제6장 고급정렬 알고리즘 (1)

과 목 명 정 보 처 리 알 고 리 즘

담당교수 김 성 훈

경북대학교 과학기술대학 소프트웨어학과

이 장에서 배울 내용

1. 기본 정렬 알고리즘:
   버블정렬,
   선택정렬,
   삽입정렬
2. 쉘정렬
3. 힙정렬
4. 정렬 문제의 하한
5. 기수정렬
6. 외부정렬

정렬 알고리즘 개요

- 기본적인 정렬 알고리즘: → $O(n^2)$
  - 버블 정렬
  - 선택 정렬
  - 삽입 정렬
- 효율적인 정렬 알고리즘: $O(nlogn)$
  - 쉘 정렬 → $O(n^{1.5})$
  - 힙 정렬
  - 합병 정렬 (3장 분할정복에서 다루었음)
  - 퀵 정렬 (3장 분할정복에서 다루었음)
- 이외에도 입력이 제한된 크기 이내에 숫자로 구성되어 있을 때에 매우 효율적인 기수 정렬이 있다. → 실행시간이 선형적이다. $O(n)$

정렬 알고리즘 개요 (2)

- 정렬 알고리즘
  - 내부정렬 (Internal sort)
  - 외부정렬 (External sort)

  - 내부정렬은 입력의 크기가 주기억 장치 (main memory)의 공간보다 크지 않은 경우에 수행되는 정렬이다.
    예: 앞에서 언급한 모든 정렬 알고리즘들

  - 입력의 크기가 주기억 장치 공간보다 큰 경우에는,
    보조 기억 장치에 있는 입력을 여러 번에 나누어 주기억 장치에 읽어 들인 후,
    정렬하여 보조 기억 장치에 다시 저장하는 과정을 반복해야 한다.
    이러한 정렬을 외부정렬이라고 한다.

6.1 버블 정렬

- 버블 정렬 (Bubbble Sort)은 이웃하는 숫자를 비교하여 작은 수를 앞쪽으로 이동시키는 과정을 반복하여 정렬하는 알고리즘이다.

- 오름차순으로 정렬한다면, 작은 수는 배열의 앞부분으로 이동하는데, 배열을 좌우가 아니라 상하로 그려보면 정렬하는 과정에서 작은 수가 마치 '거품'처럼 위로 올라가는 것을 연상케 한다.

BubbleSort 알고리즘

BubbleSort
입력: 크기가 n인 배열 A
출력: 정렬된 배열 A

```
1. for pass = 1 to n-1
2.     for i = 1 to n-pass
3.         if (A[i-1] > A[i])     // 위의 원소가 아래의 원소보다 크면
4.             A[i-1] ↔ A[i]    // 서로 자리를 바꾼다.
5. return 배열 A
```

시간복잡도

- 버블 정렬은 for-루프 속에서 for-루프가 수행되는데,
  - pass=1이면 (n-1)번 비교하고,
  - pass=2이면 (n-2)번 비교하고, …
  - pass=n-1이면 1번 비교한다.
- 총 비교 횟수: (n-1) + (n-2) + … + 1 = n(n-1)/2
- 안쪽 for-루프의 if-조건이 '참'일 때의 자리바꿈은 O(1) 시간이 걸린다.
- 시간복잡도: $n(n-1)/2 \times O(1) = O(n^2) \times O(1) = O(n^2)$

6.2 선택 정렬

- 선택 정렬 (Selection Sort)은 입력 배열 전체에서 최솟값을 '선택'하여 배열의 0번 원소와 자리를 바꾸고, 다음엔 0번 원소를 제외한 나머지 원소에서 최솟값을 선택하여, 배열의 1번 원소와 자리를 바꾼다.

- 이러한 방식으로 마지막에 2개의 원소 중 최솟값을 선택하여 자리를 바꿈으로써 오름차순의 정렬을 마친다.

SelectionSort 알고리즘

입력: 크기가 n인 배열 A
출력: 정렬된 배열 A

```
1. for i = 0 to n-2 {
2.     min = i
3.     for j = i+1 to n-1 {   // A[i]~A[n-1]에서 최솟값을 찾는다.
4.         if (A[j] < A[min])
5.             min = j
         }
6.     A[i] ↔ A[min]    // min이 최솟값이 있는 원소의 인덱스
   }
7. return 배열 A
```

시간복잡도

- 선택 정렬은 line 1의 for-루프가 (n-1)번 수행되는데,
  - i=0일 때 line 3의 for-루프는 (n-1)번 수행되고,
  - i=1일 때 line 3의 for-루프는 (n-2)번 수행되고, …,
  - 마지막으로 1번 수행
- 루프 내부의 line 4~5가 수행되는 총 횟수:

$$(n-1)+(n-2)+(n-3)+\cdots+2+1 = n(n-1)/2$$

- 루프 내부의 if-조건이 '참'일 때의 자리바꿈은 O(1) 시간이 걸리므로,
- 시간복잡도: $n(n-1)/2 \times O(1) = O(n^2)$

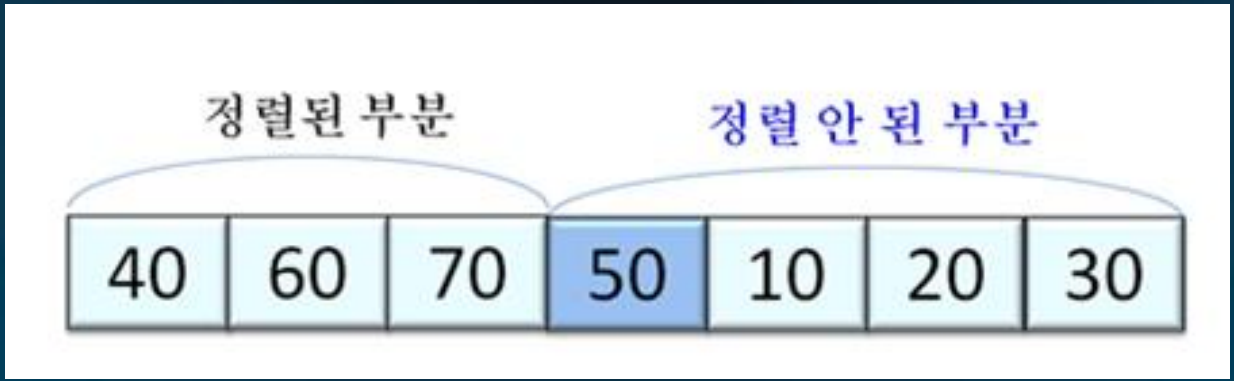
선택 정렬의 특징

- 선택 정렬의 특징은 입력이,
  - 거의 정렬되어 있든지,
  - 역으로 정렬되어 있다든지,
  - 랜덤하게 되어있든 지를 구분하지 않고,

  항상 일정한 시간복잡도를 나타낸다는 것이다.
  즉, 입력에 민감하지 않은 (input insensitive) 알고리즘이다.

6.3 삽입 정렬

- 삽입 정렬 (Insertion Sort)은 배열을 정렬된 부분 (앞부분)과 정렬 안 된 부분 (뒷부분)으로 나누고, 정렬 안 된 부분의 가장 왼쪽 원소를 정렬된 부분의 적절한 위치에 삽입하여 정렬되도록 하는 과정을 반복한다.



InsertionSort 알고리즘

```
1. for i = 1 to n-1 {
2.       CurrentElement = A[i]    // 정렬 안된 부분의 가장 왼쪽원소
3.      j ← i – 1  // 정렬된 부분의 가장 오른쪽 원소로부터 왼쪽 방향으
                    로 삽입할 곳을 탐색하기 위하여
4.      while (j >= 0) and (A[j] > CurrentElement) {
5.           A[j+1] = A[j]   // 자리 이동
6.           j ← j -1
          }
7.       A [j+1] ← CurrentElement
   }
8. return A
```

시간복잡도

- 삽입 정렬은 line 1의 for-루프가 (n-1)번 수행되는데,
  - i=1일 때 while-루프는 1번 수행되고,
  - i=2일 때 최대 2번 수행되고, …,
  - 마지막으로 최대 (n-1)번 수행되므로
- 루프 내부의 line 5~6이 수행되는 총 횟수:

$$1 + 2 + 3 + \cdots + (n-2) + (n-1) = n(n-1)/2$$

- 루프 내부의 수행시간은 O(1)이므로,
- 시간복잡도: $n(n-1)/2 \times O(1) = O(n^2)$

삽입 정렬의 특징

- 삽입 정렬은 입력의 상태에 따라 수행 시간이 달라질 수 있다.
  - 입력이 이미 정렬되어 있으면, 항상 각각 CurrentElement가 자신의 왼쪽 원소와 비교 후 자리이동 없이 원래 자리에 있게 되고, while-루프의 조건이 항상 '거짓'이 되므로 원소의 이동도 없다.
  - 따라서 (n-1)번의 비교만 하면 정렬을 마치게 된다.
  - 이때가 삽입 정렬의 최선 경우이고 시간복잡도는 $O(n)$이다.
  - 삽입 정렬은 거의 정렬된 입력에 대해서 다른 정렬 알고리즘보다 빠르다.
- 반면에 역으로 (반대로) 정렬된 입력에 대해서는 앞의 시간복잡도 분석대로 $O(n^2)$ 시간이 걸린다.
- 삽입 정렬의 평균 경우 시간복잡도는 최악 경우와 같다.

BREAK TIME

6.4 쉘 정렬

- 버블 정렬이나 삽입 정렬이 수행되는 과정을 살펴보면, 이웃하는 원소끼리의 자리이동으로 원소들이 정렬된다.
- 버블 정렬이 오름차순으로 정렬하는 과정을 살펴보면, 작은 (가벼운) 숫자가 배열의 앞부분으로 매우 느리게 이동하는 것을 알 수 있다.
  - 예를 들어, 삽입 정렬의 경우 만일 배열의 마지막 원소가 입력에서 가장 작은 숫자라면, 그 숫자가 배열의 맨 앞으로 이동해야 하므로, 모든 다른 숫자들이 1칸씩 오른쪽으로 이동해야 한다.
- 쉘 정렬 (Shell sort)은 이러한 단점을 보완하기 위해서 삽입 정렬을 이용하여 배열 뒷부분의 작은 숫자를 앞부분으로 '빠르게' 이동시키고, 동시에 앞부분의 큰 숫자는 뒷부분으로 이동시키고, 가장 마지막에는 삽입 정렬을 수행하는 알고리즘이다.

예제

- 다음의 예제를 통해 쉘 정렬의 아이디어를 이해해보자.

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

- 먼저 간격 (gap)이 5가 되는 숫자끼리 그룹을 만든다.
- 총 15개의 숫자가 있으므로,
  첫 번째 그룹은 첫 숫자인 30,
  첫 숫자에서 간격이 5되는 숫자인 80,
  그리고 80에서 간격이 5인 50으로 구성된다.

예제(계속)

- 즉, 첫 번째 그룹은 [30, 80, 50]이다. 2 번째 그룹은 [60, 40, 30]이고, 나머지 그룹은 각각 [90, 20, 40], [10, 10, 90], [40, 60, 80]이다.

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
2															
3															
4															
5															

- 각 그룹 별로 삽입 정렬을 수행한 결과를 1줄에 나열해보면 다음과 같다.

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

예제(계속)

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
2															
3															
4															
5															

그룹별 정렬 후

- 간격이 5인 그룹 별로 정렬한 결과를 살펴보면, 80과 90같은 큰 숫자가 뒷부분으로 이동하였고, 20과 30같은 작은 숫자가 앞부분으로 이동한 것을 관찰할 수 있다.

예제(계속)

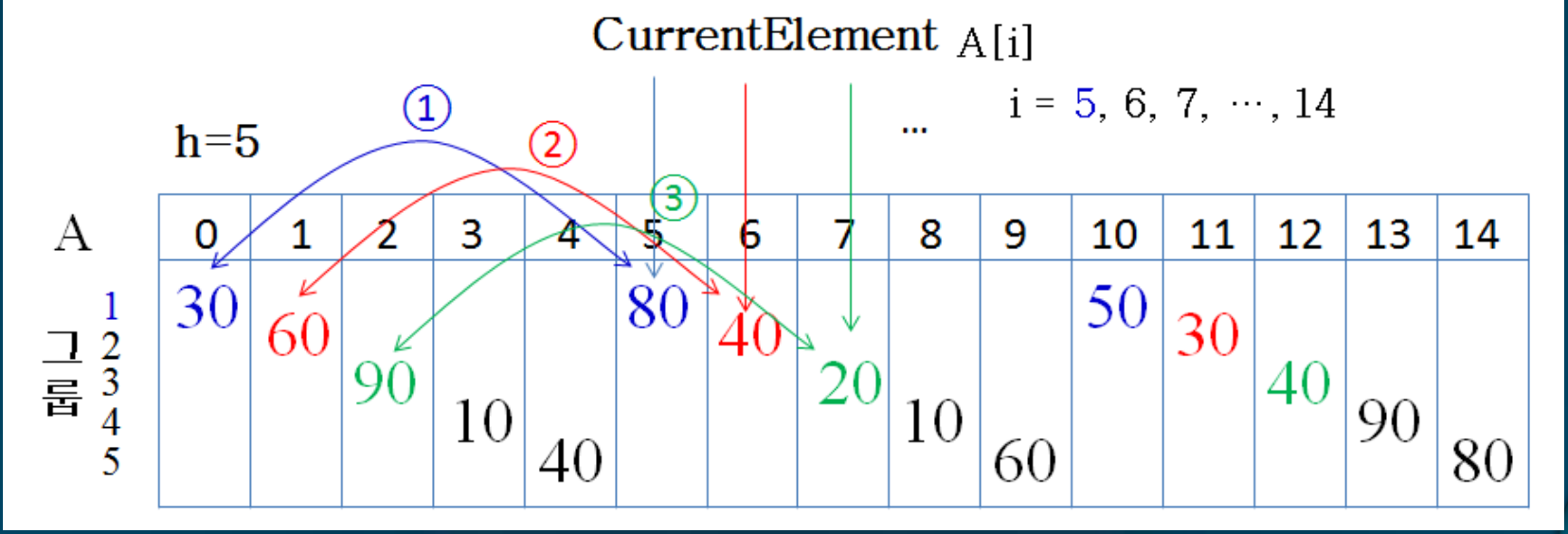
- 그 다음엔 간격을 5보다 작게 하여, 예를 들어, 3으로 하여, 3개의 그룹으로 나누어 각 그룹별로 삽입 정렬을 수행한다.
- 이때에는 각 그룹에 5개의 숫자가 있다. 마지막에는 반드시 간격을 1로 놓고 수행해야 한다.
- 왜냐하면 다른 그룹에 속해 서로 비교되지 않은 숫자가 있을 수 있기 때문이다.
- 즉, 모든 원소를 1개의 그룹으로 여기는 것이고, 이는 삽입 정렬 그 자체이다.

ShellSort 알고리즘

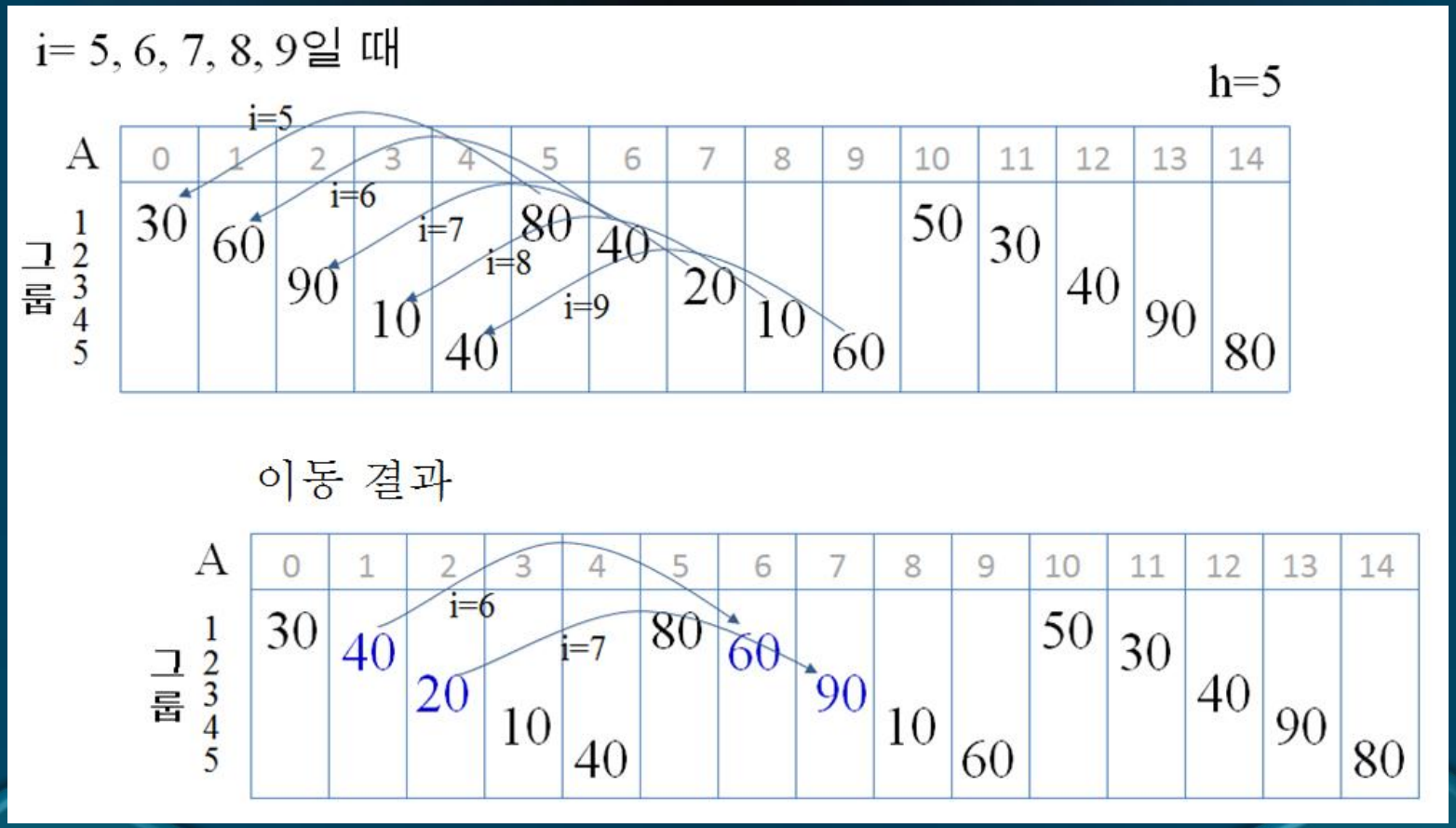
1. for each gap h = [$h_0 > h_1 > \ldots > h_k=1$] // 큰 gap부터 차례로
2. for i = h to n-1 {
3. CurrentElement=A[i];
4. j = i;
5. while (j>=h) and (A[j-h]>CurrentElement) {
6. A[j]=A[j-h];
7. j=j-h;
    }
8. A[j]=CurrentElement;
   }
9. return 배열 A

- 쉘 정렬은 간격 [$h_0 > h_1 > \cdots > h_k=1$]이 미리 정해져야 한다.
- 가장 큰 간격 h_0부터 차례로 간격에 따른 삽입 정렬이 line 2~8에서 수행된다.
- 마지막 간격 h_k는 반드시 1이어야 한다.
  - 이는 간격에 대해서 그룹별로 삽입 정렬을 수행하였기 때문에, 아직 비교 되지 않은 다른 그룹의 숫자가 있을 수 있기 때문이다.

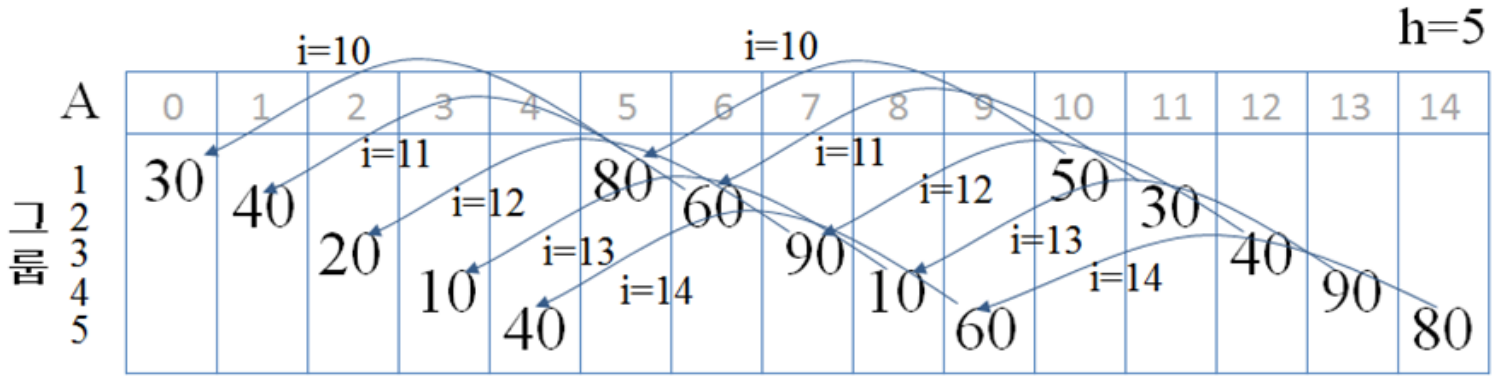
- Line 2~8의 for-루프에서는 간격 h에 대하여 삽입 정렬이 수행되는데, 핵심 아이디어에서 설명한대로 그룹 별로 삽입 정렬을 수행하지만 자리바꿈을 위한 원소간의 비교는 다음과 같은 순서로 진행된다.



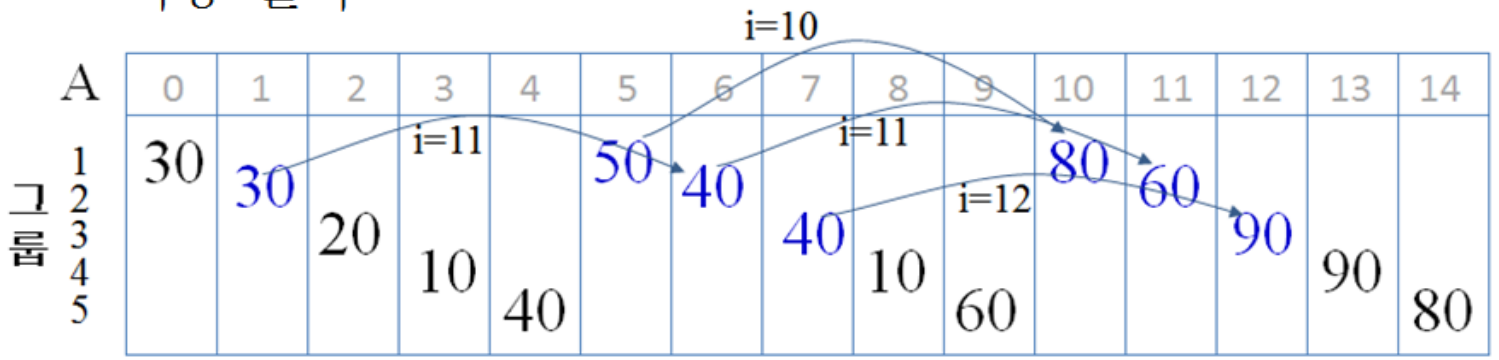
- 간격이 5일 때, 쉘 정렬의 수행 과정



i=10, 11, 12, 13, 14



이동 결과



- h=5일 때의 결과를 한 줄에 나열해보면 다음과 같다.

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

- 이제 간격을 줄여서 h=3이 되면, 배열의 원소가 3개의 그룹으로 나누어진다.
  - 그룹1은 0번째, 3번째, 6번째, 9번째, 12번째 숫자이고,
  - 그룹2는 1번째, 4번째, 7번째, 10번째, 13번째 숫자로 구성되고,
  - 마지막으로 그룹3은 2번째, 5번째, 8번째, 11번째, 14번째 숫자이다.
- 각 그룹 별로 삽입 정렬하면,

그룹1	그룹2	그룹3
30	30	20
10	40	50
40	40	10
60	80	60
90	90	80

그룹1	그룹2	그룹3
10	30	10
30	40	20
40	40	50
60	80	60
90	90	80

- 각 그룹 별로 정렬한 결과를 한 줄에 나열해보면 다음과 같다. 즉, 이것이 h=3일 때의 결과이다.

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

- 마지막으로 위의 배열에 대해 h=1일 때 알고리즘을 수행하면 아래와 같이 정렬된 결과를 얻는다.
- h=1일 때는 간격이 1 (즉, 그룹이 1개)이므로, 삽입 정렬과 동일하다.

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

- 쉘 정렬의 수행 속도는 간격 선정에 따라 좌우된다.
- 지금까지 알려진 가장 좋은 성능을 보인 간격:
  - 1, 4, 10, 23, 57, 132, 301, 701
  - 701 이후는 아직 밝혀지지 않았다.

시간복잡도

- 쉘 정렬, 최악의 경우 시간복잡도는 $O(n^2)$이다.
- 히바드(Hibbard)의 간격:
  - 2^k-1 (즉, 2^k-1, …, 15, 7, 5, 3, 1)을 사용하면, 시간복잡도는 $O(n^{1.5})$이다.
- 많은 실험을 통한 쉘 정렬의 시간복잡도는 $O(n^{1.25})$으로 알려지고 있다.
- 쉘 정렬의 시간복잡도는 아직 풀리지 않은 문제
  - 이는 가장 좋은 간격을 알아내야 하는 것이 선행되어야 하기 때문이다.

응 용

- 쉘 정렬은 입력 크기가 매우 크지 않은 경우에 매우 좋은 성능을 보인다.
- 쉘 정렬은 임베디드(Embedded) 시스템에서 주로 사용되는데,
  쉘 정렬의 특징인 간격에 따른 그룹 별 정렬 방식이
  H/W로 정렬 알고리즘을 구현하는데 매우 적합하기 때문이다.

요약

- 정렬 알고리즘은 크게 내부정렬 (Internal sort)과 외부정렬 (External sort)로도 분류한다.
- 버블 정렬 (Bubble Sort)은 이웃하는 숫자를 비교하여 작은 수를 앞쪽으로 이동시키는 과정을 반복하여 정렬이고, 시간복잡도는 $O(n^2)$이다.
- 선택 정렬 (Selection Sort)은 매번 최소값을 선택하여 정렬하며, 시간복잡도는 $O(n^2)$이다.
- 삽입 정렬 (Insertion Sort)은 정렬 안 된 부분에 있는 원소 하나를 정렬된 부분의 알맞은 위치에 삽입하여 정렬하며, 시간복잡도는 $O(n^2)$이다. 또한 최선 경우 시간복잡도는 $O(n)$이고, 평균 경우 시간복잡도는 $O(n^2)$이다.

요약(계속)

- 쉘 정렬 (Shell Sort)은 삽입 정렬을 이용하여 배열 뒷부분의 작은 숫자를 앞부분으로, 동시에 앞부분의 큰 숫자는 뒷부분으로 '빠르게' 이동시키는 과정을 반복하여 정렬하는 알고리즘이다. 시간복잡도는 최악의 경우 $O(n^2)$이다.

실 습

실습1: 버블정렬 알고리즘 구현하기

실습2: 삽입정렬 알고리즘 구현하기

숙제:

1. 쉘정렬 알고리즘 구현하기
2. 버블정렬, 삽입정렬, 쉘정렬 실행시간 비교

Q&A