Divide and Conquer

수학적 귀납법

- $P(1)$이 참이고, $P(n-1) \rightarrow P(n)$이 참이면, $P(n)$은 모든 자연수 n에 대해서 참이다.
- 직관과는 거리가 있지만, 수학적으로 증명하는 것에 익숙해져야 함

수학적 귀납법과 재귀

- $P(1)$이 참이고, $P(n-1) \rightarrow P(n)$이 참이면, $P(n)$은 모든 자연수 n에 대해서 참이다.
- 1부터 x까지의 합을 구하는 함수가 정말 우리가 원하는 값을 리턴하는지 증명해 보자.

```
int sum(int x){
    if (x <= 0) return 0;
    return x + sum(x-1);
}
```

수학적 귀납법과 재귀

- $P(1)$이 참이고, $P(n-1) \rightarrow P(n)$이 참이면, $P(n)$은 모든 자연수 n에 대해서 참이다.

- 1부터 x까지의 합을 구하는 함수가 정말 우리가 원하는 값을 리턴하는지 증명해 보자.

```
int sum(int x){
    if (x <= 0) return 0;
    return x + sum(x-1);
}
```

$$
\begin{aligned}
&\texttt{sum}(4) \\
&\rightarrow 4 + \texttt{sum}(3) \\
&\rightarrow 4 + (3 + \texttt{sum}(2)) \\
&\rightarrow 4 + (3 + (2 + \texttt{sum}(1))) \\
&\rightarrow 4 + (3 + (2 + (1 + \texttt{sum}(0)))) \\
&\rightarrow 4 + (3 + (2 + (1 + 0))) \\
&\rightarrow 10
\end{aligned}
$$

수학적 귀납법과 재귀

- Base: $x = 0$일 때, 합은 0이다. (참)
- Step: $x = k - 1$일 때, 함수가 1부터 $k - 1$까지의 합을 리턴한다면,
  그 값에 k를 더한 값은 1부터 k까지의 합과 같다.

- $P(1)$이 참이고, $P(n-1) \rightarrow P(n)$이 참이면, $P(n)$은 모든 자연수 n에 대해서 참이다.
- 만약 위 식에서 $P(n-1)$이 거짓이라면, $P(n)$은 참이 되기 때문에 생각할 필요가 없다.
- 따라서 $P(n-1)$이 참이라고 가정해도 된다.

```
int sum(int x){
    if (x <= 0) return 0;
    return x + sum(x-1);
}
```

투에-모스 문자열 BOJ 18222

- 투에-모스 문자열의 k번째 자리에 오는 문자를 출력하자
- 0 – 01 – 0110 – 01101001 – 0110100110010110 - …
- $k \leq 10^{18}$

투에-모스 문자열 BOJ 18222

- 투에-모스 문자열의 k번째 자리에 오는 문자를 출력하자
- 0 – 01 – 0110 – 01101001 – 0110100110010110 - …
- $k \leq 10^{18}$
- 직접 처음부터 문자열을 구축해나가는 것은 시간/메모리 초과

투에-모스 문자열 BOJ 18222

투에-모스 문자열 BOJ 18222

- 투에-모스 문자열의 k번째 자리에 오는 문자를 출력하자
- 0 – 01 – 0110 – 01101001 – 0110100110010110 - …
- 0과 1로 이루어진 문자열이므로, 두 번 뒤집으면 자신과 같은 수
- 절반보다 왼쪽에 있으면 뒤집을 필요 없고, 오른쪽에 있다면 결과를 뒤집어야 함

투에-모스 문자열 BOJ 18222

- 투에-모스 문자열의 k번째 자리에 오는 문자를 출력하자
- 0 – 01 – 0110 – 01101001 – 0110100110010110 - …
- $f(n, k)$ = n 길이의 투에-모스 문자열의 k번째 문자
- Base: 첫 번째 글자이면 0
- Step: k가 절반 왼쪽에 위치한다면 $f(\frac{n}{2}, k)$, 아니라면 $1 - f(\frac{n}{2}, k - \frac{n}{2})$

투에-모스 문자열 BOJ 18222

- $f(n,k)$ = n 길이의 투에-모스 문자열의 k번째 문자
- Base: 첫 번째 글자이면 0
- Step: k가 절반 왼쪽에 위치한다면 $f(\frac{n}{2},k)$, 아니라면 $1-f(\frac{n}{2},k-\frac{n}{2})$

```
int f(ll len, ll idx){
    if (idx == 1) return 0;
    if (len/2 >= idx) return f(len/2, idx);
    else return 1 - f(len/2, idx-len/2);
}
```

분할 정복

- 주어진 문제를 여러 개의 작은 문제로 **분할**
- 작은 문제들의 답을 이용해 주어진 문제의 답을 구함 (**정복**)

쉬운 곱셈

- $A^k \ mod \ M$을 구하자.

곱셈 BOJ 1629

- $A^k \ mod \ M$을 구하자. $(A, M, k \le 2\ 147\ 483\ 647)$

곱셈 BOJ 1629

- $A^k \ mod \ M$을 구하자. $(A, M, k \le 2\ 147\ 483\ 647)$

- Naïve approach

$$A \times A \times A \times A \times \cdots \times A \ mod \ M$$

- Overflow

$$\left(\left(\left((A \times A) \ mod \ M\right) \times A\right) mod \ M\right) \times \cdots$$

- 시간복잡도는 $O(N)$

결과
시간 초과

곱셈 BOJ 1629

- $A^k \ mod \ M$을 구하자. $(A, M, k \le 2\ 147\ 483\ 647)$
- $A^k = (A^2)^{\frac{k}{2}}$ 임을 활용할 수 있다
- 2^8을 단순하게 계산한다면 8번의 연산이 필요하다
- $2^8 = (2^2)^4 = ((2^2)^2)^2$
- 제곱 한 번이 계산이므로 총 3번의 계산으로 2^8을 계산할 수 있다.

곱셈 BOJ 1629

- $A^k \ mod \ M$을 구하자. $(A, M, k \leq 2\ 147\ 483\ 647)$

- $f(n, k)$가 n^k를 계산해준다고 하자.

- Base: $k = 0$인 경우, 1을 리턴한다. (참)
- Step: $k \geq 1$인 짝수의 경우, $f(n,\ \frac{k}{2}) \times f\left(n,\ \frac{k}{2}\right)$를 기계적으로 계산하면 된다.
  $k \geq 1$인 홀수의 경우, $n \times f(n,\ \ k-1)$를 기계적으로 계산하면 된다.

- 나머지 연산은 분배법칙을 활용해 오버플로우나지 않도록 해주자.

곱셈 BOJ 1629

- $A^k \ mod \ M$을 구하자. $(A, M, k \le 2\ 147\ 483\ 647)$

```
long long f(long long a, long long b){
    if (b == 0) return 1;
    ll val = f(a, b / 2);
    if (b % 2) return a * val * val;
    else return val * val;
}
```

충분히 작은 문제는 직접 해결

큰 문제는 작은 문제로 쪼갬

작은 문제의 정답으로 큰 문제의 답을 구함

- 시간복잡도 $T(k) = T\left(\frac{k}{2}\right) + O(1) = O(logk)$

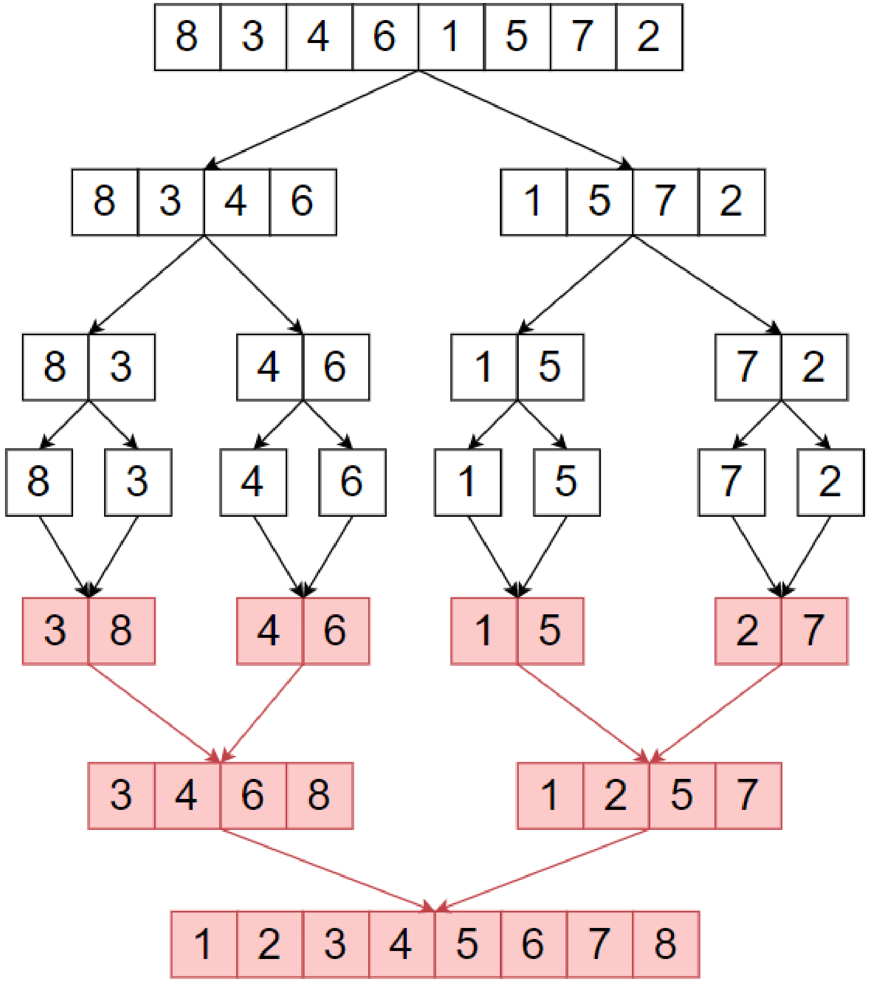
정렬

- 주어진 배열을 비내림차순으로 정렬하자.

정렬

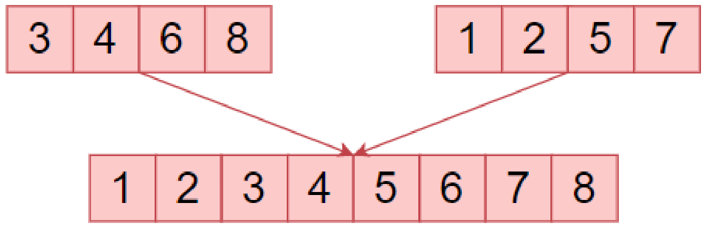
- $f(l,r)$이 $arr[l:r]$을 정렬해주는 함수
- Base: $l = r$인 경우, 배열은 이미 정렬돼 있다. (원소가 한 개인 배열, 직접 해결)
- Step: $mid = \frac{l+r}{2}$인 mid에 대해서
- 만약 $f(l, mid)$와 $f(mid+1, r)$이 각각 해당 구간을 올바르게 정렬해준다면,
- 정렬된 두 구간을 **정렬되도록 합쳐주면** 된다. (큰 경우에는 작게 분할 후 답을 정복)

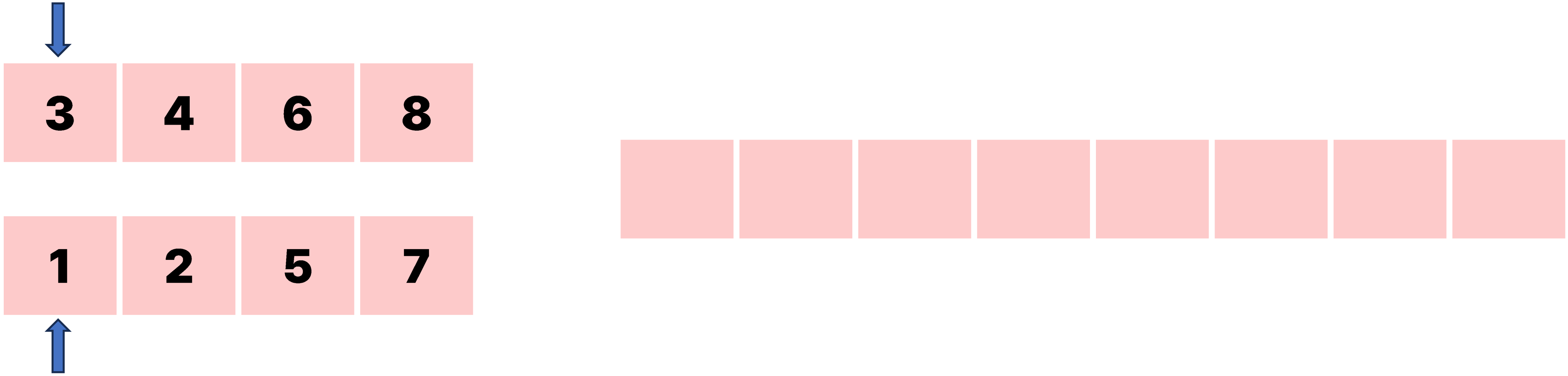
Merge Sort BOJ 2751



Merge

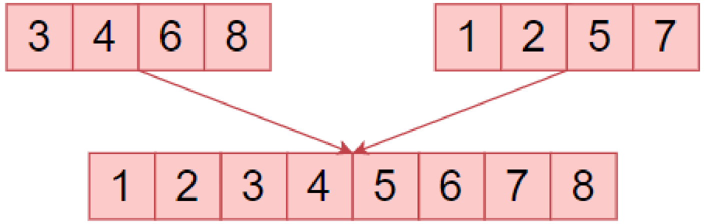
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

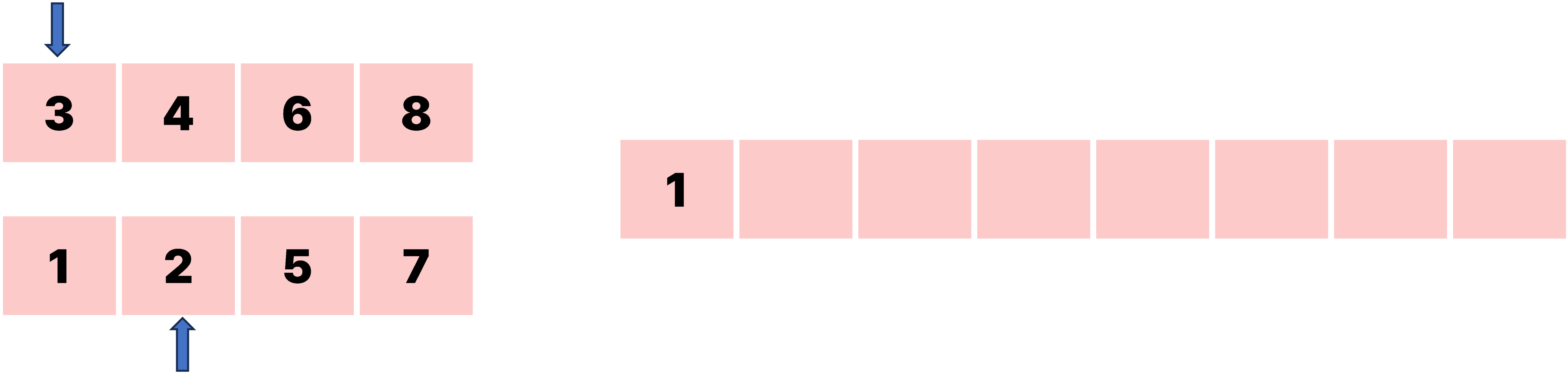




Merge

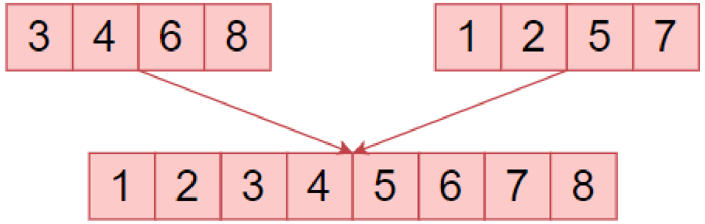
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

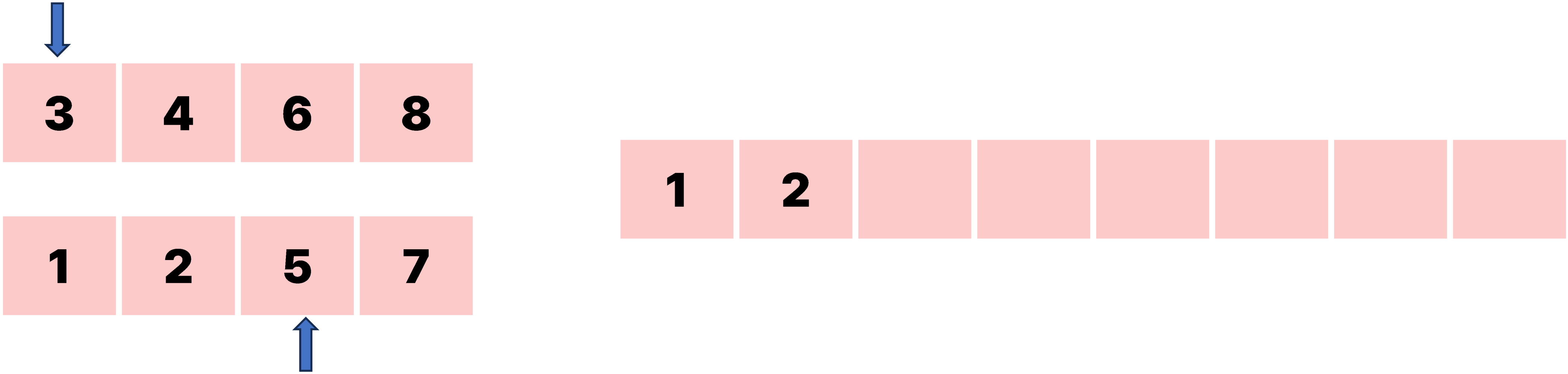




Merge

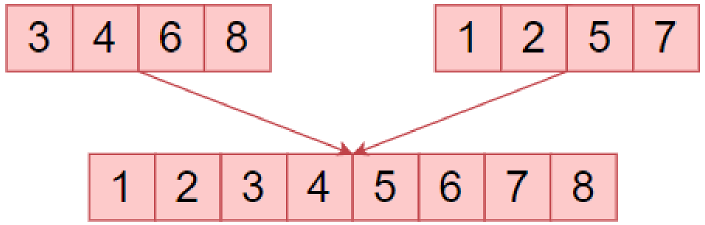
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

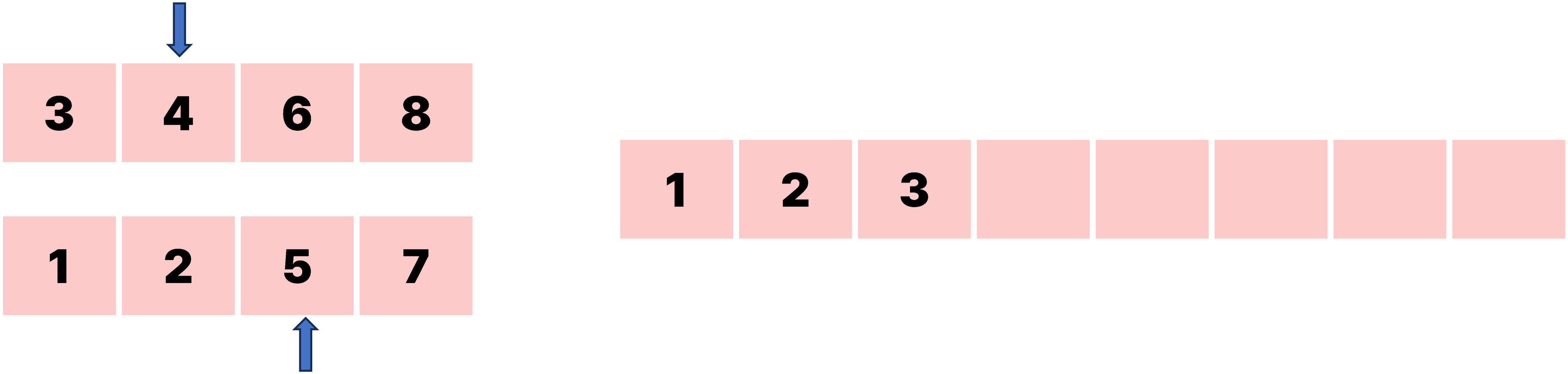




Merge

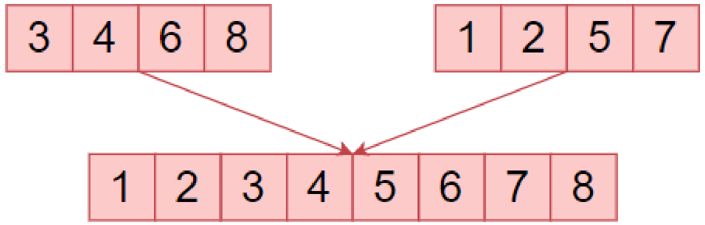
• 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

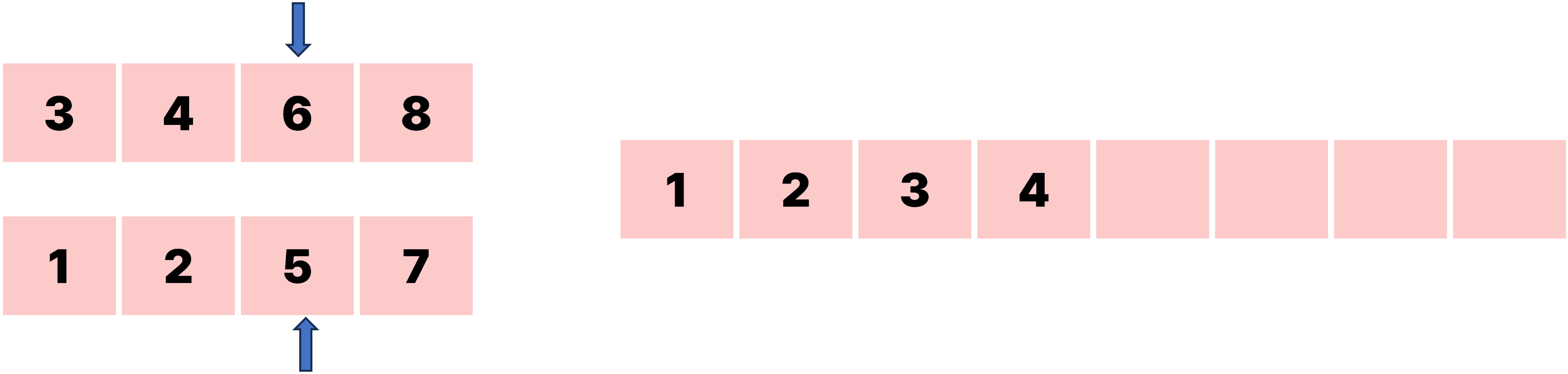




Merge

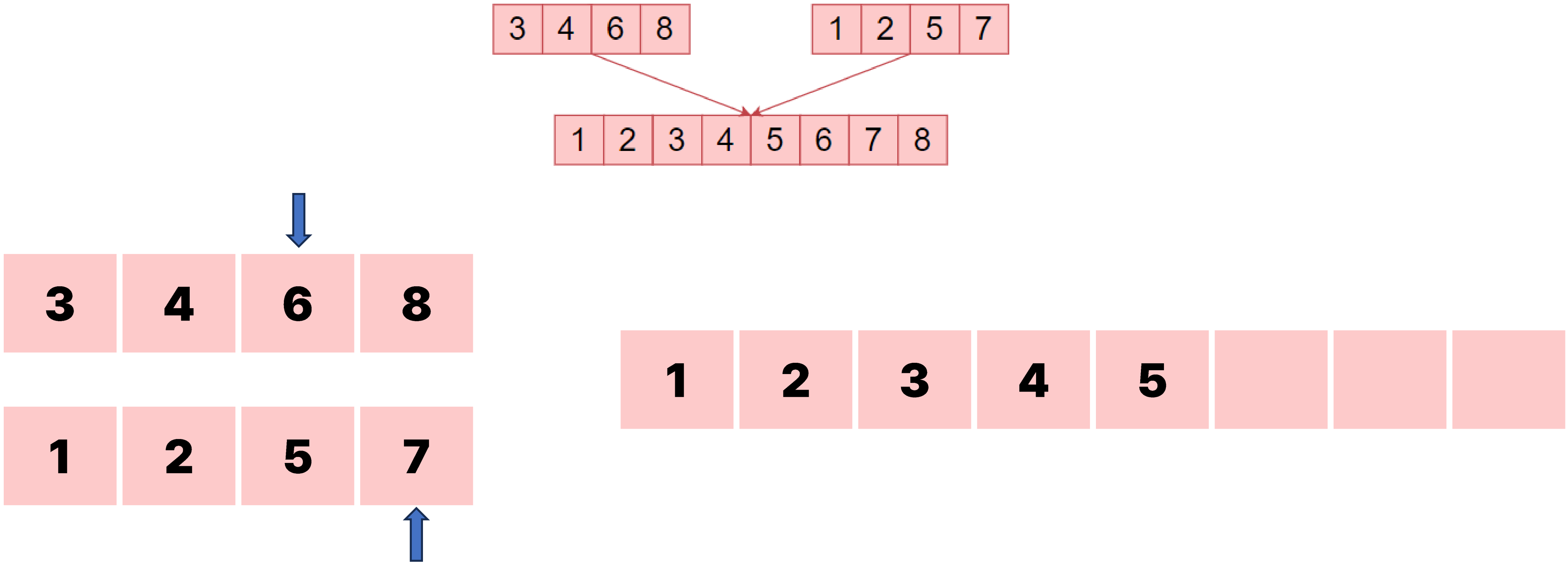
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정





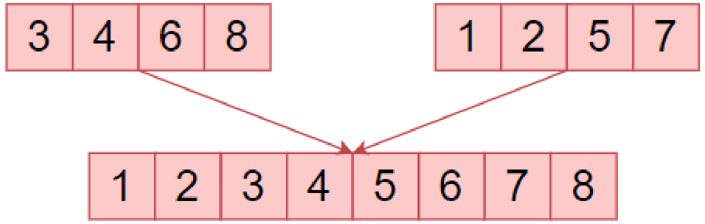
Merge

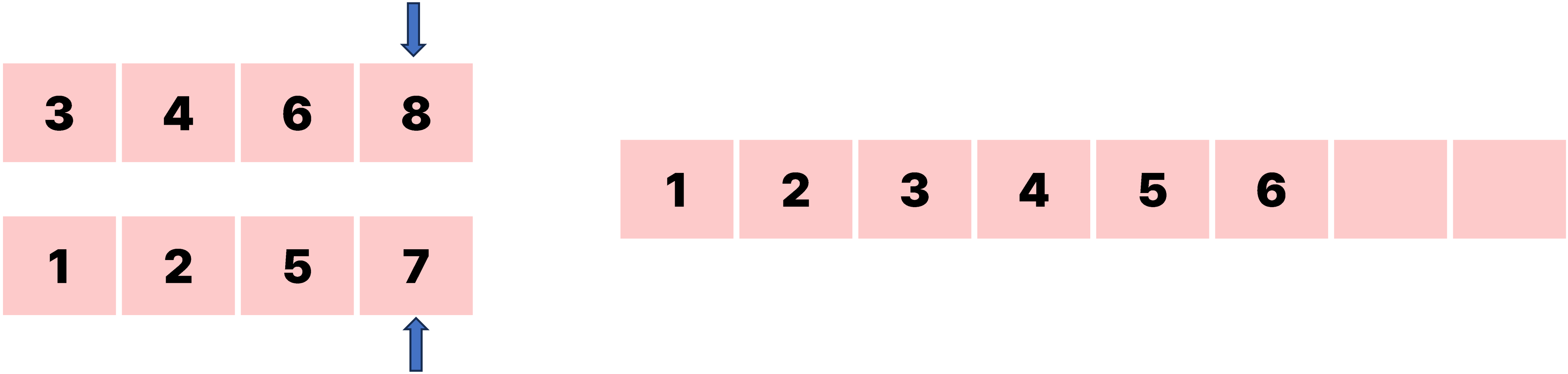
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정



Merge

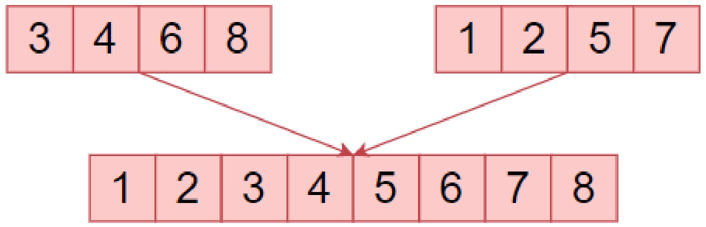
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

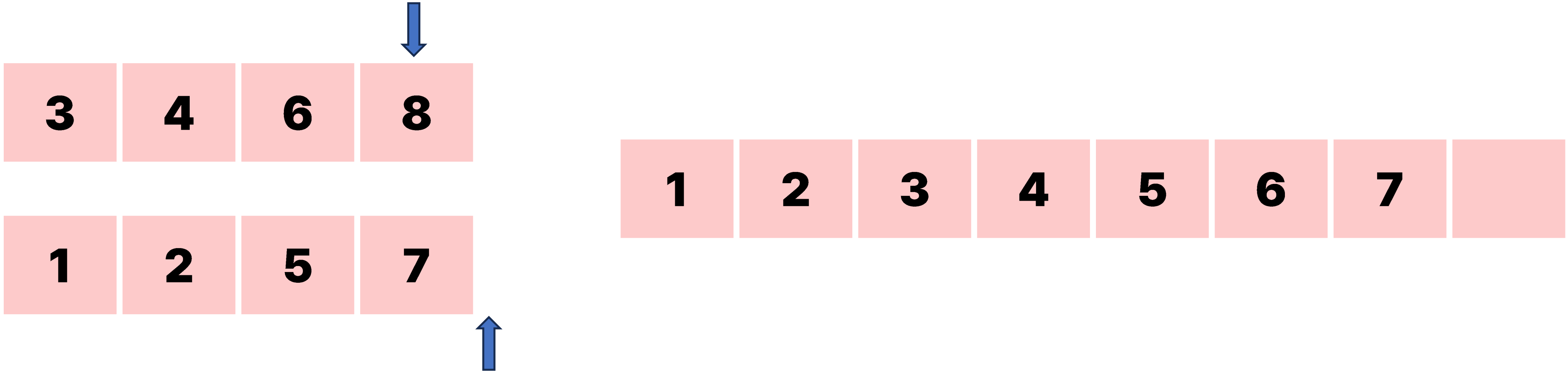




Merge

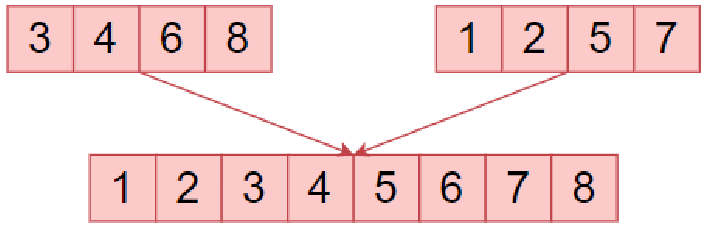
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

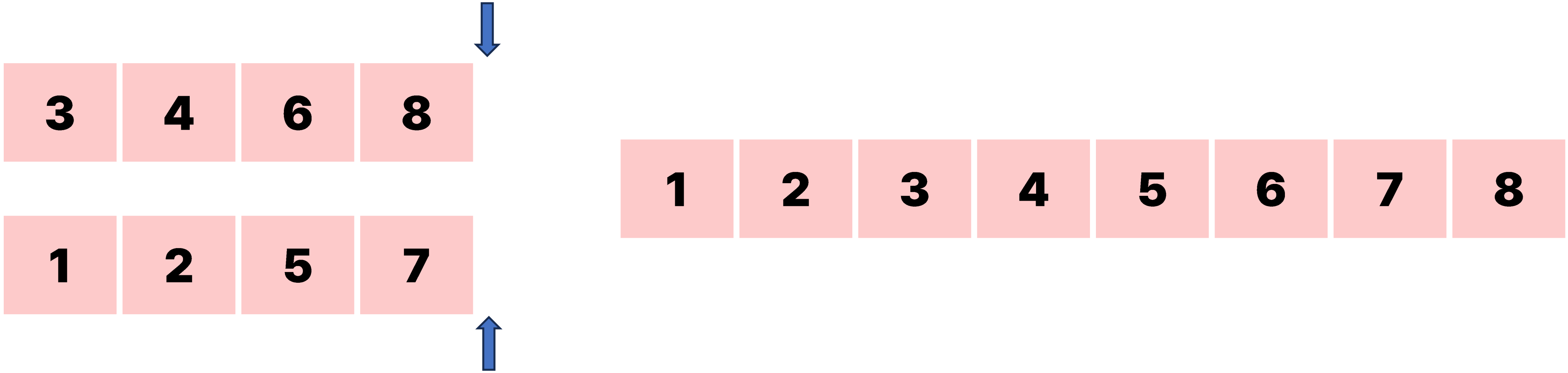




Merge

- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정





Merge

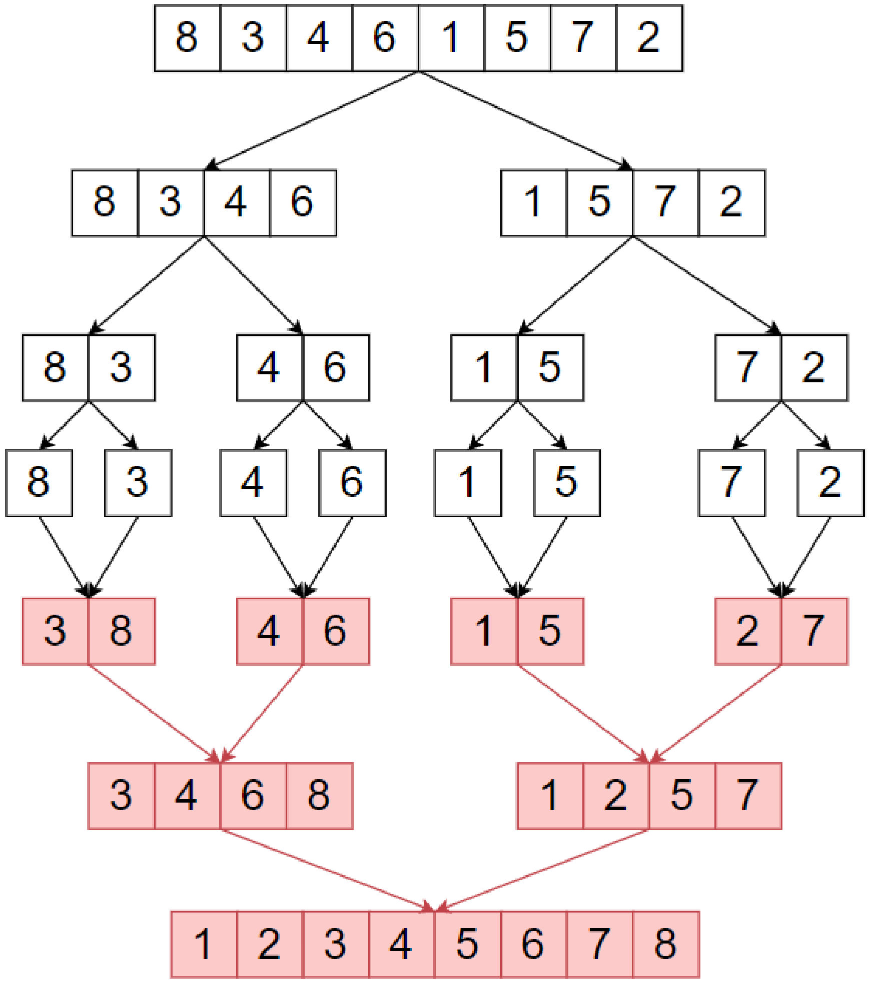
- 이미 정렬된 두 배열을 정렬된 하나의 배열로 합치는 과정

```
int i = l, j = m+1, idx = l;
while(i <= m && j <= r){
    if (a[i] < a[j]) s[idx++] = a[i++];
    else s[idx++] = a[j++];
}
while(i <= m) s[idx++] = a[i++];
while(j <= r) s[idx++] = a[j++];
```

Merge Sort BOJ 2751

- $T(N)$은 배열을 정렬하는 데 드는 시간
- $T(N) = 2 \times T\left(\frac{N}{2}\right) + O(N)$
- 두 개의 구역으로 나눠 각각 정렬한 뒤,
- 두 배열을 합쳐주는 시간이 추가로 발생

$= O(NlogN)$



Merge Sort

```
void f(int l, int r) {
    if (l > r) return;
    if (l == r) {
        s[l] = a[l];
        return;
    }
    int m = (l + r) / 2;
    f(l, m); f(m+1, r);
    int i = l, j = m+1, idx = l;
    while(i <= m && j <= r){
        if (a[i] < a[j]) s[idx++] = a[i++];
        else s[idx++] = a[j++];
    }
    while(i <= m) s[idx++] = a[i++];
    while(j <= r) s[idx++] = a[j++];
    for(int i = l; i <= r; i++) a[i] = s[i];
}
```

Counting Inversions BOJ 10090

- $i < j$이면서 $arr[i] > arr[j]$인 순서쌍 (i, j)의 개수를 구하는 문제

4	**2**	**7**	**1**	**5**	**6**	**3**

- 쉽게 풀면, 자신보다 뒤에 있는 수 중, 자신보다 작은 수의 총 개수를 구하는 문제
- 위 배열에서는 총 10개의 inversion이 존재함
- $(1,2), (1,4), (1,7), (2,4), (3,4), (3,5), (3,6), (3,7), (5,7), (6,7)$

Counting Inversions BOJ 10090

- $i < j$이면서 $arr[i] > arr[j]$인 순서쌍 (i, j)의 개수를 구하는 문제

4	**2**	**7**	**1**	**5**	**6**	**3**

- Naïve approach: 각 배열의 원소에 대해서, 모든 inversion을 탐색 - $O(N^2)$
- $N \leq 1\,000\,000$ 이기 때문에, 더 나은 시간복잡도 알고리즘이 필요함

Counting Inversions BOJ 10090

- 배열을 절반으로 쪼개서 생각해 보자

4	2	7

1	5	6	3

Counting Inversions BOJ 10090

- 배열을 절반으로 쪼개서 생각해 보자

4	2	7		1	5	6	3

- 분할 정복에서 사용하듯이, 쪼갠 배열 내에서 inversion의 수는 재귀적으로 계산

Counting Inversions BOJ 10090

- 배열을 절반으로 쪼개서 생각해 보자

4	2	7		1	5	6	3

- 분할 정복에서 사용하듯이, 쪼갠 배열 내에서 Inversion의 수는 재귀적으로 계산

1	2	3	4	5	6	7

- 추가적으로 구해야하는 inversion은 왼쪽 절반과 오른쪽 절반 사이의 inversion 개수

Counting Inversions BOJ 10090

- 추가적으로 구해야하는 inversion은 왼쪽 절반과 오른쪽 절반 사이의 inversion 개수
- Merge Sort의 아이디어를 활용해 보자

2	4	7		1	3	5	6

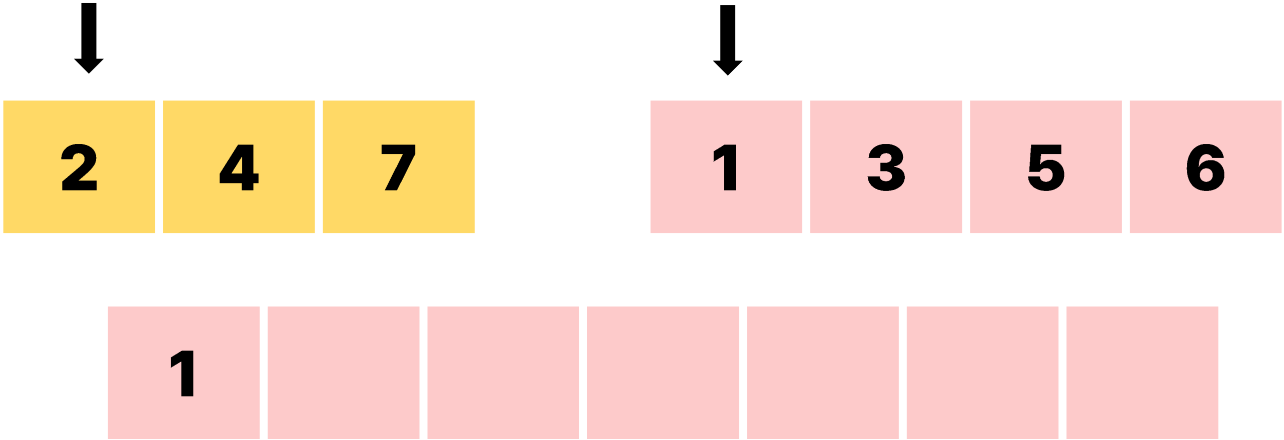
- 왼쪽 절반과 오른쪽 절반의 inversion은 이미 구했고, Merge Sort로 이미 정렬돼 있다

1	2	3	4	5	6	7

- Merge할 때 inversion의 개수를 어떻게 셀까?

Counting Inversions BOJ 10090

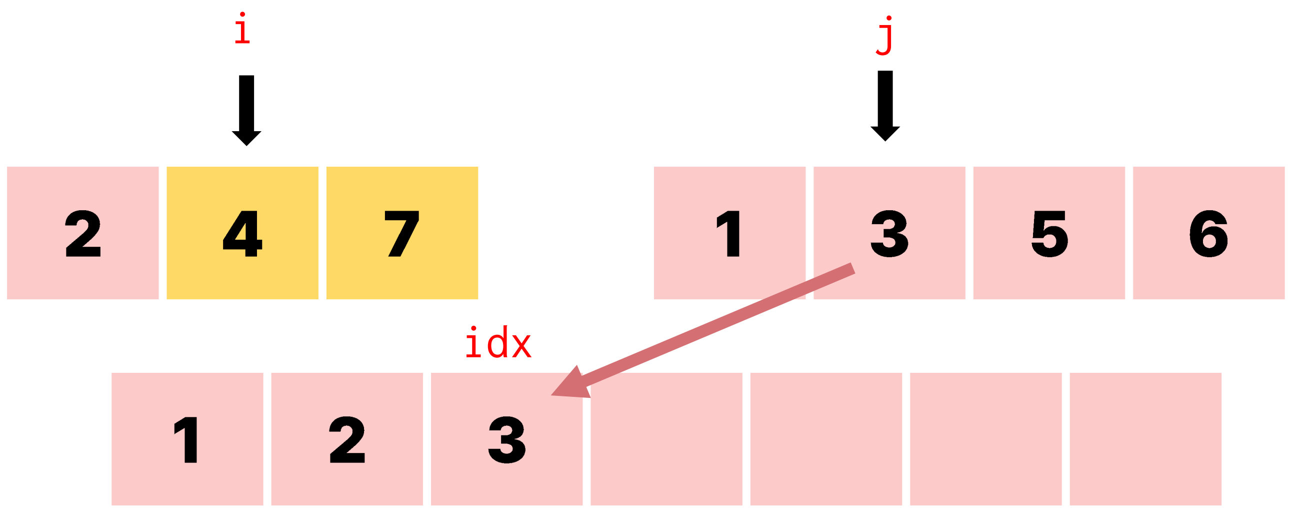
- 2, 4, 7은 1보다 크지만 왼쪽 절반에 위치해 있다
- 화살표의 위치로 한 번에 몇 개의 inversion이 생기는지 찾아낼 수 있음



- 오른쪽 절반에서 더 작은 값이 나왔을 때,
- 배열이 정렬돼있으므로, 왼쪽 절반의 병합되지 않은 원소의 개수를 추가하자.

Counting Inversions BOJ 10090

```
void f(int l, int r){
    if (l > r) return;
    if (l == r){
        s[l] = a[l];
        return;
    }
    int m = (l + r) / 2;
    f(l, m); f(m+1, r);
    int i = l, j = m+1, idx = l;
    while(i <= m && j <= r){
        if (a[i] < a[j]) s[idx++] = a[i++];
        else s[idx++] = a[j++], ans += m - i + 1;
    }
    while(i <= m) s[idx++] = a[i++];
    while(j <= r) s[idx++] = a[j++];
    for(int i = l; i <= r; i++) a[i] = s[i];
}
```



버블 소트 BOJ 1517

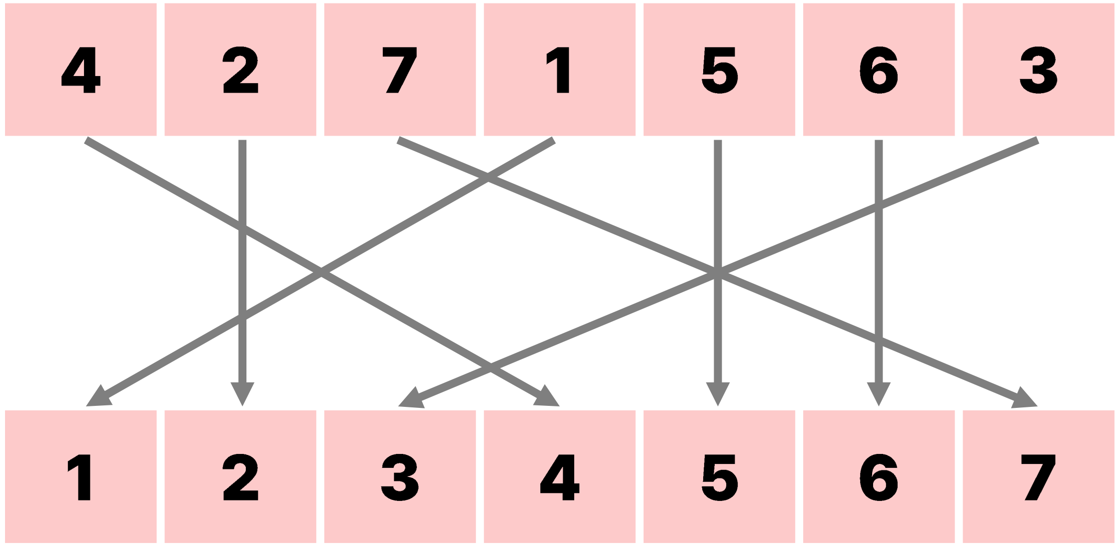
- 버블 소트를 사용해 정렬할 때까지 발생하는 Swap 수의 총합을 구하는 문제

4	2	7	1	5	6	3

- 실제 버블 소트를 구현하면 $O(N^2)$

버블 소트 BOJ 1517

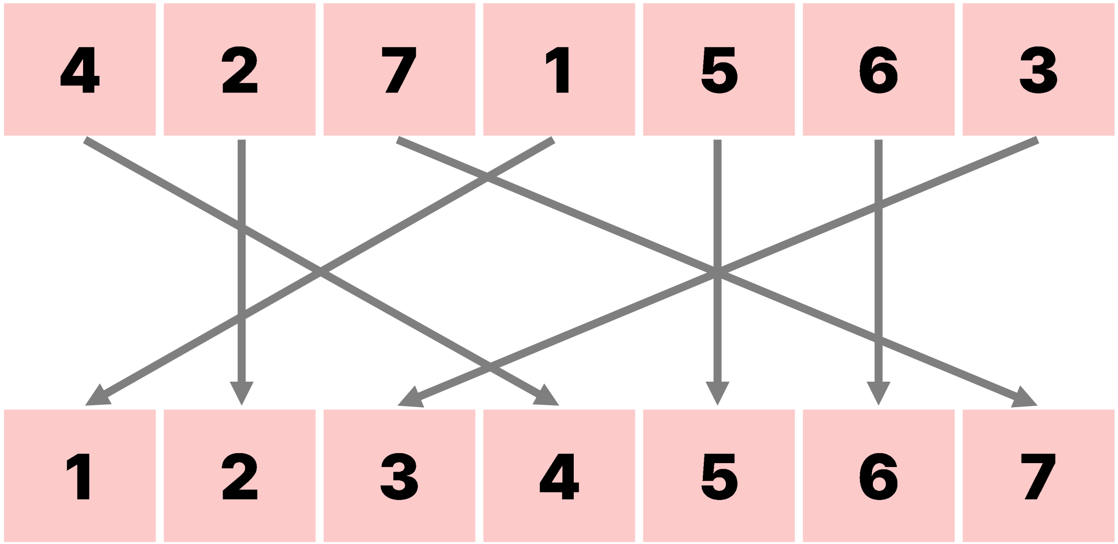
- 버블 소트를 사용해 정렬할 때까지 발생하는 Swap 수의 총합을 구하는 문제



= 자신보다 뒤에 있으면서 작은 수의 개수 찾는 문제

버블 소트 BOJ 1517

- 정렬된 배열로 향하는 선을 그었을 때, 교점의 개수와 같다



행렬 제곱 BOJ 10830

- 앞에서 설명했던 자연수 제곱과 같은 방법으로 접근할 수 있음
- 행렬의 곱셈의 항등원은 단위행렬 I임에 주의하자

피보나치 수 6 BOJ 11444

- 10^{18}번째 피보나치 수를 구해야하는 문제
- 피보나치 수는 간단하지만, 행렬식으로도 표현할 수 있다
- $F_{i+1} = F_i + F_{i-1},\ \ F_i = F_i + 0$
- $\begin{pmatrix} F_{i+1} \\ F_i \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_i \\ F_{i-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^i \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^i \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

- 간편히 만든 행렬을 i번 제곱한 뒤, 수를 확인하면 된다.
- $O(logN)$ 의 시간복잡도를 가지며, 위 제한을 통과한다.

연습 문제

[1629](#) : 곱셈
[2751](#) : 수 정렬하기 2 (Merge Sort 구현)
[18222](#) : 투에-모스 문자열
[16974](#) : 레벨 햄버거
[21870](#) : 시철이가 사랑한 GCD
[10090](#) : Counting Inversions
[1725](#) : 히스토그램 (분할 정복 사용) **
[2261](#) : 가장 가까운 두 점 ***
[18253](#) : 최단경로와 쿼리 ****

References

- https://blog.hoony.me/2022/09/02/proof-recursive-using-mathematical-induction/
- https://www.codingninjas.com/studio/problems/merge-sort_920442
- https://github.com/justiceHui/SSU-SCCC-Study/blob/master/2023-summer-basic/slide/05-2-basic-divide-and-conquer.pdf
- https://ohgym.tistory.com/1