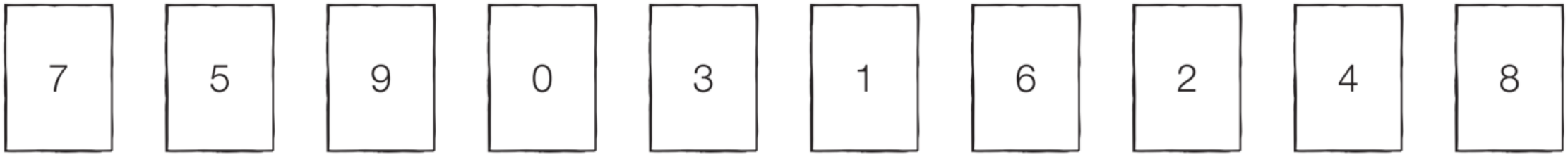
고급 알고리즘

컴퓨터소프트웨어공학과 강 은 영

정렬 알고리즘

정렬 알고리즘

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



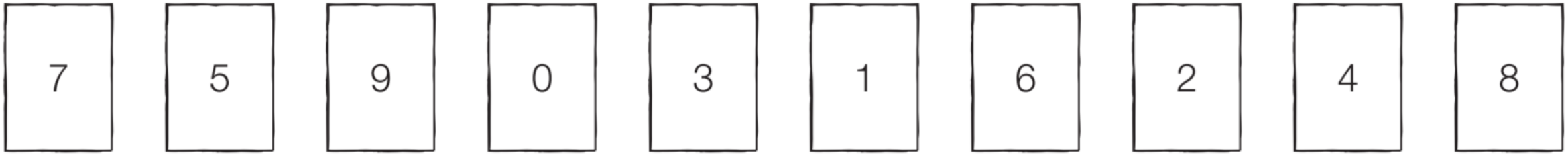
여러 개의 데이터(카드)를 어떻게 정렬할 수 있을까요?

선택 정렬

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

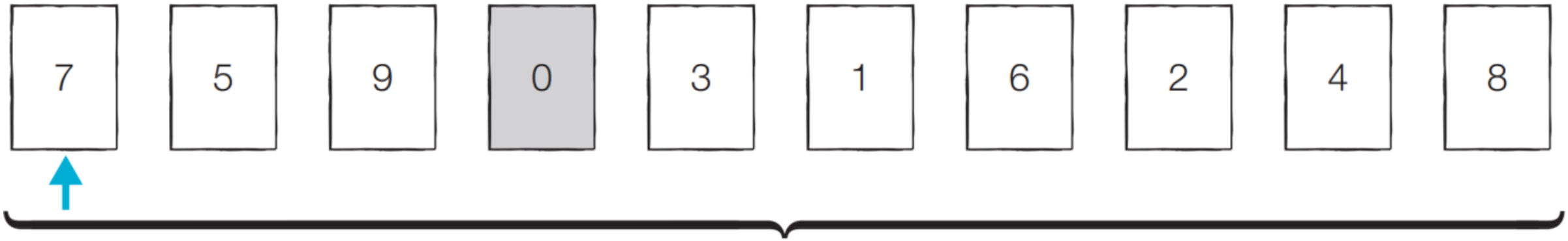
선택 정렬 동작 예시

- 정렬할 데이터를 준비합니다.



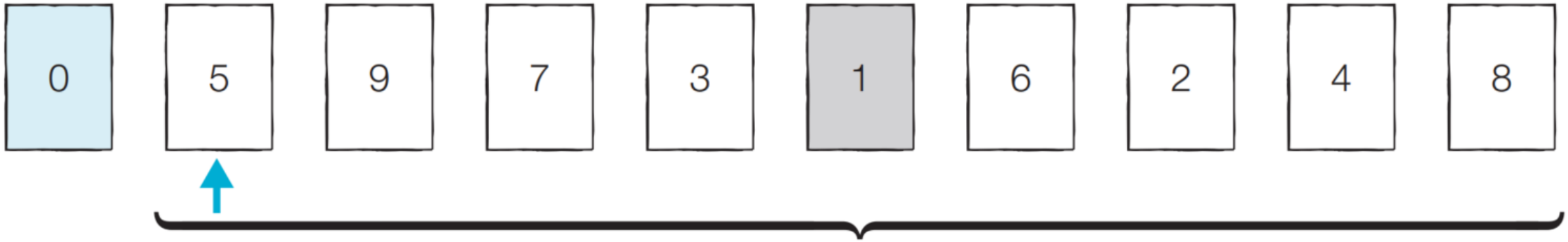
선택 정렬 동작 예시

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



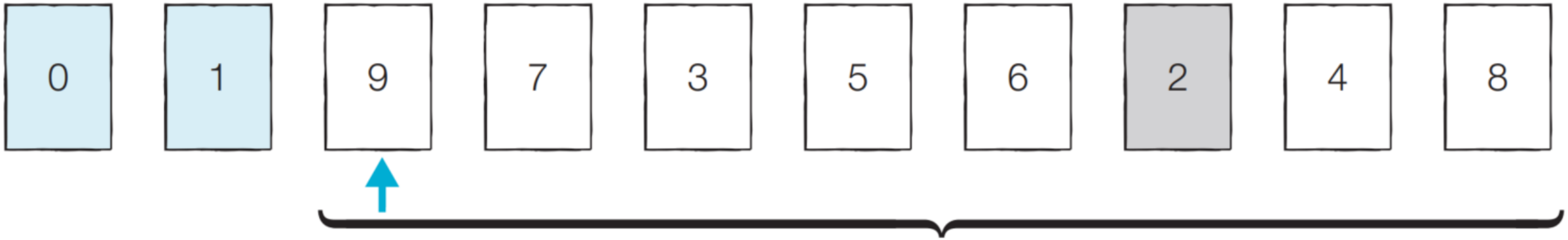
선택 정렬 동작 예시

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



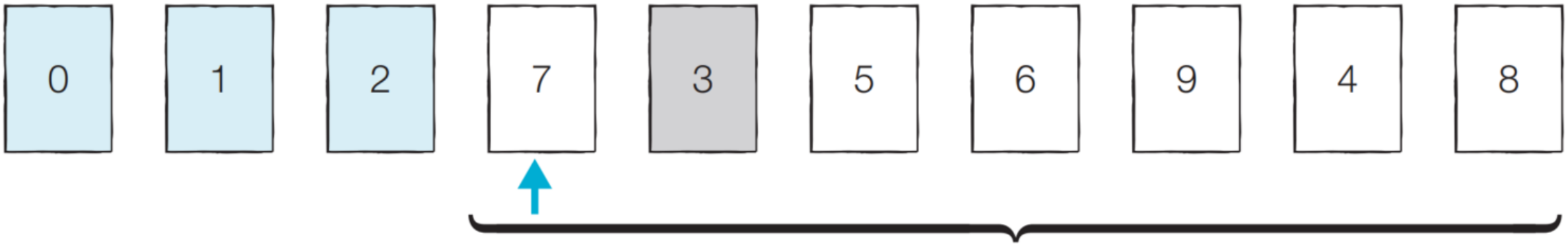
선택 정렬 동작 예시

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



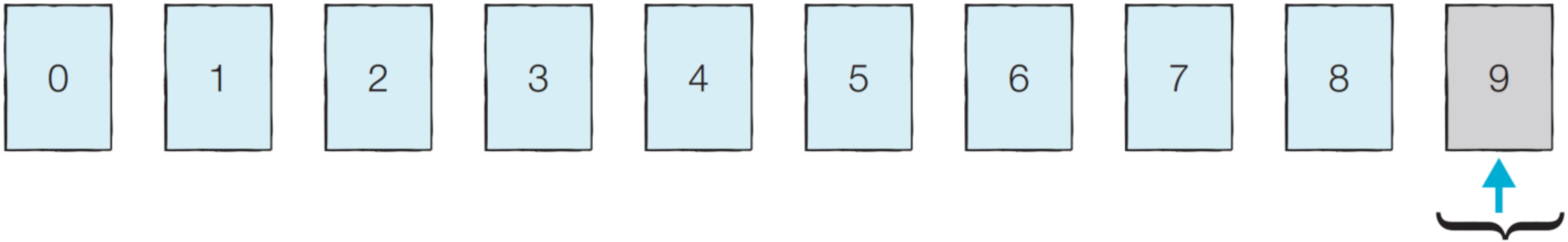
선택 정렬 동작 예시

- [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.

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

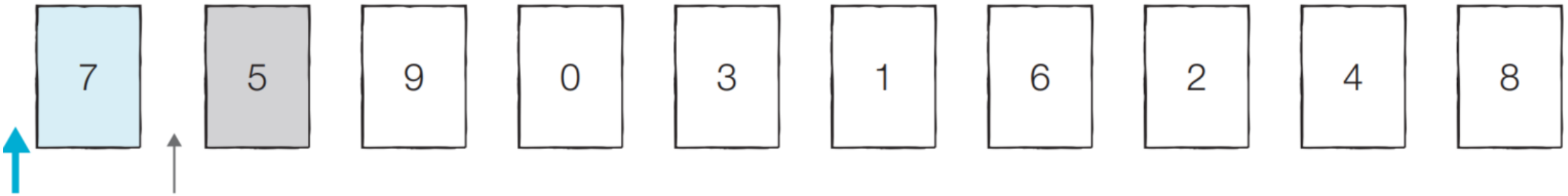
- 이는 $(N^2 + N - 2) / 2$ 로 표현할 수 있는데, 빅오 표기법에 따라서 $\boldsymbol{O(N^2)}$이라고 작성합니다.

삽입 정렬

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

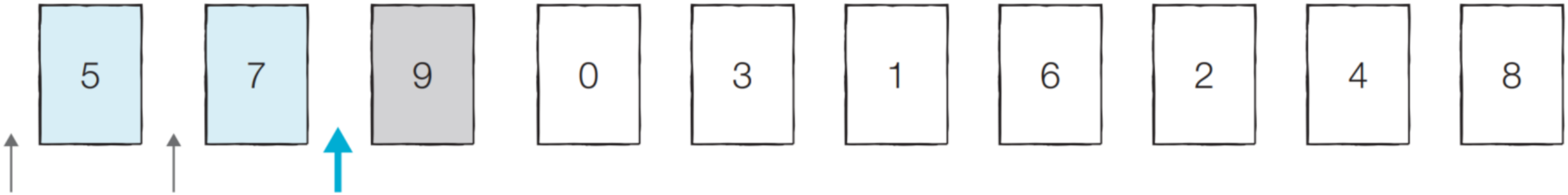
삽입 정렬 동작 예시

- [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.



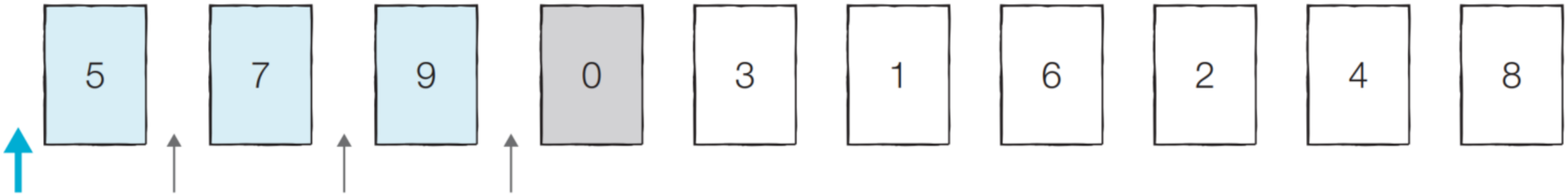
삽입 정렬 동작 예시

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



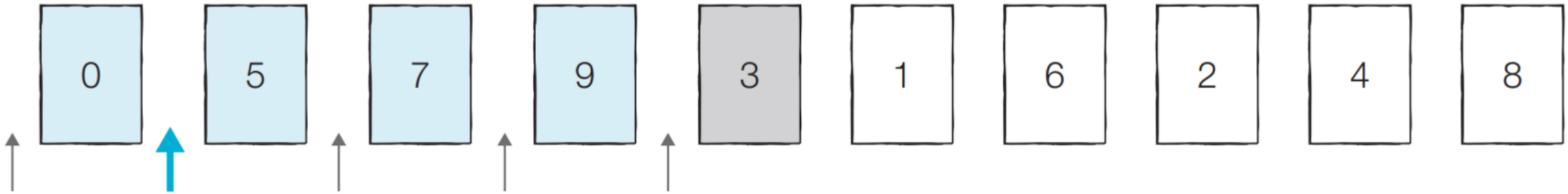
삽입 정렬 동작 예시

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



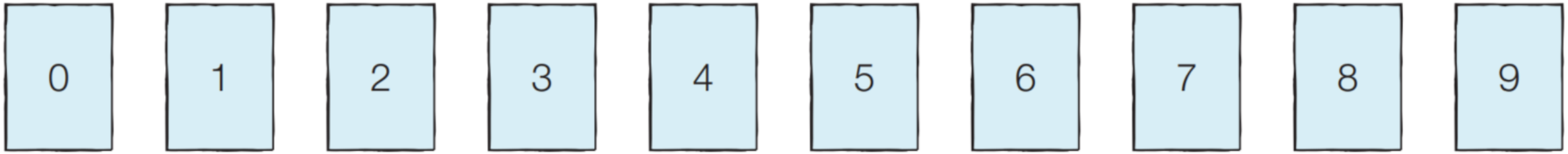
삽입 정렬 동작 예시

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



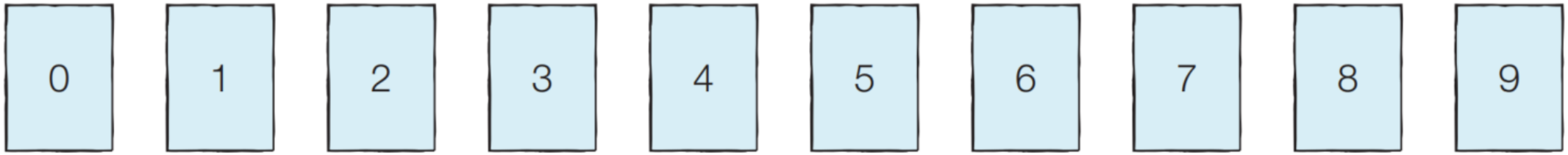
삽입 정렬 동작 예시

- 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 사용됩니다.
- 삽입 정렬은 현재 리스트의 데이터가 거의 정렬되어 있는 상태라면 매우 빠르게 동작합니다.
  - 최선의 경우 $O(N)$의 시간 복잡도를 가집니다.



퀵 정렬

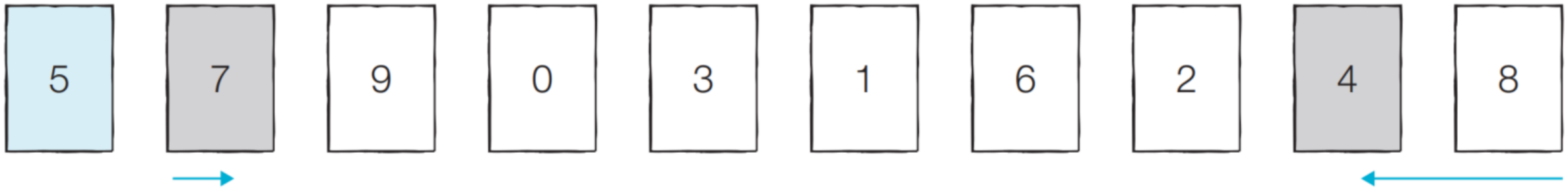
- 기준 데이터를 설정하고 그 기준보다 큰 데이터와 작은 데이터의 위치를 바꾸는 방법입니다.
- 일반적인 상황에서 가장 많이 사용되는 정렬 알고리즘 중 하나입니다.
- 병합 정렬과 더불어 대부분의 프로그래밍 언어의 정렬 라이브러리의 근간이 되는 알고리즘입니다.
- 가장 기본적인 퀵 정렬은 첫 번째 데이터를 기준 데이터(Pivot)로 설정합니다.

퀵 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.

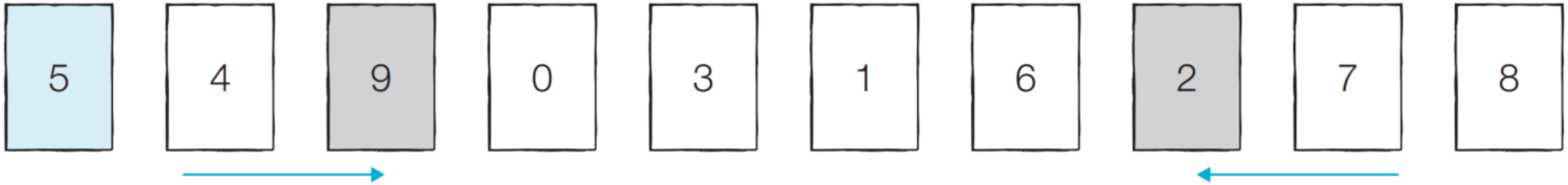
퀵 정렬 동작 예시

- [Step 0] 현재 피벗의 값은 '5'입니다. 왼쪽에서부터 '5'보다 큰 데이터를 선택하므로 '7'이 선택되고, 오른쪽에서부터 '5'보다 작은 데이터를 선택하므로 '4'가 선택됩니다. 이제 이 두 데이터의 위치를 서로 변경합니다.



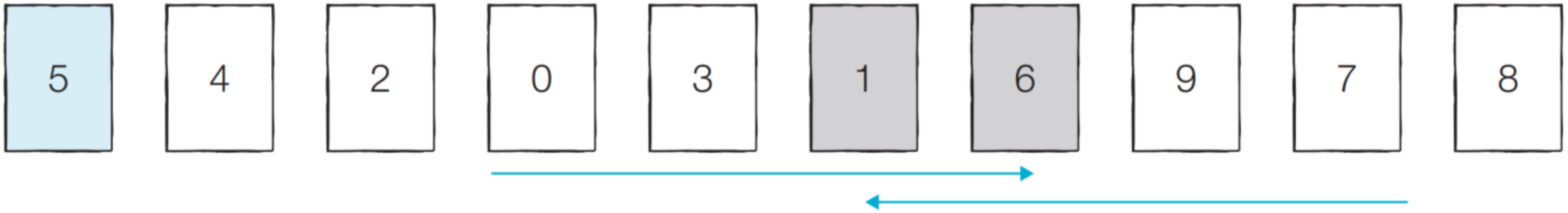
퀵 정렬 동작 예시

- [Step 1] 현재 피벗의 값은 '5'입니다. 왼쪽에서부터 '5'보다 큰 데이터를 선택하므로 '9'가 선택되고, 오른쪽에서부터 '5'보다 작은 데이터를 선택하므로 '2'가 선택됩니다. 이제 이 두 데이터의 위치를 서로 변경합니다.



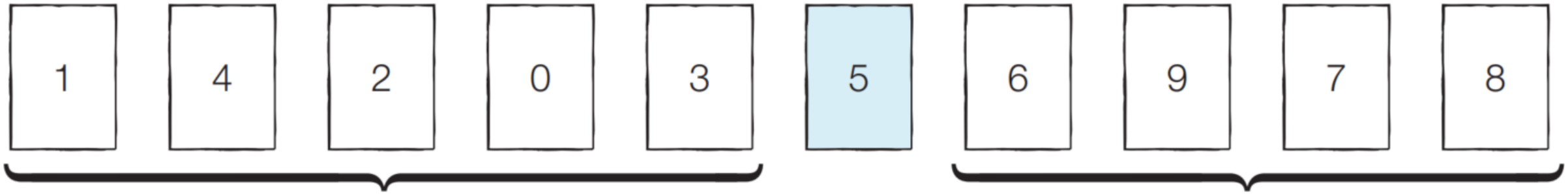
퀵 정렬 동작 예시

- [Step 2] 현재 피벗의 값은 '5'입니다. 왼쪽에서부터 '5'보다 큰 데이터를 선택하므로 '6'이 선택되고, 오른쪽에서부터 '5'보다 작은 데이터를 선택하므로 '1'이 선택됩니다. 단, 이처럼 위치가 엇갈리는 경우 '피벗'과 '작은 데이터'의 위치를 서로 변경합니다.



퀵 정렬 동작 예시

- [분할 완료] 이제 '5'의 왼쪽에 있는 데이터는 모두 5보다 작고, 오른쪽에 있는 데이터는 모두 '5'보다 크다는 특징이 있습니다. 이렇게 피벗을 기준으로 데이터 묶음을 나누는 작업을 분할(Divide)이라고 합니다.

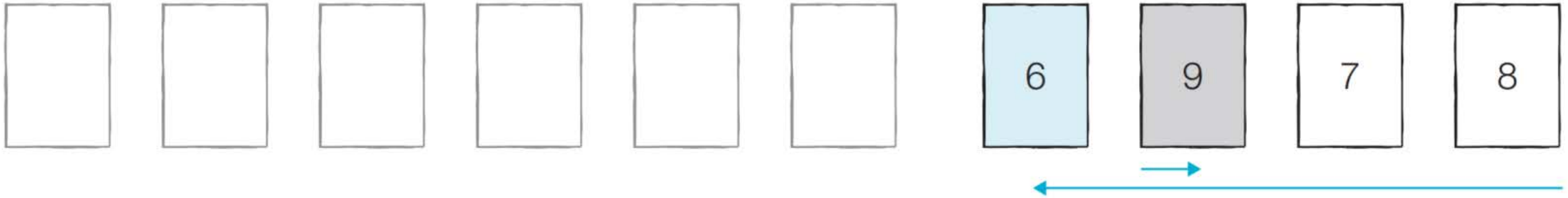


퀵 정렬 동작 예시

- [왼쪽 데이터 묶음 정렬] 왼쪽에 있는 데이터에 대해서 마찬가지로 정렬을 수행합니다.

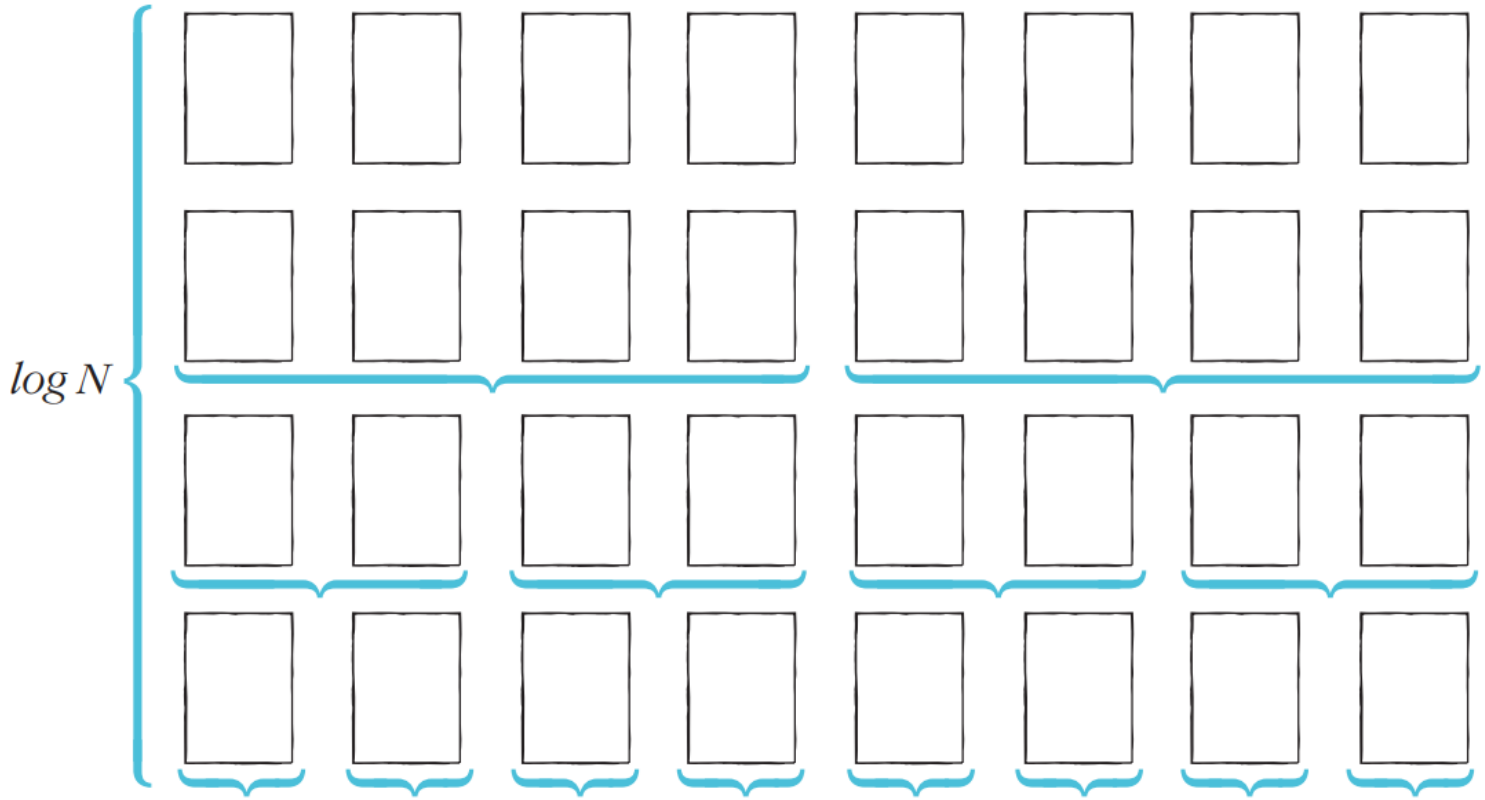
퀵 정렬 동작 예시

- [오른쪽 데이터 묶음 정렬] 오른쪽에 있는 데이터에 대해서 마찬가지로 정렬을 수행합니다.
  - 이러한 과정을 반복하면 전체 데이터에 대해서 정렬이 수행됩니다.



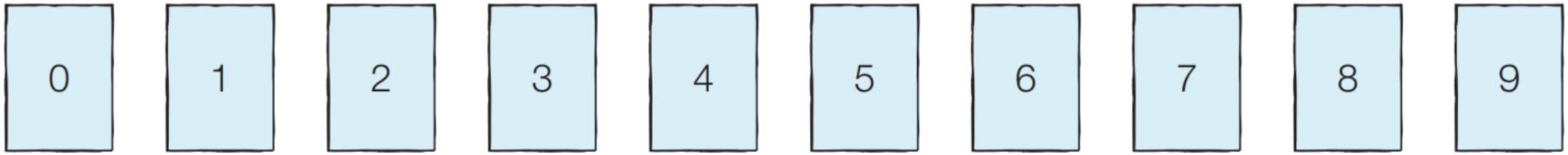
퀵 정렬이 빠른 이유: 직관적인 이해

- 이상적인 경우 분할이 절반씩 일어난다면 전체 연산 횟수로 $O(NlogN)$를 기대할 수 있습니다.
  - 너비 X 높이 = $N \times logN = NlogN$



퀵 정렬의 시간 복잡도

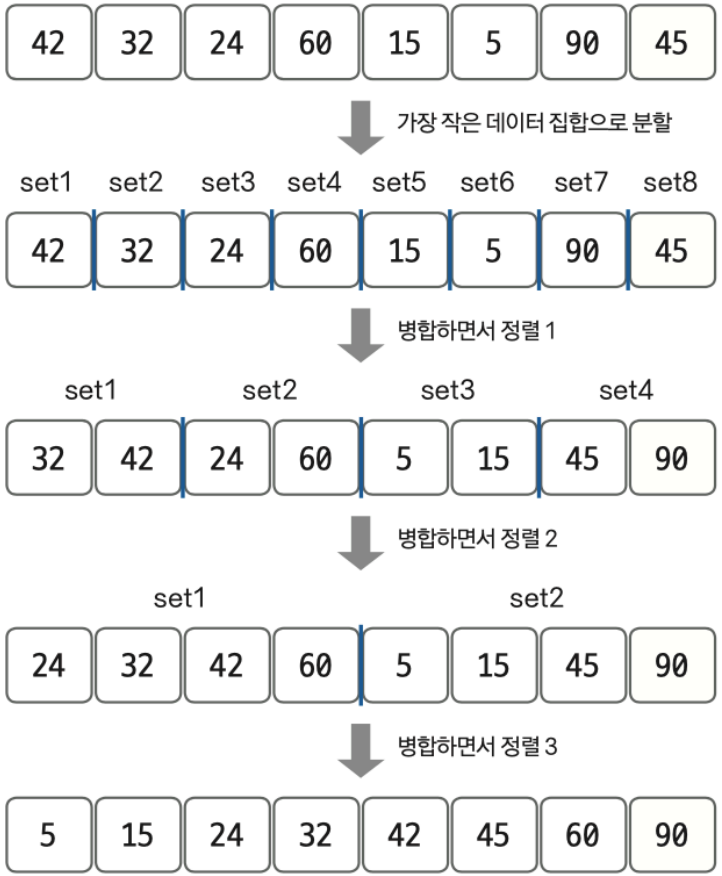
- 퀵 정렬은 평균의 경우 $O(NlogN)$의 시간 복잡도를 가집니다.
- 하지만 최악의 경우 $O(N^2)$의 시간 복잡도를 가집니다.



병합정렬(Merge Sort)

병합 정렬(merge sort)은 분할 정복(divide and conquer)방식을 사용해 데이터를 분할하고
분할한 집합을 정렬하며 합치는 알고리즘 이다.
시간 복잡도는 O(nlogn)이다.

병합정렬(Merge Sort)



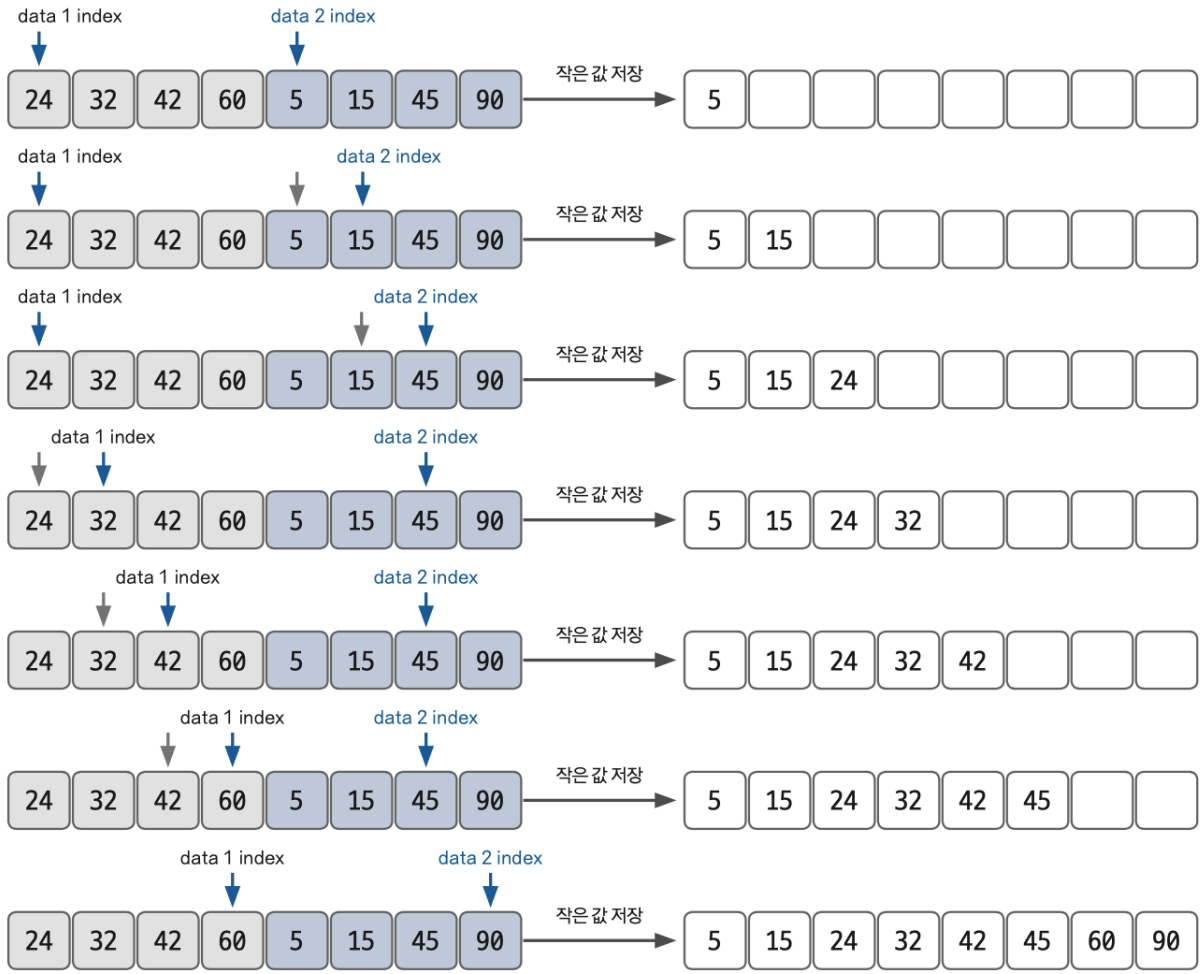
병합정렬(Merge Sort)

그림을 보면 최초에는 8개의 그룹으로 나뉜다.
이 상태에서 2개씩 그룹을 합치며 오름차순 정렬한다. 그 결과 (32, 42), (24, 60), (5, 15), (45, 90)이 된다.
이어서 2개씩 그룹을 합치며 다시 오름차순 정렬한다. 그 결과 (24, 32, 42, 60), (5, 15, 45, 90)이 된다.
이런 방식으로 병합 정렬 과정을 거치면 (5, 15, 24, 32, 42, 45, 60, 90)이 되어 전체 오름차순으로 정렬된다.

투 포인터 개념을 사용하여 왼쪽, 오른쪽 그룹을 병합한다.
왼쪽 포인터와 오른쪽 포인터의 값을 비교하여 작은 값을 결과 배열에 추가하고 포인터를 오른쪽으로 1칸 이동시킨다.

병합 정렬은 코딩 테스트의 정렬 관련 문제에 자주 등장한다.

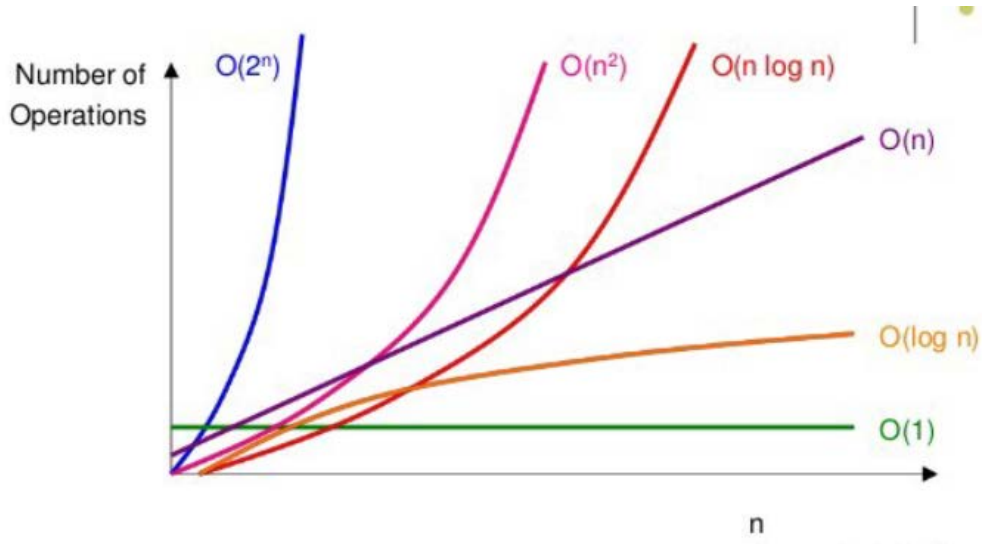
병합정렬(Merge Sort)



연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2) **정렬 (sorting)**

Algorithm	Time Complexity		
	Best	Average	Worst
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)
Merge Sort	Ω(n log(n))	θ(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 from 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.

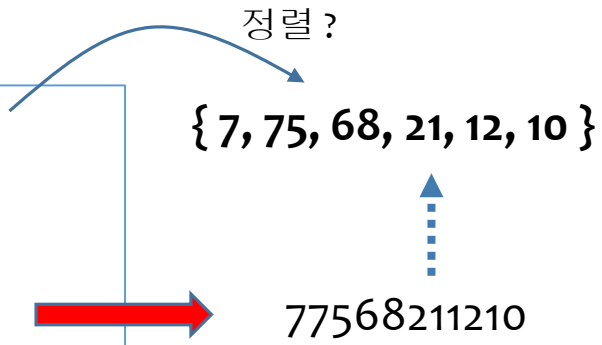
Find largest number possible from set of given numbers. The numbers should be appended to each other in any order to from the largest number.

(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

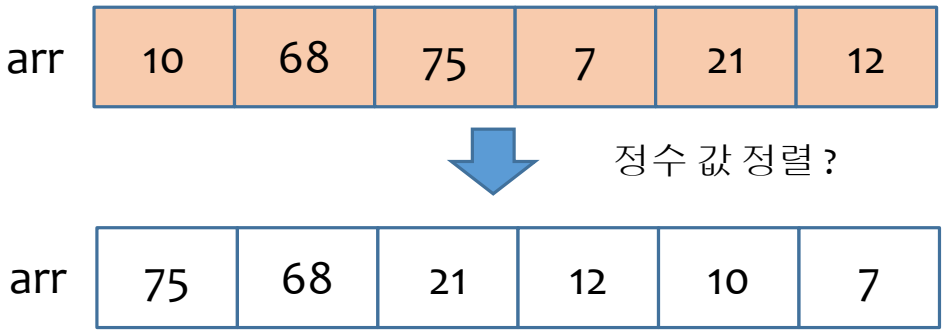
- (입력) { 2, 3, 4, 0, 6}
- (정렬) { 6, 4, 3, 2, 0 }
- (연결) 64320

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

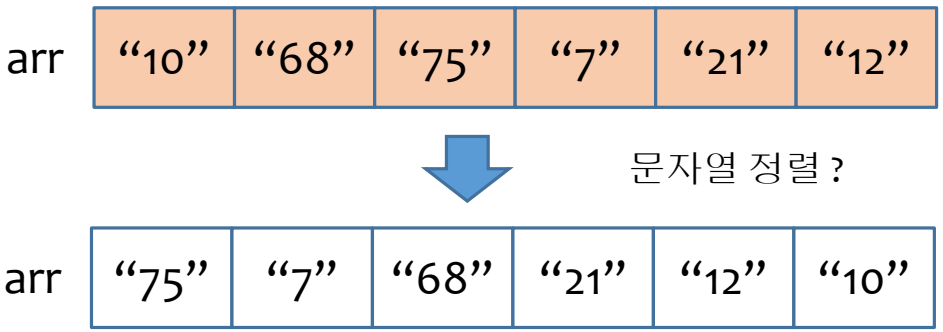


연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (1) **크기의 비교**

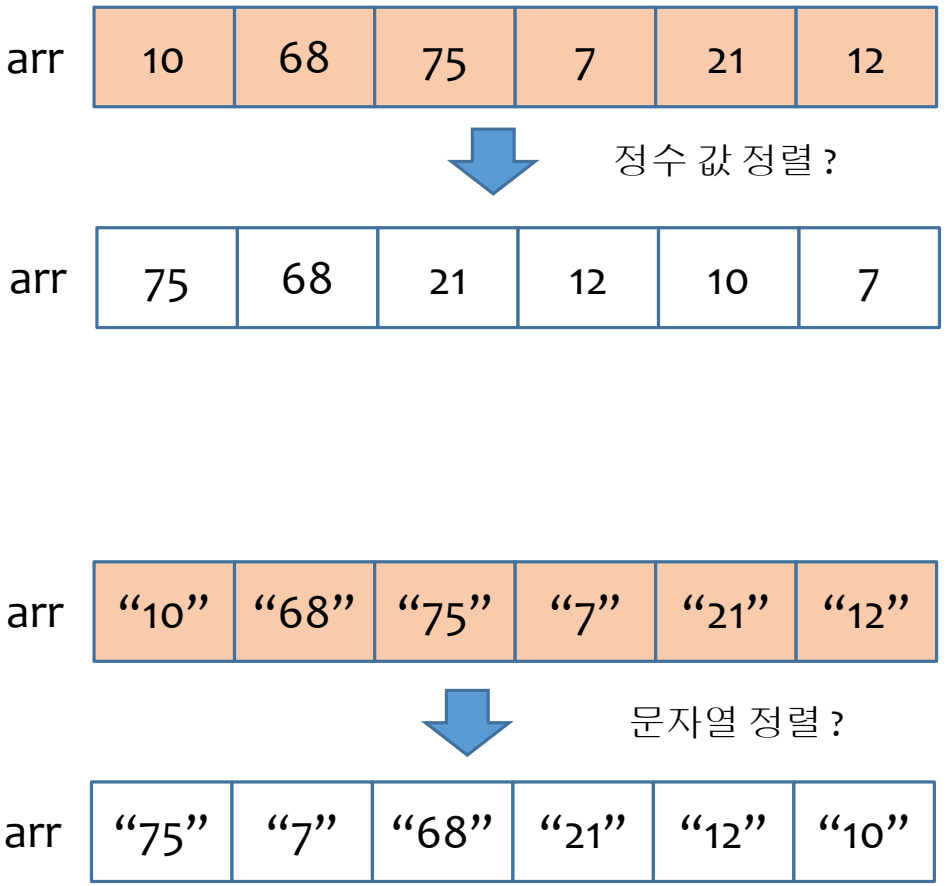


? { 7, 75, 68, 21, 12, 10 }



연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (1) **크기의 비교**



? { 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**

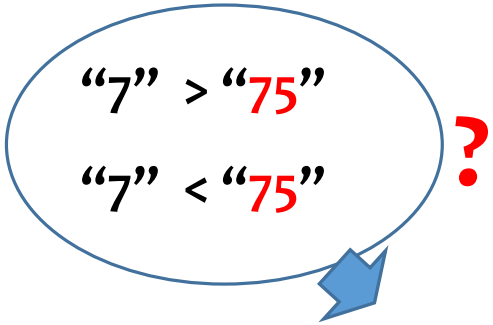
값의 정렬을 위한 수의 크기 비교

“xy”와 “ab” 중 누가 큰가?

If “xyab” > “abxy”

then “xy” > “ab”

else “ab” > “xy”



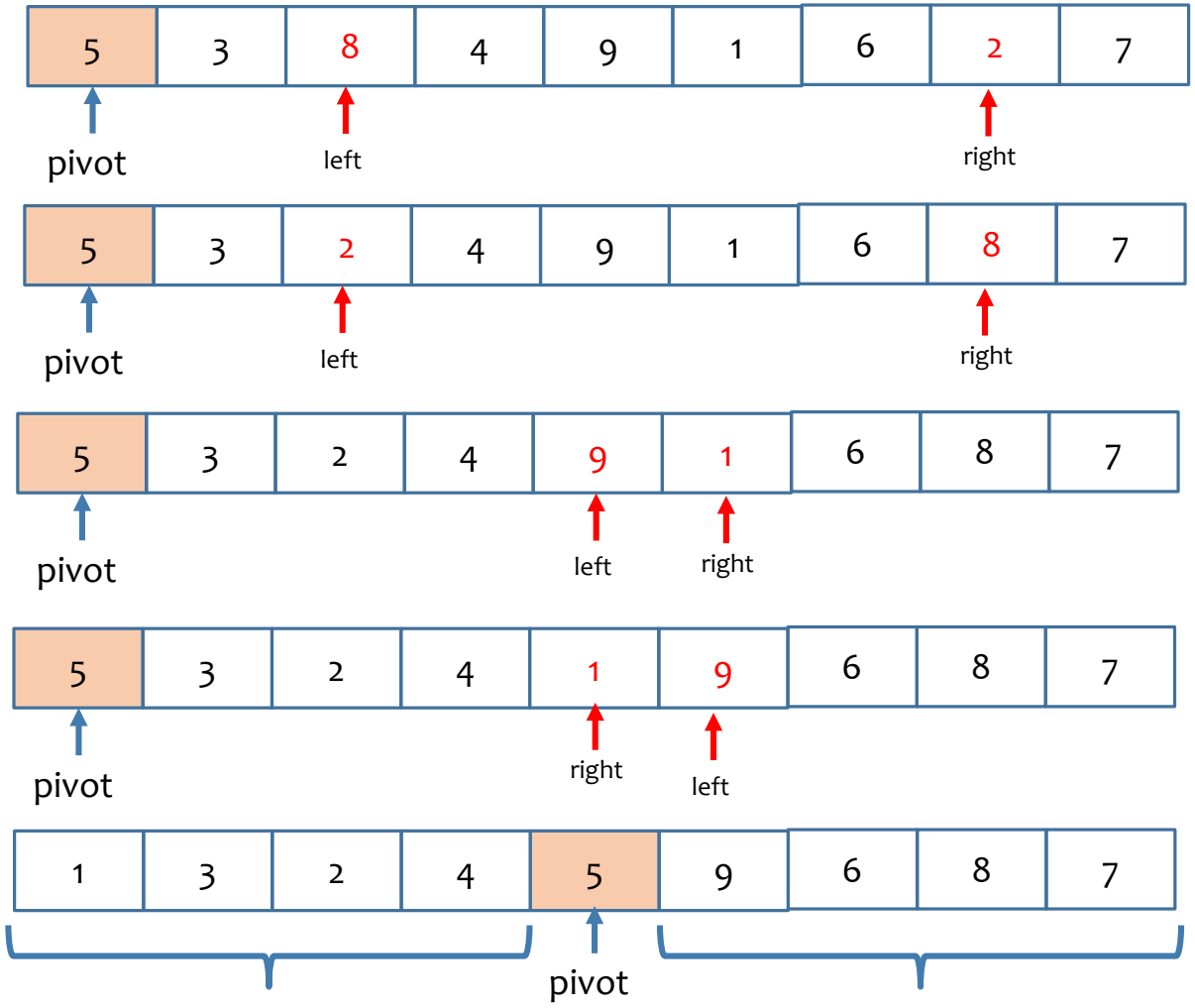
“775” > “757”

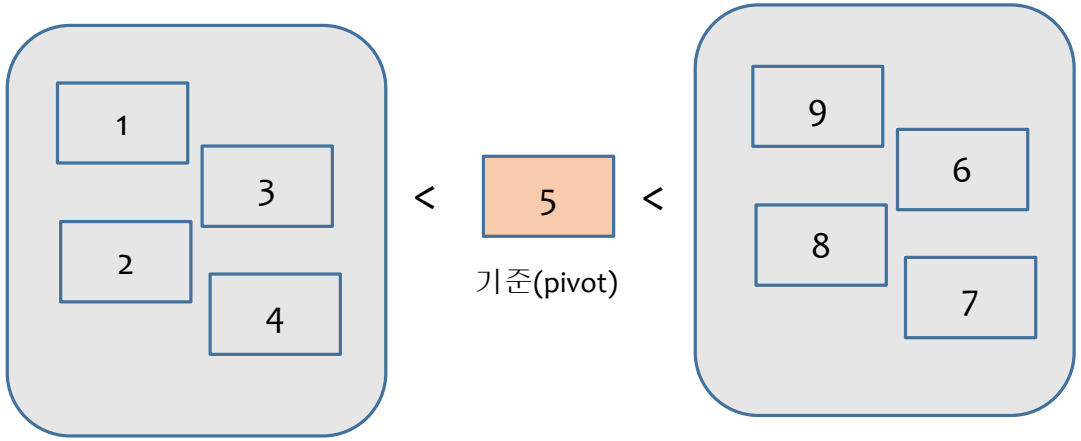
“7” > “75”

비교함수

연결하여 만들 수 있는 가장 큰 수 찾기

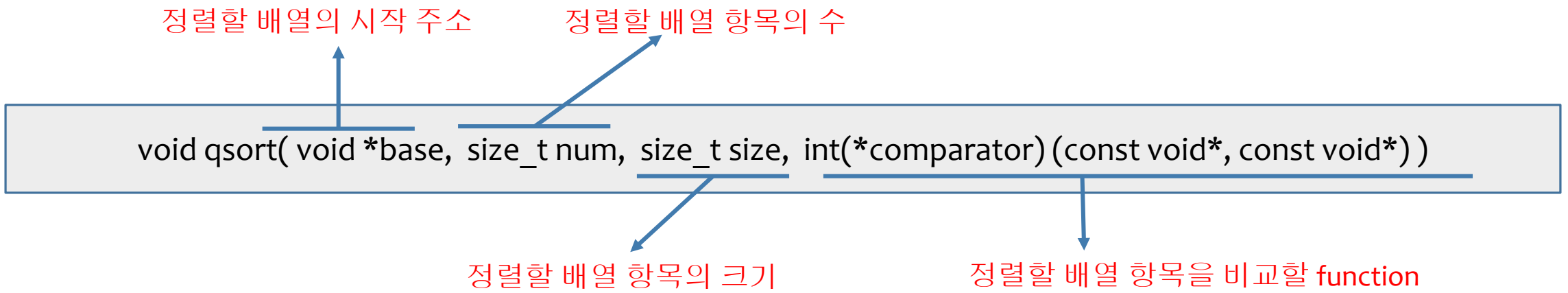
배경지식 (2) **정렬 (sorting)** **Quick sort()**





연결하여 만들 수 있는 가장 큰 수 찾기

배경지식 (2) **정렬 (sorting)** **qsort() in C**




```
# 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**

C

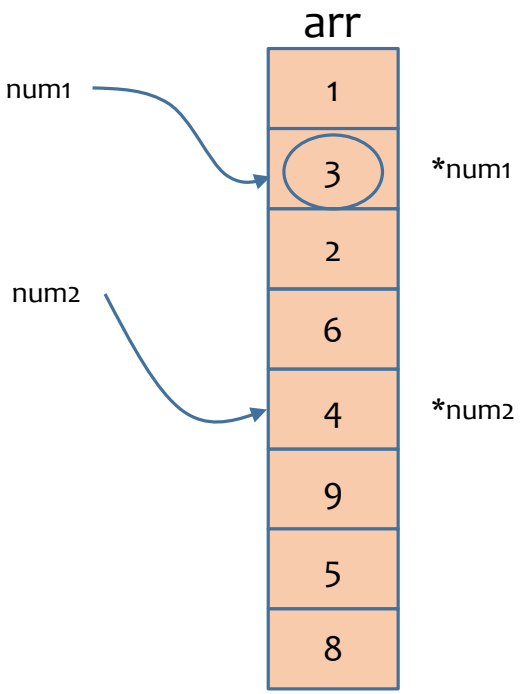
```c
# include <stdio.h>
# include <stdlib.h>

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;
}
```

Input Goes Here..

Time(sec) : 0

Output:

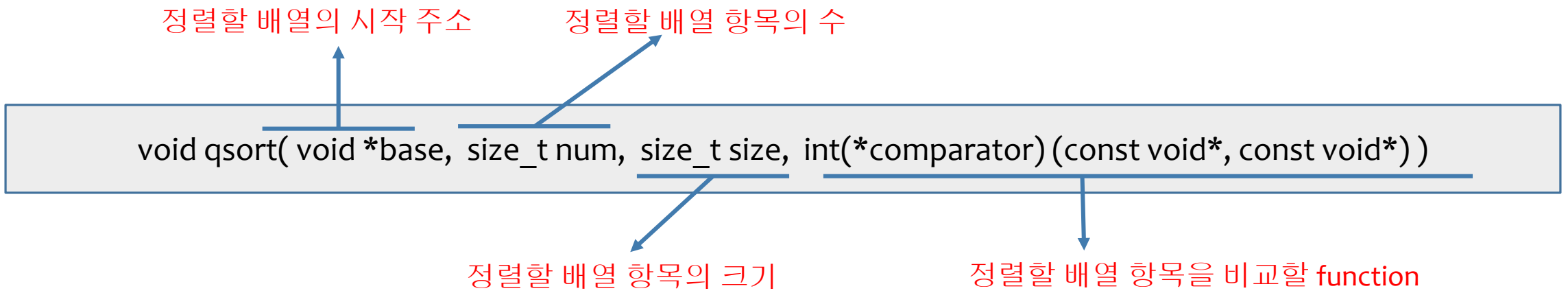
```
1 2 3 4 5 6 8 9
```




```c
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**




```
# 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**

```c
# 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;
}
```

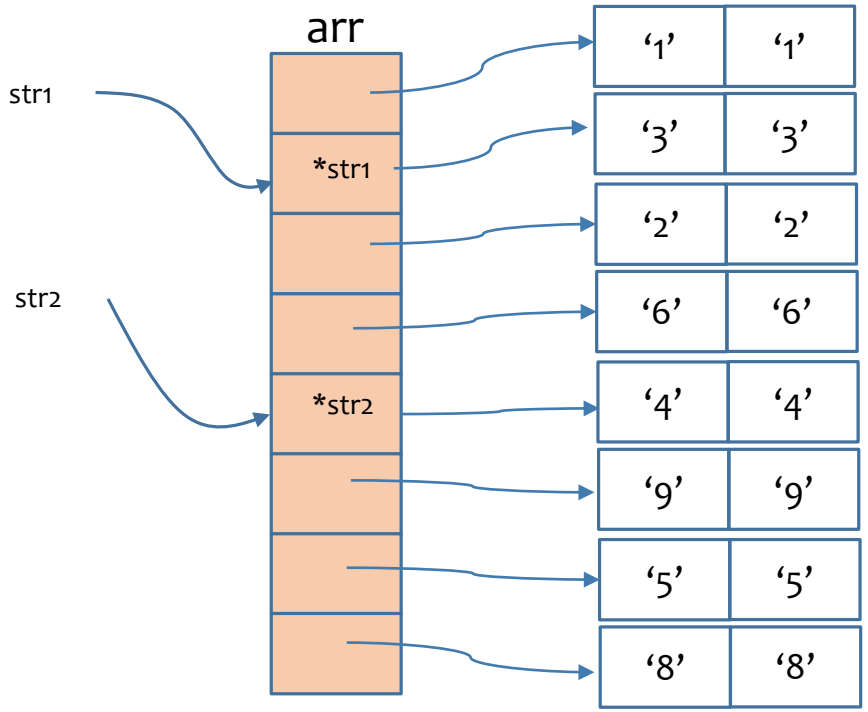
Input Goes Here..

Time(sec) : 0

Output:

```
11 22 33 44 55 66 88 99
```

```c
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

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

```
#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"

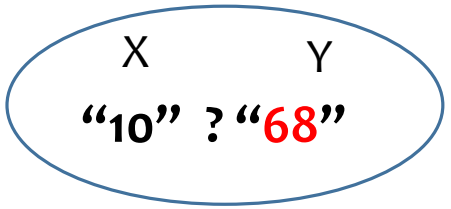
then "xy" > "ab"

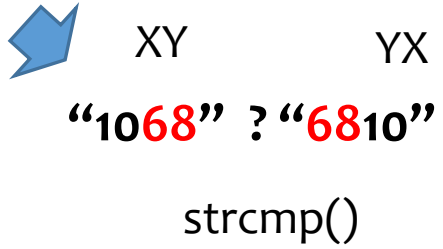
else "ab" > "xy"

연결하여 만들 수 있는 가장 큰 수 찾기(C언어예시)

배경지식 (2) **정렬 (sorting)** **qsort() in C**

```c
#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]);
    }
}
```





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

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
class Util1
{
	public static void main(String[] args)
	{
		List<String> numbers = Arrays.asList("10", "68", "75", “7", "21", "12");
		// sort using a custom function object
		Collections.sort(numbers, (a, b) -> (b + a).compareTo(a + b));
		// print the sorted sequence
		numbers.stream().forEach(System.out::print);
		for (int i = 0; i < numbers.size(); i++) {
			System.out.print(numbers.get(i));
		}
	}
}
```

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 반환**해주는 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	입력 **Input: {10, 68, 75, 7, 21, 12 }**	출력 **Output: 77568211210**

```
def largestNumber(array):
    if len(array)==1:
        return str(array[0])
    for i in range(len(array)):
        array[i]=str(array[i])
    # [10,68,75,7,21,12]=>['10','68','75','7','21','12']
    for i in range(len(array)):
        for j in range(1+i,len(array)):
            if array[j]+array[i]>array[i]+array[j]:
                array[i],array[j]=array[j],array[i]
    #['7', '75', '68', '21'... ]
    #Refer JOIN function in Python
    result=''.join(array)
    if(result=='0'*len(result)):
        return '0'
    else:
        return result
#main
arr = [10, 68, 75, 7, 21, 12]
print(largestNumber(arr))
```

연결하여 만들 수 있는 가장 큰 수 찾기(Python예시-2 sorted문사용)

문제해결 **연결하여 만들 수 있는 가장 큰 수 찾기**

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

```
from functools import cmp_to_key
def compare_to_xyab(a, b):
    if  a+b > b+a:
        return 1
    elif a+b == b+a:
        return 0
    else:
        return -1
def largestNumber(array):
    for i in range(len(array)):
        array[i] = str(array[i])
    result = sorted(array, key = cmp_to_key(compare_to_xyab) ,reverse = True)
    retult = ''.join(result)???????????
    return result
#-------------
arr = [10, 68, 75, 7, 21, 12]
```