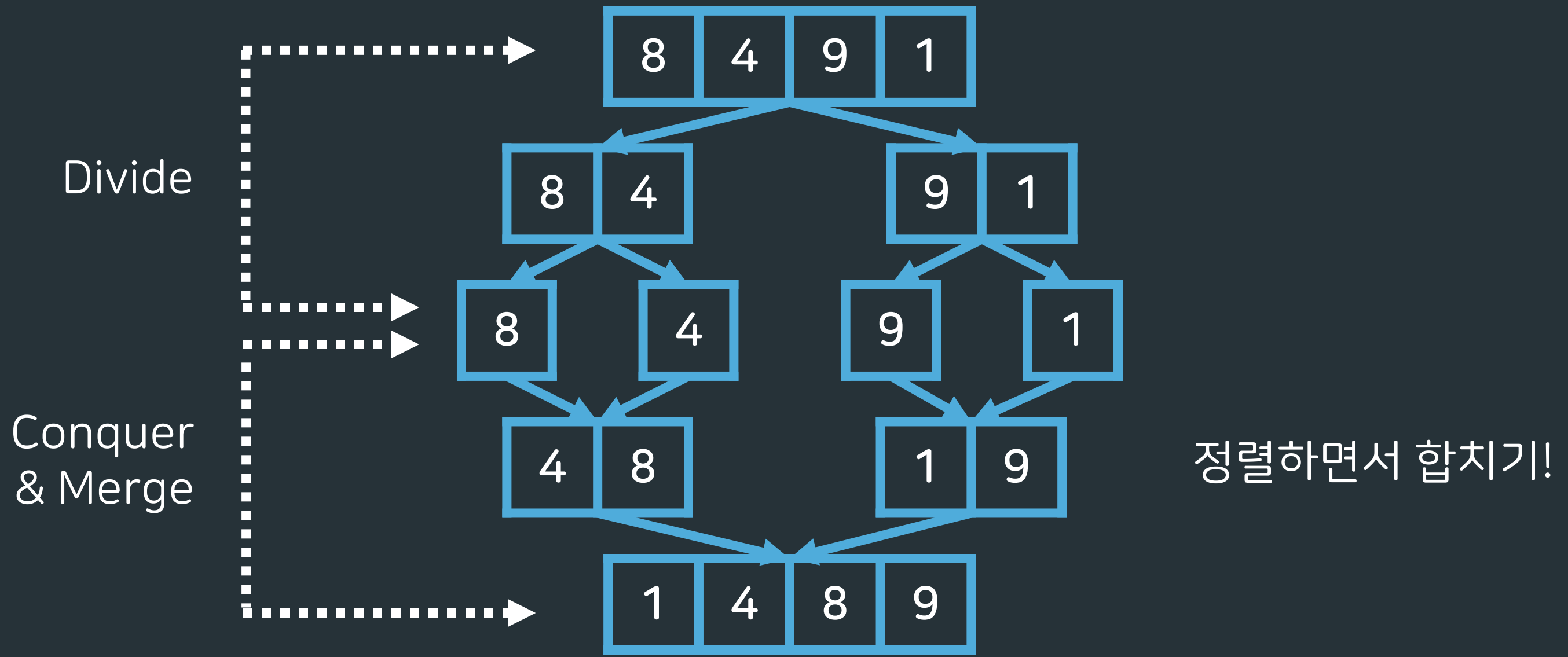
→ N의 범위가 최대 1,000이기 때문에  $O(n^2)$ 의 알고리즘이라도 시간초과가 발생하지 않음!

## Merge sort

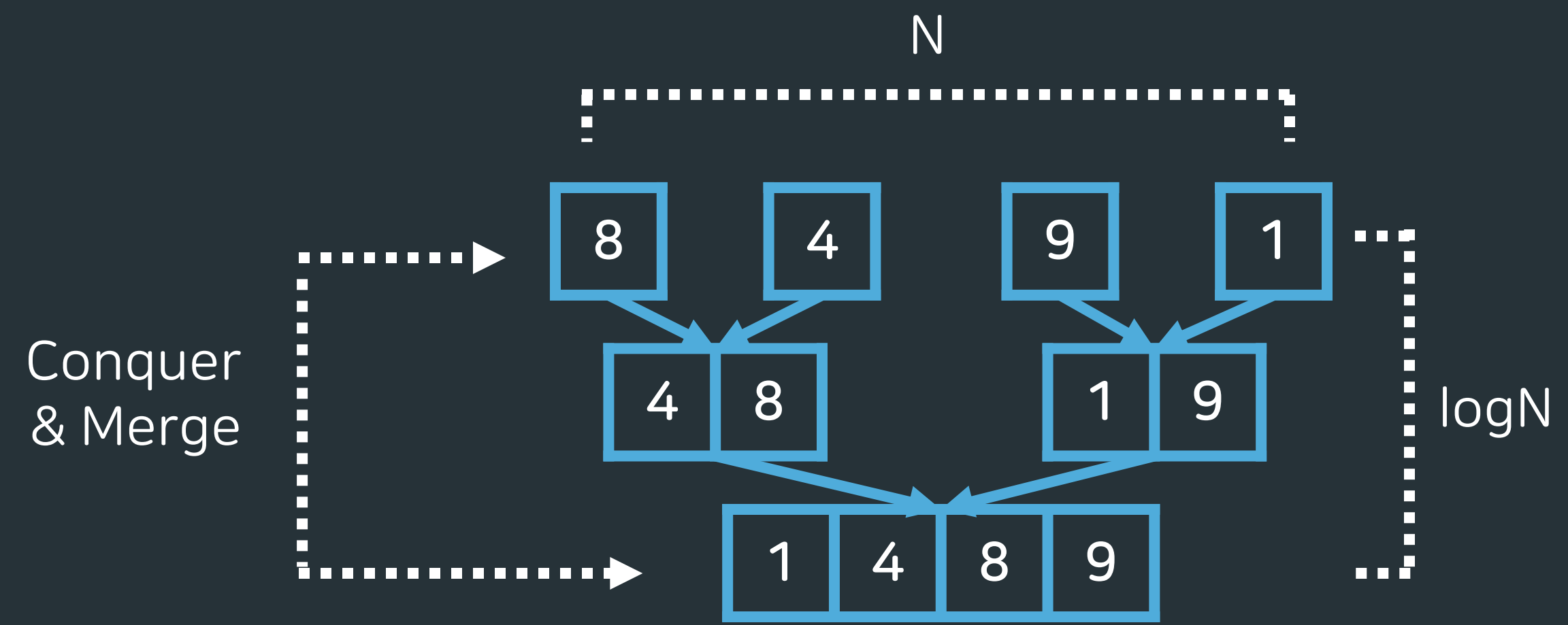
- 분할 정복(Divide and Conquer) 방식으로 설계된 알고리즘
- 하나의 배열을 정확히 반으로 나눔 (Divide)
- 나뉜 배열들을 정렬 (Conquer)
- 다시 하나의 배열로 합치기 (Merge)

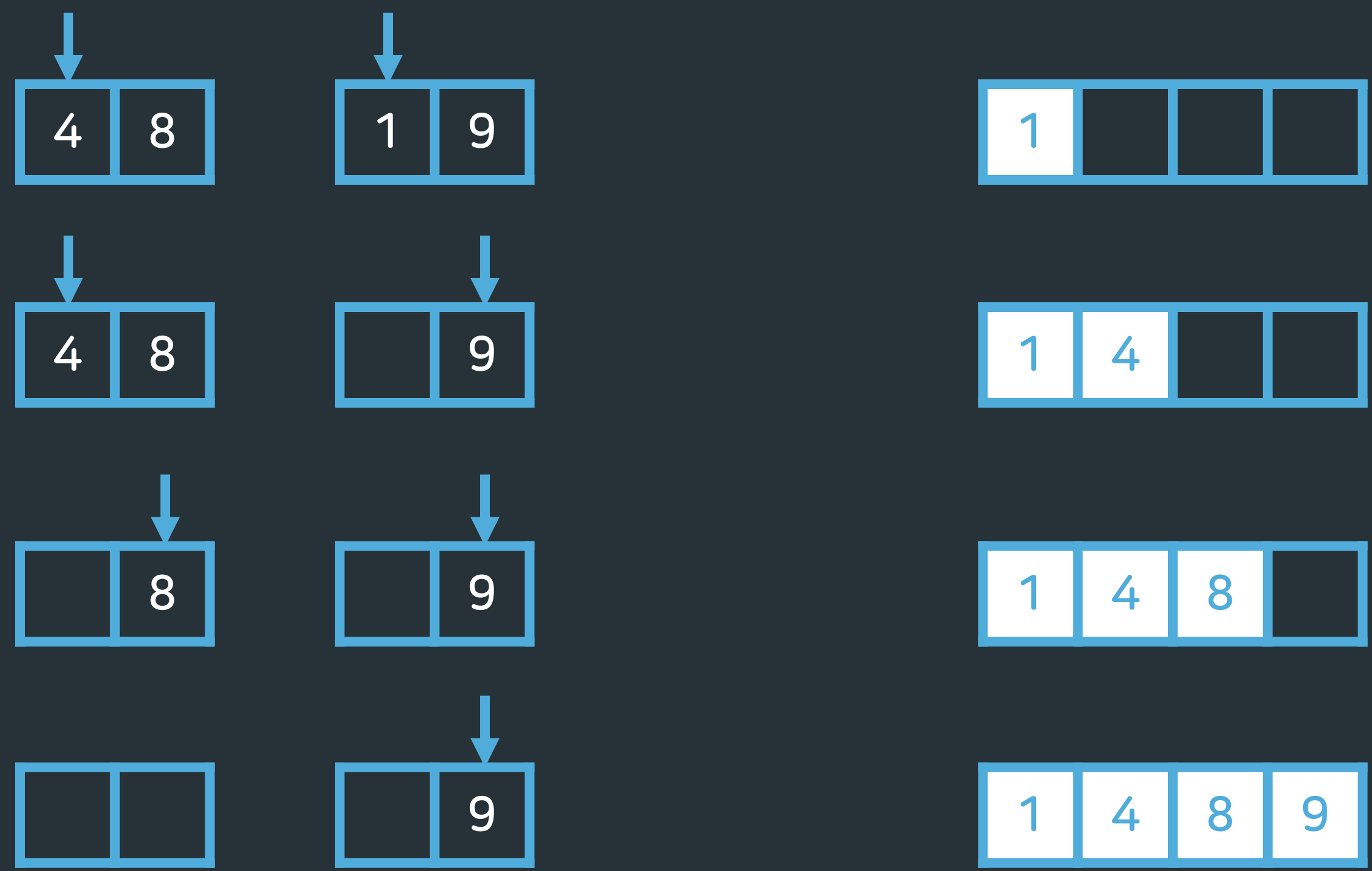
## 분할 정복

- 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 <algorithm>

**std::sort**

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);

**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!

`class list([iterable])`

Lists may be constructed in several ways:

- Using a pair of square brackets to denote the empty list: `[]`
- Using square brackets, separating items with commas: `[a], [a, b, c]`
- Using a list comprehension: `[x for x in iterable]`
- Using the type constructor: `list()` or `list(iterable)`

The constructor builds a list whose items are the same and in the same order as *iterable*'s items. *iterable* may be either a sequence, a container that supports iteration, or an iterator object. If *iterable* is already a list, a copy is made and returned, similar to `iterable[:]`. For example, `list('abc')` returns `['a', 'b', 'c']` and `list((1, 2, 3))` returns `[1, 2, 3]`. If no argument is given, the constructor creates a new empty list, `[]`.

Many other operations also produce lists, including the `sorted()` built-in.

Lists implement all of the **common** and **mutable** sequence operations. Lists also provide the following additional method:

`sort(*, key=None, reverse=False)`

This method **sorts** the list in place, using only  $\lt$  comparisons between items. Exceptions are not suppressed - if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

`sort()` accepts two arguments that can only be passed by keyword (**keyword-only arguments**):

**key** specifies a function of one argument that is used to extract a comparison key from each list element (for example, `key=str.lower`). The key corresponding to each item in the list is calculated once and then used for the entire **sorting** process. The default value of `None` means that list items are sorted directly without calculating a separate key value.

The `functools.cmp_to_key()` utility is available to convert a 2.x style *cmp* function to a *key* function.

**reverse** is a boolean value. If set to `True`, then the list elements are **sorted** as if each comparison were reversed.


This method modifies the sequence in place for economy of space when **sorting** a large sequence. To remind users that it operates by side effect, it does not return the **sorted** sequence (use `sorted()` to explicitly request a new sorted list instance).

The `sort()` method is guaranteed to be stable. A **sort** is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).


For **sorting** examples and a brief **sorting** tutorial, see [Sorting HOW TO](#).

**CPython implementation detail:** While a list is being **sorted**, the effect of attempting to mutate, or even inspect, the list is undefined. The C implementation of Python makes the list appear empty for the duration, and raises `ValueError` if it can detect that the list has been mutated during a sort.

## ● list.sort()는 리스트의 메소드

-  **list.sort** (Python method, in Built-in Types)

## ● sorted()는 리스트 등 iterable을 인자로 받는 함수

-  **sorted** (Python function, in Built-in Functions)

`sorted(iterable, /, *, key=None, reverse=False)`

Return a new **sorted** list from the items in *iterable*.

Has two optional arguments which must be specified as keyword arguments.

**key** specifies a function of one argument that is used to extract a comparison key from each element in *iterable* (for example, `key=str.lower`). The default value is `None` (compare the elements directly).

**reverse** is a boolean value. If set to `True`, then the list elements are **sorted** as if each comparison were reversed.

Use `functools.cmp_to_key()` to convert an old-style *cmp* function to a *key* function.

The built-in `sorted()` function is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

The sort algorithm uses only  $\lt$  comparisons between items. While defining an `__lt__()` method will suffice for sorting, **PEP 8** recommends that all six **rich comparisons** be implemented. This will help avoid bugs when using the same data with other ordering tools such as `max()` that rely on a different underlying method. Implementing all six comparisons also helps avoid confusion for mixed type comparisons which can call reflected the `__gt__()` method.

For sorting examples and a brief sorting tutorial, see [Sorting HOW TO](#).



## list.sort()

- default 값은 오름차순 정렬
- key값에 함수(람다함수)를 보내줘서 원하는 조건대로 정렬 가능!
- Key값이 정렬의 기준이 됨
- 내림차순 정렬은 reverse에 true를 보내주면 됨
- C++과 다르게 비교함수를 정의하는 것이 아니라 기준 값을 주어 정렬하는 것임을 주의!
- list.sort()는 리스트의 원본을 바꿔주는 리스트의 메소드
- sorted()는 정렬된 리스트를 반환하는 함수

```
a = ["pineapple", "banana", "mango", "kiwi"]  
a.sort()
```

기본 형식

```
a = ["pineapple", "banana", "mango", "kiwi"]  
a.sort(key=lambda x:len(x))
```

key 정렬 - 문자열 길이를 기준으로



## 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

```
arr = [['Junkyu', 50, 60, 100],  
        ['Sangkeun', 80, 60, 50],  
        ['Sunyoung', 80, 70, 100]  
        ...  
        ]
```

- 위와 같은 형태로 입력을 받은 상황에서는?

## 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

```
arr = [['Junkyu', 50, 60, 100],  
        ['Sangkeun', 80, 60, 50],  
        ['Sunyoung', 80, 70, 100]  
        ...  
        ]
```

```
arr.sort(key=lambda x:(-x[1], x[2], -x[3], x[0]))
```

- 정렬의 우선 순위를 그대로 담은 튜플(리스트)을 리턴하는 람다 함수를 key에 전달

## 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!

- 비교함수 작성 시 인자를 넘겨줄 때 왜 `const`와 `&`를 사용할까요?
- 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?

- 문자열로 들어오는 입력을 어떻게 이렇게 받을 수 있을까요?
- 2중 반복문을 쓰지 않고 처리할 수는 없을까?

```
arr = [['Junkyu', 50, 60, 100],  
       ['Sangkeun', 80, 60, 50],  
       ['Sunyoung', 80, 70, 100]  
       ...  
       ]
```

## Hint

- 입력을 받을 때 자주 쓰는 map() 함수...
- OT자료에서 본거 같은데...?

## 필수

- /<> 10804번 : 카드 역배치 - Bronze 2
- /<> 12840번 : 창용이의 시계 - Bronze 3

## 3문제 이상 선택

- /<> 11651번 : 좌표 정렬하기 2 - Silver 2
- /<> 1758번 : 알바생 강호 - Silver 4
- /<> 1431번 : 시리얼 번호 - Silver 3
- /<> 1946번 : 신입 사원 - Silver 1
- /<> 1026번 : 보물 - Silver 4