정렬

- 간단한 정렬 알고리즘

  선택정렬, 버블정렬, 삽입정렬
- 분할과 정복 (Divide and Conquer) 방법
- 분할과 정복에 의한 알고리즘

  병합정렬과 퀵정렬
- **힙정렬 (Heapsort)**
- **정렬 하한계**
- 기수정렬(Radix Sort)
- Counting Sort (계수 정렬)

- 두 원소의 (키) 비교에 의한 정렬 방법이 아닌 정렬 알고리즘: 기수정렬(radix sort), 계수정렬(counting sort)

Radix (기수) 정렬

- 두 원소의 (키) 비교에 의한 정렬 방법이 아닌 정렬 알고리즘:
  기수정렬(radix sort), 계수정렬(counting sort)

- 정렬할 자료들: $x_0, x_1, x_2, x_3, \ldots, x_{n-1}$
  (d: 정렬 자료의 최대 자리 개수)

  r: 진수

(각 자료의 가장 오른쪽 자릿수가 1번째 자릿수이다.)

- LSD(Least Significant Digit) Radix Sort

```
for (i = 1; i <= d; i++)
{
    bin[0], bin[1], bin[2], …, bin[r-1]을 초기화
    for (j = 0; j < n; j++)
        xj의 i번째 자릿수의 digit를 검사하여 xj를  bin[digit]에 넣는다.
    bin[0]부터 bin[r-1]까지 원소들을 차례로 모은다.
}
```

- 시간복잡도 O(d (n+r))

Radix sort (계속)

– 10 1234 9 7234 67 9181 733 197 7 3

Bin 먼저들어온 수가 먼저 나감.

분배하기 (가장 오른쪽 자리수 기준으로) – 큐를 이용

Bin[]

0	1	2	3	4	5	6	7	8	9
10	9181		3 733	7234 1234			7 197 67		9

모우기

– 10 9181 733 3 1234 7234 67 197 7 9

분배하기 (오른쪽에서 두번째 자리수 기준으로) – 큐를 이용

Bin[]

0	1	2	3	4	5	6	7	8	9
9 7 3	10		7234 1234 733			67		9181	197

모우기

– 3 7 9 10 733 1234 7234 67 9181 197

Radix sort (계속)

d번 만큼 루프가 돌게될 것 같다. d=4 이므로 4번만에 정렬됨.

- 3 7 9 10 733 1234 7234 67 9181 197

67 10 9 7 3	**197** **9181**	**7234** **1234**					**733**		
0	1	2	3	4	5	6	7	8	9

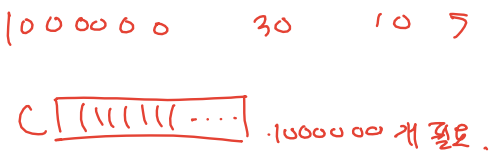
- 3 7 9 10 67 9181 197 1234 7234 733

733 197 67 10 9 7 3	**1234**						**7234**		**9181**
0	1	2	3	4	5	6	7	8	9

- 3 7 9 10 67 197 733 1234 7234 9181

계수 정렬 (Counting Sort)

- Counting sort:
  - No (key) comparisons!
  - 가정: input is in the range 0..k
  - Basic idea:
    - Count number of elements $k \le$ each element i
    - Use that number to place i in position k of sorted array
  - 수행시간: O(n + k)
  - Stable sort
  - Does not sort in place:
    - O(n) array to hold sorted output
    - O(k) array for scratch storage



계수정렬(Counting Sort)

◆ 가정:

정수 (0 ~ k)

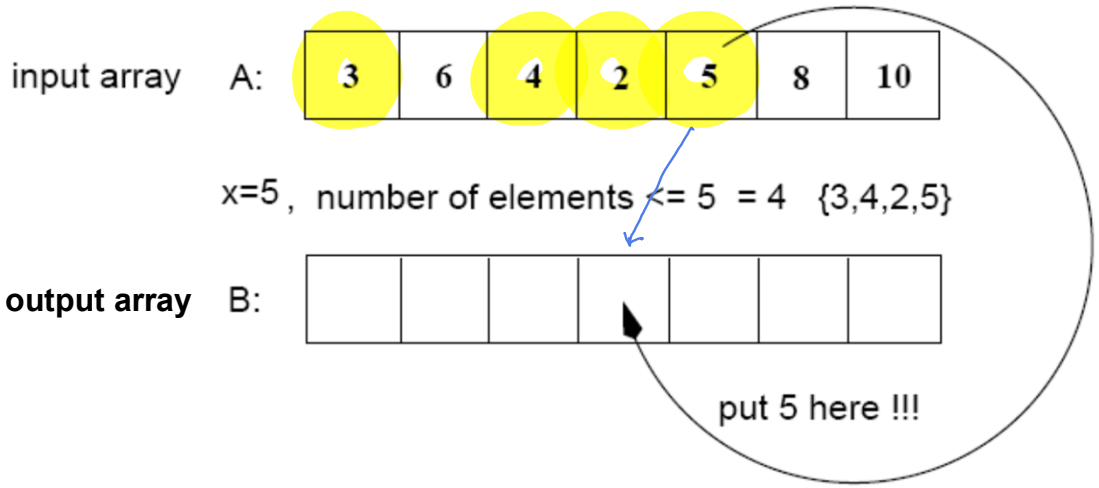
– n integers which are in the range [0 ... k]

↳ 최대값

◆ Idea:

– For each element x, find the number of elements ≤ x

– Place x into its correct position in the output array



계수정렬(Counting Sort)

```
Algorithm CountingSort(A, B, k)
// A: 정렬하고자 하는 리스트
// B: 정렬 결과를 저장하는 리스트
// C[i]: A에서 i보다 같거나 작은 수의 개수
    for i = 0 to k
        C[i]= 0
    for j = 0 to n-1
        C[A[j]] += 1
    for i = 1 to k
        C[i] = C[i] + C[i-1]
    for j = n-1 downto 0
        B[C[A[j]]-1] = A[j]
        C[A[j]] -= 1
```

K는 6까지가 최대값.

O(n) memory

O(k) 메모리

K의 최대값

원소 개수

for 구문돌면서 C[i] i가 나타나는 횟수 count 1증가

A

0	1	2	3	4	5	6	7
3	6	4	1	3	4	1	4

C

0	1	2	3	4	5	6
0	2	0	2	3	0	1

K=6이므로 0~6 배열길이

같거나 작은수.

C 누적합

4보다작거나같은거 7개

0	1	2	3	4	5	6
0	2	2	4	7	7	8

0 0+2 0+2+0 0+2+0+2 4+3 4+3+0 4+3+1=8

5보다작거나같은거 7개

6보다 작거나같은거 8개

B

0	1	2	3	4	5	6	7
						4	

A 7번째 4 (4보다 작거나 같은 수 8개)

C

0	1	2	3	4	5	6
0	2	2	4	6	7	8

B

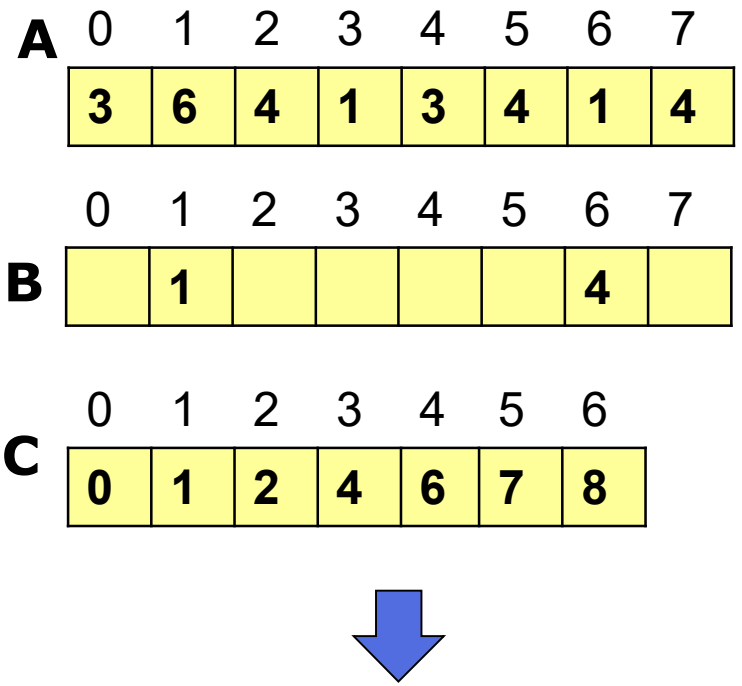
0	1	2	3	4	5	6	7
	1					4	

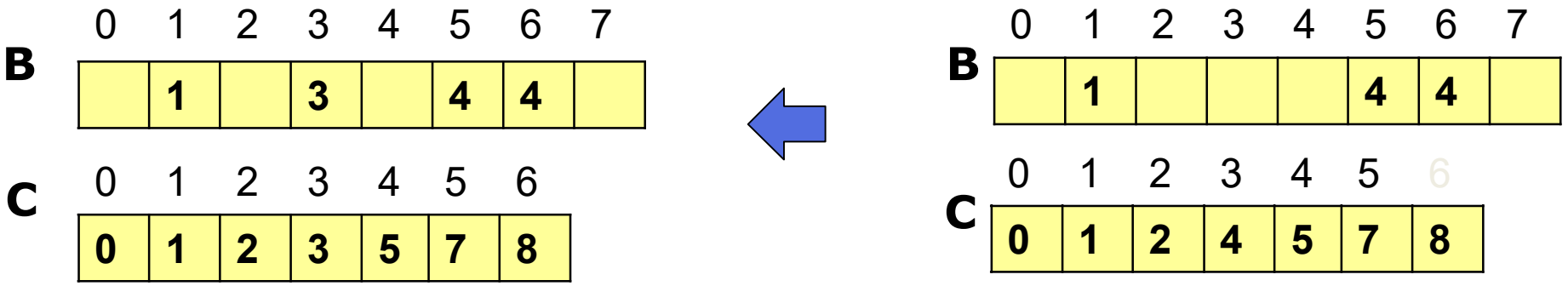
C

0	1	2	3	4	5	6
0	1	2	4	6	7	8

계수정렬(Counting Sort)

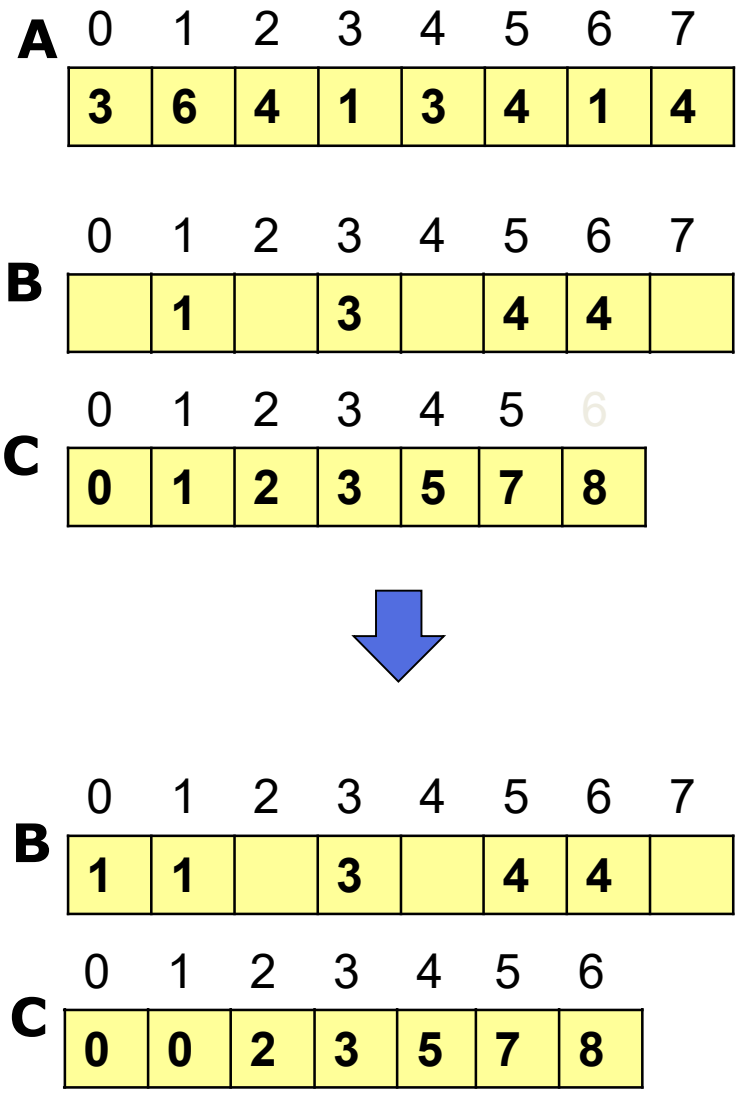
```
Algorithm CountingSort(A, B, k)
// A: 정렬하고자 하는 리스트
// B: 정렬 결과를 저장하는 리스트
// C[i]: A에서 i보다 같거나 작은 수의 개수
    for i = 0 to k
        C[i]= 0
    for j = 0 to n-1
        C[A[j]] += 1
    for i = 1 to k
        C[i] = C[i] + C[i-1]
    for j = n-1 downto 0
        B[C[A[j]]-1] = A[j]
        C[A[j]] -= 1
```





계수정렬(Counting Sort)

```
Algorithm CountingSort(A, B, k)
  for i = 0 to k
    C[i]= 0
  for j = 0 to n-1
    C[A[j]] += 1
  for i = 1 to k
    C[i] = C[i] + C[i-1]
  for j = n-1 downto 0
    B[C[A[j]]-1] = A[j]
    C[A[j]] -= 1
```



계수정렬(Counting Sort)

```
Algorithm CountingSort(A, B, k)
  for i = 0 to k
    C[i]= 0
  for j = 0 to n-1
    C[A[j]] += 1
  for i = 1 to k
    C[i] = C[i] + C[i-1]
  for j = n-1 downto 0
    B[C[A[j]]-1] = A[j]
    C[A[j]] -= 1
```