SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 기본

SAMSUNG SW ACADEMY FOR YOUTH

APS(Algorithm Problem Solving)

LIST 4

- 2차원 배열
- 카운팅 정렬(Counting Sort)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

2차원 배열

2차원 배열

2차원 배열의 선언

- 2차원 이상의 다차원 배열은 차원에 따라 Index를 선언
- 2차원 배열의 선언 : 세로길이(행의 개수), 가로길이(열의 개수)를 필요로 함

```
int[][] twoDimArr = new int[2][4]
```

(2행 4열의 2차원 배열 선언)

0	1	2	3
4	5	6	7

twoDimArr

2차원 배열의 접근

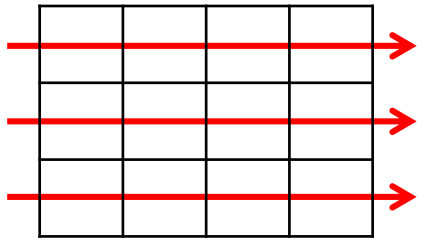
배열 순회

- n X m 배열의 n * m 개의 모든 원소를 빠짐 없이 조사하는 방법

행 우선 순회

```
int i; // 행의 좌표
int j; // 열의 좌표

for i from 0 to n-1
   for j from 0 to m-1
        Array[i][j] //필요한 연산 수행
```



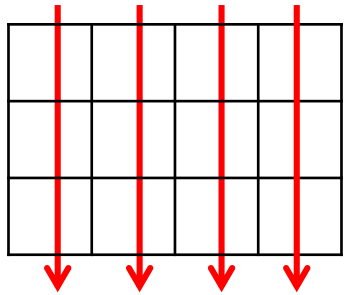
행 우선 순회

2차원 배열의 접근

열 우선 순회

```
int i; // 행의 좌표
int j; // 열의 좌표

for j from 0 to m-1
   for i from 0 to n-1
        Array[i][j] //필요한 연산 수행
```

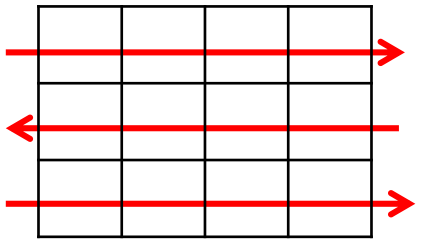


열 우선 순회

지그재그 순회

```
int i; // 행의 좌표
int j; // 열의 좌표

for i from 0 to n-1
   for j from 0 to m-1
        Array[i][j+(m-1-2*j)*(i%2)]
        //필요한 연산 수행
```



지그재그 순회

2차원 배열의 접근

델타를 이용한 2차 배열 탐색

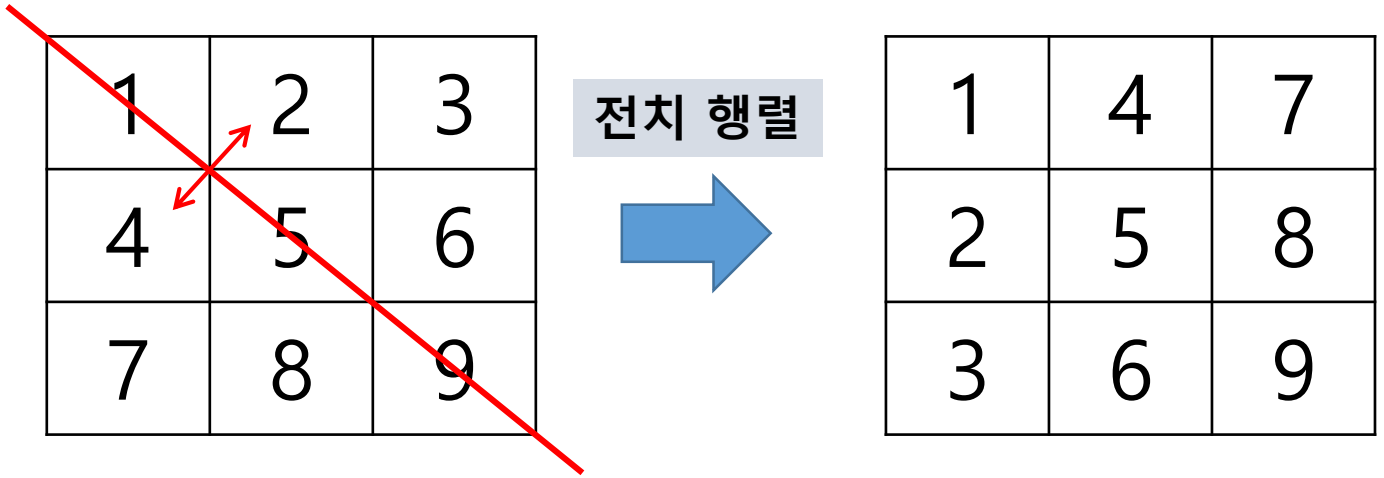
- 2차 배열의 한 좌표에서 4방향의 인접 배열 요소를 탐색하는 방법

```
array[0…n-1][0…n-1] //2차원 배열
dr[] ← {-1, 1, 0, 0} # 상하좌우
dc[] ← {0, 0, -1, 1}

for r in from 0 to n-1
    for c in from 0 to n-1
        for d from 0 to 3
            nr ← r + dr[d];
            nc ← c + dc[d];
            // 조건
            array[nr][nc]; // 필요한 연산 수행
```

2차원 배열의 활용

전치 행렬




```
arr = {{1,2,3},{4,5,6},{7,8,9}} // 3*3 행렬
int i; //행의 좌표
int j; //열의 좌표

for i from 0 to 2
    for j from 0 to 2
         if (i < j)
             swap(arr[i][j], arr[j][i]);
```

<연습문제1>

- 5x5 2차 배열에 무작위로 25개의 숫자로 초기화 한 후
- 25개의 각 요소에 대해서 그 요소와 이웃한 요소와의 차의 절대값을 구하시오.
- 예를 들어 아래 그림에서 7 값의 이웃한 값은 2, 6, 8, 12 이며 차의 절대값의 합은 12 이다.
  - $|2-7|+|6-7|+|8-7|+|12-7|=12$

…	2	…
6	7	8
…	12	…

- 25개의 요소에 대해서 모두 조사하여 총합을 구하시오.
- 벽에 있는 요소는 이웃한 요소가 없을 수 있음을 주의하시오.
  - 예를 들어 [0][0]은 이웃한 요소가 2개이다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

카운팅 정렬
(Counting Sort)

카운팅 정렬(Counting Sort)

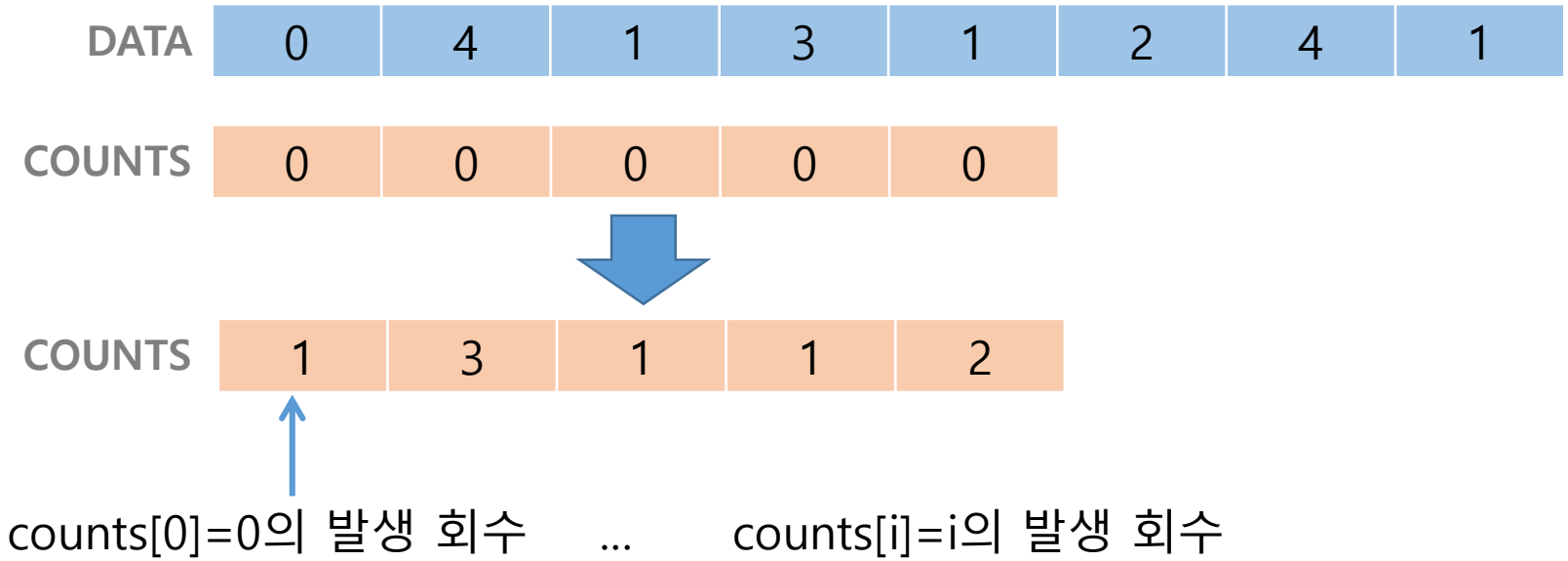
- **항목들의 순서를 결정하기 위해 집합에 각 항목이 몇 개씩 있는지 세는 작업을 하여, 선형 시간에 정렬하는 효율적인 알고리즘**

- **제한 사항**
  - 정수나 정수로 표현할 수 있는 자료에 대해서만 적용 가능
  - 각 항목의 발생 회수를 기록하기 위해, 정수 항목으로 인덱스 되는 카운트들의 배열을 사용하기 때문이다.
  - 카운트들을 위한 충분한 공간을 할당하려면 집합 내의 가장 큰 정수를 알아야 한다.

- **시간 복잡도**
  - O(n + k) : n은 배열의 길이, k는 정수의 최대값

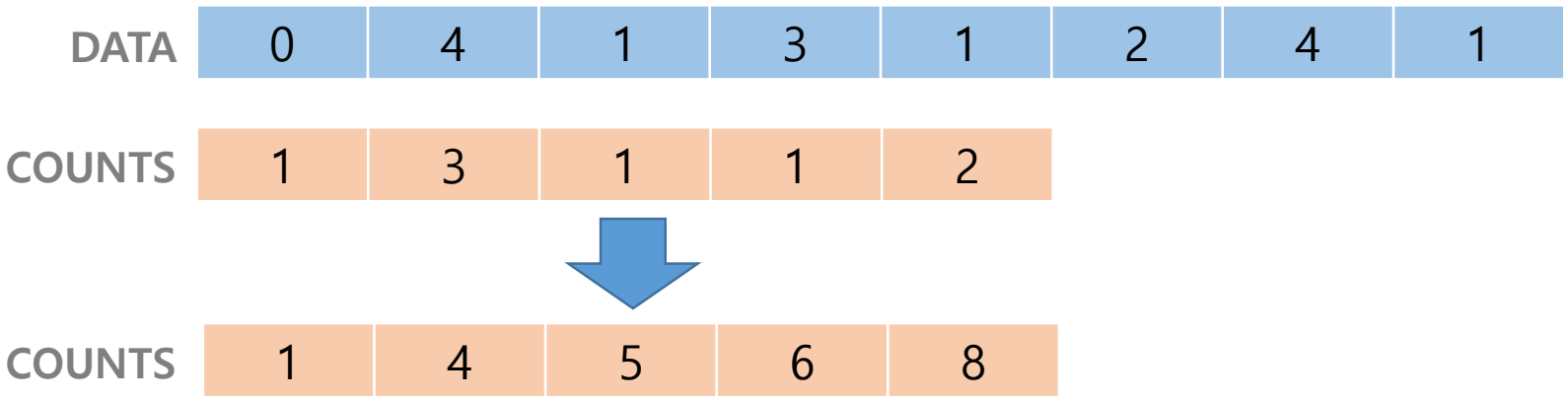
카운팅 정렬(Counting Sort)

- [0, 4, 1, 3, 1, 2, 4, 1] 을 카운팅 정렬하는 과정
- 1단계
  - Data에서 각 항목들의 발생 회수를 세고, 정수 항목들로 직접 인덱스 되는 카운트 배열 counts에 저장한다.



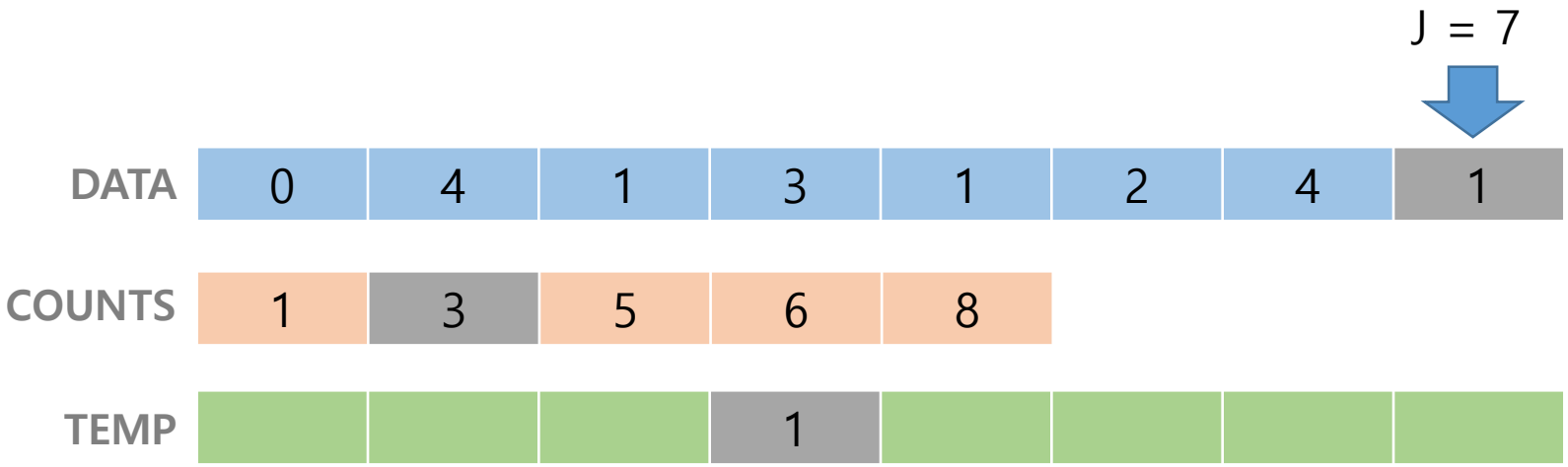
카운팅 정렬(Counting Sort)

- 정렬된 집합에서 각 항목의 앞에 위치할 항목의 개수를 반영하기 위해 counts의 원소를 조정한다.



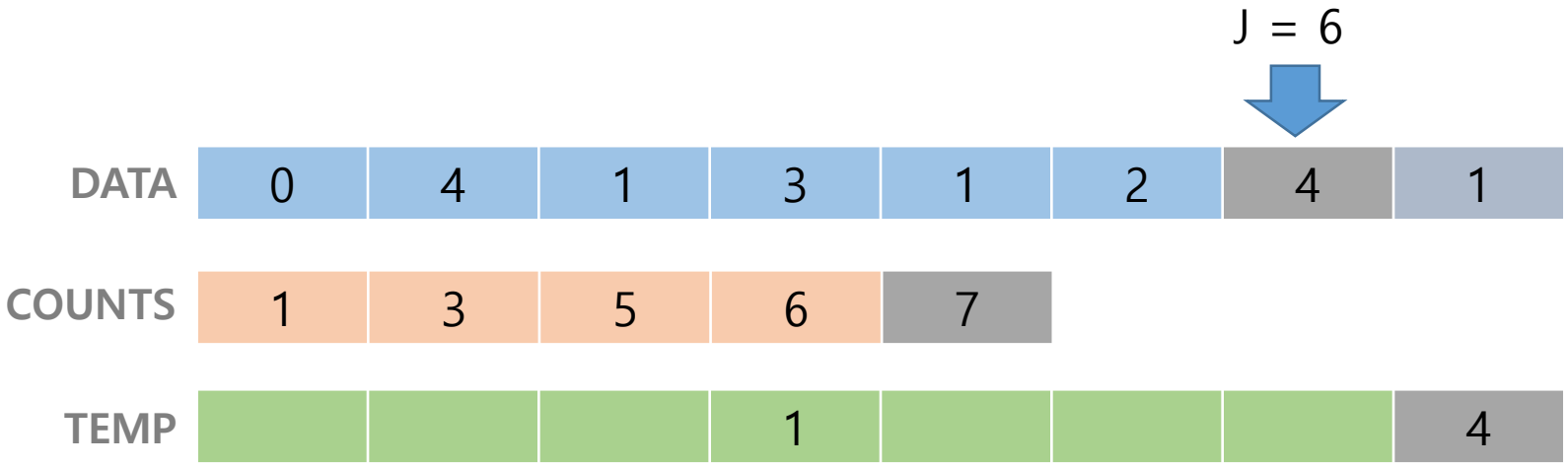
카운팅 정렬(Counting Sort)

- counts[1]을 감소시키고 Temp에 1을 삽입한다.



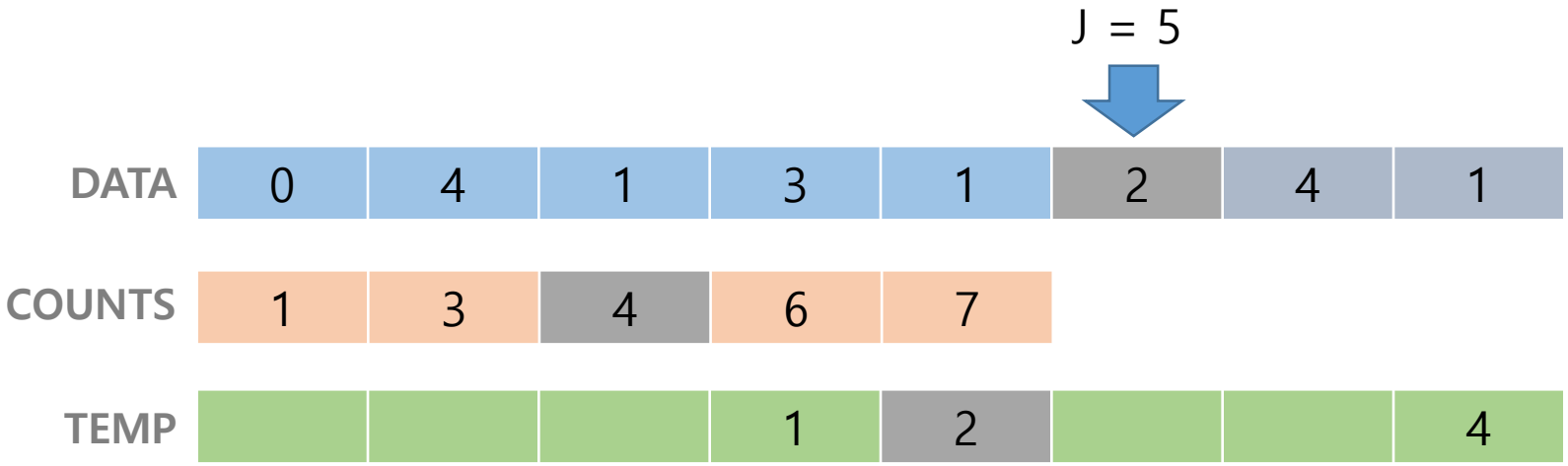
카운팅 정렬(Counting Sort)

- counts[4]를 감소시키고 Temp에 4를 삽입한다.



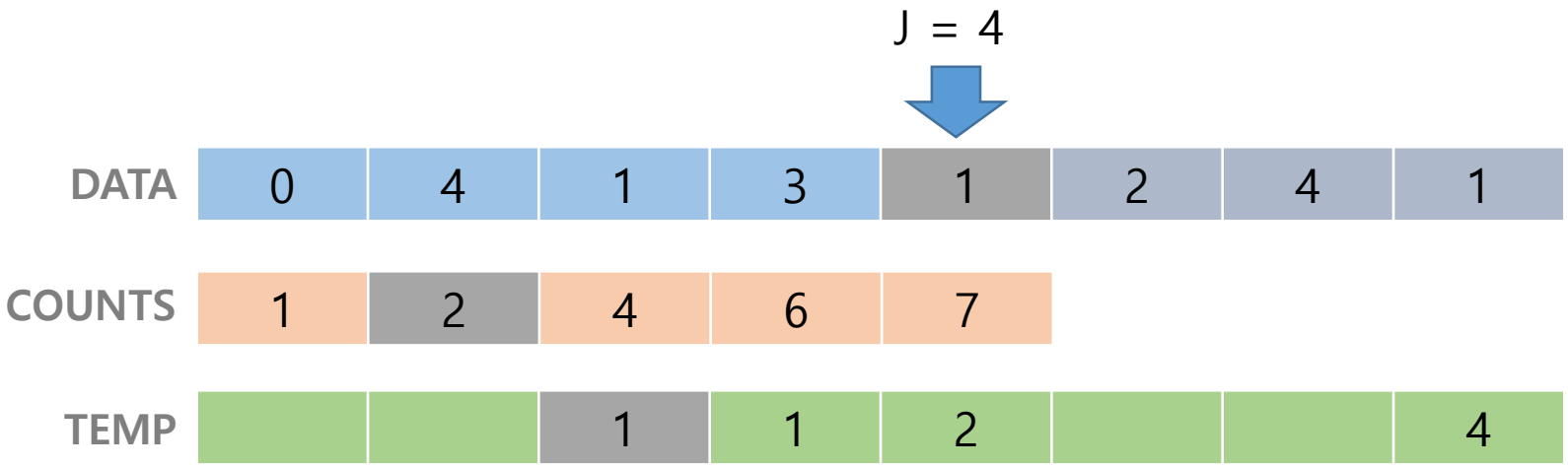
카운팅 정렬(Counting Sort)

- counts[2]를 감소시키고 Temp에 2를 삽입한다.



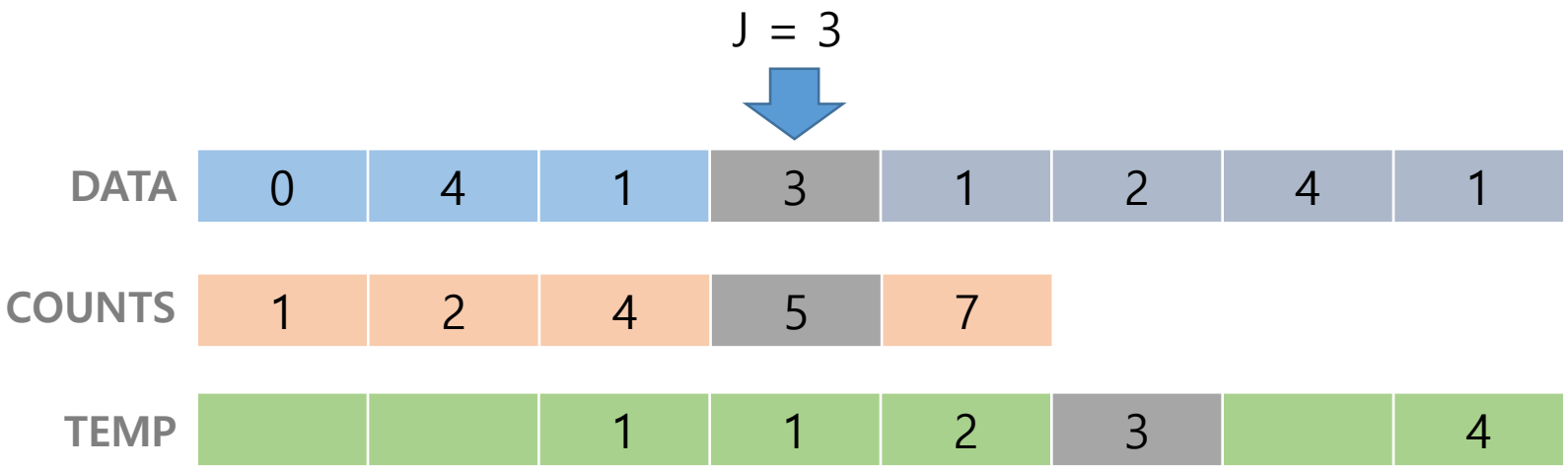
카운팅 정렬(Counting Sort)

- counts[1]을 감소시키고 Temp에 1을 삽입한다.



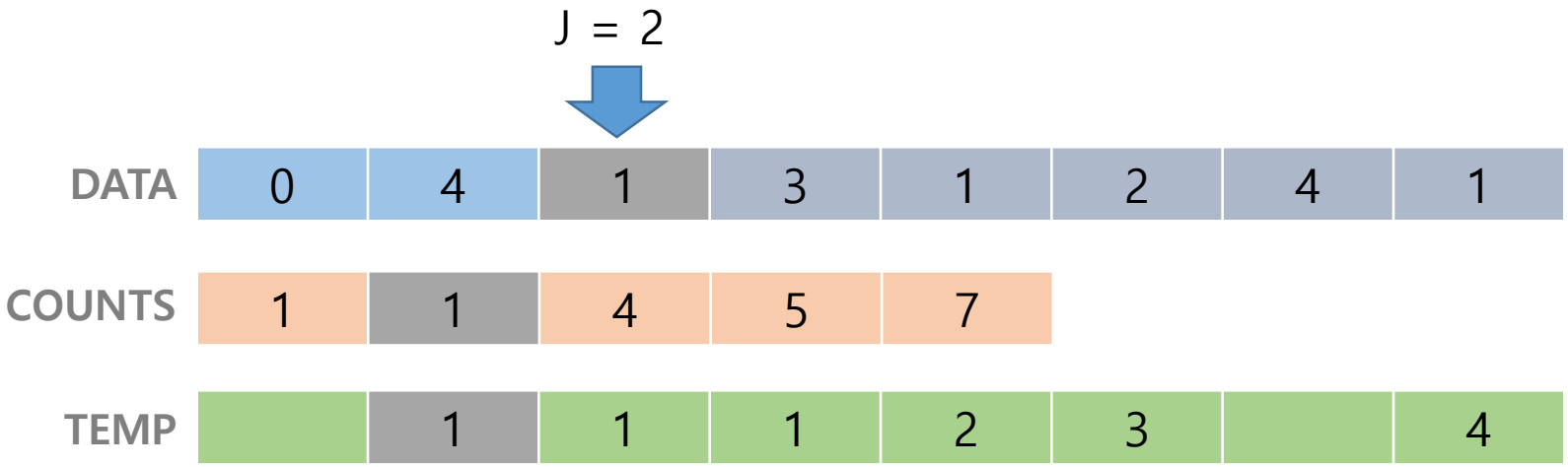
카운팅 정렬(Counting Sort)

- counts[3]을 감소시키고 Temp에 3을 삽입한다.



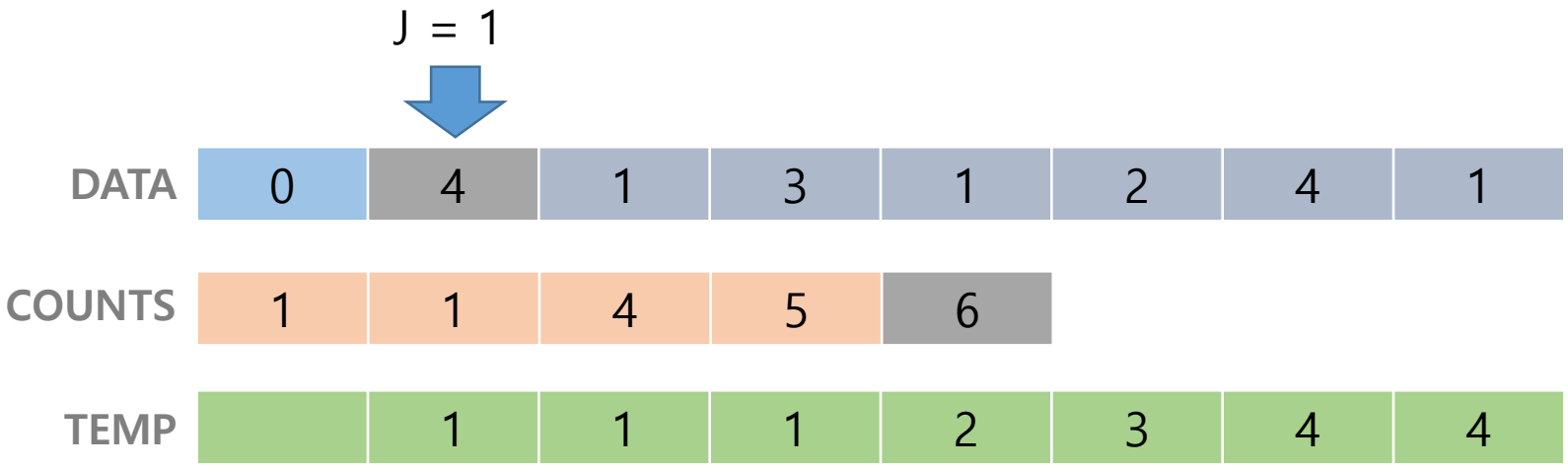
카운팅 정렬(Counting Sort)

- counts[1]을 감소시키고 Temp에 1을 삽입한다.



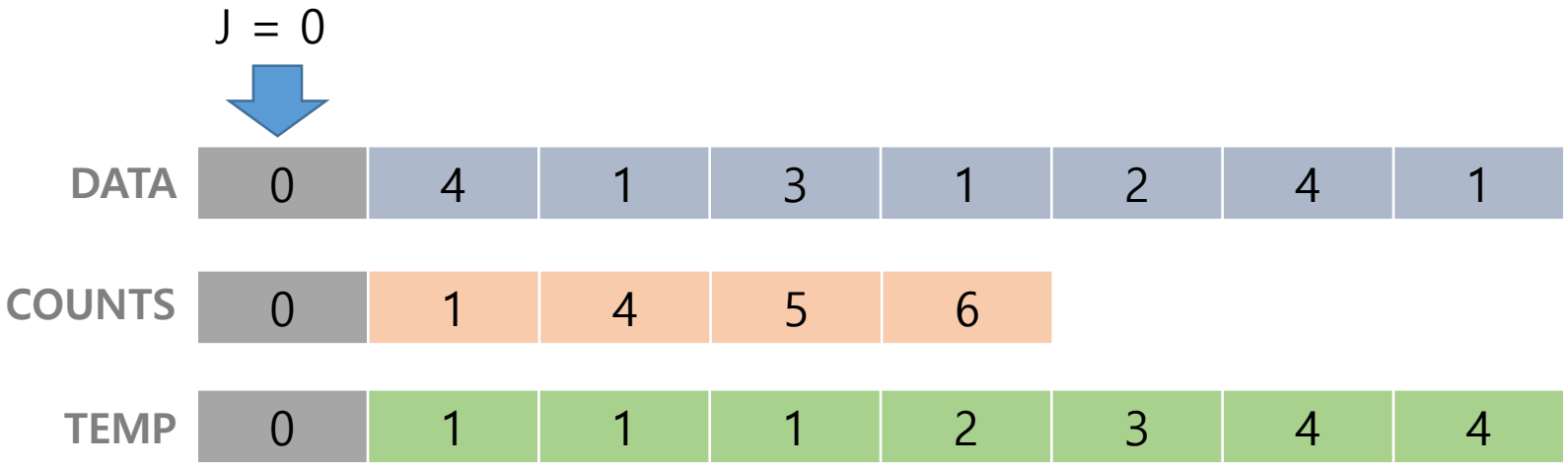
카운팅 정렬(Counting Sort)

- counts[4]를 감소시키고 Temp에 4를 삽입한다.



카운팅 정렬(Counting Sort)

- counts[0]을 감소시키고 Temp에 0을 삽입한다.



- Temp 업데이트 완료하고 정렬 작업을 종료한다.

카운팅 정렬(Counting Sort)

```
Counting_Sort(A, B, k)
// A [] -- 입력 배열
// B [] -- 정렬된 배열.
// C [] -- 카운트 배열.
// k    -- 최대 값, n : 입력 배열 길이

C = new int[k];

for i from 0 to n
    C[A[i]] += 1; //개수세기

for i from 1 to k
        C[i] += C[i-1] //누적합

for i from n-1 to 0
        B[C[A[i]]-1] = A[i]
        C[A[i]] -= 1
```

카운팅 정렬(Counting Sort)

- 카운팅 정렬을 항상 사용 하는게 좋은가??
  - {1, 2, 10억, 1} 을 정렬하고 했을 때 메모리 낭비가 발생
- 카운팅 정렬은 안정정렬이라고 하는데 무슨 뜻인가??
  - 같은 값을 가지는 복수의 원소들이 정렬 후에도 정렬 전과 같은 순서를 가진다.

0	1	2	3	4
0	2^A	2^B	1	3

카운팅 정렬(Counting Sort)

복수의 원소를 카운팅 정렬하자.

두번째 기준 정렬

0	1	2	3	4	5	6	7
(0 , 3)	(4 , 0)	(1 , 2)	(3 , 3)	(1 , 4)	(2 , 1)	(4 , 1)	(1 , 3)

0	1	2	3	4

0	1	2	3	4	5	6	7

카운팅 정렬(Counting Sort)

복수의 원소를 카운팅 정렬하자.

첫번째 기준 정렬

0	1	2	3	4	5	6	7
(4 , 0)	(2 , 1)	(4 , 1)	(1 , 2)	(0 , 3)	(3 , 3)	(1 , 3)	(1 , 4)

0	1	2	3	4

0	1	2	3	4	5	6	7

카운팅 정렬(Counting Sort)

- 복수의 원소를 카운팅 정렬하자.

정렬 끝

0	1	2	3	4	5	6	7
(0 , 3)	(1 , 2)	(1 , 3)	(1 , 4)	(2 , 1)	(3 , 3)	(4 , 0)	(4 , 1)

정렬 알고리즘 비교

- 학습한 정렬 알고리즘의 특성을 다른 정렬들과 비교해보자.

알고리즘	평균 수행시간	최악 수행시간	알고리즘 기법	비고
버블 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	코딩이 가장 손쉽다.
선택 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	교환의 회수가 버블, 삽입정렬보다 작다.
카운팅 정렬	O(n+k)	O(n+k)	비교환 방식	n이 비교적 작을 때만 가능하다.
삽입 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	n의 개수가 작을 때 효과적이다.
병합 정렬	O(n log n)	O(n log n)	분할 정복	연결리스트의 경우 가장 효율적인 방식
퀵 정렬	O(n log n)	$O(n^2)$	분할 정복	최악의 경우 $O(n^2)$ 이지만, 평균적으로는 가장 빠르다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미