1 힙 정렬

1 힙 정렬

1 힙 정렬(Heap Sort)

- 힙이라는 특수한 자료구조를 사용하는 정렬 알고리즘
- 주어진 배열을 힙으로 만든 다음 차례로 하나씩 힙에서 제거함으로써 정렬함

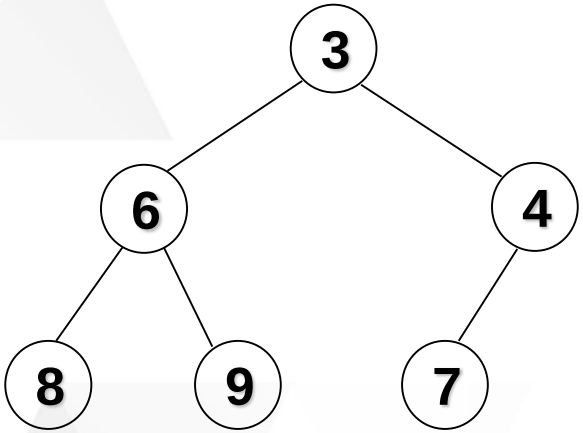
1 힙 정렬

2 힙(Heap)

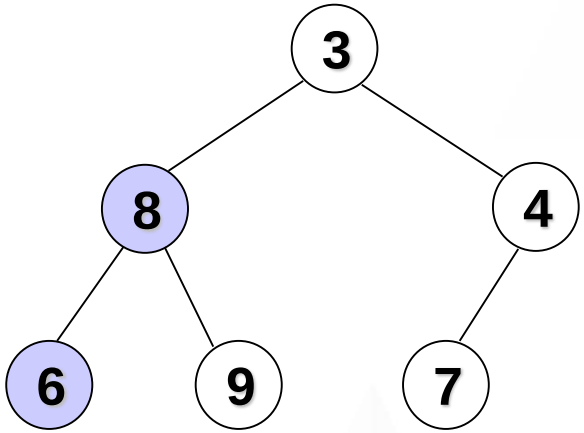
- 맨 아래 층을 제외하고는 완전히 채워져 있고
  맨 아래층은 왼쪽부터 꽉 채워져 있음(완전 이진 트리)
- 각 노드의 값은 자식의 값보다 작거나 같음
- 최대힙과 최소힙이 있음

1 힙 정렬

3 힙(최소힙)

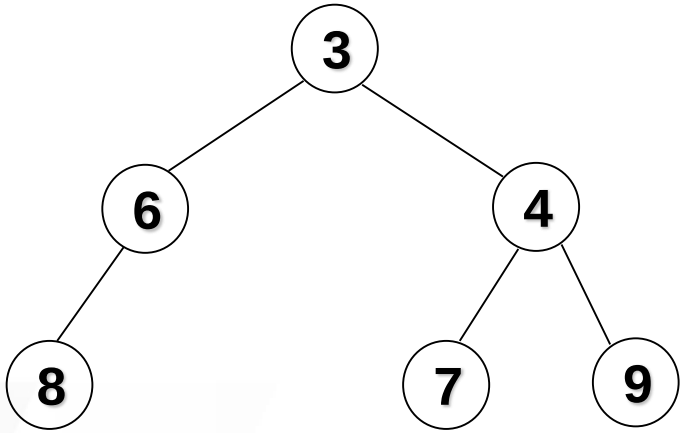


[힙]



[힙 아님]
(8이 자식의 값 6보다 큼)

3 힙(최소힙)



[힙 아님]
(**완전 이진 트리가 아님**)

1 힙 정렬

4 힙 정렬 방법

- 힙은 배열을 이용해서 표현할 수 있음
- 힙 정렬은 먼저 주어진 배열을 힙으로 만든 다음 힙에서 가장 작은 값을 차례로 하나씩 제거하면서 힙의 크기를 줄여나감
- 나중에 힙에 아무 원소도 남지 않으면 힙 정렬이 끝남
- 정렬은 힙에서 원소들이 제거된 순서대로 함

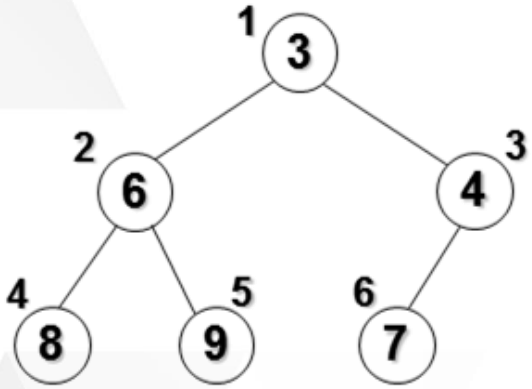
1 힙 정렬

4 힙 정렬 방법

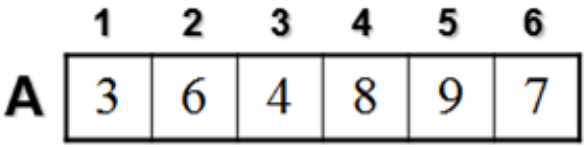
- 힙 정렬 과정

① 주어진 배열을 힙으로 만드는 과정
② 힙에서 최소 원소를 제거하고 나서 힙 성질을 만족하도록 수선해 주는 과정이 필요

4 힙 정렬 방법



[힙]



[힙을 배열로 표현]

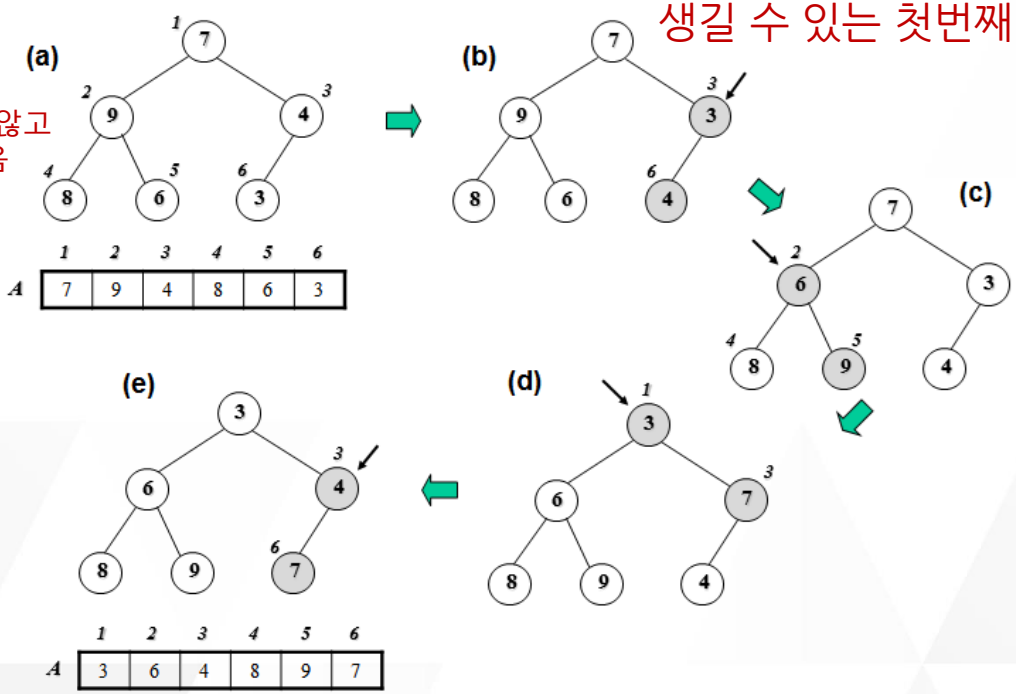
1 힙 정렬

5 힙 정렬의 작동 예(힙 만들기)

- 주어진 배열을 힙으로 만들기

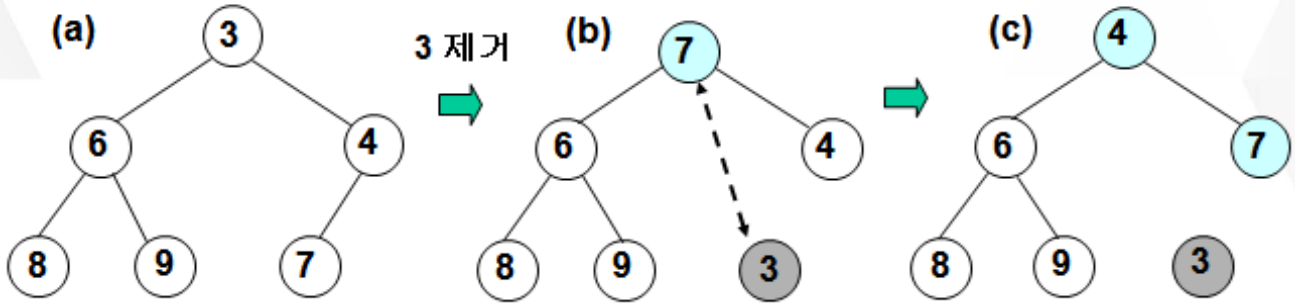
➔ 맨 뒤에서부터 따져 힙 성질에 문제가 생길 수 있는 첫번째 원소를 대상으로 체크

힙 성질 만족하지 않고 제멋대로 들어있음



5 힙 정렬의 작동 예

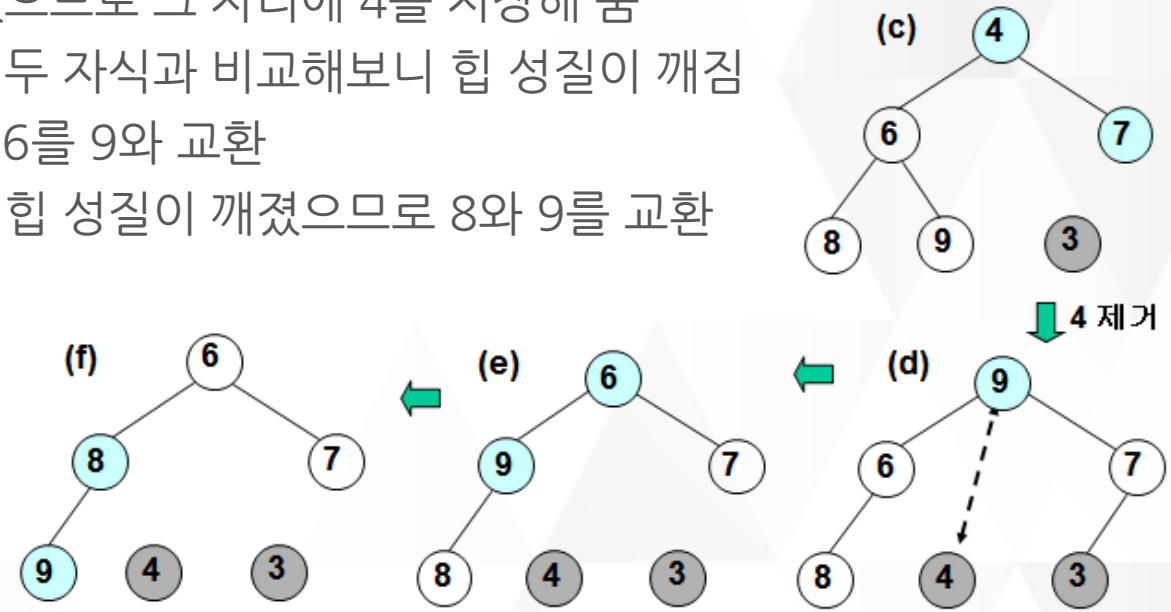
◆ 루트 노드의 원소를 제거하여 다른 곳에 저장하면서 정렬



① 루트에 있는 3 제거하고 배열의 맨 끝에 있는 7을 루트로 올림
② 7을 옮긴 자리가 비었으므로 그 자리에 3을 저장해 둠
③ 7이 루트로 올라가서 두 자식과 비교해보니 힙 성질이 깨짐
④ 두 자식 중 작은 값인 4를 7과 교환

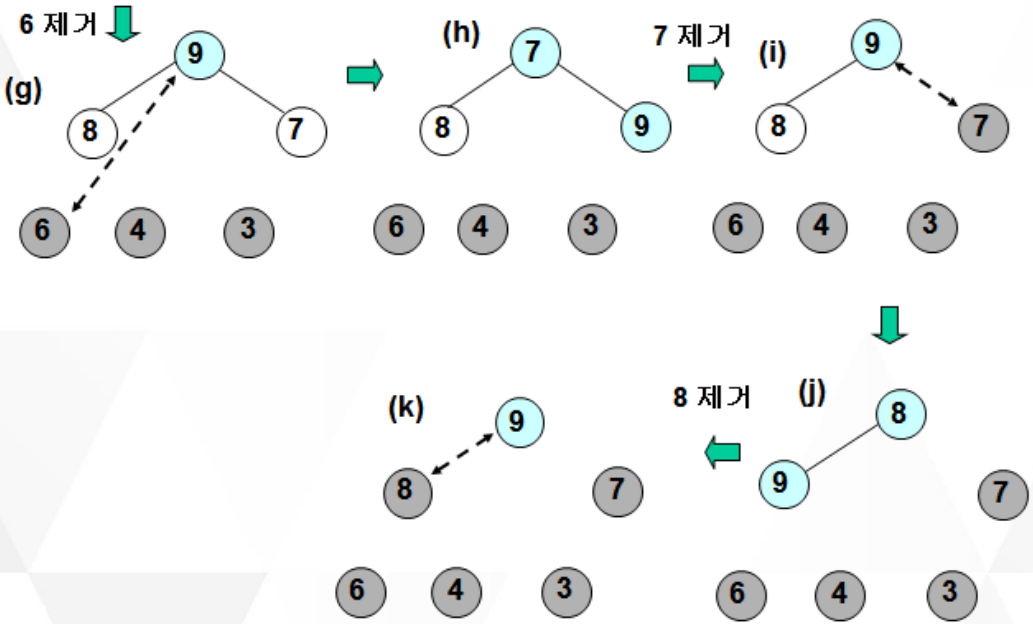
5 힙 정렬의 작동 예

⑤ 다시 루트에 있는 4 제거하고 배열의 맨 끝에 있는 9를 루트로 올림
⑥ 9를 옮긴 자리가 비었으므로 그 자리에 4를 저장해 둠
⑦ 9가 루트로 올라가서 두 자식과 비교해보니 힙 성질이 깨짐
⑧ 두 자식 중 작은 값인 6를 9와 교환
⑨ 9의 자식과 비교하여 힙 성질이 깨졌으므로 8와 9를 교환



5 힙 정렬의 작동 예

⑩ 나머지도 같은 방법으로 반복하여 정렬시킴



6 힙 정렬 알고리즘

```
heapSort(A[ ], n)
▷ A[1 ... n] 을 정렬한다
{
    buildHeap(A, n);        ▷ 힙 만들기
    for i ← n downto 2 {
        A[1] ↔ A[i];        ▷ 맨 마지막 원소와 루트 교환
        heapify(A, 1, i-1); ▷ A[1]을 루트로 하는 트리를 힙 성질을
                               만족하도록 수선함
    }
}
```

1 힙 정렬

7 수행시간

- 힙을 만드는 BuildHeap() 은 O(n)의 시간이 걸림
- for 루프는 (n-1)번 반복
- 힙의 높이는 log n이기 때문에 Heapify()는 충분히 잡아서 O(log n)의 시간이 듬
- 힙 정렬의 **총 수행시간은 O(n log n)** 임

  ➔ 최악의 경우에도 O(n log n) 시간 소요

2 기수 정렬

2 기수 정렬

1 기수 정렬(Radix Sort)

- 기수 정렬은 비교 정렬이 아니며 숫자를 부분적으로 비교하는 정렬 방법
- 입력이 모두 k 자릿수 이하의 자연수인 특수한 경우에 사용할 수 있는 방법
- 기(Radix)는 특정 진수를 나타내는 숫자들임
  - 예를 들어, 10진수의 기는 0, 1, 2, …, 9이고, 2진수의 기는 0, 1임

2 기수 정렬

1 기수 정렬(Radix Sort)

- 제한적인 범위 내에 있는 숫자에 대해서
  각 자릿수 별로 정렬하는 알고리즘

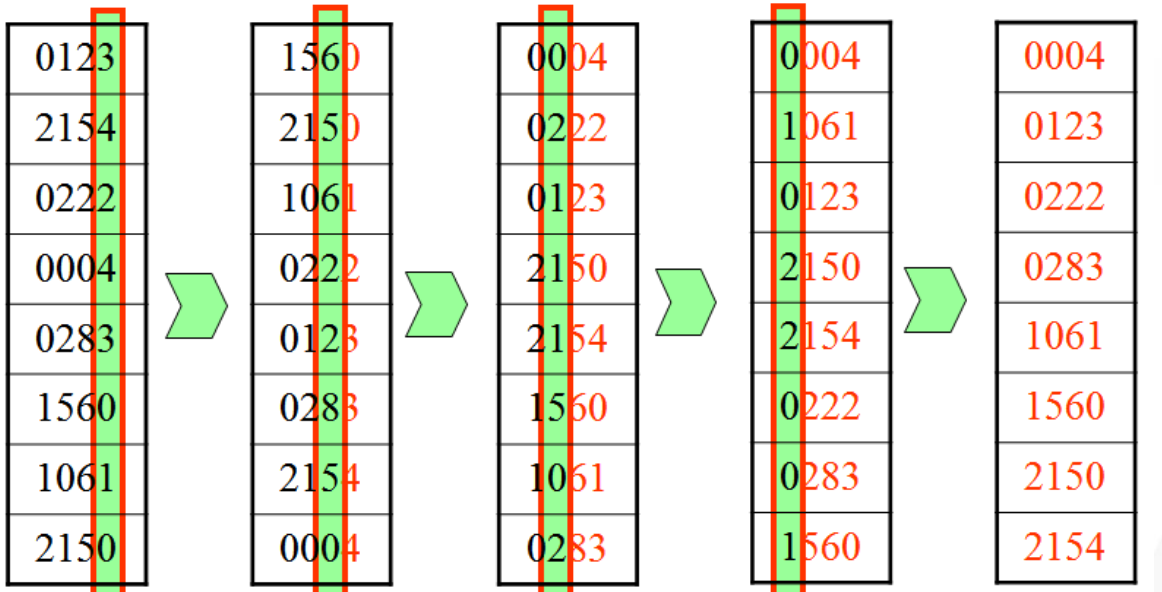
- 기수 정렬의 가장 큰 장점은
  어느 비교 정렬 알고리즘 보다 빠름

2 기수 정렬 방법

- 정렬하고자 하는 숫자들을 먼저 가장 낮은 자릿수만 가지고 모든 수를 재배열(정렬)함
- 그런 다음 가장 낮은 자릿수는 제외하고 나머지 자릿수에 대해 다시 앞과 같이 반복함
- 더 이상 자릿수가 남지 않을 때까지 계속하면 마지막에는 정렬된 배열을 갖게 됨

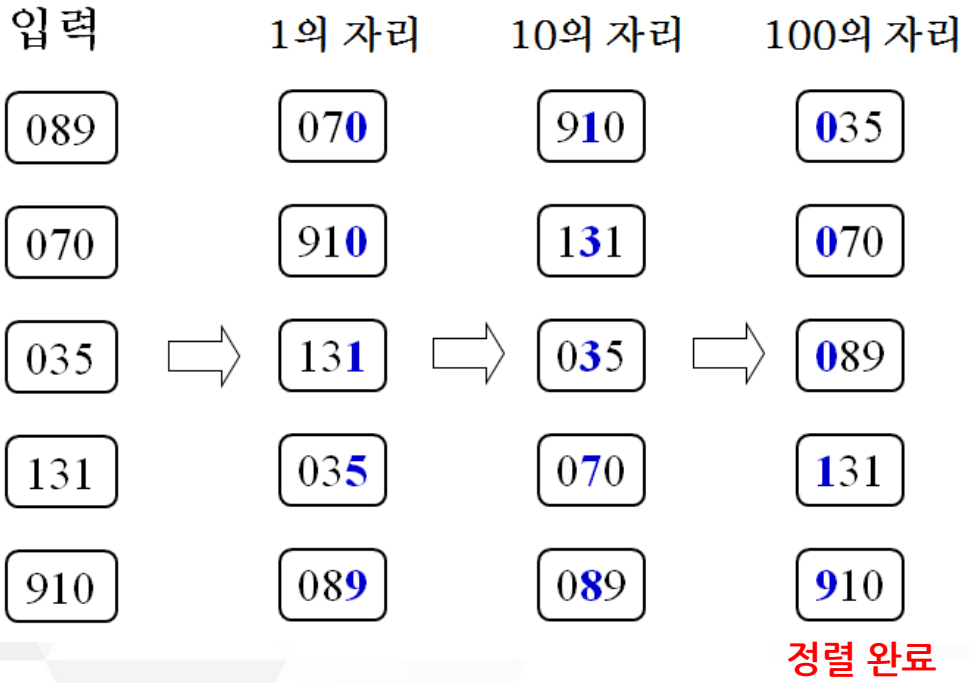
2 기수 정렬

3 기수 정렬의 작동 예



2 기수 정렬

3 기수 정렬의 작동 예



2 기수 정렬

4 기수 정렬 알고리즘

```
radixSort(A[ ], n, k)
▷ 원소들이 각각 최대 k 자리수인 A[1…n]을 정렬한다
▷ 가장 낮은 자리수를 1번째 자리수라 한다
{
      for i ← 1 to k
            i 번째 자리수에 대해 A[1…n]을 안정을 유지하면서 정렬한다;
}
```

- 기수 정렬 수행시간: O(n)

2 기수 정렬

5 안정성 정렬(Stable Sort)

- 값이 같은 원소끼리는 정렬 후에도 원래의 순서가 바뀌지 않는 성질을 가진 정렬

6 기수 정렬의 응용

- 기수 정렬은 계좌 번호, 날짜, 주민등록번호 등으로 대용량의 상용 데이터베이스 정렬, 랜덤 128 비트 숫자로 된 초대형 파일(예, 인터넷 주소)의 정렬, 지역 번호를 기반한 대용량의 전화 번호 정렬에 매우 적절함

- 다수의 프로세서들이 사용되는 병렬(Parallel) 환경에서의 정렬 알고리즘에 기본 아이디어로 사용되기도 함

3 계수 정렬

3 계수 정렬

1 계수 정렬(Counting Sort)

- 숫자가 등장한 횟수를 세서 그 기준으로 정렬하는 방법
- 데이터가 가질 수 있는 값의 범위(도메인)나,
  사전 지식을 바탕으로 정렬하는 알고리즘
- 계수를 이용하여 정렬하는 방법이며 배열에 저장된
  숫자를 세는 방법으로 숫자가 몇 개인지 기록함

3 계수 정렬

1 계수 정렬(Counting Sort)

- 범위 조건이 있는 경우에 굉장히 빠른 알고리즘

- 먼저 배열의 원소를 훑어보고 1부터 k까지의 자연수가 각각 몇 번 나타나는지를 셈

  ↳ 계수 정렬의 **수행시간은 O(n)** 임

3 계수 정렬

1 계수 정렬(Counting Sort)

- 선택 정렬, 버블 정렬, 삽입 정렬, 병합 정렬, 퀵 정렬, 힙 정렬들은 정렬하기 위해서 반드시 데이터 간의 값을 비교해야만 하는 비교 정렬 알고리즘
  (비교 정렬의 시간 복잡도 하한은 O(n logn) 임)
  - 계수 정렬은 비교 정렬이 아니며 데이터 간의 상대적 크기 관계를 이용하는 것

3 계수 정렬

2 계수 정렬의 작동 예

◆ 예) 100점 만점인 수학 시험을 응시한 사람 1,000명을 점수 순으로 정렬하는 문제

풀이)

- 모든 학생의 점수는 [0, 100] 사이의 값
- 학생들의 시험지를 0점끼리, 1점끼리, 2점끼리, …, 99점끼리, 100점끼리 묶어 놓고 0점부터 순서대로 나열하기만 하면 정렬이 끝남

3 계수 정렬

2 계수 정렬의 작동 예

◆ 예) 데이터 2, 5, 3, 0, 2, 3, 0, 3를 계수 정렬하시오.

풀이)
정렬할 데이터가 8개이므로 크기가 8인 배열이
필요하며 정렬할 데이터가 0부터 5까지의 수이므로
계수 정렬할 카운트의 인덱스는 0~5가 됨

배열

2	5	3	0	2	3	0	3

* 정렬하고자 하는 초기 데이터가 들어있음(0~5사이의 수)

3 계수 정렬

2 계수 정렬의 작동 예

Count

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

배열

정렬완료

0	0	2	2	3	3	3	5

3 계수 정렬 알고리즘

```
countingSort(A, B,  n)
▷ A[1…n]: 입력 배열
▷ B[1…n]: 배열 A를 정렬한 결과
{
        for i = 1 to k
                C[i] ← 0;
        for j = 1 to n
                C[A[j]]++;
        ▷ 이 시점에서의 C[i] : 값이 i인 원소의 총 수
        for i = 1 to k
                C[i] ← C[i] + C[i-1] ;
        ▷ 이 시점에서의 C[i] : i보다 작거나 같은 원소의 총 개수
        for j ← n downto 1 {
                B[C[A[j]] ← A[j];
                C[A[j]]--;
        }
}
```

3 계수 정렬

4 정렬 알고리즘의 효율성 비교

	Worst Case	Average Case
선택 정렬	n^2	n^2
버블 정렬	n^2	n^2
삽입 정렬	n^2	n^2
병합 정렬	nlogn	nlogn
퀵 정렬	n^2	nlogn
힙 정렬	nlogn	nlogn
기수 정렬	n	n
계수 정렬	n	n

3 계수 정렬

5 정렬 알고리즘을 선택할 때 고려사항

- 키값들의 분포 상태
- 소요공간 및 작업시간
- 정렬에 필요한 기억공간의 크기
- 데이터의 양
- 사용 컴퓨터 시스템의 특성