1 알고리즘의 복잡도

1 알고리즘의 복잡도

1 알고리즘 성능 분석 방법

- 알고리즘 중 가장 효율성이 좋은 것을 선택하기 위해서는 알고리즘의 수행 시간, 차지하는 기억 공간 등의 비용으로 판단
- 처리 속도는 빠르게, 저장 장소는 적게 사용하는 알고리즘
- 공간 복잡도와 시간 복잡도가 있음
- 알고리즘은 복잡도가 낮을수록 효율적

1 알고리즘의 복잡도

2 알고리즘의 수행시간

◆ 예) a = (a + b) / c + 1 의
연산 과정에 대한 실행 횟수를 구하시오

(풀이)
연산 순서
x = a + b
y = x/c
a = y + 1

그러므로
a = (a + b) / c + 1 의 연산 실행 횟수는 3 회

1 알고리즘의 복잡도

3 점근적 성능(Asymptotic Performance)

- 입력 크기가 작은 문제는 알고리즘의 효율성이 중요하지 않지만 입력 크기가 충분히 큰 문제에서 비효율적인 알고리즘은 치명적임

- 입력 크기 n이 무한대로 커질 때의 복잡도를 간단히 표현하기 위해 사용하는 표기법

- O(Big-Oh) 표기, Ω(Big-Omega) 표기, Θ(Big-Theta) 표기법이 있으며 주로 O(Big-Oh) 표기법을 사용함

3 점근적 성능(Asymptotic Performance)

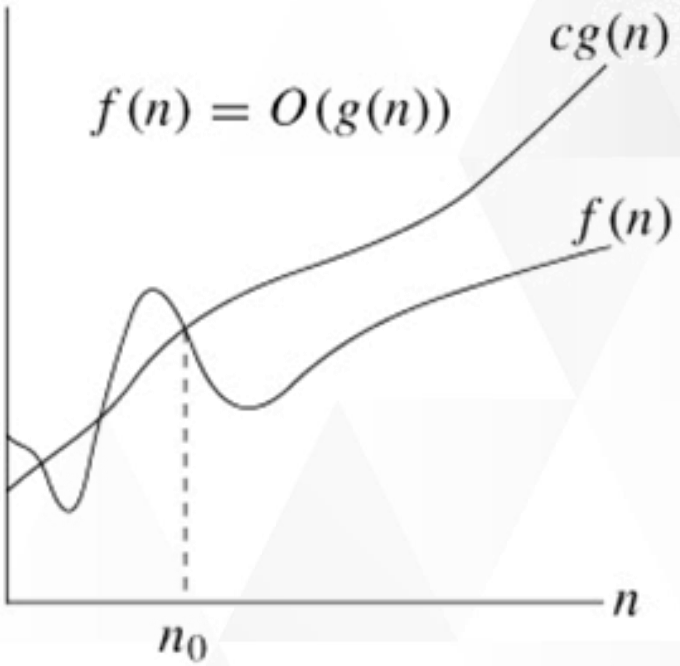
- O(Big-Oh) 표기법
  - 복잡도의 점근적 상한을 나타냄
    (상한선, 최악의 시간)
    예) O(n), O(n logn), $O(n^2)$, $O(2^n)$, ...

  - 정의
    f(n)과 g(n)이 주어졌을 때 모든 $n \geq n_0$에 대하여
    $f(n) \leq cg(n)$을 만족하는 상수 c와 n_0가 존재하면
    f(n) = O(g(n))임

3 점근적 성능(Asymptotic Performance)

- O(Big-Oh) 표기법

n이 증가함에 따라 O(g(n))이 점근적 상한이라는 것을 보여줌
즉, **g(n)이 n_0보다 큰 모든 n에 대해서 항상 f(n)보다 크다는 것**을 보여줌



4 복잡도 간의 관계

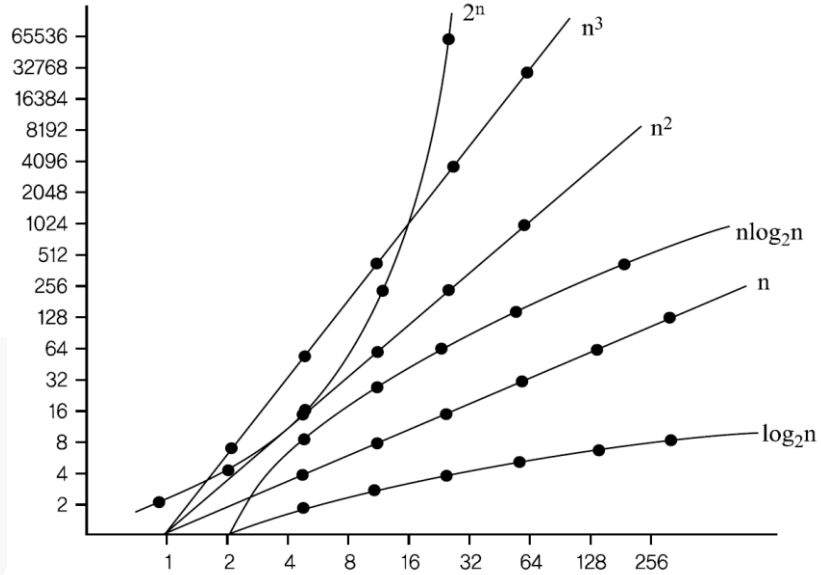
◆ O(Big-Oh) 표기법

$$O(1) < O(\log_2 n) < O(n) < O(n \log_2 n)$$
$$O(n^2) < O(n^3) < O(2^n)$$

- O(1) : 상수 복잡도
- $O(n \log_2 n)$: 로그 복잡도
- O(n) : 선형 복잡도
- $O(n \log n)$: $n \log n$복잡도
- $O(n^2)$: 2차 복잡도(Quadratic Complexity)
- $O(n^3)$: 3차 복잡도(Cubic Complexity)
- $O(2^n)$: 지수 복잡도(Exponential Complexity)

5 입력 크기에 따른 복잡도 비율의 변화

- 입력값 n에 따른 알고리즘 수행 시간



※출처: 수학으로 이해하는 디지털 논리,박주미, 한빛아카데미

5 입력 크기에 따른 복잡도 비율의 변화

Function	n = 10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

5 입력 크기에 따른 복잡도 비율의 변화

◆ 예) 다음의 시간 복잡도를 구하시오

(1)

```
1  algorithm A( ){
2    i = 0;
3    print i;
4  }
5
6
7
8
```

(2)

```
1  algorithm B(int n){
2    x = 0;
3    for i=0 to n
4      x += 1;
5    next i
6  }
7
8
```

(3)

```
1  algorithm C(int n){
2    x = 0;
3    for i=0 to n
4      for j=0 to n
5        x += 1;
6      next j
7    next i
8  }
```

(풀이)

(1) 연산횟수$f(n) = 1$ ∴ 복잡도$O(1)$

(2) 연산횟수$f(n) = n + 1$ ∴ 복잡도$O(\mathbf{n})$

(3) 연산횟수$f(n) = (n+1)^2$ ∴ 복잡도$O(\mathbf{n^2})$

1 알고리즘의 복잡도

5 입력 크기에 따른 복잡도 비율의 변화

- 왜 효율적인 알고리즘이 필요한가?
  - 10억 개의 숫자를 정렬하는데 PC에서 $O(n^2)$ 알고리즘은 300여년이 걸리는 반면에 O(n logn) 알고리즘은 5분 만에 정렬함

$O(n^2)$	1,000	1백만	10억
PC	< 1초	2시간	300년
슈퍼컴	< 1초	1초	1주일
O(nlogn)	**1,000**	**1백만**	**10억**
PC	< 1초	<1초	5분
슈퍼컴	< 1초	<1초	<1초

2 버블 정렬, 삽입 정렬, 쉘 정렬

2 버블 정렬, 삽입 정렬, 쉘 정렬

1 정렬(Sorting)

- 데이터의 순서를 결정하는 것
- 사전과 같이 어떤 기준에 따라 원소들이 나열되는 것

2 버블 정렬, 삽입 정렬, 쉘 정렬

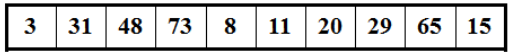
2 버블 정렬

- 이웃하는 숫자를 비교하여 작은 수를 앞쪽으로 이동시키는 과정을 반복하여 정렬하는 알고리즘
- 주어진 파일에서 인접한 2개의 레코드 키값을 비교하여 그 크기에 따라 레코드 위치를 서로 교환하는 정렬 방식
- 오름차순으로 정렬한다면 작은 수는 배열의 앞부분으로 이동

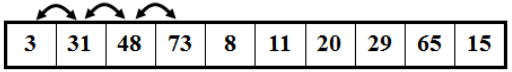
2 버블 정렬

버블 정렬 작동 예

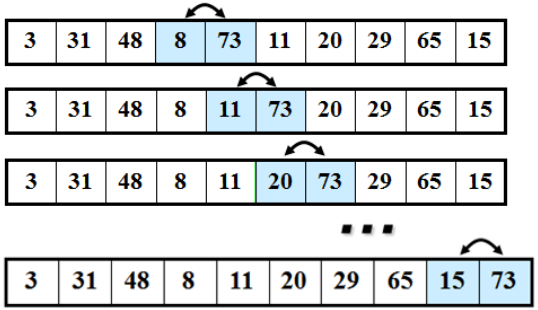
정렬할 배열이 주어진다



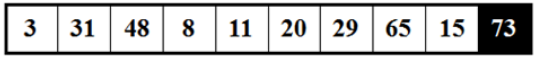
왼쪽부터 시작해 이웃한 쌍들을 비교해 나간다



순서대로 되어 있지 않으면 자리 바꾼다



맨 오른쪽 수(73)를 대상에서 제외한다



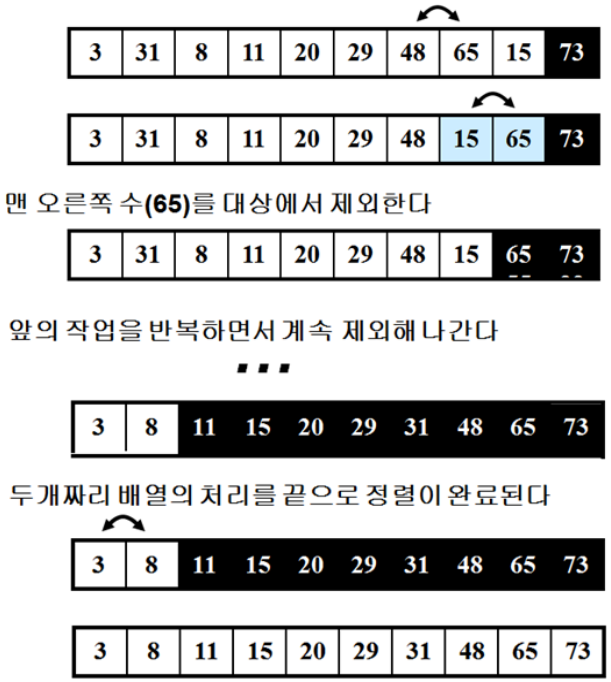
2 버블 정렬

버블 정렬 작동 예

왼쪽부터 시작해 이웃한 쌍들을 비교해간다

2 버블 정렬

버블 정렬 작동 예



2 버블 정렬, 삽입 정렬, 쉘 정렬

3 삽입 정렬

- 새로운 데이터를 정렬된 데이터에 삽입하는 과정을 반복하여 전체 데이터를 정렬하는 방식
- 가장 간단한 정렬 방식
- 이미 순서화된 파일에 새로운 하나의 레코드를 순서에 맞게 삽입시켜 정렬하는 방식
- 어느 정도 정렬이 되어 있을 경우 매우 효과적

3 삽입 정렬

- 삽입 정렬 방식
  1. 처음 A[0]은 정렬된 데이터로 취급
  2. 다음 데이터 A[1]은 정렬된 데이터 A[0]과 비교하여 적절한 위치에 삽입
  3. 다음 데이터 A[2]는 정렬된 데이터 A[0], A[1]과 비교하여 적절한 위치에 삽입
  4. 같은 방식으로 나머지 데이터들을 삽입하여 정렬

3 삽입 정렬

삽입 정렬 작동 예

2 버블 정렬, 삽입 정렬, 쉘 정렬

4 쉘 정렬

- 주어진 입력 데이터를 적당한 매개 변수의 값만큼 서로 떨어진 데이터들과 비교하여 교환하는 과정을 매개 변수의 값을 바꾸어가며 되풀이하는 것
- 영역을 나눈 후 삽입하는 **보완된 삽입 정렬** 개념
- 입력 파일을 어떤 매개변수의 값으로 서브파일을 구성하고, 각 서브파일을 삽입 정렬 방식으로 배열하는 과정을 반복하는 정렬 방식

4 쉘 정렬

쉘 정렬 작동 예

◆ 예) 다음의 데이터를 쉘 정렬로 정렬하시오

30 60 90 10 40 80 40 20 10 60 50 30 40 90 80

(풀이)

✓ 먼저 간격(gap)이 5가 되는 숫자끼리 그룹을 만듦
✓ 첫번째 그룹은 [30, 80, 50]
✓ 두번째 그룹은 [60, 40, 30],
✓ 나머지 그룹은 각각
[90, 20, 40], [10, 10, 90], [40, 60, 80]임

4 쉘 정렬

쉘 정렬 작동 예

h=5

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
그룹 1	30	60	90	10	40	80	40	20	10	60	50	30	40	90	80
그룹 2															
그룹 3															
그룹 4															
그룹 5															

그룹별 정렬 후

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
그룹 1	30	30	20	10	40	50	40	40	10	60	80	60	90	90	80
그룹 2															
그룹 3															
그룹 4															
그룹 5															

4 쉘 정렬

쉘 정렬 작동 예

- 그 다음엔 간격을 5보다 작게 하여 예를 들어 3으로 하여 3개의 그룹으로 나누어 각 그룹별로 삽입 정렬을 수행하는데 이때에는 각 그룹에 5개의 숫자가 있음
- 마지막에는 **반드시 간격을 1**로 놓고 수행해야 하는데 다른 그룹에 속해 서로 비교되지 않은 숫자가 있을 수 있기 때문임

3 선택 정렬, 퀵 정렬, 병합 정렬

3 선택 정렬, 퀵 정렬, 병합 정렬

1 선택 정렬

- 주어진 데이터에서 가장 큰 값이나 가장 작은 값을 찾은 후 그 값만을 교환하는 방식으로 단계적으로 정렬함

3 선택 정렬, 퀵 정렬, 병합 정렬

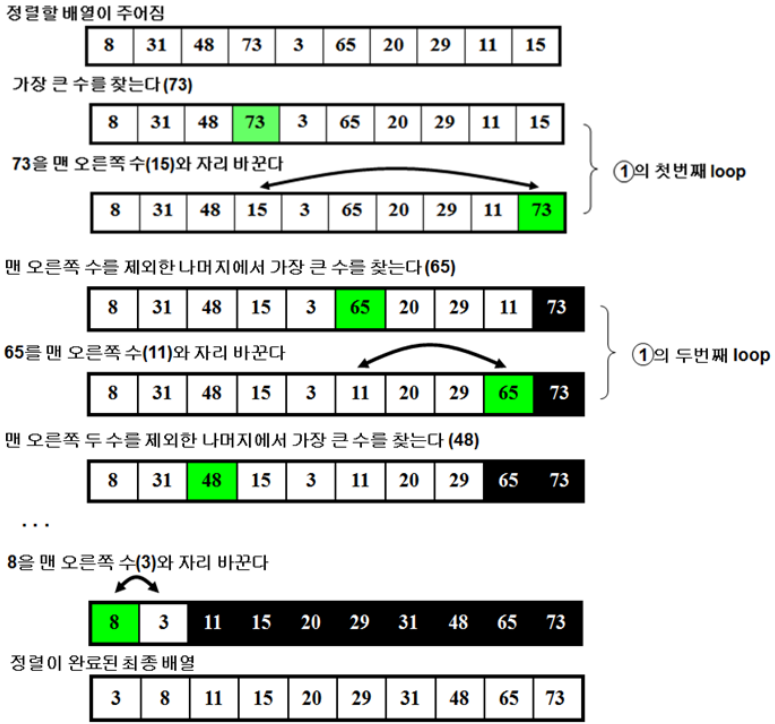
1 선택 정렬

- 각 루프마다
  - 최대 원소를 찾음
  - 최대 원소와 맨 오른쪽 원소를 교환함
  - 맨 오른쪽 원소를 제외함
- 하나의 원소만 남을 때까지 위의 루프를 반복

3 선택 정렬, 퀵 정렬, 병합 정렬

1 선택 정렬

선택 정렬 작동 예



3 선택 정렬, 퀵 정렬, 병합 정렬

2 퀵 정렬

- 기준 키를 기준으로 작거나 같은 값을 지닌 데이터는 앞으로, 큰 값을 지닌 데이터는 뒤로 가도록 하여 작은 값을 갖는 데이터와 큰 값을 갖는 데이터로 분리해가며 정렬하는 방법

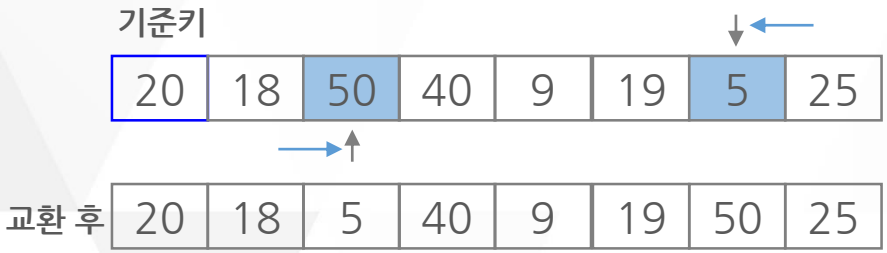
- 평균적으로 가장 좋은 성능을 가져 현장에서 가장 많이 쓰이는 정렬 알고리즘

2 퀵 정렬

◆ 정렬하고자 하는 데이터

20	18	50	40	9	19	5	25

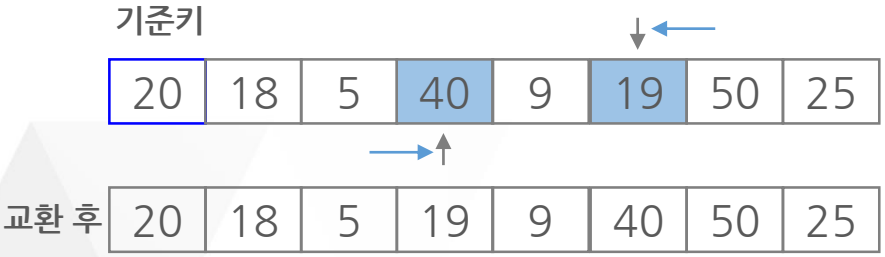
① 맨 앞의 20을 기준키로 하고 기준키 다음부터 기준키보다 큰 데이터를 찾아 50을 선택하고, 마지막 데이터부터 기준키보다 작은 데이터를 찾아 5를 선택함
그리고 선택된 50과 5를 교환함



2 퀵 정렬

◆ 정렬하고자 하는 데이터

② 계속해서 진행하여 기준키보다 큰 데이터인 40을 선택하고, 기준키보다 작은 데이터인 19를 선택한 후 두 수를 교환함



나머지도 이 방법으로 진행하여 정렬 완료함

3 병합 정렬

- 정렬할 데이터들을 2개로 나누고 2개로 나뉜 데이터들을 각각 정렬한 다음에 다시 합병하여 하나의 정렬된 데이터들로 완성하는 방법

- 이미 정렬되어 있는 두 개의 파일을 1개의 파일로 합병하는 정렬 방식으로 1개의 파일에서는 각각의 데이터를 하나의 파일로 취급하여 정렬함

3 병합 정렬

병합 정렬 작동 예