



고급 알고리즘

컴퓨터소프트웨어공학과 강은영

정렬 알고리즘

정렬 알고리즘

- 정렬(Sorting)이란 데이터를 특정한 기준에 따라 순서대로 나열하는 것을 말합니다.
- 일반적으로 문제 상황에 따라서 적절한 정렬 알고리즘이 공식처럼 사용됩니다.



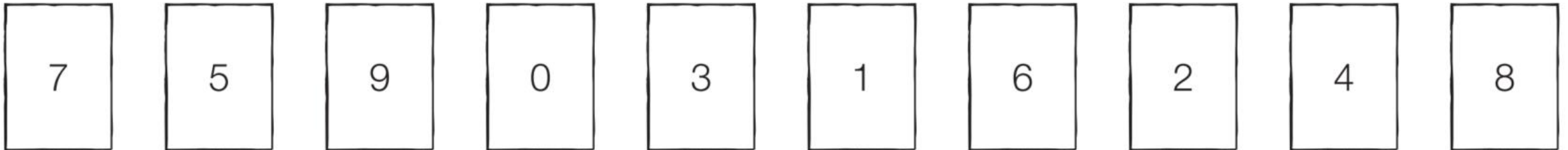
여러 개의 데이터(카드)를 어떻게 정렬할 수 있을까요?

선택 정렬

- 처리되지 않은 데이터 중에서 가장 작은 데이터를 **선택**해 맨 앞에 있는 데이터와 바꾸는 것을 반복합니다.

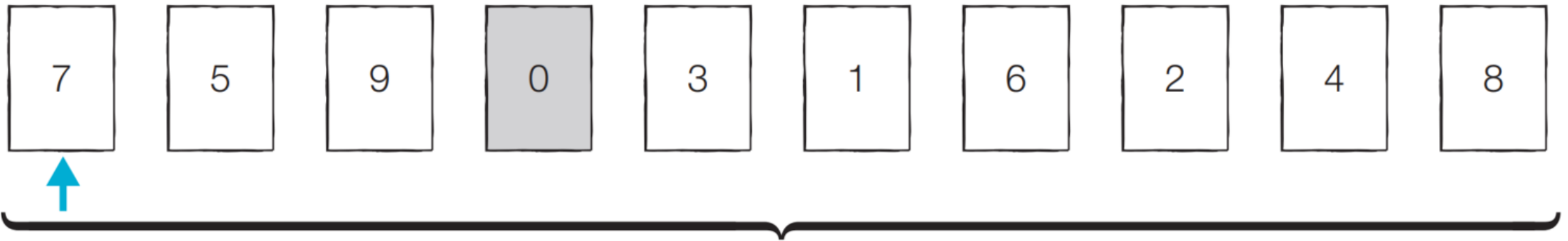
선택 정렬 동작 예시

- 정렬할 데이터를 준비합니다.



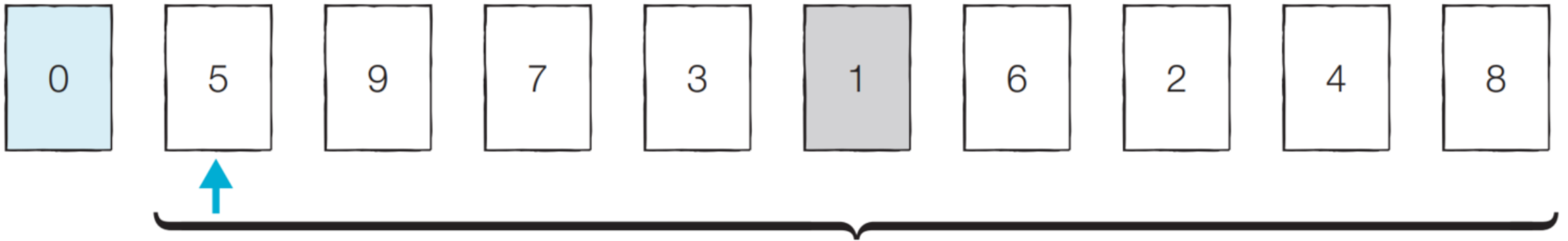
선택 정렬 동작 예시

- [Step 0] 처리되지 않은 데이터 중 가장 작은 '0' 을 선택해 가장 앞의 '7' 과 바꿉니다.



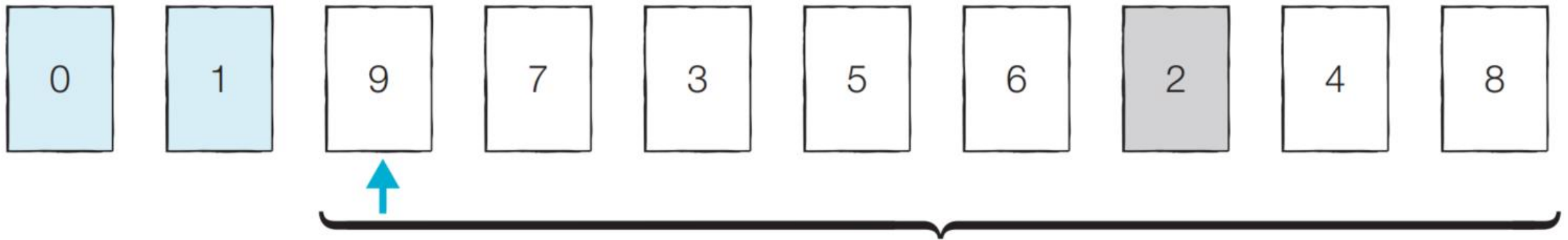
선택 정렬 동작 예시

- [Step 1] 처리되지 않은 데이터 중 가장 작은 '1' 을 선택해 가장 앞의 '5' 와 바꿉니다.



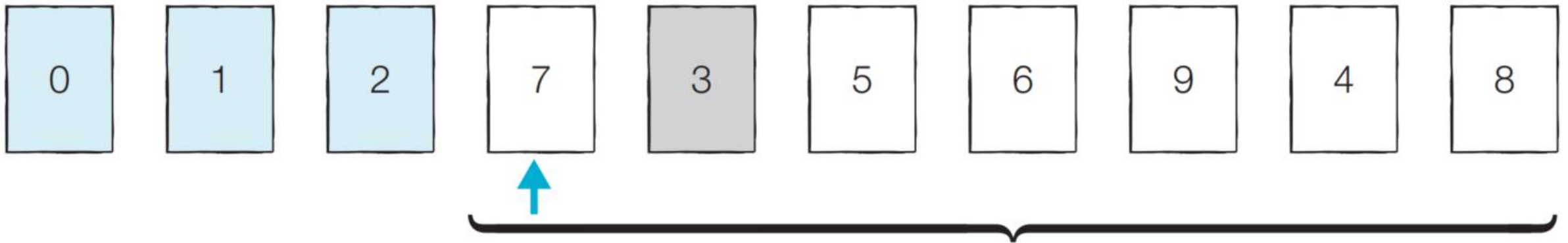
선택 정렬 동작 예시

- [Step 2] 처리되지 않은 데이터 중 가장 작은 '2' 를 선택해 가장 앞의 '9' 와 바꿉니다.



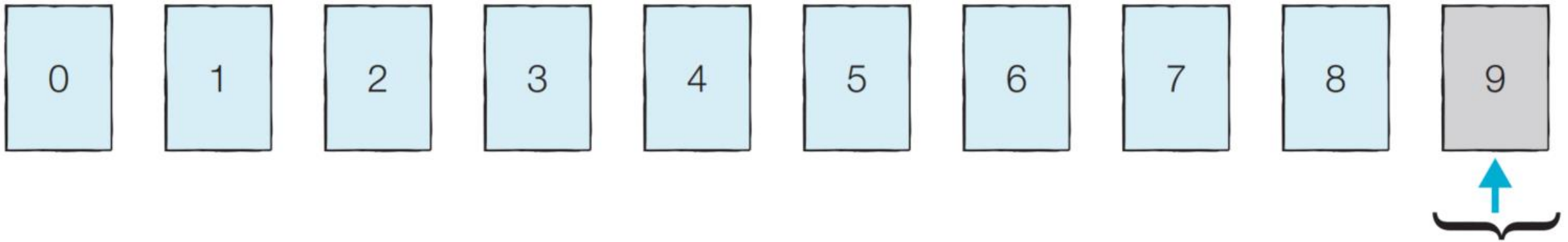
선택 정렬 동작 예시

- [Step 3] 처리되지 않은 데이터 중 가장 작은 '3' 을 선택해 가장 앞의 '7' 과 바꿉니다.



선택 정렬 동작 예시

- 이러한 과정을 반복하면 다음과 같이 정렬이 완료됩니다.



선택 정렬의 시간 복잡도

- 선택 정렬은 N번 만큼 가장 작은 수를 찾아서 맨 앞으로 보내야 합니다.
- 구현 방식에 따라서 사소한 오차는 있을 수 있지만, 전체 연산 횟수는 다음과 같습니다.

$$N + (N - 1) + (N - 2) + \dots + 2$$

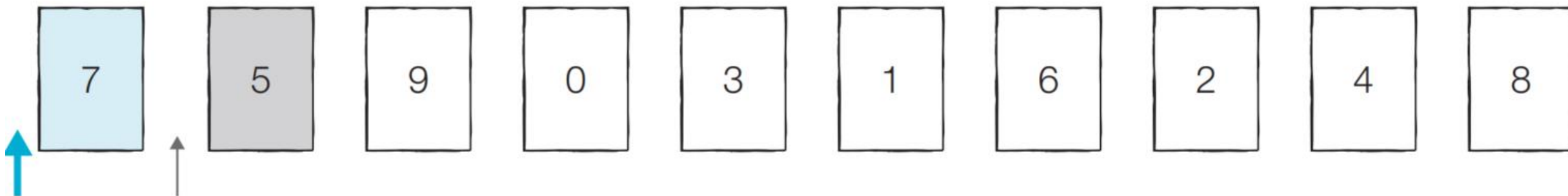
- 이는 $(N^2 + N - 2) / 2$ 로 표현할 수 있는데, 빅오 표기법에 따라서 **$O(N^2)$** 이라고 작성합니다🌀

삽입 정렬

- 처리되지 않은 데이터를 하나씩 골라 적절한 위치에 **삽입**합니다.
- 선택 정렬에 비해 구현 난이도가 높은 편이지만, 일반적으로 더 효율적으로 동작합니다.

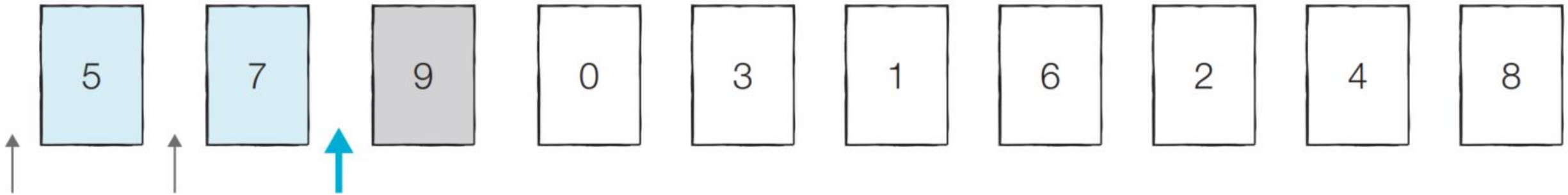
삽입 정렬 동작 예시

- [Step 0] 첫 번째 데이터 '7'은 그 자체로 정렬이 되어 있다고 판단하고, 두 번째 데이터인 '5'가 어떤 위치로 들어갈지 판단합니다. '7'의 왼쪽으로 들어가거나 오른쪽으로 들어가거나 두 경우만 존재합니다.



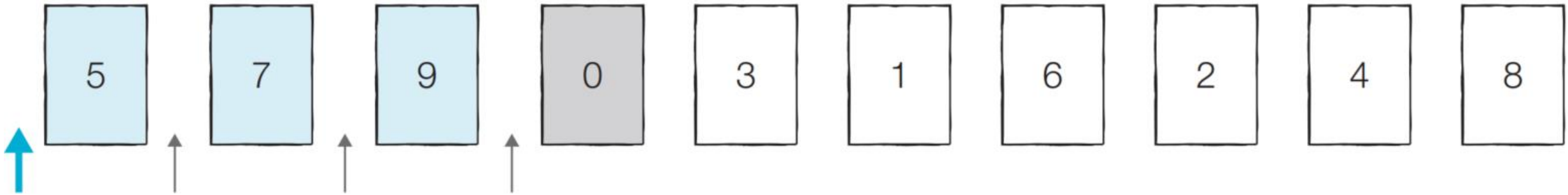
삽입 정렬 동작 예시

- [Step 1] 이어서 '9'가 어떤 위치로 들어갈지 판단합니다.



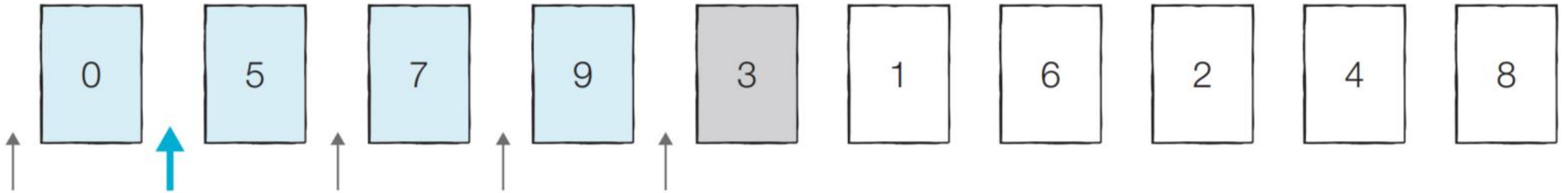
삽입 정렬 동작 예시

- [Step 2] 이어서 '0'이 어떤 위치로 들어갈지 판단합니다.



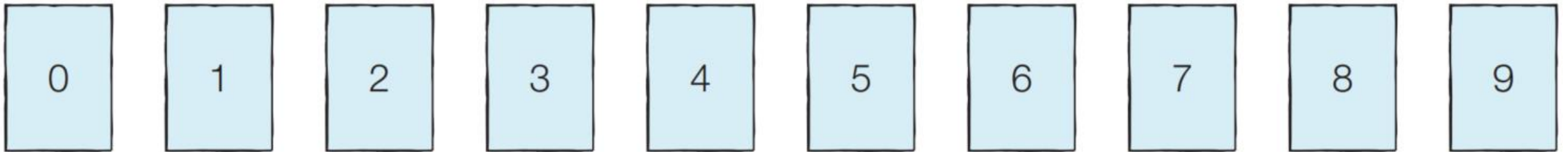
삽입 정렬 동작 예시

- [Step 3] 이어서 '3'이 어떤 위치로 들어갈지 판단합니다.



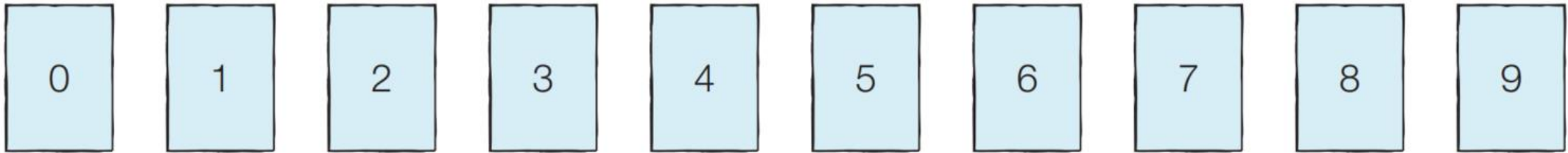
삽입 정렬 동작 예시

- 이러한 과정을 반복하면 다음과 같이 정렬이 완료됩니다.



삽입 정렬의 시간 복잡도

- 삽입 정렬의 시간 복잡도는 $O(N^2)$ 이며, 선택 정렬과 마찬가지로 반복문이 두 번 중첩되어 사용 됩니다.
- 삽입 정렬은 현재 리스트의 데이터가 거의 정렬되어 있는 상태라면 매우 빠르게 동작합니다.
 - 최선의 경우 $O(N)$ 의 시간 복잡도를 가집니다.
 - 이미 정렬되어 있는 상태에서 다시 삽입 정렬을 수행하면 어떻게 될까요?

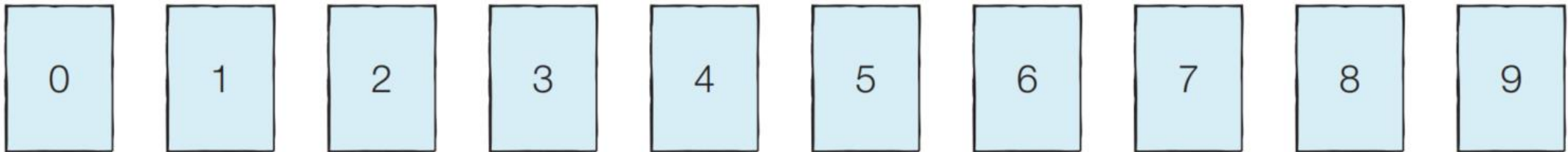


퀵 정렬

- 기준 데이터를 설정하고 그 기준보다 큰 데이터와 작은 데이터의 위치를 바꾸는 방법입니다.
- 일반적인 상황에서 가장 많이 사용되는 정렬 알고리즘 중 하나입니다.
- 병합 정렬과 더불어 대부분의 프로그래밍 언어의 정렬 라이브러리의 근간이 되는 알고리즘입니다.
- 가장 기본적인 퀵 정렬은 첫 번째 데이터를 기준 데이터(Pivot)로 설정합니다.

퀵 정렬의 시간 복잡도

- 퀵 정렬은 평균의 경우 $O(N \log N)$ 의 시간 복잡도를 가집니다.
- 하지만 최악의 경우 $O(N^2)$ 의 시간 복잡도를 가집니다.
 - 첫 번째 원소를 피벗으로 삼을 때, 이미 정렬된 배열에 대해서 퀵 정렬을 수행하면 어떻게 될까요?

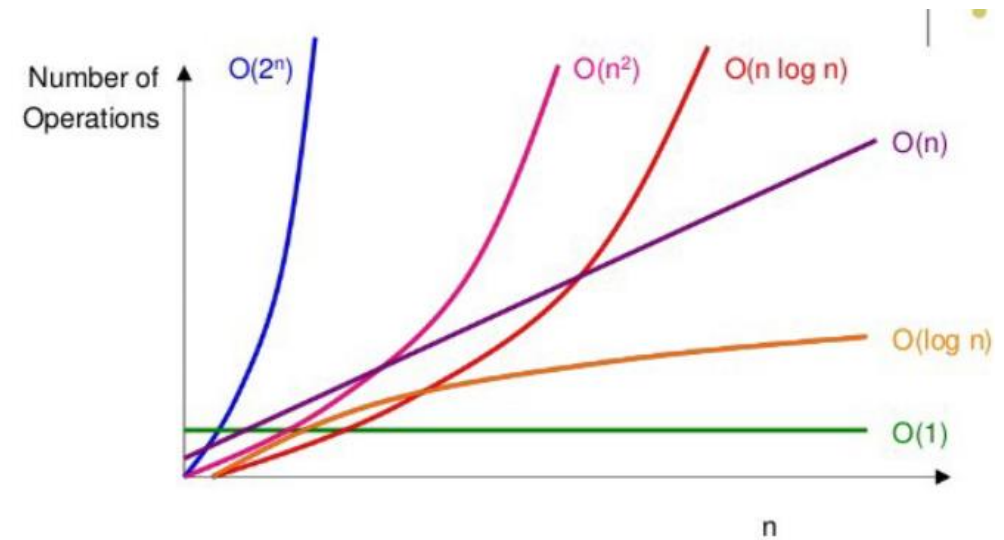


연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting)

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$
Bubble Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$
Heap Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$
Quick Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$
Merge Sort	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$



연결하여 만들 수 있는 가장 큰 수 찾기

(출처)
<https://www.geeksforgeeks.org/>

연결하여 만들 수 있는 가장 큰 수 찾기

숫자가 주어지면, 이를 연결하여 만들 수 있는 가장 큰 수를 찾아라. 각각의 숫자는 연결하여 만들 수 있는 가장 큰 숫자를 만들기 위해 서로 연결된다.

Find largest number possible from set of given numbers. The numbers should be appended to each other in any order to form the largest number.

(예) {2, 3, 4, 0, 6}의 숫자가 주어지면,
연결하여 만들 수 있는 가장 큰 수는 64320

입력	Input: {2, 3, 4, 0, 6}	
출력	Output: {2, 3, 4, 0, 6}의 숫자를 연결하여 만들 수 있는 가장 큰 수 64320	
입출력 예	입력 Input: {10, 68, 75, 7, 21, 12 }	출력 Output: 77568211210

연결하여 만들 수 있는 가장 큰 수 찾기

(출처)

<https://www.geeksforgeeks.org/>

연결하여 만들 수 있는 가장 큰 수 찾기

숫자가 주어지면, 이를 연결하여 만들 수 있는 가장 큰 수를 찾아라. 각각의 숫자는 연결하여 만들 수 있는 가장 큰 숫자를 만들기 위해 서로 연결된다.

Find largest number possible from set of given numbers. The numbers should be appended to each other in any order to form the largest number.

(예) {2, 3, 4, 0, 6}의 숫자가 주어지면, 연결하여 만들 수 있는 가장 큰 수는 64320

1 입력 데이터를 내림차순으로 정렬하고, 서로 연결하여 수를 구성

- (입력) { 2, 3, 4, 0, 6 }
- (정렬) { 6, 4, 3, 2, 0 }
- (연결) 64320

- (입력) { 10, 68, 75, 7, 21, 12 }
- (정렬) { 75, 68, 21, 12, 10, 7 }
- (연결) 75682112107

정렬?

{ 7, 75, 68, 21, 12, 10 }

77568211210



연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (1)

크기의 비교

arr

10	68	75	7	21	12
----	----	----	---	----	----



정수 값 정렬 ?

arr

75	68	21	12	10	7
----	----	----	----	----	---

?

{ 7, 75, 68, 21, 12, 10 }

arr

"10"	"68"	"75"	"7"	"21"	"12"
------	------	------	-----	------	------



문자열 정렬 ?

arr

"75"	"7"	"68"	"21"	"12"	"10"
------	-----	------	------	------	------

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (1)

크기의 비교

arr

10	68	75	7	21	12
----	----	----	---	----	----



정수 값 정렬?

arr

75	68	21	12	10	7
----	----	----	----	----	---

arr

"10"	"68"	"75"	"7"	"21"	"12"
------	------	------	-----	------	------



문자열 정렬?

arr

"75"	"7"	"68"	"21"	"12"	"10"
------	-----	------	------	------	------

?

{ 7, 75, 68, 21, 12, 10 }

"7" > "75"

"7" < "75"

?

"775" > "757"



"7" > "75"

비교함수

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (1)

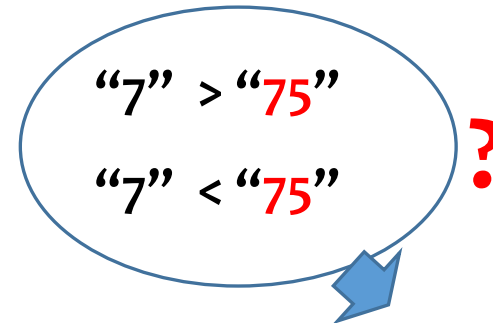
크기의 비교

값의 정렬을 위한 수의 크기 비교

“xy”와 “ab” 중 누가 큰가?



If “xyab” > “abxy”
then “xy” > “ab”
else “ab” > “xy”



“775” > “757”



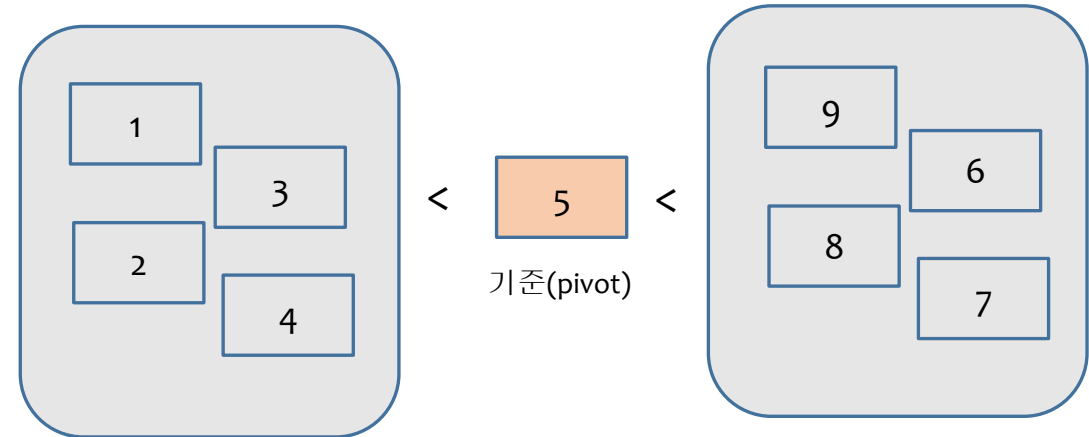
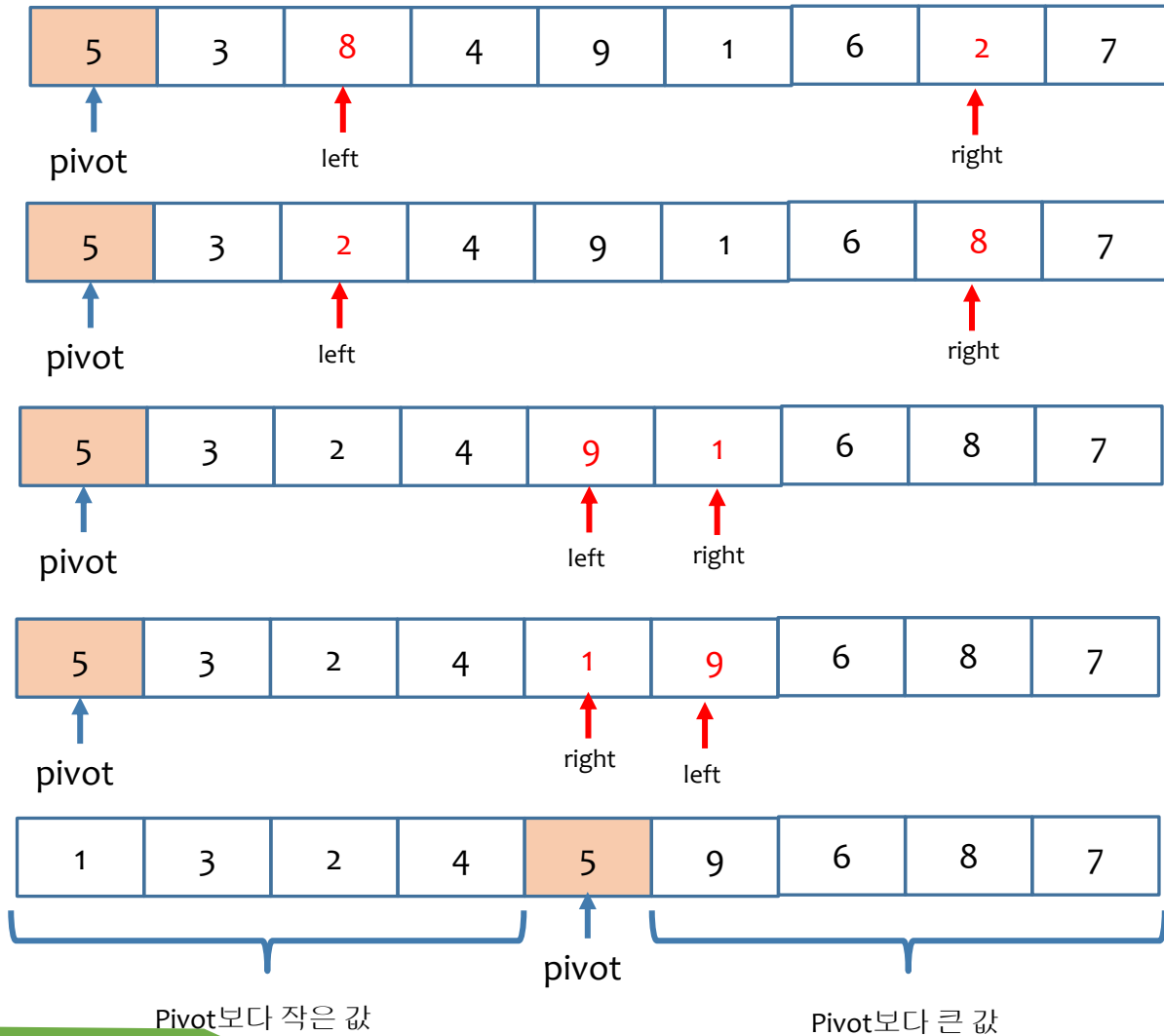
“7” > “75”

} 비교함수

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting) Quick sort()



연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting) **qsort() in C**

정렬할 배열의 시작 주소 정렬할 배열 항목의 수

```
void qsort( void *base, size_t num, size_t size, int(*comparator)(const void*, const void*))
```

정렬할 배열 항목의 크기 정렬할 배열 항목을 비교할 function

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

int compare(const void *num1, const void *num2)
{
    return (*(int *)num1 - *(int *)num2);
}
```

```
int main(void) {
    int arr[] = {1, 3, 2, 6, 4, 9, 5, 8};
    int n = sizeof(arr)/sizeof(arr[0]);

    qsort(arr, n, sizeof(arr[0]), compare);

    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```



1 2 3 4 5 6 8 9

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting) **qsort() in C**

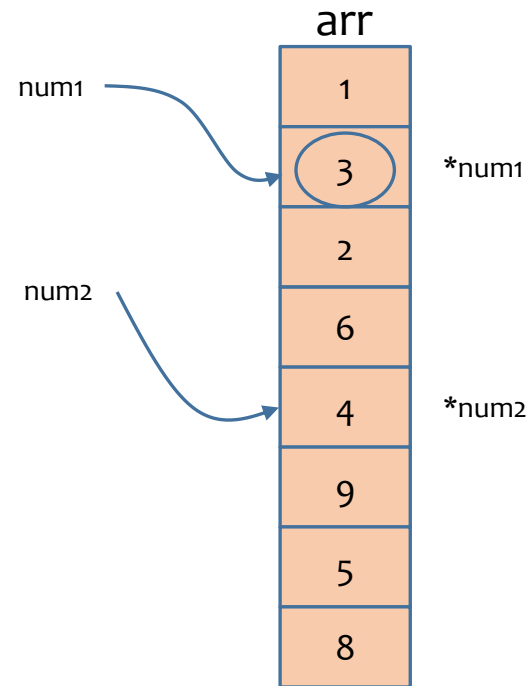
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int compare(const void *num1, const void *num2)
5 {
6     return (*(int *)num1 - *(int *)num2);
7 }
8
9 int main(void) {
10     int arr[] = {1, 3, 2, 6, 4, 9, 5, 8};
11     int n = sizeof(arr)/sizeof(arr[0]);
12
13     qsort(arr, n, sizeof(arr[0]), compare);
14
15     for(int i = 0; i < n; i++)
16         printf("%d ", arr[i]);
17     return 0;
18 }
19
```

Input Goes Here..

Time(sec) : 0

Output:

1 2 3 4 5 6 8 9



```
int compare(const void *num1, const void *num2)
{
    return (*(int *)num1 - *(int *)num2);
}
```

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting) **qsort() in C**

정렬할 배열의 시작 주소 정렬할 배열 항목의 수

```
void qsort( void *base, size_t num, size_t size, int(*comparator)(const void*, const void*))
```

정렬할 배열 항목의 크기 정렬할 배열 항목을 비교할 function

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int compare(const void *str1, const void *str2)
{
    return (strcmp(*(char **)str1, *(char **)str2));
}
```

```
int main(void) {
    char *arr[] = {"11", "33", "22", "66", "44", "99", "55", "88"};
    int n = sizeof(arr)/sizeof(arr[0]);

    qsort(arr, n, sizeof(arr[0]), compare);

    for(int i = 0; i < n; i++)
        printf("%s ", arr[i]);

    return 0;
}
```

11 22 33 44 55 66 88 99

연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2)

정렬 (sorting) **qsort() in C**

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int compare(const void *str1, const void *str2)
6 {
7
8     return (strcmp(*(char **)str1, *(char **)str2));
9 }
10
11 int main(void) {
12     char *arr[] = {"11", "33", "22", "66", "44", "99", "55", "88" };
13     int n = sizeof(arr)/sizeof(arr[0]);
14
15     qsort(arr, n, sizeof(arr[0]), compare);
16
17     for(int i = 0; i < n; i++)
18         printf("%s ", arr[i]);
19     return 0;
20 }
21
```

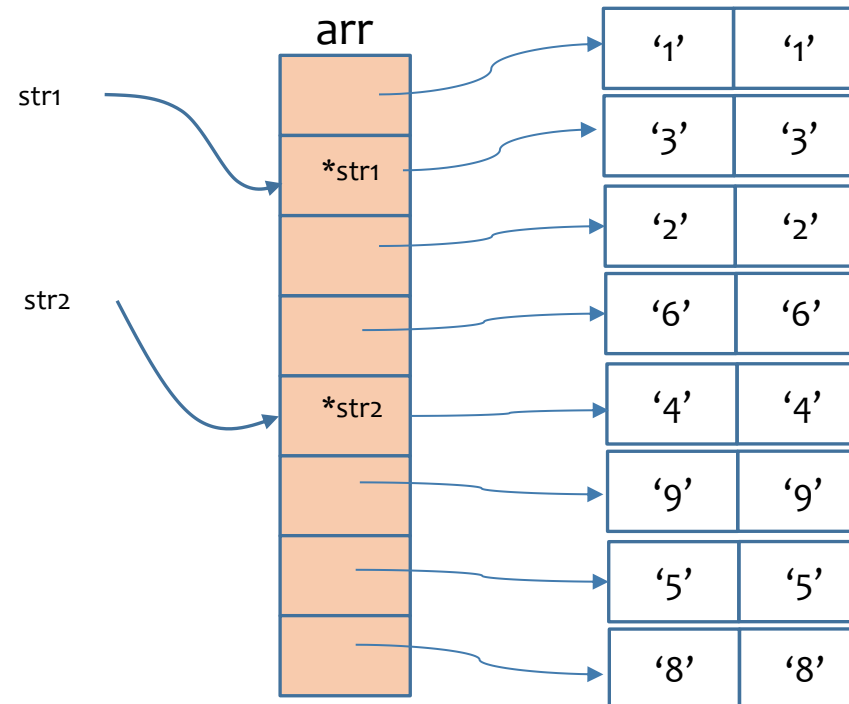
Input Goes Here..

Time(sec) : 0

Output:

11 22 33 44 55 66 88 99

```
int compare(const void *str1, const void *str2)
{
    return (strcmp(*(char **)str1, *(char **)str2));
}
```



연결하여 만들 수 있는 가장 큰 수 찾기

문제해결

연결하여 만들 수 있는 가장 큰 수 찾기

입출력	입력	출력
	Input: {10, 68, 75, 7, 21, 12}	Output: 77568211210

Quick Sort



"7" > "75"

"7" < "75"

?

"775" > "757"



"7" > "75"

비교함수

연결하여 만들 수 있는 가장 큰 수 찾기

문제해결

연결하여 만들 수 있는 가장 큰 수 찾기

입출력

입력

Input: {10, 68, 75, 7, 21, 12}

출력

Output: 77568211210

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int compare(const void *a, const void *b)
{
    const char **X = (const char **)a;
    const char **Y = (const char **)b;

    int len = strlen(*X) + strlen(*Y) + 1;

    char XY[len];
    strcpy(XY, *X);
    strcat(XY, *Y);

    char YX[len];
    strcpy(YX, *Y);
    strcat(YX, *X);

    return strcmp(YX, XY);
}
```

```
int main(void)
{
    char *arr[] = { "10", "68", "75", "7", "21", "12" };
    int n = sizeof(arr)/sizeof(arr[0]);

    qsort(arr, n, sizeof(arr[0]), compare);

    for (int i = 0; i < n; i++)
        printf("%s", arr[i]);

    return 0;
}
```

“xy”와 “ab” 중 누가 큰가?



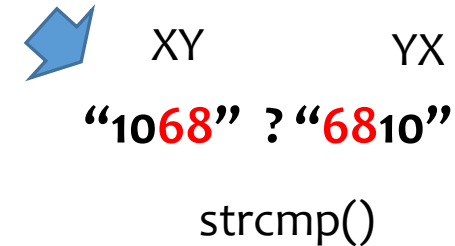
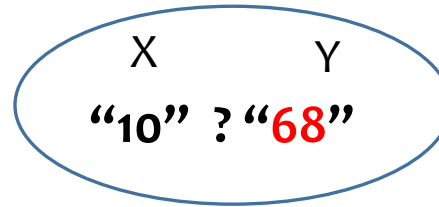
If “xyab” > “abxy”
then “xy” > “ab”
else “ab” > “xy”

연결하여 만들 수 있는 가장 큰 수 찾기(c언어예시)

배경지식 (2)

정렬 (sorting) **qsort() in C**

```
3 #include <stdlib.h>
4
5 int compare(const void* a, const void* b)
6 {
7     const char** X = (const char**)a;
8     const char** Y = (const char**)b;
9     int len = strlen(*X) + strlen(*Y) + 1;
10    char XY[len];
11    strcpy(XY, *X);
12    strcat(XY, *Y);
13    char YX[len];
14    strcpy(YX, *Y);
15    strcat(YX, *X);
16    return strcmp(YX, XY);
17 }
18
19 int main(void)
20 {
21     char* arr[] = { "10", "68", "75", "7", "21", "12" };
22     int n = sizeof(arr) / sizeof(arr[0]);
23     qsort(arr, n, sizeof(arr[0]), compare);
24     for (int i = 0; i < n; i++)
25     {
26         printf("%s", arr[i]);
27     }
28 }
```



Time(sec) : 0

Memory(MB) : 1.4910809414673

Output:

77568211210

연결하여 만들 수 있는 가장 큰 수 찾기(Java 예시)

문제해결

연결하여 만들 수 있는 가장 큰 수 찾기

입출력	<u>입력</u> Input: {10, 68, 75, 7, 21, 12 }	<u>출력</u> Output: 77568211210
-----	--	----------------------------------

연결하여 만들 수 있는 가장 큰 수 찾기(Python join함수)

- 함수의 모양
- `''.join(리스트)`
- `'구분자'.join(리스트)`
- join 함수는 매개변수로 들어온 리스트에 있는 요소 하나하나를 합쳐서 하나의 문자열로 바꾸어 반환하는 함수입니다.
- `''.join(리스트)`
`''.join(리스트)`를 이용하면 매개변수로 들어온 `['a', 'b', 'c']` 이런 식의 리스트를 `'abc'`의 문자열로 합쳐서 반환해주는 함수인 것입니다.
- `'구분자'.join(리스트)`
`'구분자'.join(리스트)`를 이용하면 리스트의 값과 값 사이에 '구분자'에 들어온 구분자를 넣어서 하나의 문자열로 합쳐줍니다.
`'_'.join(['a', 'b', 'c'])` 라 하면 `"a_b_c"` 와 같은 형태로 문자열을 만들어서 반환해 줍니다.

연결하여 만들 수 있는 가장 큰 수 찾기(Python예시)

문제해결

연결하여 만들 수 있는 가장 큰 수 찾기

입출력	<u>입력</u> Input: {10, 68, 75, 7, 21, 12 }	<u>출력</u> Output: 77568211210
-----	--	----------------------------------