

# Counting Sort

(주)한컴에듀케이션 이주현

# $O(n)$ 정렬

- ❖ 비교정렬의 하한은  $O(n \log n)$ 이지만 입력된 자료의 원소가 제한적인 성질을 만족하는 경우  $O(n)$ 정렬이 가능하다.
- ❖ 계수 정렬(Counting Sort)
  - 원소의 범위가  $k$  (  $-O(n) \sim O(n)$  ) 범위의 정수인 경우
- ❖ 기수 정렬(Radix Sort)
  - 원소의 자리수가  $d$  이하의 자리인 경우 ( $d$  : 상수로 간주될 크기)

# 계수 정렬(Counting Sort)

- ❖ 원소의 범위가 제한적일때 원소의 개수를 세어 정렬하는 방법이다.
- ❖ 개수를 셀 배열과 정렬된 결과가 저장될 배열이 추가로 필요하다.
- ❖ 일반적으로 정수 또는 문자를 정렬하는 경우에 많이 사용될 수 있다.
- ❖ 다양한 구현 방법이 있다.  
여기서는 3가지 방법으로 구현해본다.

# 1. 계수정렬(counting sort) 구현 1

- ❖ 단순히 정렬하는 방법이다.
- ❖ 수를 세는 cnt[]배열이 추가적으로 필요하다.
- ❖ 정렬된 결과를 A[]에 저장할 수 있다.
- ❖ A[]배열에 아래와 같은 수들이 담겨있다고 하자.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

# 계수 정렬 예제

❖ A[]에 담긴 원소의 개수를 세어 cnt[]에 저장한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

```
N = 10, k = 6;  
for(i=0; i < k; ++i) cnt[i] = 0;    /// initialize cnt array  
for(i=0; i < N; ++i) cnt[A[i]] ++;  /// counting
```

❖ cnt[]에 담긴 결과이다.

index	0	1	2	3	4	5
cnt[i]	0	3	2	3	1	1



# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	2->1	2	3	1	1

```
int i, j = 0;
for( i = 1; i <= 5; ++i){  /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1								

# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	1->0	2	3	1	1

```
int i, j = 0;
for( i = 1; i <= 5; ++i){  /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1							



# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	0	2->1	3	1	1

```
int i, j = 0;
for( i = 1; i <= 5; ++i){    /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2						

# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	0	1->0	3	1	1

```
int i, j = 0;
for( i = 1; i <= 5; ++i){    /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2					

# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	0	0	3->0	1	1

```
int i, j = 0;
for( i = 1; i <= 5; ++i){    /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3		

# 계수 정렬 예제 cont.

❖ cnt[]만을 이용하여 정렬된 결과를 A[]에 저장한다.

index	0	1	2	3	4	5
cnt[i]	0	0	0	0	1->0	1->0

```
int i, j = 0;
for( i = 1; i <= 5; ++i){  /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3	4	5

# 계수 정렬 예제 cont.

❖ A[]에 정렬된 결과가 저장되었다.

index	0	1	2	3	4	5
cnt[i]	0	0	0	0	0	0

```
int i, j = 0;
for( i = 1; i <= 5; ++i){  /// sorting
    while(cnt[i] --)
        A[j++] = i;
}
```

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3	4	5

# 계수정렬(counting sort) – code sample 1

```
// 원소의 범위 k ( 0 ~ 5) 라고 가정
int A[10], cnt[10], k = 6;
void countingSort(int A[], int N){
    int i, j = 0;
    for(i=0; i< N; ++i) cnt[i] = 0;      // init
    for(i=0; i< N; ++i) cnt[A[i]] ++;    // counting
    for(i=0; i< k; ++i){                 // sorting
        while(cnt[i]){
            A[j++] = i;
        }
    }
}
```

## 2. 계수정렬(counting sort) 구현 2

- ❖ 기수 정렬(Radix Sort)와 연관된 버전으로 stable(안정적)하게 정렬하는 방법이다.
- ❖ cnt[]와 sorted[]라는 배열이 추가적으로 필요하다.  
: stable & out-of-place 정렬
- ❖ A[]배열에 아래와 같은 수들이 담겨있다고 하자.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

# 계수 정렬 예제

❖ A[]에 담긴 원소의 개수를 세어 cnt[]에 저장한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

```
N = 10, k = 6;  
for(i=0; i < k; ++i) cnt[i] = 0;           // initialize cnt array  
for(i=0; i < N; ++i) cnt[A[i]] ++;         // counting
```

❖ cnt[]에 담긴 결과이다.

index	0	1	2	3	4	5
cnt[i]	0	3	2	3	1	1



# 계수 정렬 예제 cont.

❖ cnt[]에 담긴 결과이다.

index	0	1	2	3	4	5
cnt[i]	0	3	2	3	1	1

```
for(i=1;i<k;++i) cnt[i] += cnt[i-1]; // accumulate
```

❖ cnt[]에 누적합을 구한다.

index	0	1	2	3	4	5
cnt[i]	0	3	5	8	9	10

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	3->2	5	8	9	10

```
for(i= N -1;i>=0;--i){ // sorting
    sortedA[--cnt[A[i]]] = A[i];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1							

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	5	8->7	9	10

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[8]]] = A[8];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1					3		

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	5	7	9	10->9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[7]]] = A[7];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1					3		5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	5	7->6	9	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[6]]] = A[6];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1				3	3		5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	5	6	9->8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[5]]] = A[5];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1				3	3	4	5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	5->4	6	8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[4]]] = A[4];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1		2		3	3	4	5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	4->3	6	8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[3]]] = A[3];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1	2	2		3	3	4	5



# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2	3	6->5	8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[2]]] = A[2];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]			1	2	2	3	3	3	4	5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	2->1	3	5	8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[1]]] = A[1];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]		1	1	2	2	3	3	3	4	5

# 계수 정렬 예제 cont.

❖ cnt[]과 A[]를 이용하여 정렬된 결과 sortedA[]를 구한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	1 → 0	3	5	8	9

```
for(i= N -1 ~ 0){ // sorting
    sortedA[--cnt[A[0]]] = A[0];
}
```

index	0	1	2	3	4	5	6	7	8	9
sortedA[i]	1	1	1	2	2	3	3	3	4	5

## 계수정렬(counting sort) – code sample 2

```
// 원소의 범위 k ( 0 ~ 5 ) 라고 가정
int A[10], sortedA[10], cnt[10], k = 6;
void countingSort(int A[], int N){
    int i ;
    for(i=0; i< k; ++i) cnt[i] = 0;           // initialize cnt array
    for(i=0; i< N; ++i) cnt[A[i]] ++;         // counting
    for(i=1; i< k; ++i) cnt[i] += cnt[i-1];    // accumulate
    for(i= N -1; i>=0; --i){                  // stable sorting
        sortedA[--cnt[ A[i] ] ] = A[i];
    }
}
```

### 3. 계수정렬(counting sort) 구현 3

- ❖ 기수 정렬(Radix Sort)과 연관된 버전으로 정렬하는 경우 안정적인 정렬을 위하여 sortedA[]라는 추가적인 배열이 필요하다. 그런데 정렬된 각 수들에 여분의 비트가 있다면 이를 이용하여 추가 배열을 사용하지 않고 정렬할 수 있다.
- ❖ unstable sort이다. in-place sort이다.
- ❖ A[]배열에 아래와 같은 수들이 담겨있다고 하자.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

# 계수 정렬 예제

❖ A[]에 담긴 원소의 개수를 세어 cnt[]에 저장한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

```
N = 10, k = 6;  
for(i=0; i < k; ++i) cnt[i] = 0;           // initialize cnt array  
for(i=0; i < N; ++i) cnt[A[i]] ++;         // counting
```

❖ cnt[]에 담긴 결과이다.

index	0	1	2	3	4	5
cnt[i]	0	3	2	3	1	1

# 계수 정렬 예제 cont.

❖ cnt[]에 담긴 결과이다.

index	0	1	2	3	4	5
cnt[i]	0	3	2	3	1	1

```
for(i=1;i<k;++i) cnt[i] += cnt[i-1]; // accumulate
```

❖ cnt[]에 누적합을 구한다.

index	0	1	2	3	4	5
cnt[i]	0	3	5	8	9	10

# 계수 정렬 예제 cont.

- ❖ A[9]에 있는 1은 정렬된 후에 A[2]에 위치한다.  
그런데 A[2]에는 3이 있다. 따라서 두 수를 교환하고 수 1에 FLAG를 달아 둔다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	3	2	2	4	3	5	3	1

index	0	1	2	3	4	5
cnt[i]	0	3->2	5	8	9	10

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```



# 계수 정렬 예제 cont.

- ❖ 1은 정렬이 끝난(자기 자리를 찾은) 상태이고 정렬할(자기 자리를 찾아줄) 수는 3이 되었다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	4	3	5	3	1 <sup>-&gt;3</sup>

index	0	1	2	3	4	5
cnt[i]	0	3 <sup>-&gt;2</sup>	5	8	9	10

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[9]에 있는 3은 정렬된 후에 A[7]에 위치한다.  
그런데 A[7]에는 5가 있다. 따라서 두 수를 교환하고 수 3에 FLAG를 달아 둔다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	4	3	5	3	3

index	0	1	2	3	4	5
cnt[i]	0	2	5	8→7	9	10

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ 3은 정렬이 끝난(자기 자리를 찾은) 상태이고 정렬할(자기 자리를 찾아줄) 수는 5가 되었다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	4	3	3F	3	3->5

index	0	1	2	3	4	5
cnt[i]	0	2	5	7	9	10

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[9]에 있는 5는 정렬된 후에 A[9]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	4	3	3F	3	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	7	9	10->9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[8]에 있는 3은 정렬된 후에 A[6]에 위치한다.  
그런데 A[6]에는 3이 있다. 따라서 두 수를 교환하고 수 3에 FLAG를 달아 둔다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	4	3	3 <sup>F</sup>	3	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	7→6	9	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ 3은 정렬이 끝난(자기 자리를 찾은) 상태이고 정렬할(자리 자리를 찾아줄) 수는 그대로 3이 되었다. 같은 수이지만 과정의 단순화를 위하여 그대로 진행한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	4	3F	3F	3->3	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	7->6	9	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[8]에 있는 3은 정렬된 후에 A[5]에 위치한다.  
그런데 A[5]에는 4가 있다. 따라서 두 수를 교환하고 3에 FLAG를 달아 둔다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	4	3 <sup>F</sup>	3 <sup>F</sup>	3	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	6->5	9	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ 3은 정렬이 끝난(자기 자리를 찾은) 상태이고 정렬할(자기 자리를 찾아줄) 수는 4가 되었다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	3 <sup>F</sup>	3 <sup>F</sup>	3 <sup>F</sup>	3 <sup>F</sup> →4	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	6→5	9	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```



# 계수 정렬 예제 cont.

- ❖ A[8]에 있는 4는 정렬된 후에 A[8]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <sup>F</sup>	2	2	3 <sup>F</sup>	3 <sup>F</sup>	3 <sup>F</sup>	4	5


  

index	0	1	2	3	4	5
cnt[i]	0	2	5	5	9→8	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[7]에 있는 3은 이전에 정렬된 수이므로 FLAG를 제거하고 i를 감소시켜 정렬 대상을 변경한다.

										
index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	3F	3F	3F	4	5


  

index	0	1	2	3	4	5
cnt[i]	0	2	5	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[6]에 있는 3은 이전에 정렬된 수이므로 FLAG를 제거하고 i를 감소시켜 정렬 대상을 변경한다.

	index	0	1	2	3	4	5	6	7	8	9
	A[i]	1	1	1 <sup>F</sup>	2	2	3 <sup>F</sup>	3 <sup>F</sup>	3	4	5


  

index	0	1	2	3	4	5
cnt[i]	0	2	5	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[5]에 있는 3은 이전에 정렬된 수이므로 FLAG를 제거하고 i를 감소시켜 정렬 대상을 변경한다.

										
index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	3F	3	3	4	5

index	0	1	2	3	4	5
cnt[i]	0	2	5	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[4]에 있는 2는 정렬된 후에 A[4]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	3	3	3	4	5

index	0	1	2	3	4	5
cnt[i]	0	2	5->4	5	8	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[3]에 있는 2는 정렬된 후에 A[3]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1F	2	2	3	3	3	4	5



  

index	0	1	2	3	4	5
cnt[i]	0	2	4->3	5	8	9

```
constexpr int FLAG = 1 << 20 ; // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){ // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i!=j; j = --cnt[A[i]]){
            int tmp = A[j]; // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG; // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp; // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[2]에 있는 1은 이전에 정렬된 수이므로 FLAG를 제거하고 i를 감소시켜 정렬 대상을 변경한다.


										
index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1 <del>F</del>	2	2	3	3	3	4	5

index	0	1	2	3	4	5
cnt[i]	0	2	3	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

- ❖ A[1]에 있는 1은 정렬된 후에 A[1]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.



index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3	4	5

index	0	1	2	3	4	5
cnt[i]	0	2->1	3	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```



# 계수 정렬 예제 cont.

- ❖ A[0]에 있는 1은 정렬된 후에 A[0]에 위치한다.  
이미 정렬된 상태이므로 i를 감소시켜 정렬 대상을 변경한다.

index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3	4	5


  

index	0	1	2	3	4	5
cnt[i]	0	1 → 0	3	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수 정렬 예제 cont.

❖ 정렬이 완성되었다.

										
index	0	1	2	3	4	5	6	7	8	9
A[i]	1	1	1	2	2	3	3	3	4	5

index	0	1	2	3	4	5
cnt[i]	0	0	3	5	8	9

```
constexpr int FLAG = 1 << 20 ;    // 배열의 각 수에 영향을 주지 않는 비트로 설정
int i, j;
for(i = N - 1; i >= 0; --i){      // sorting
    if(A[i] & FLAG) A[i] ^= FLAG; // 이미 정렬된 수이므로 원래 값으로 복원
    else{
        for(j = --cnt[A[i]]; i != j; j = --cnt[A[i]]){
            int tmp = A[j];        // A[i]가 갈 자리에 있던 수를 임시로 tmp에 저장
            A[j] = A[i] | FLAG;    // 정렬된 위치에 표시를 달아 저장
            A[i] = tmp;            // 정렬해야 하는 새로운 수
        }
    }
}
```

# 계수정렬(counting sort) – code sample 3

```
constexpr int FLAG = 1 << 20;
int N, A[10], cnt[10], k = 6; // 원소의 범위 k ( 0 ~ 5 ) 라고 가정
void countingSort(int A[], int N){
    int i, j;
    for(i=0; i< k; ++i) cnt[i] = 0;           // initialize cnt array
    for(i=0; i< N; ++i) cnt[A[i]] ++;         // counting
    for(i=1; i< k; ++i) cnt[i] += cnt[i-1];   // accumulate
    for(i = N -1; i >= 0; --i){               // unstable sorting
        if(A[i] & FLAG) A[i] ^= FLAG;
        else {
            for(j = --cnt[A[i]]; i!=j; j = -- cnt[A[i]]){
                int tmp = A[j];
                A[j] = A[i] | FLAG;
                A[i] = tmp;
            }
        }
    }
}
```

감사합니다.