13주차 2차시 정렬 2

【학습목표】

1. 자료의 정렬 방법을 설명할 수 있다.
2. 정렬 방법에 대한 알고리즘을 설명할 수 있다.

학습내용1 : 삽입 정렬

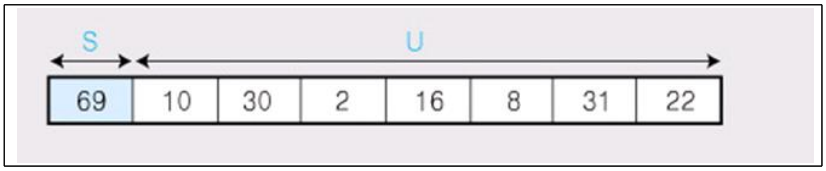
1. 개요

* 정렬되어있는 부분집합에 정렬할 새로운 원소의 위치를 찾아 삽입하는 방법

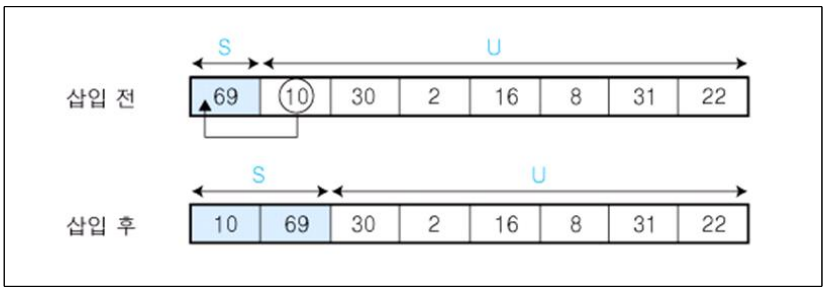
* 정렬할 자료를 두 개의 부분집합 S와 U로 가정
- 부분집합 S : 정렬된 앞부분의 원소들
- 부분집합 U : 아직 정렬되지 않은 나머지 원소들
- 정렬되지 않은 부분집합 U의 원소를 하나씩 꺼내서 이미 정렬되어있는 부분집합 S의 마지막 원소부터 비교하면서 위치를 찾아 삽입
- 삽입 정렬을 반복하면서 부분집합 S의 원소는 하나씩 늘리고 부분집합 U의 원소는 하나씩 감소하게 한다. 부분집합 U가 공집합이 되면 삽입 정렬이 완성된다.

2. 삽입 정렬 수행 과정

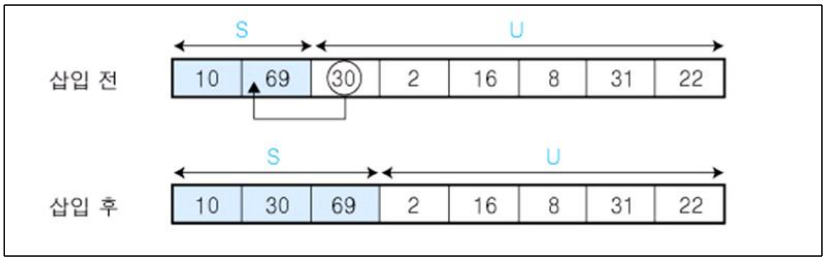
* 정렬되지 않은 {69, 10, 30, 2, 16, 8, 31, 22 }의 자료들을 삽입 정렬 방법으로 정렬하는 과정을 살펴보자.
- 초기 상태 : 첫 번째 원소는 정렬되어있는 부분 집합 S로 생각하고 나머지 원소들은 정렬되지 않은 원소들의 부분 집합 U로 생각한다.
- S= {69 } , U= {10, 30, 2, 16, 8, 31, 22 }



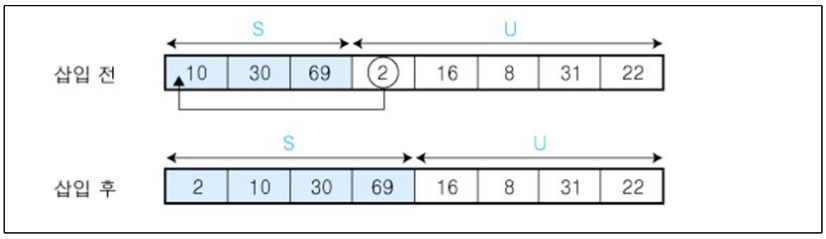
① U의 첫 번째 원소 10을 S의 마지막 원소 69와 비교하여 (10 < 69) 이므로 원소 10은 원소 69의 앞자리가 된다. 더 이상 비교할 S의 원소가 없으므로 찾은 위치에 원소 10을 삽입한다.
- S= {10, 69 } , U= {30, 2, 16, 8, 31, 22 }



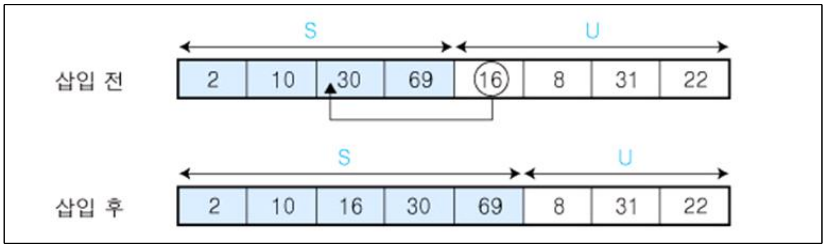
② U의 첫 번째 원소 30을 S의 마지막 원소 69와 비교하여 (30 < 69)이므로 원소 69의 앞자리 원소 10과 비교한다.
- (30 > 10) 이므로 원소 10과 69 사이에 삽입한다.
- S= {10, 30, 69 } , U= {2, 16, 8, 31, 22 }



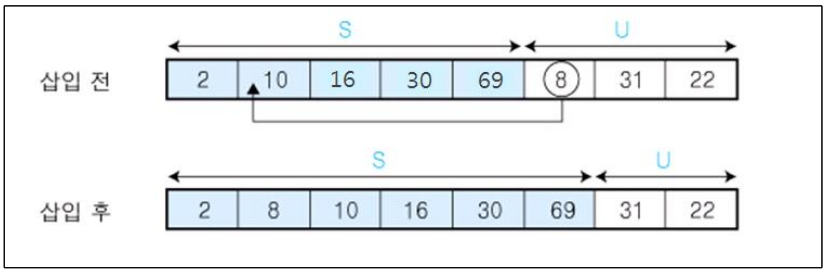
③ U의 첫 번째 원소 2를 S의 마지막 원소 69와 비교하여 (2 < 69) 이므로 원소 69의 앞자리 원소 30과 비교하고, (2 < 30) 이므로 다시 그 앞자리 원소 10과 비교하는데, (2 < 10) 이면서 더 이상 비교할 S의 원소가 없으므로 원소 10의 앞에 삽입한다.
- S= {2, 10, 30, 69} , U= {16, 8, 31, 22 }



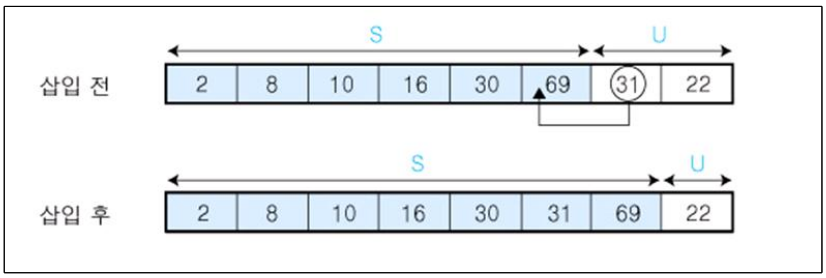
④ U의 첫 번째 원소 16을 S의 마지막 원소 69와 비교하여 (16 < 69)이므로 그 앞자리 원소 30과 비교한다. (16 < 30) 이므로 다시 그 앞자리 원소 10과 비교하여, (16 > 10)이므로 원소 10과 30 사이에 삽입한다.
- S= {2, 10, 16, 30, 69 } , U= {8, 31, 22 }



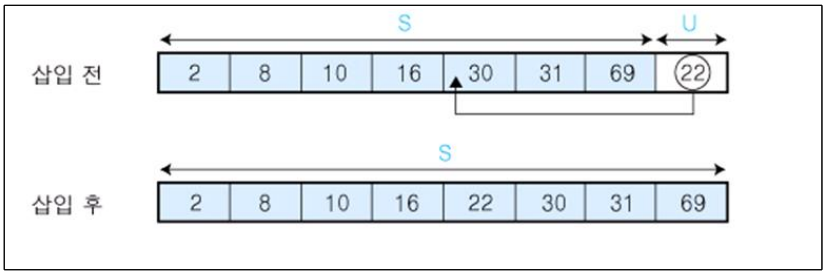
⑤ U의 첫 번째 원소 8을 S의 마지막 원소 69와 비교하여 (8 < 69)이므로 그 앞자리 원소 30과 비교한다.
(8 < 30) 이므로 그 앞자리 원소 16과 비교하여, (8 < 16) 이므로 그 앞자리 원소 10과 비교하여, (8 < 10) 이므로 다시 그 앞자리 원소 2와 비교하는데, (8 > 2)이므로 원소 2와 10 사이에 삽입한다.
- S= {2, 8, 10, 16, 30, 69 } , U= {31, 22 }



⑥ U의 첫 번째 원소 31을 S의 마지막 원소 69와 비교하여 (31 < 69)이므로 그 앞자리 원소 30과 비교한다.
(31 > 30) 이므로 원소 30과 69 사이에 삽입한다.
- S= {2, 8, 10, 16, 30, 31, 69 } , U= {22 }



⑦ U의 첫 번째 원소 22를 S의 마지막 원소 69와 비교하여 (22 < 69)이므로 그 앞자리 원소 31과 비교한다.
(22 < 31) 이므로 그 앞자리 원소 30과 비교하고, (22 < 30) 이므로 다시 그 앞자리 원소 16과 비교하여, (22 > 16) 이므로 원소 16과 30 사이에 삽입한다.
- S= {2, 8, 10, 16, 22, 30, 31, 69} , U={ }



- U가 공집합이 되었으므로 실행을 종료하고 삽입 정렬이 완성

3. 삽입 정렬 알고리즘

알고리즘 10-5 삽입 정렬 알고리즘

```
insertionSort(a[],n)
    for (i←1; i<n; i←i+1) {
        temp ← a[i];
        j ← i;
        if (a[j-1]>temp) then move ← true;
        else move ← false;
        while (move) do {
            a[j] ← a[j-1];
            j ← j-1;
            if (j>0 and a[j-1]>temp) then move ← true;
            else move ← false;
        }
        a[j] ← temp;
    }
end insertionSort()
```

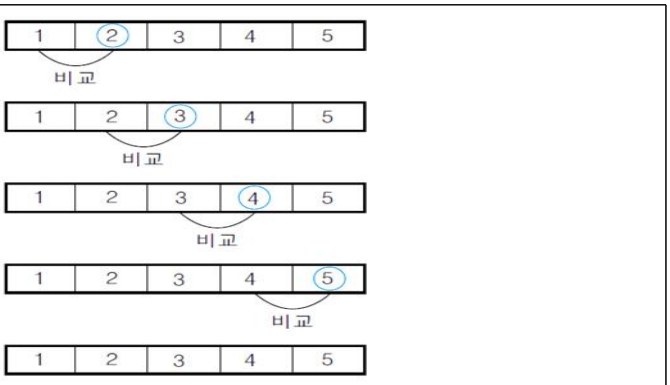
* 메모리 공간
- n개의 원소에 대하여 n개의 메모리 사용

* 연산 시간
- 최선의 경우 : 원소들이 이미 정렬되어있어서 비교횟수가 최소인 경우
    - 이미 정렬되어있는 경우에는 바로 앞자리 원소와 한번만 비교한다.
    - 전체 비교횟수 = n-1
    - 시간 복잡도 : O(n)

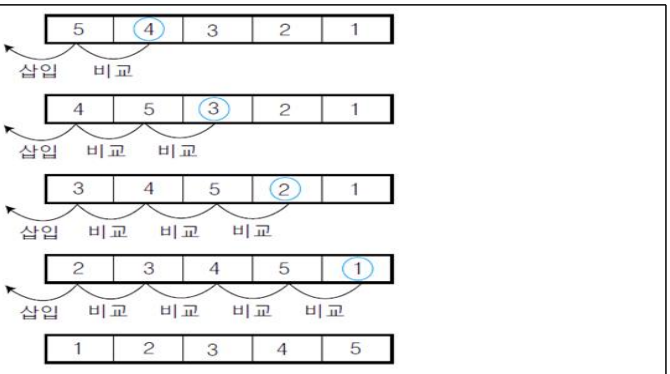
- 최악의 경우 : 모든 원소가 역순으로 되어있어서 비교횟수가 최대인 경우
    - 전체 비교횟수 = 1+2+3+ … +(n-1) = n(n-1)/2
    - 시간 복잡도 : $O(n^2)$

- 삽입 정렬의 평균 비교횟수 = n(n-1)/4
- 평균 시간 복잡도 : $O(n^2)$

- 최선의 경우와 최악의 경우 예



[그림 10-2] 이미 정렬되어 있는 원소에서의 삽입 정렬



[그림 10-3] 역순으로 정렬되어 있는 원소에서의 삽입 정렬

학습내용2 : 셸 정렬

1. 개요

* 일정한 간격(interval)으로 떨어져있는 자료들끼리 부분집합을 구성하고 각 부분집합에 있는 원소들에 대해서 삽입 정렬을 수행하는 작업을 반복하면서 전체 원소들을 정렬하는 방법
- 전체 원소에 대해서 삽입 정렬을 수행하는 것보다 부분집합으로 나누어 정렬하게 되면 비교연산과 교환연산 감소

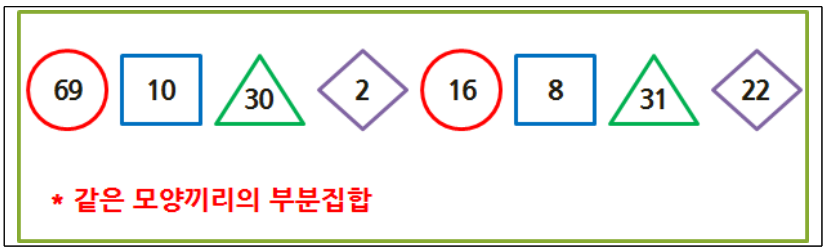
* 셸 정렬의 부분집합
- 부분집합의 기준이 되는 간격을 매개변수 h에 저장
- 한 단계가 수행될 때마다 h의 값을 감소시키고 셸 정렬을 순환 호출
    - h가 1이 될 때까지 반복

* 셸 정렬의 성능은 매개변수 h의 값에 따라 달라진다.
- 정렬할 자료의 특성에 따라 매개변수 생성 함수를 사용
- 일반적으로 사용하는 h의 값은 원소 개수의 1/2을 사용하고 한 단계 수행될 때마다 h의 값을 반으로 감소시키면서 반복 수행

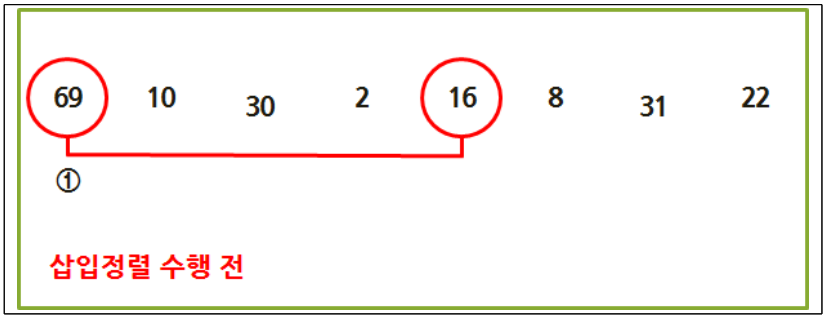
2. 셸 정렬 수행과정

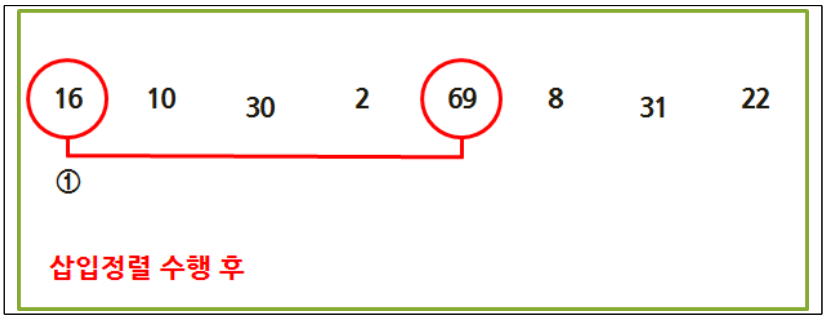
* 정렬되지 않은 {69, 10, 30, 2, 16, 8, 31, 22 }의 자료들을 셸 정렬 방법으로 정렬하는 과정을 살펴보자.

① 원소의 개수가 8개이므로 매개변수 h는 4에서 시작함
- h=4 이므로 간격이 4인 원소들을 같은 부분 집합으로 만들면 4개의 부분 집합이 만들어짐

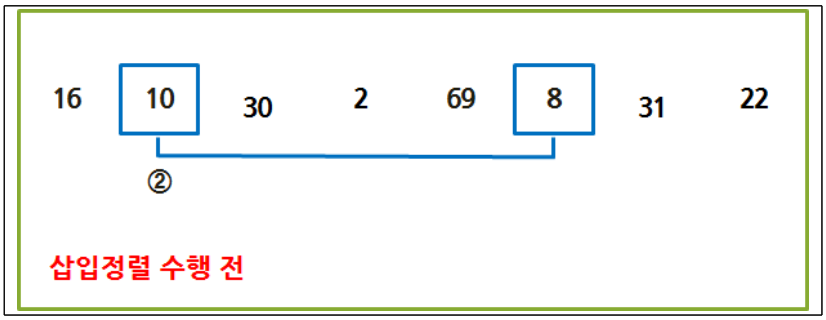


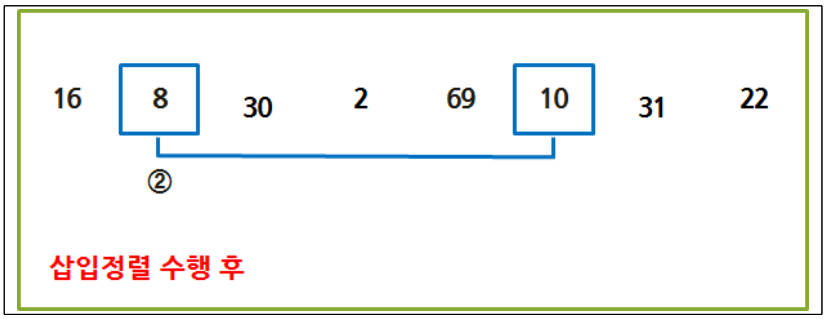
- 첫 번째 부분 집합 {69, 16 }에 대해서 삽입 정렬을 수행하여 정렬



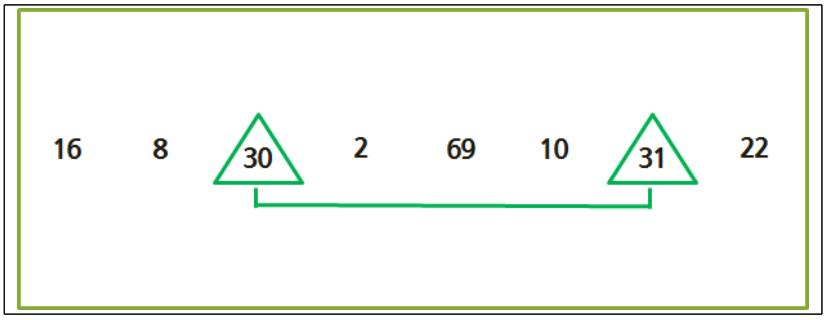


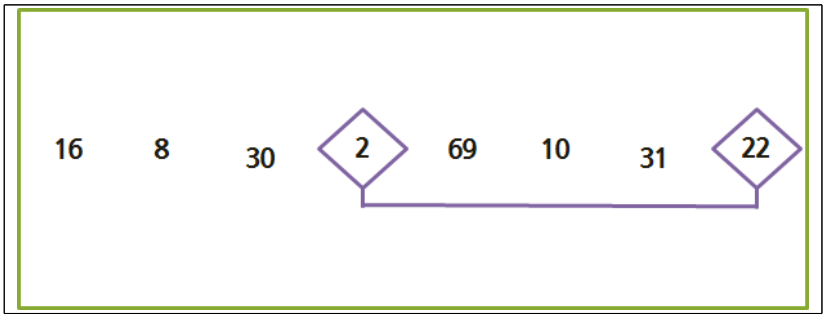
- 두 번째 부분 집합 {10, 8 }에 대해서 삽입 정렬 수행





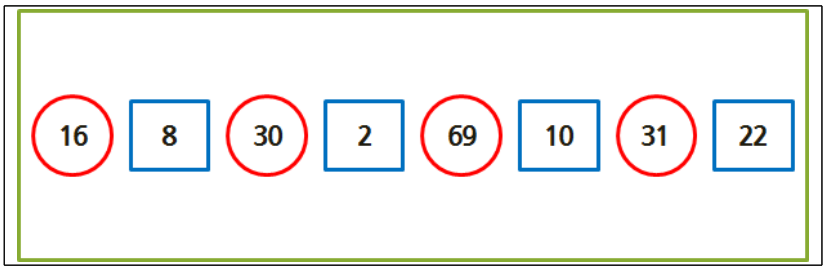
- 세 번째 부분집합 {30, 31 }에 대해서 삽입 정렬을 수행하는데, (30 < 31) 이므로 자리 교환은 이루어지지 않음



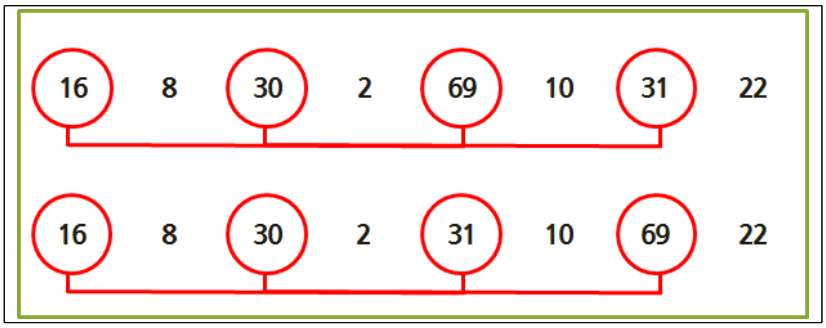


② 이제 h를 2로 변경하고 다시 셸 정렬 시작

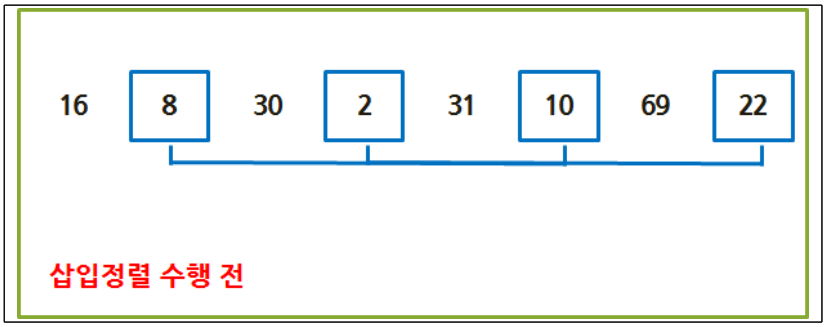
- h=2 이므로 간격이 2인 원소들을 같은 부분 집합으로 만들면 2개의 부분 집합이 만들어짐

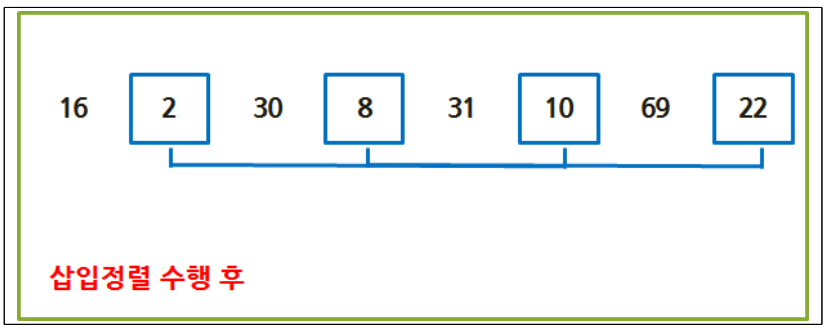


- 첫 번째 부분 집합 {16, 30, 69, 31 }에 대해서 삽입 정렬을 수행하여 정렬



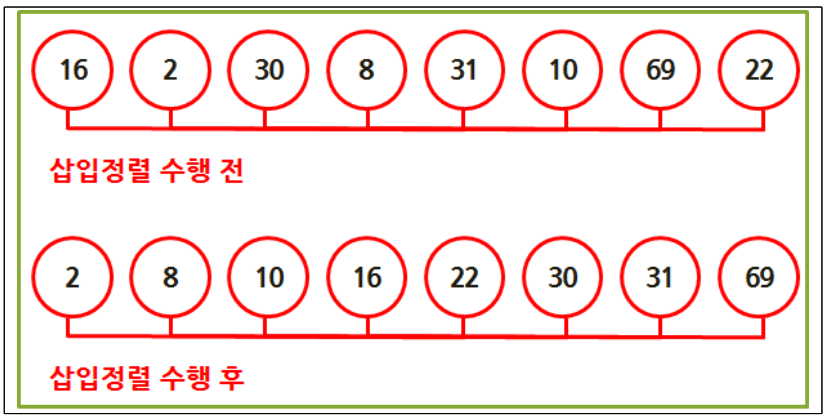
- 두 번째 부분 집합 {8, 2, 10, 22 }에 대해서 삽입 정렬을 수행하여 정렬





③ 이제 h를 1로 변경하고 다시 셸 정렬 시작

- h=1 이므로 간격이 1인 원소들을 같은 부분 집합으로 만들면 1개의 부분 집합이 만들어짐
- 즉, 전체 원소에 대해서 삽입 정렬을 수행하고 셸 정렬이 완성됨



3. 셸 정렬 알고리즘

알고리즘 10-6 셸 정렬 알고리즘

```
shellSort(a[], n)
    interval ← n;
    while (interval ≥ 1) do {
        interval ← interval/2;
        for (i←0; i<interval; i←i+1) do {
            intervalSort(a[], i, n, interval);
        }
    }
end shellSort()
```

* 메모리 사용공간 : n개의 원소에 대하여 n개의 메모리와 매개변수 h에 대한 저장공간 사용

* 연산 시간
- 비교횟수
    - 처음 원소의 상태에 상관없이 매개변수 h에 의해 결정
- 일반적인 시간 복잡도 : $O(n^{1.25})$
- 셸 정렬은 삽입 정렬의 시간 복잡도 $O(n^2)$ 보다
- 개선된 정렬 방법

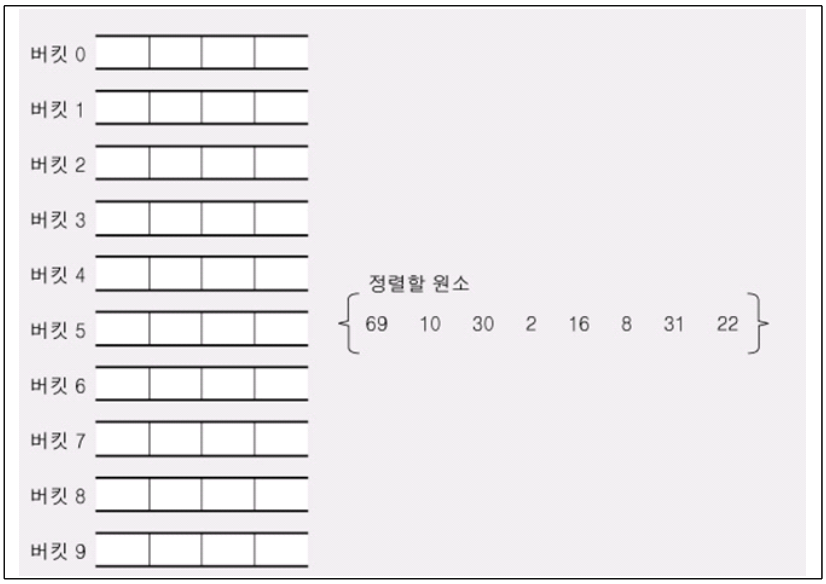
학습내용3 : 퀵 정렬

1. 개요

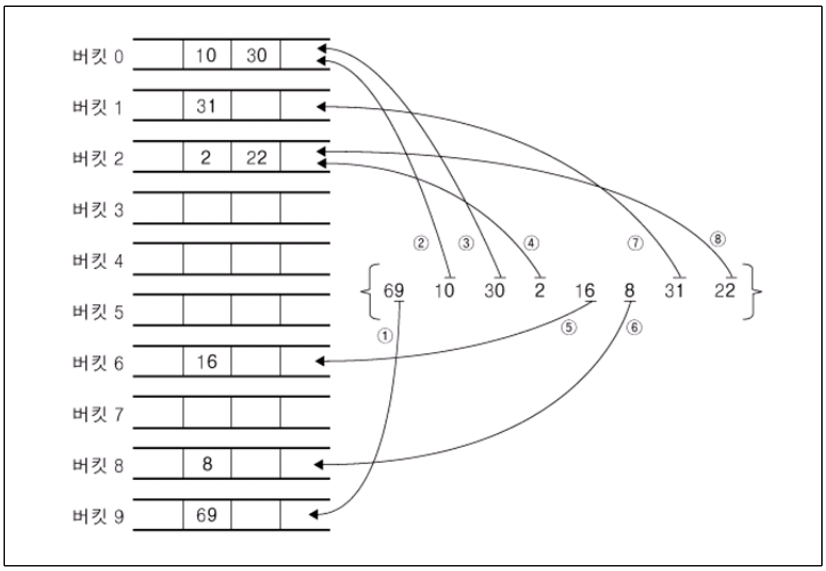
* 원소의 키값을 나타내는 기수를 이용한 정렬 방법
- 정렬할 원소의 키 값에 해당하는 버킷(bucket)에 원소를 분배하였다가 버킷의 순서대로 원소를 꺼내는 방법을 반복하면서 정렬
    - 원소의 키를 표현하는 기수만큼의 버킷 사용
    - 예) 10진수로 표현된 키 값을 가진 원소들을 정렬할 때에는 0부터 9까지 10개의 버킷 사용
- 키 값의 자리수 만큼 기수 정렬을 반복
    - 키 값의 일의 자리에 대해서 기수 정렬을 수행하고,
    - 다음 단계에서는 키 값의 십의 자리에 대해서,
    - 그리고 그 다음 단계에서는 백의 자리에 대해서 기수 정렬 수행
- 한 단계가 끝날 때마다 버킷에 분배된 원소들을 버킷의 순서대로 꺼내서 다음 단계의 기수 정렬을 수행해야 하므로 큐를 사용하여 버킷을 만든다.

2. 기수 정렬 수행과정

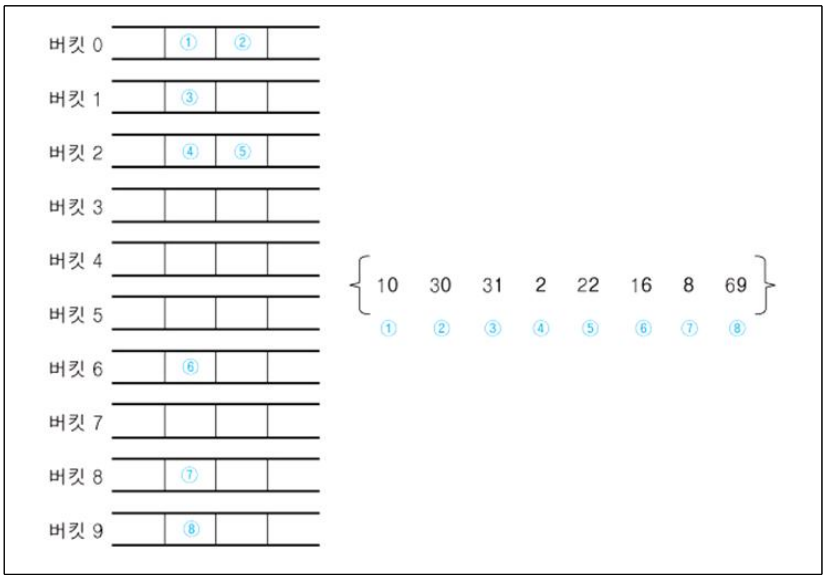
* 정렬되지 않은 {69, 10, 30, 2, 16, 8, 31, 22 }의 자료들을 기수 정렬 방법으로 정렬하는 과정
- 키 값이 두 자리 십진수 이므로, 10개의 버킷을 사용하여 기수 정렬을 두 번 반복한다.
- 초기 상태 : 큐를 사용하여 0부터 9까지 10개의 버킷을 만든다



① 키 값의 일의 자리에 대해서 기수 정렬 수행
- 정렬할 원소의 일의 자리 값에 따라서 순서대로 버킷에 분배

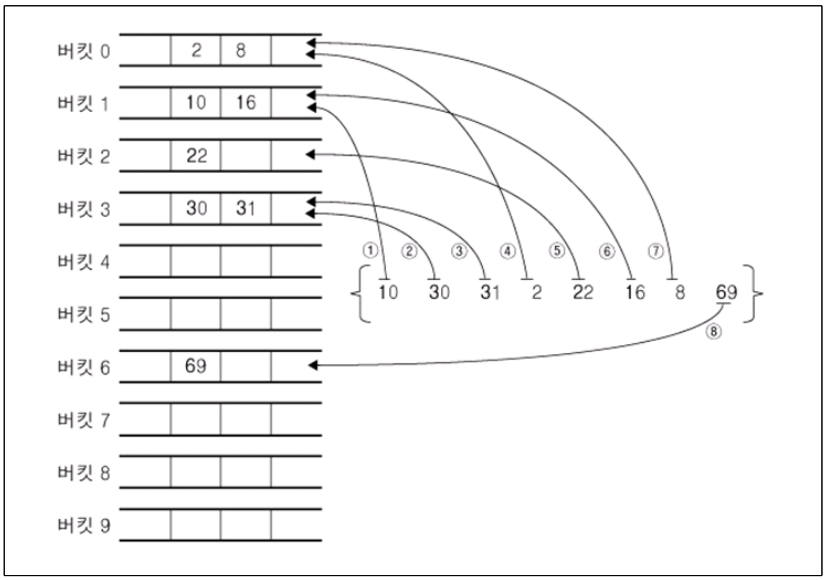


- 버킷에 분배된 원소들을 순서대로 꺼내서 저장하기

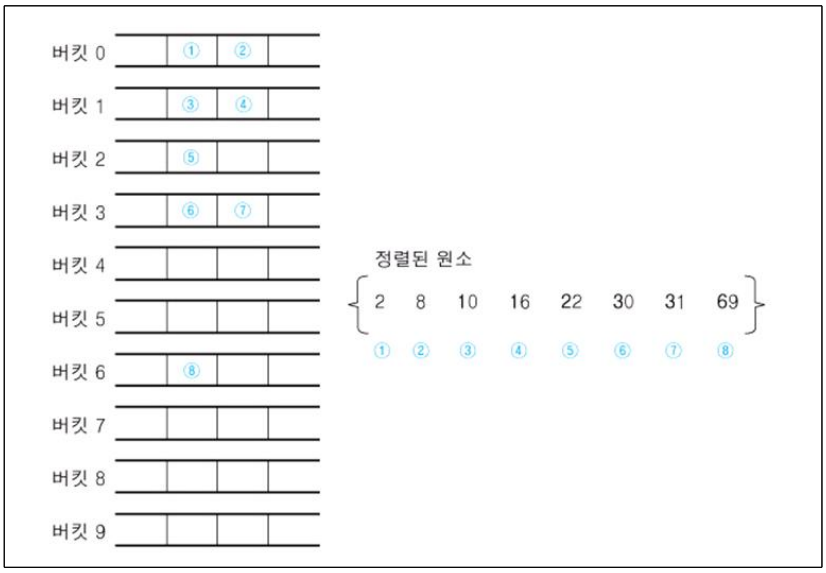


② 키 값의 십의 자리에 대해서 기수 정렬 수행

- 정렬할 원소의 십의 자리 값에 따라서 순서대로 버킷에 분배



- 버킷에 분배된 원소들을 순서대로 꺼내서 저장하기



3. 기수 정렬 알고리즘

알고리즘 10-8 기수 정렬 알고리즘

```
radixSort(a[], n)
    for (k ← 1; k≤ n; k ← k+1) do {
        for (i ← 0; i < n; i ← i+1) do {
            k번째 자릿수 값에 따라서 해당 버킷에 저장한다;
            enQueue(Q[k], a[i]);
        }
        p ← -1;
        for (i ← 0; i ≤ 9; i ≤ i+1) do {
            while (Q[i] ≠ NULL) do {
                p ← p+1;
                a[p] ← deQueue(Q[i]);
            }
        }
    }
end radixSort()
```

* 메모리 사용공간
- 원소 n개에 대해서 n개의 메모리 공간 사용
- 기수 r에 따라 버킷 공간이 추가로 필요

* 연산 시간
- 연산 시간은 정렬할 원소의 수 n과 키 값의 자릿수 d와 버킷의 수를 결정하는 기수 r 따라서 달라진다.
    - 정렬할 원소 n개를 r개의 버킷에 분배하는 작업 : (n+r)
    - 이 작업을 자릿수 d 만큼 반복
- 수행할 전체 작업 : d(n+r)
- 시간 복잡도 : O(d(n+r))

【학습정리】

1. 삽입 방식에는 삽입 정렬과 셸 정렬이 있다.
- 삽입 정렬 : 정렬되어 있는 부분집합에 정렬할 새로운 원소의 순서를 찾아 삽입하는 방식
- 셸 정렬 : 일정한 간격으로 떨어져있는 자료들로 구성한 부분집합 단위로 삽입 정렬을 수행하는 방식
- 전체 원소에 대해서 정렬을 수행하는 삽입 정렬 방법에서 비교 연산과 교환 연산을 수행하기보다 부분집합으로 나누어 정렬하면 비교 횟수와 교환 연산을 줄일 수 있다

2. 분배 방식에는 기수 정렬이 있다. 기수 정렬은 분배 방식의 정렬 방법으로 정렬할 원소의 킷값에 해당하는 버킷에 원소를 분배하였다가 버킷의 순서대로 원소를 꺼내는 방법을 반복한다.
- 기수 정렬은 원소의 키를 표현하는 값의 기수만큼의 버킷이 필요하고, 킷값의 자릿수만큼 기수 정렬을 반복한다.