정렬 알고리즘_1

2020년도 2학기 최 희석

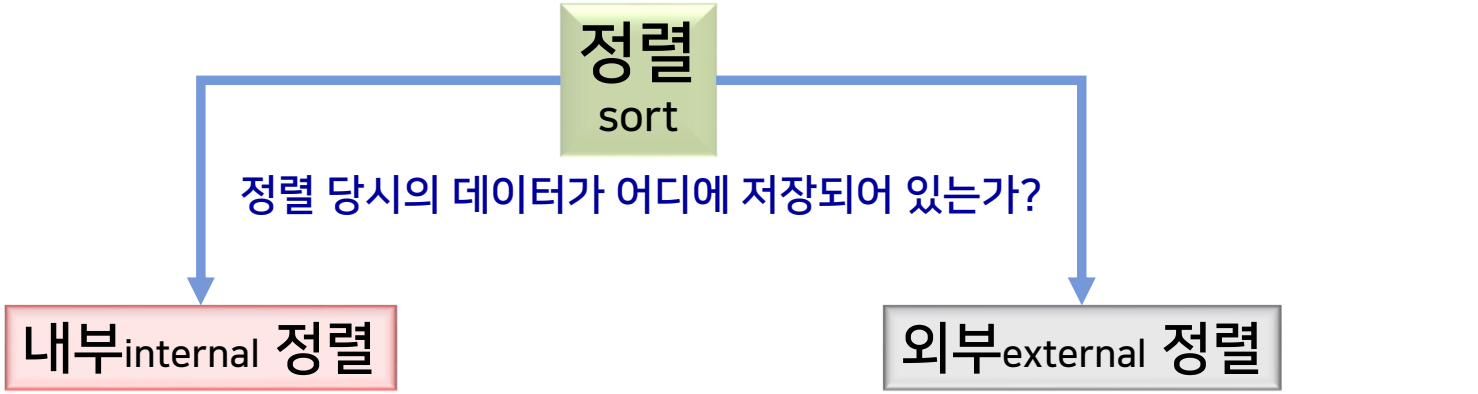
목차

- 정렬의 개념
- 버블 정렬
- 선택 정렬
- 삽입 정렬
- 셸 정렬

정렬의 개념

정렬이란?

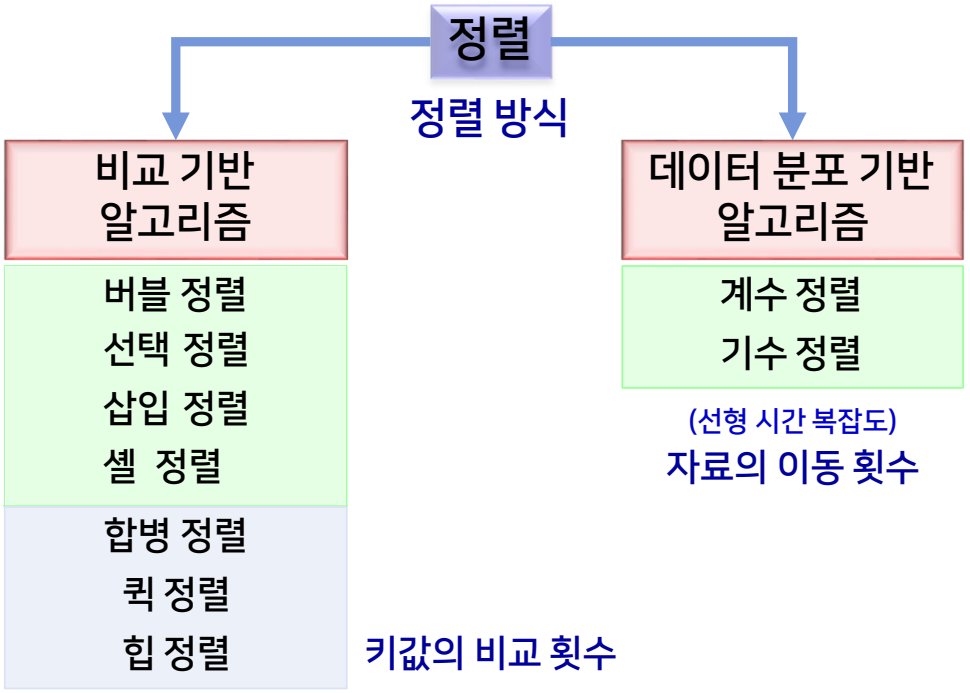
- 정렬 → 주어진 데이터를 값의 크기 순서에 따라 재배열하는 것



모든 데이터를
주기억장치에 저장한 후
정렬하는 방식

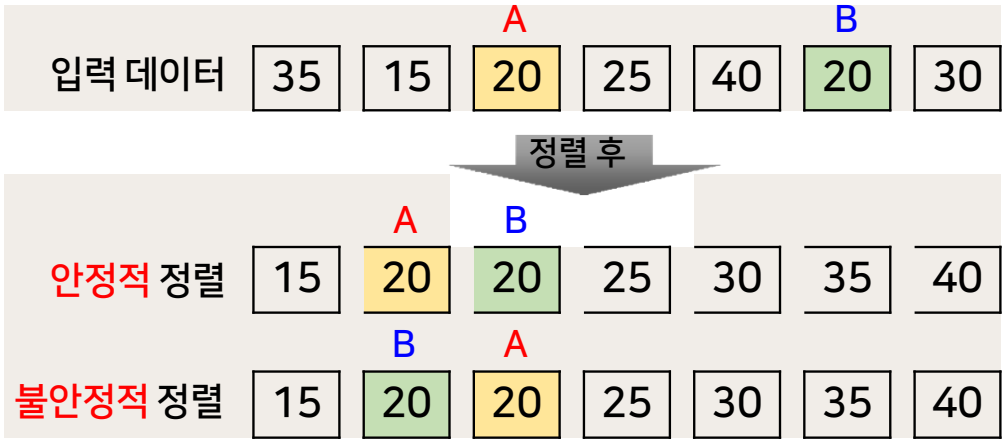
모든 데이터를 주기억장치에 저장할 수 없는 경우,
일부 데이터만 주기억장치에 있고 나머지는
외부기억장치에 저장한 채 정렬하는 방식

내부 정렬 알고리즘의 종류



기본 개념

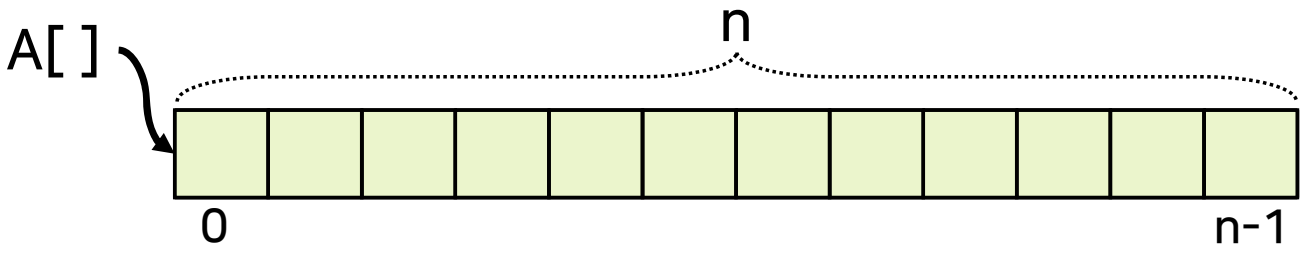
- 안정적stable 정렬
  - 동일한 값을 갖는 데이터가 여러 개 있을 때 정렬 전의 상대적인 순서가 정렬 후에도 그대로 유지되는 정렬 방식



기본 개념

- 제자리in-place 정렬
- 입력 데이터를 저장하는 공간 이외에 추가적인 저장공간을 상수 개만 필요로 하는 정렬 방식
  - 입력 크기 n이 증가함에도 불구하고 추가적인 저장공간은 증가하지 않음

정렬을 위한 기본 가정



$A[i] > 0, (0 \le i \le n-1)$

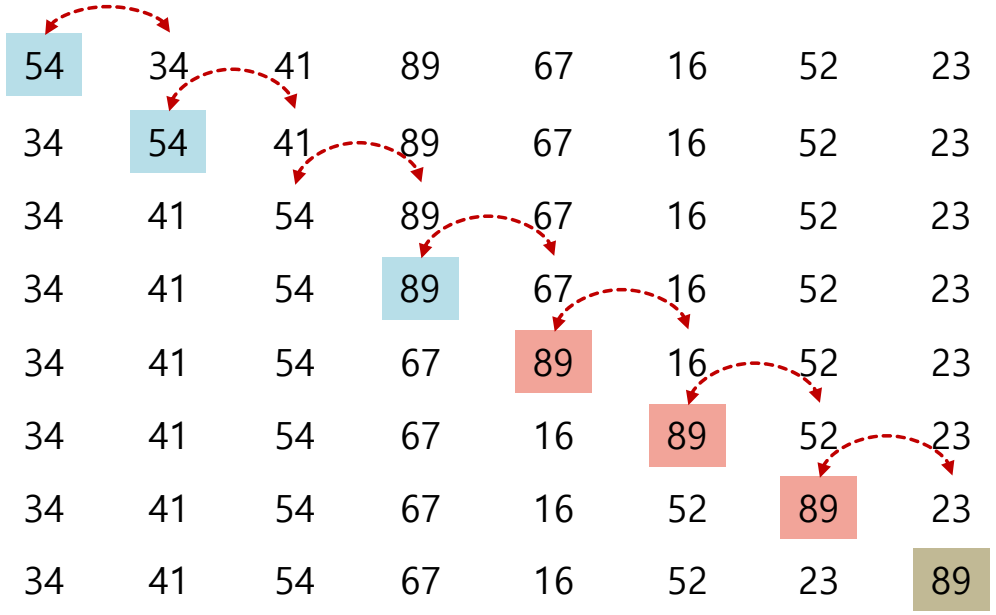
$\text{if } i < j \text{ then } A[i] \le A[j], (0 \le i,j \le n-1)$

- 키값 → 양의 정수
- 정렬 방식 → 오름차순
- 키의 개수 → n
- 키 저장 → A[0..n-1]

버블 정렬

버블 정렬의 개념과 원리

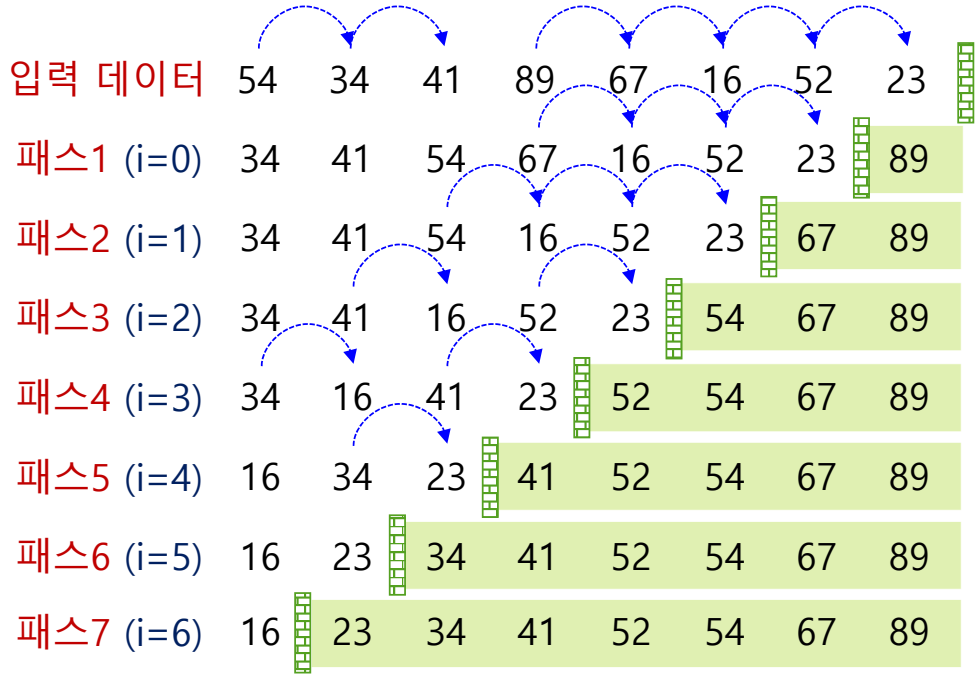
- **모든 인접한 두 값을 비교하여 왼쪽의 값이 더 큰 경우에는 자리를 바꾸는 과정을 반복해서 정렬하는 방식**



버블 정렬의 알고리즘

```
BubbleSort (A[], n)
{
  for (i=0; i<n-1; i++)          //(n-1)번의 패스
    for (j=0; j<(n-1)-i; j++)    //왼쪽 → 오른쪽
      if ( A[j] > A[j+1] )
        A[j]와 A[j+1]의 자리바꿈;
  return A;
}
```

버블 정렬의 적용 예


입력 데이터 54 34 41 89 67 16 52 23
패스1 (i=0) 34 41 54 67 16 52 23 89
패스2 (i=1) 34 41 54 16 52 23 67 89
패스3 (i=2) 34 41 16 52 23 54 67 89
패스4 (i=3) 34 16 41 23 52 54 67 89
패스5 (i=4) 16 34 23 41 52 54 67 89
패스6 (i=5) 16 23 34 41 52 54 67 89
패스7 (i=6) 16 23 34 41 52 54 67 89

버블 정렬의 성능 분석

```
BubbleSort (A[], n)
{
  for (i=0; i<n-1; i++)          //(n-1)번의 패스
    for (j=0; j<(n-1)-i; j++)    //왼쪽 → 오른쪽
      if ( A[j] > A[j+1] )
        A[j]와 A[j+1]의 자리바꿈;
  return A;
}
```

$O(n^2)$

바깥 루프 i	안쪽 루프 j에서의 비교 횟수
i=0	n-1
i=1	n-2
i=2	n-3
…	…
i=(n-3)	2
i=(n-2)	1

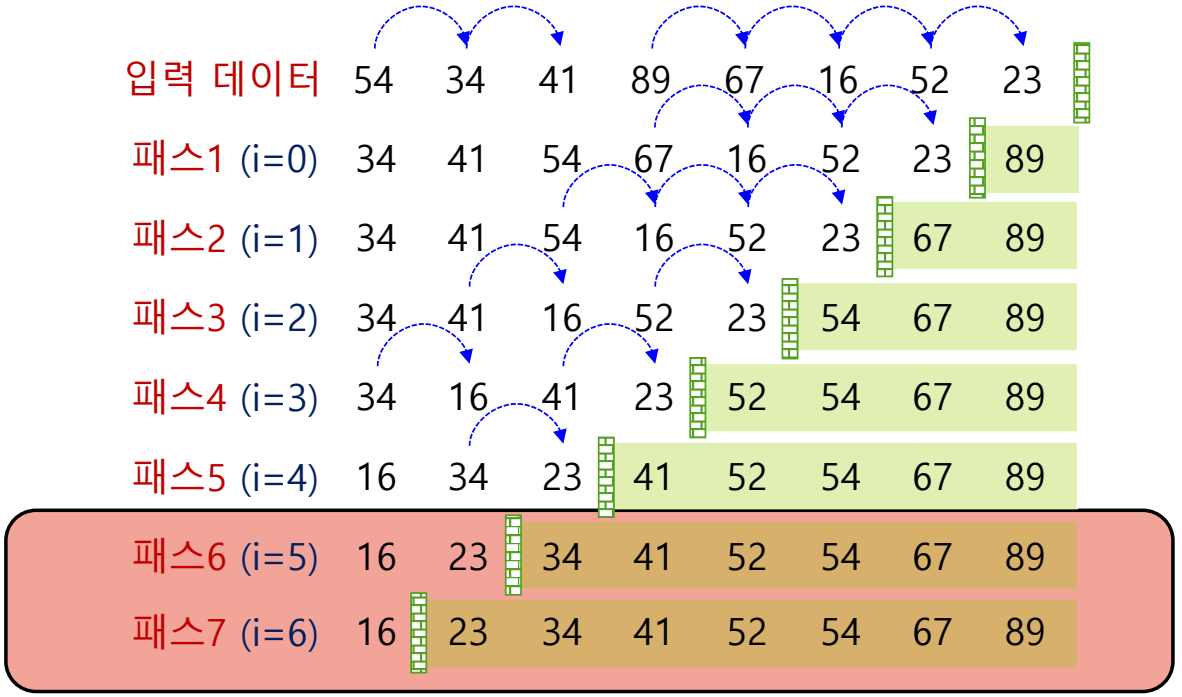
총 비교횟수

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$$

버블 정렬의 특징

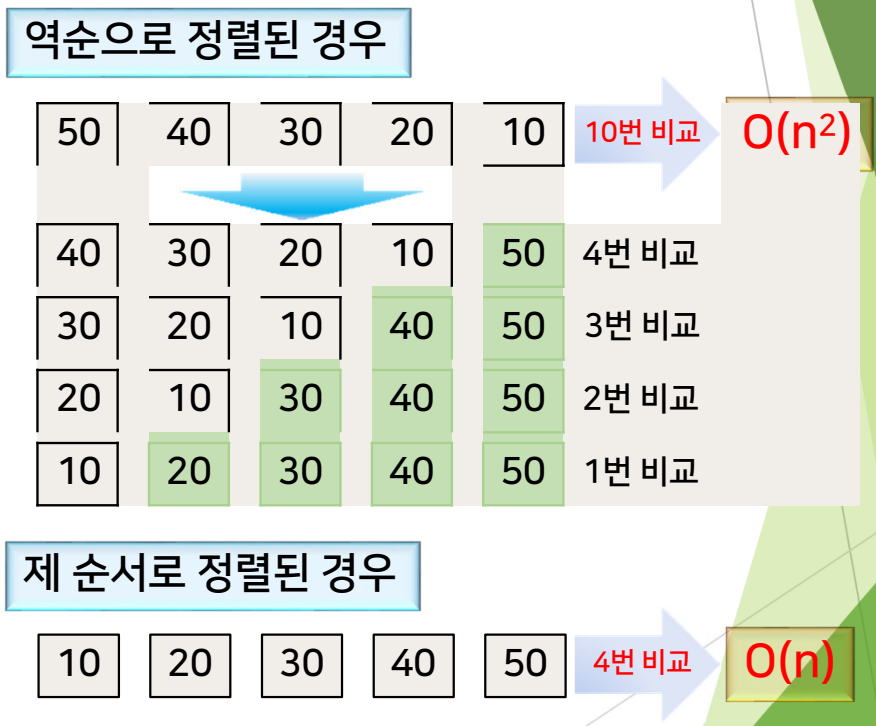
- 안정적 정렬 알고리즘
- 제자리 정렬 알고리즘
  - 입력 배열 A[], 입력크기 n
    + 추가적인 저장공간은 상수 개 (제어 변수 i 와 j, 데이터 교환을 위한 변수)

버블 정렬의 개선된 알고리즘



버블 정렬의 개선된 알고리즘

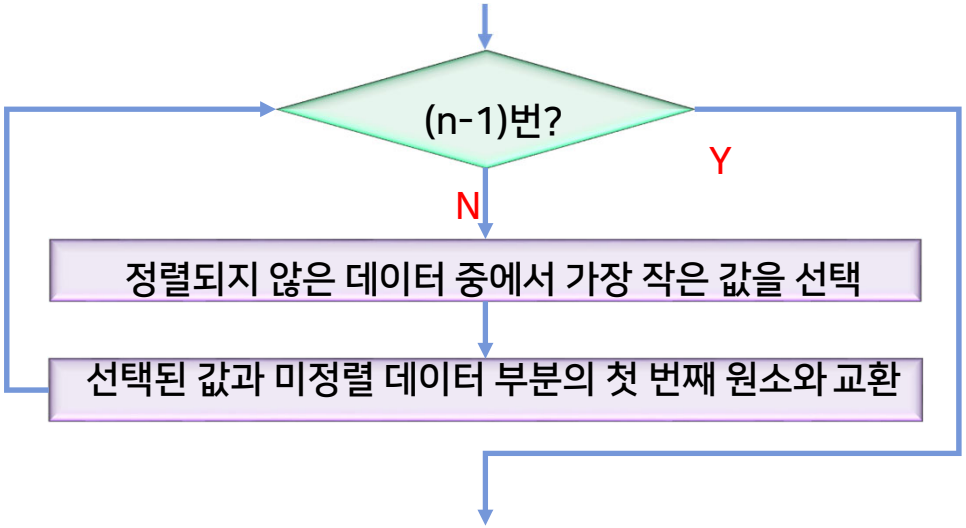
```
BubbleSort (A[], n)
{
  for (i=0; i<n-1; i++) {
    exchange = false;
    for (j=0; j<(n-1)-i; j++)
      if ( A[j] > A[j+1] ) {
        A[j]와 A[j+1]의 자리바꿈;
        exchange = true;
      }
    if ( exchange == false ) break;
  }
  return A;
}
```



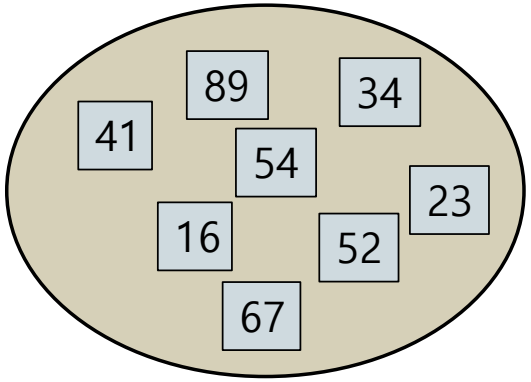
선택 정렬

선택 정렬의 개념과 원리

- 주어진 데이터 중에서 가장 작은 값부터 차례대로 '**선택**'해서 나열하는 방식



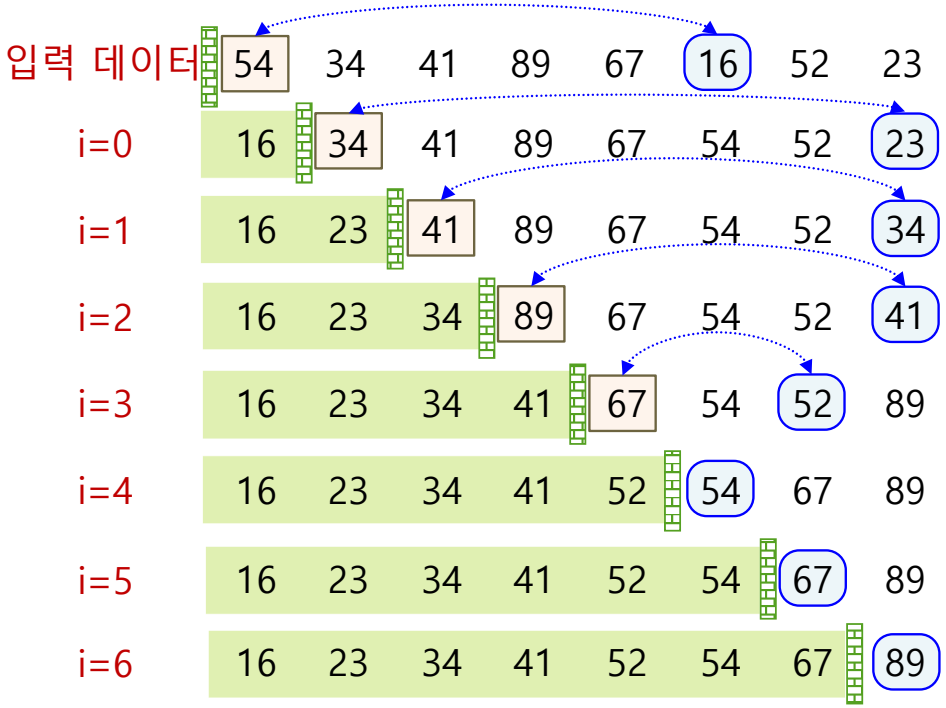
선택 정렬의 개념과 원리



선택 정렬의 알고리즘

```
SelectionSort (A[], n)
{
  for (i=0; i<n-1; i++) {
    Min = i;
    //미정렬 부분 A[i..n-1]에서 최소값 찾기
    for (j=i+1; j<n; j++)
      if ( A[Min] > A[j] )
        Min = j;
    //미정렬 부분의 첫 번째 원소와 최소값 교환
    A[i]와 A[Min]의 자리바꿈;
  }
  return A;
}
```

선택 정렬의 적용 예


입력 데이터 54 34 41 89 67 16 52 23
i=0 16 34 41 89 67 54 52 23
i=1 16 23 41 89 67 54 52 34
i=2 16 23 34 89 67 54 52 41
i=3 16 23 34 41 67 54 52 89
i=4 16 23 34 41 52 54 67 89
i=5 16 23 34 41 52 54 67 89
i=6 16 23 34 41 52 54 67 89

선택 정렬의 성능 분석

```
SelectionSort (A[], n)
{
  for (i=0; i<n-1; i++) {
    Min = i;
    for (j=i+1; j<n; j++)
      if ( A[Min] > A[j] )
        Min = j;
    A[i]와 A[Min]의 자리바꿈;
  }
  return A;
}
```

바깥 루프 i	i=0	i=1	i=2	…	i=(n-2)
j에서의 비교횟수	n-1	n-2	n-3	…	1

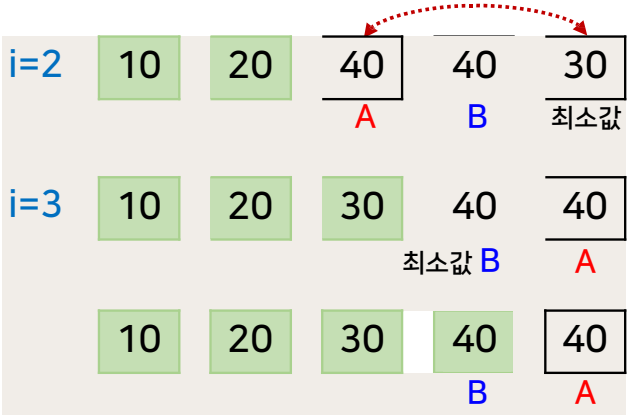
총 비교횟수

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$$

$O(n^2)$

선택 정렬의 특징

- **언제나 동일**한 시간 복잡도 O(n2)
  - 최소값을 찾는 과정이 데이터의 입력 상태에 민감하지 않음
- 제자리 정렬 알고리즘
- 안정적이지 않은 정렬 알고리즘



삽입 정렬

삽입 정렬의 개념과 원리

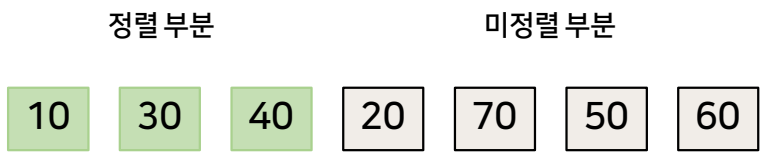
- 주어진 데이터를 하나씩 뽑은 후, 나열된 데이터들이 항상 정렬된 형태를 가지도록 뽑은 데이터를 바른 위치에 '**삽입**' 해서 나열하는 방식
- 입력 배열을 정렬 부분과 미 정렬 부분으로 구분

  → 미 정렬 부분에서 첫 번째 데이터를 뽑은 후,

  정렬 부분에서 ***제자리를 찾아*** 뽑은 데이터를 삽입

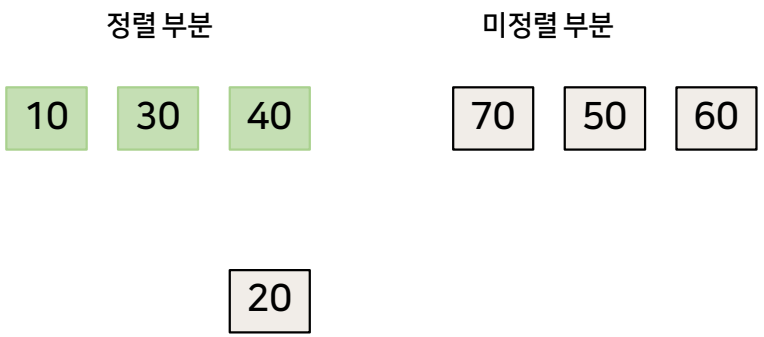
삽입 정렬의 개념과 원리

- 삽입할 제자리를 찾는 과정 (1/5)



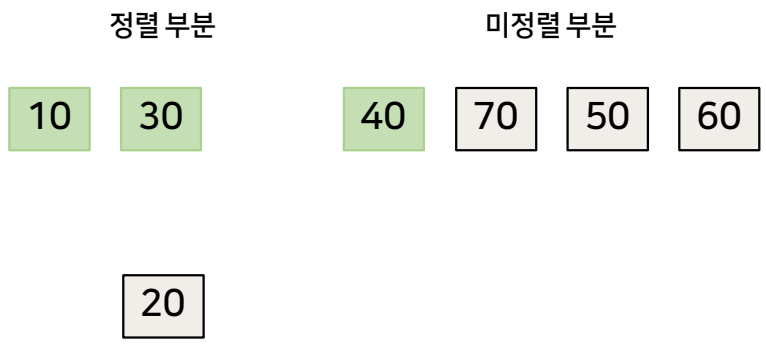
삽입 정렬의 개념과 원리

- 삽입할 제자리를 찾는 과정 (2/5)



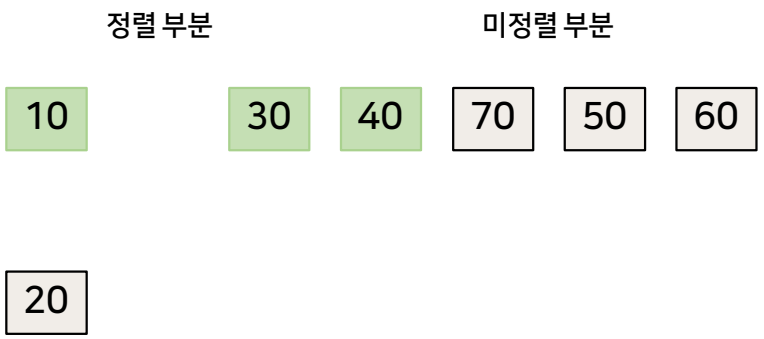
삽입 정렬의 개념과 원리

- 삽입할 제자리를 찾는 과정 (3/5)



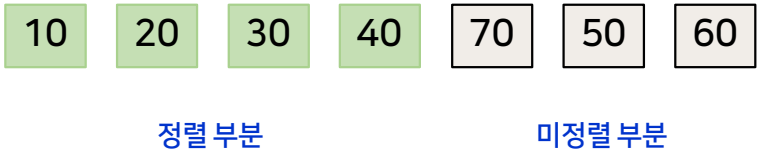
삽입 정렬의 개념과 원리

- 삽입할 제자리를 찾는 과정 (4/5)



삽입 정렬의 개념과 원리

- 삽입할 제자리를 찾는 과정 (5/5)



삽입 정렬의 알고리즘

```
InsertionSort (A[], n)
{
  for (i=1; i<n; i++) {
    val = A[i];
    for (j=i; j>0 && A[j-1]>val; j--)
      A[j] = A[j-1];
    A[j] = val;
  }
  return A;
}
```

삽입 정렬의 적용 예


입력 데이터 54 34 41 89 67 16 52 23
i=1 34 54 41 89 67 16 52 23
i=2 34 41 54 89 67 16 52 23
i=3 34 41 54 89 67 16 52 23
i=4 32 41 54 67 89 16 52 23
i=5 16 32 41 54 67 89 52 23
i=6 16 32 41 52 54 67 89 23

삽입 정렬의 성능 분석

```
InsertionSort (A[], n)
{
  for (i=1; i<n; i++) {
    val = A[i];
    for (j=i; j>0 && A[j-1]>val; j--)
      A[j] = A[j-1];
    A[j] = val;
  }
  return A;
}
```

바깥 루프 i	i=1	i=2	i=3	…	i=(n-1)
j에서의 비교횟수	1	2	3	…	n-1

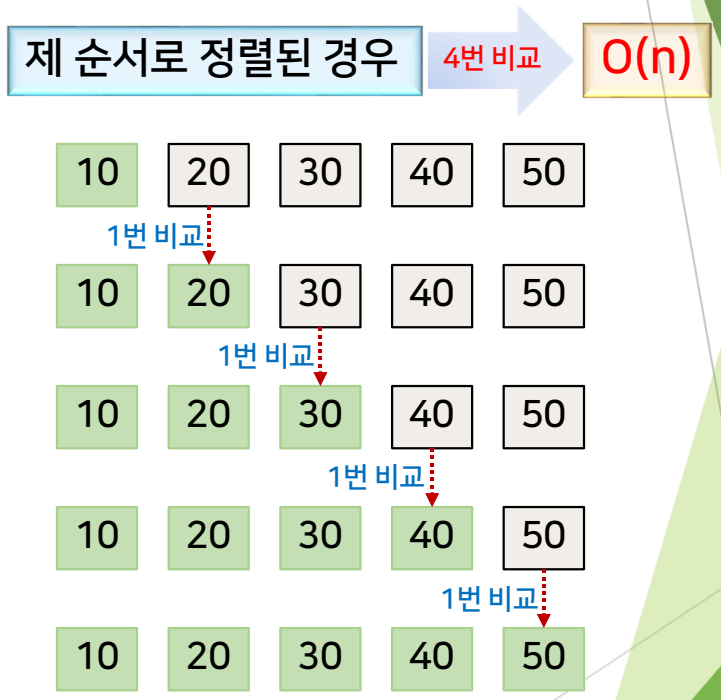
총 비교횟수

$$1+2+\cdots+(n-2)+(n-1)=\frac{n(n-1)}{2}$$

$O(n^2)$

삽입 정렬의 특징


역순으로 정렬된 경우
10번 비교
O(n²)
50 40 30 20 10
1번 비교
40 50 30 20 10
2번 비교
30 40 50 20 10
3번 비교
20 30 40 50 10
4번 비교
10 20 30 40 50


제 순서로 정렬된 경우
4번 비교
O(n)
10 20 30 40 50
1번 비교
10 20 30 40 50
1번 비교
10 20 30 40 50
1번 비교
10 20 30 40 50
1번 비교
10 20 30 40 50

삽입 정렬의 특징

- **입력이 거의 정렬된 경우** 다른 어떤 정렬 알고리즘보다 **빠른 수행 시간 O(n)**을 가짐
- 안정적인 정렬 알고리즘
- 제자리 정렬 알고리즘
- 삽입될 위치를 찾기 위해 한 번에 한 자리씩만 이동
  - 자료의 이동이 여러 번 발생

셸 정렬

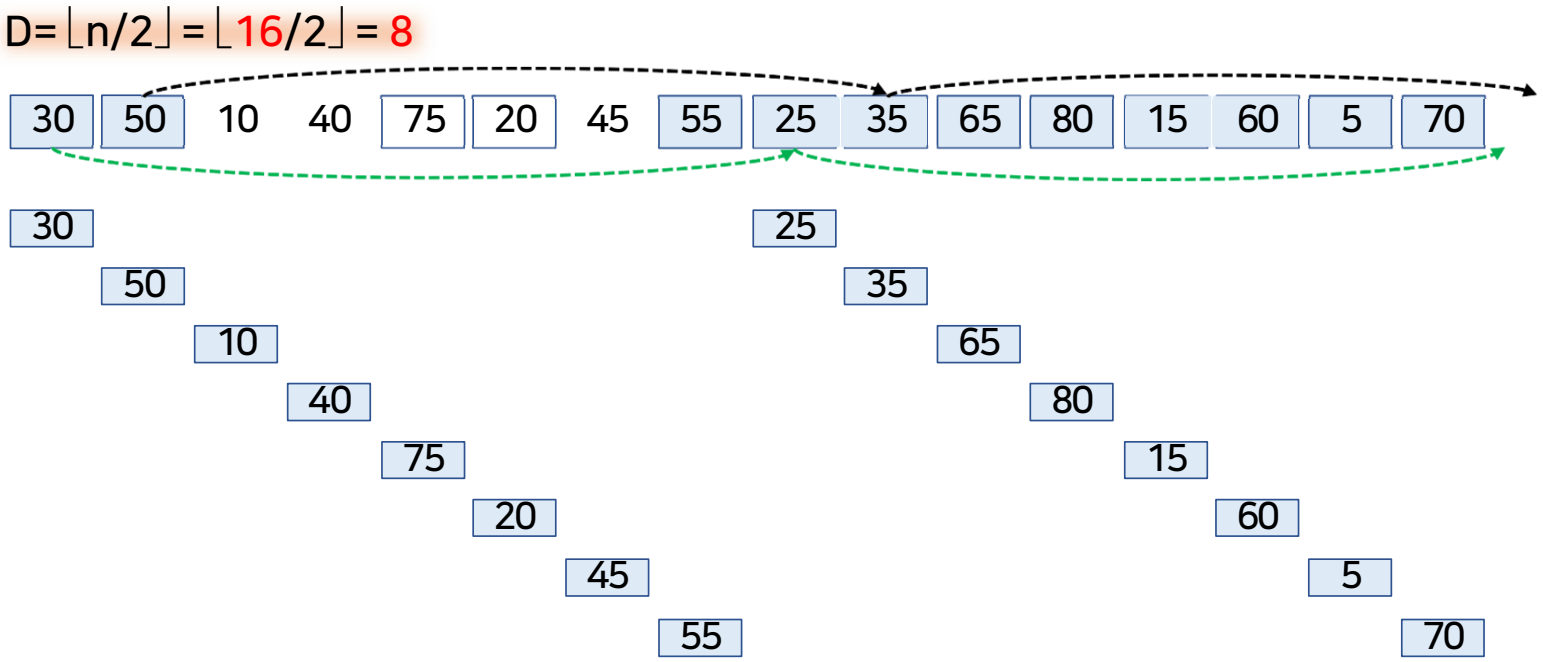
셸 정렬의 개념과 원리

- 삽입 정렬의 단점 보완 by Donald L. Shell
  - "데이터가 삽입될 위치에서 많이 떨어져 있어도 한 번에 한 자리 씩만 이동하므로 제자리를 찾아가는 데 느리다."
- 기본 아이디어
- 멀리 떨어진 원소를 교환하여 처리 속도 향상
  - 처음에는 멀리 떨어진 두 원소를 비교하여 필요시 위치를 교환하고 점차 가까운 위치의 원소를 비교 · 교환 한 뒤, 맨 마지막에는 인접한 원소를 비교 · 교환하는 방식
  - 입력 배열을 부분배열로 나누어 삽입 정렬을 수행하는 과정을 부분배열의 크기와 개수를 변화시켜 가면서 반복

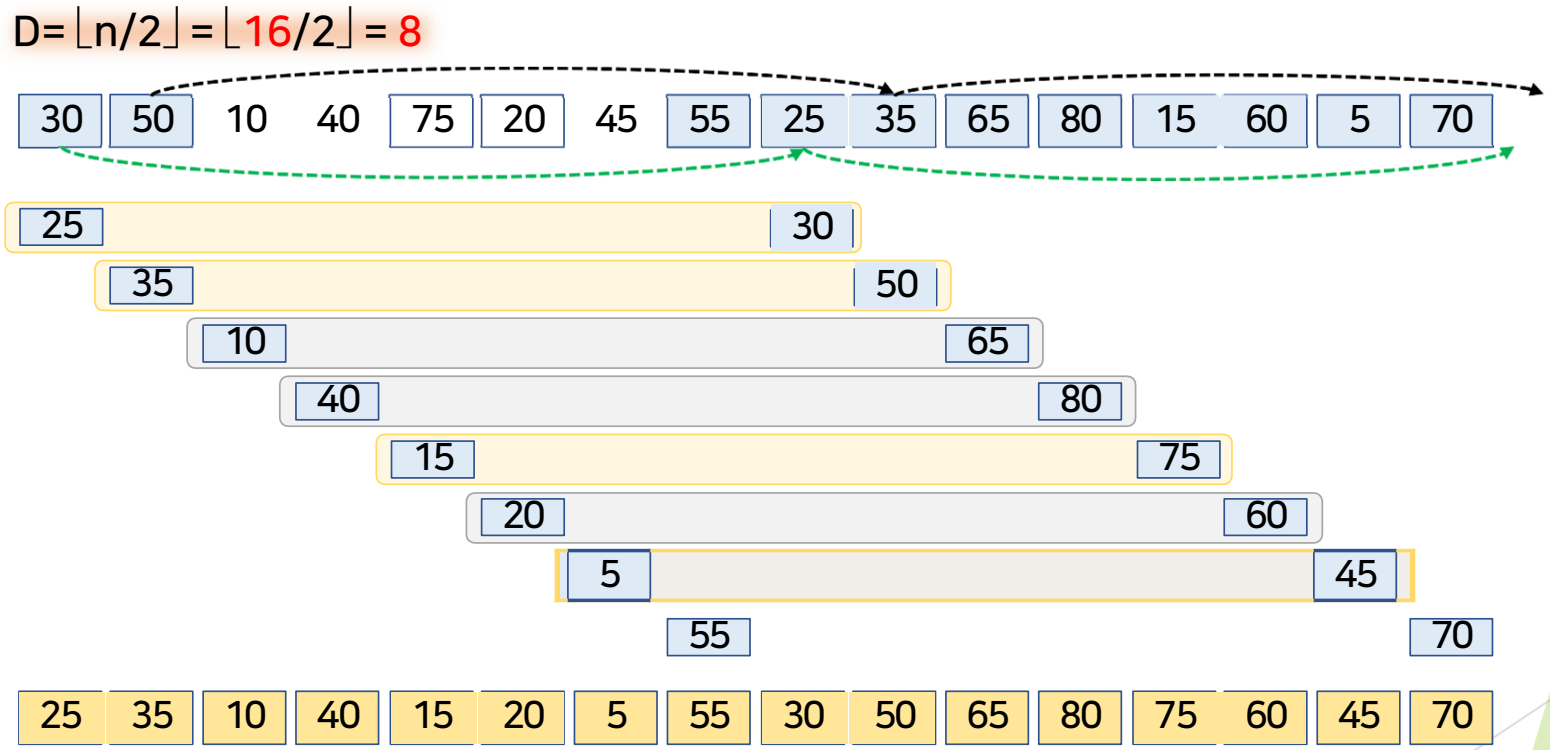
셸 정렬의 알고리즘

```
ShellSort (A[], n)
{
  for(D=⌊n/2⌋; D>=1; D=⌊D/2⌋) {
    for (i=D; i<n; i++) {
      val = A[i];
      for (j=i; j>=D && A[j-D]>val; j=j-D)
        A[j] = A[j-D];
      A[j] = val;
    }
  }
  return A;
}
```

셸 정렬의 적용 예

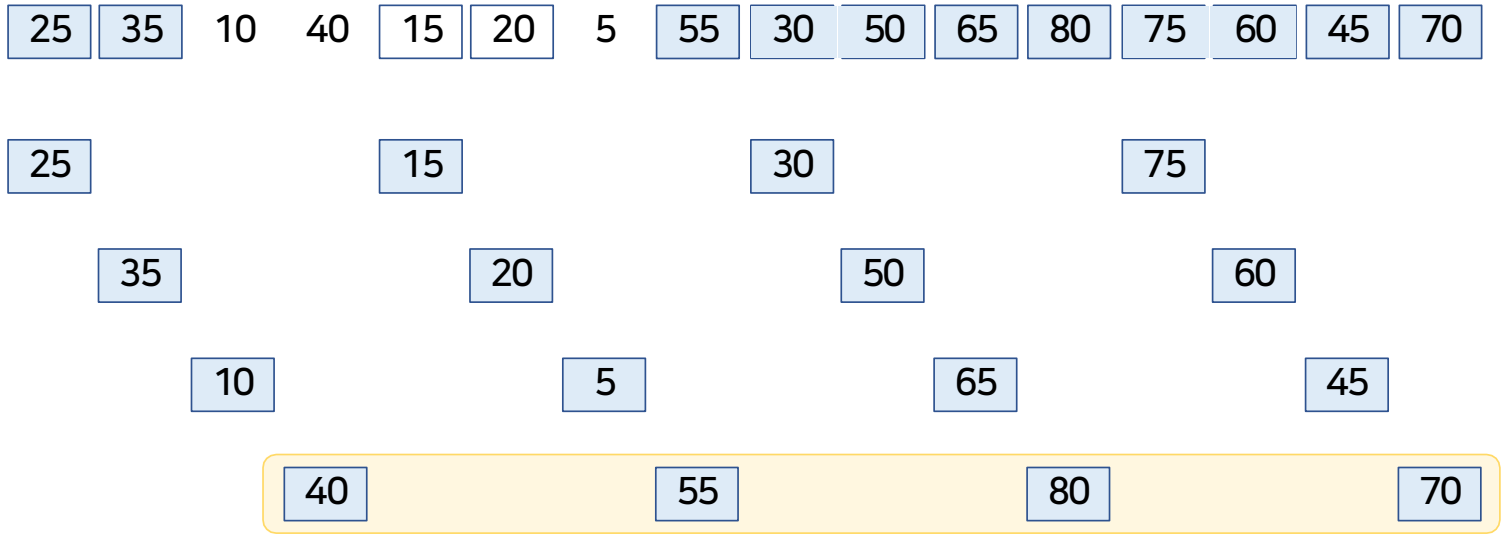

D=⌊n/2⌋=⌊16/2⌋=8
30 50 10 40 75 20 45 55 25 35 65 80 15 60 5 70
30 25
50 35
10 65
40 80
75 15
20 60
45 5
55 70

셸 정렬의 적용 예


D=⌊n/2⌋=⌊16/2⌋=8
30 50 10 40 75 20 45 55 25 35 65 80 15 60 5 70
25 30
35 50
10 65
40 80
15 75
20 60
5 45
55 70
25 35 10 40 15 20 5 55 30 50 65 80 75 60 45 70

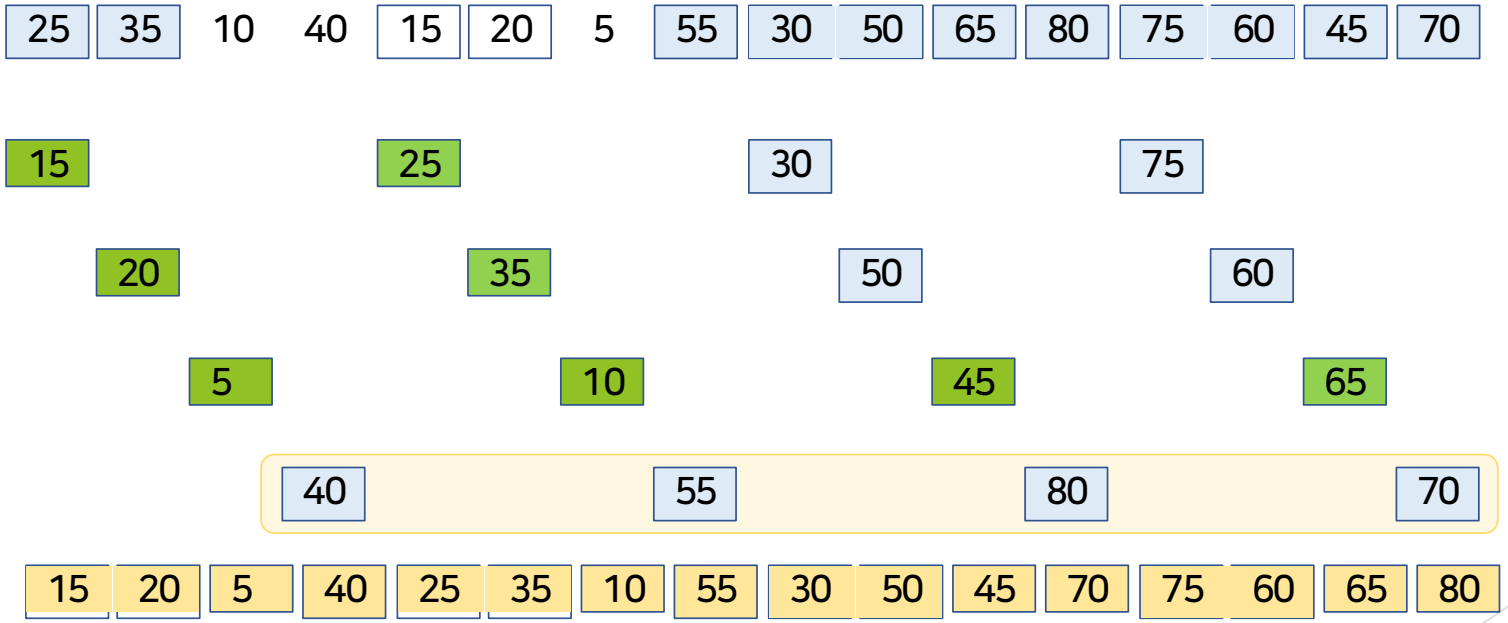
셸 정렬의 적용 예

D=⌊D/2⌋=⌊8/2⌋=4



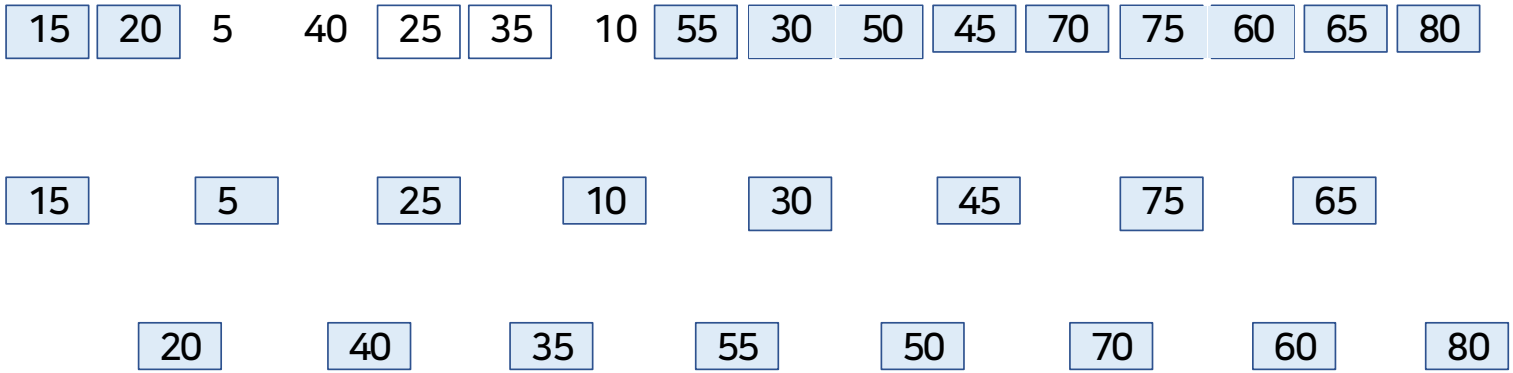
셸 정렬의 적용 예

D=⌊D/2⌋=⌊8/2⌋=4



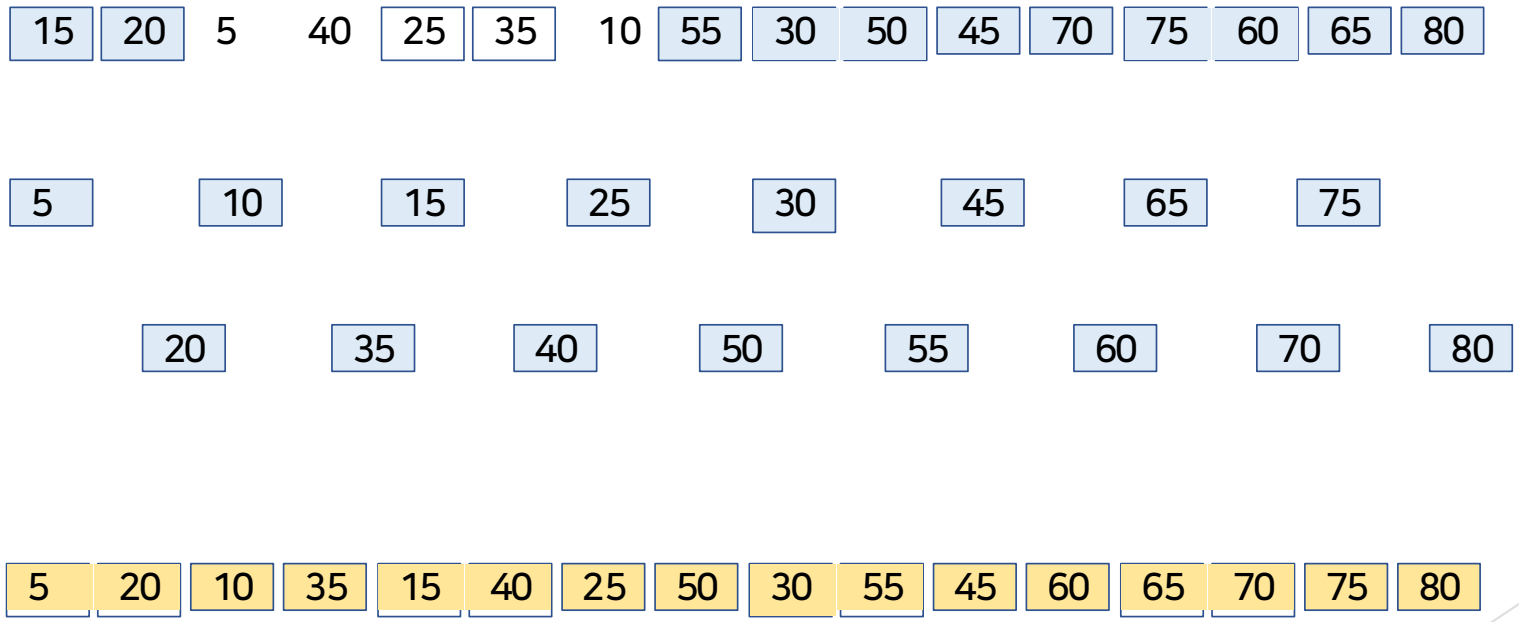
셸 정렬의 적용 예

D=⌊D/2⌋=⌊4/2⌋=2



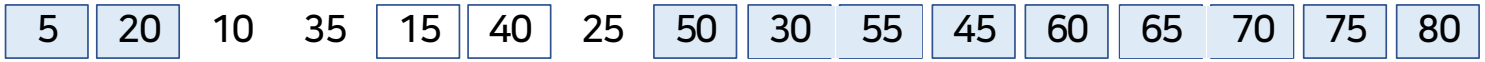
셸 정렬의 적용 예

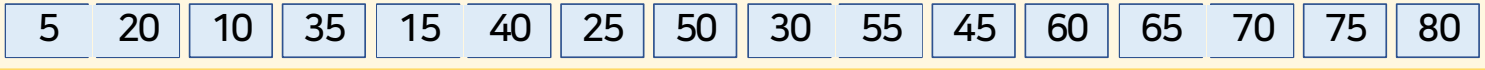
D=⌊D/2⌋=⌊4/2⌋=2

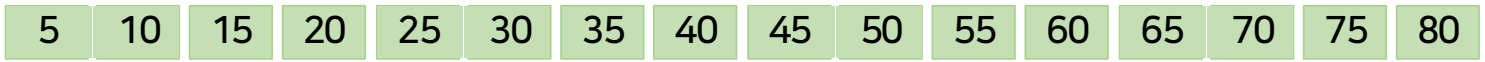


셸 정렬의 적용 예

D=⌊D/2⌋=⌊2/2⌋=1







셸 정렬의 성능 분석

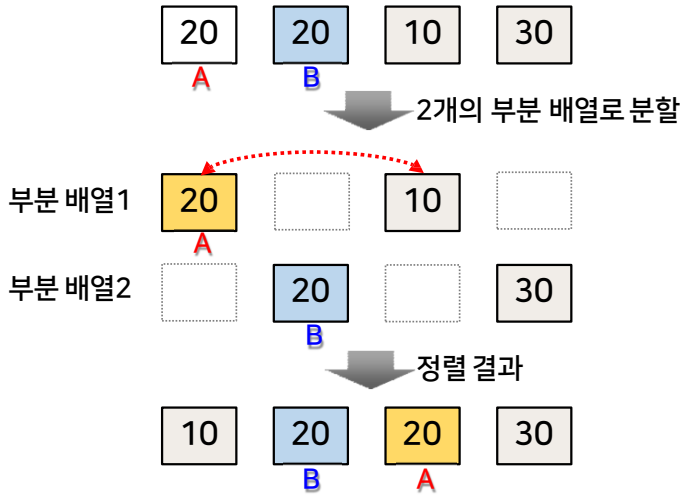
- 최악의 경우 → **O(n2)**
- 최선의 경우 → **O(nlogn)**

셸 정렬의 특징

- 간격의 크기 D를 계산하는 방식에 따라 성능이 달라짐
  - $D = n / 2^i$ (n : 데이터 개수, i=1, 2, 3, …)
  - 가장 좋은 간격을 찾는 것은 미해결 과제
    - 1, 4, 13, 40, 121, 364, 1093, 3280, … ($h_{i+1} = 3h + 1,\ h_1 = 1$)
    - 1, 3, 7, 15, 32, 63, … (2^i -1)
    - 1, 3, 7, 21, 48, 112, 336, 861, 1968, 4592, …
    - 1, 4, 10, 23, 57, 132, 391, 701
  - 간격의 크기 D의 적용은 역순으로 차례대로 사용
    - …, 121, 40, 13, 4, 1

셸 정렬의 특징

- 안정적이지 않은 정렬 알고리즘



- 제자리 정렬 알고리즘

과제 안내

8주차 과제

- 버블 정렬, 삽입 정렬 문제 구현하기
  - e-Class 업로드
- 양식 (한글, 워드,PDF -> 자유)
- 파일명 (이름_학번_전공)
  - 예) 최희석_2014182009_게임공학

- 질의 응답은 e-Class 질의응답 게시판에 남겨 주시길 바랍니다.