자료구조

Chap 6. Sorting

2017년 2학기

컴퓨터과학과
민 경 하

Contents

내용

6.1 소개

- 정렬 (sorting)
  - 데이터를 정해진 키에 따라서 크기 순으로 배열하는 것
    - 오름차순 (ascending order)
    - 내림차순 (descending order)

34 64 19 8 23

8 19 23 34 64 (오름차순)

64 34 23 19 8 (내림차순)

6.1 소개

- 정렬 알고리즘의 성능
  - 정렬할 데이터의 개수: n
  - O(f(n))으로 표시

  - $O(n^2)$ 정렬 알고리즘
    - 버블 정렬, 삽입 정렬, 선택 정렬, ……

  - O(n log n) 정렬 알고리즘
    - 합병 정렬, 쾌속 정렬, ……

6.2 버블 정렬 (bubble sort)

1) 기본 개념

- 배열의 가장 앞자리에 최소값을 옮김
  - 배열의 가장 뒷자리로부터 원소들을 차례로 비교하면서 작은 값을 가장 앞 자리로 옮김
- 배열의 두 번째 자리에 두 번째 작은 값을 옮김
  - 배열의 가장 뒷자리로부터 원소들을 차례로 비교하면서 작은 값을 두 번째 자리로 옮김
- 이 과정을 n번 반복함
- 마치 거품 (bubble)이 물밑에서 올라오는 듯 함

6.2 버블 정렬 (bubble sort)

2) 알고리즘 설계

– 배열의 임의의 위치 (i번째 위치)에서 다음의 연산을 수행함

- i번째 위치에 (i) ~ (n – 1)의 원소들 중에서 가장 작은 값이 옴

  → i번째에는 i번째 작은 값이 옴

- (n – 1)번째 위치부터 (i + 1)번째 위치까지 차근차근 비교하면서 진행함

6.2 버블 정렬 (bubble sort)

3) 알고리즘 구현

– 배열의 임의의 위치 (i번째 위치)에서 다음의 연산을 수행함

```
void bubble_sort ( int n, int a[] )
{
    int i;
    for ( i = 0; i < n-1; i++ ) {

    }
}
```

0번째부터 (n – 1)번째까지 차례대로 가장 작은 값을 저장함

i번째에는 i번째 작은 값을 저장

6.2 버블 정렬 (bubble sort)

3) 알고리즘 구현

– (n – 1)번째 위치부터 (i – 1)번째 위치까지 차근차근 비교하면서 진행함

```
void bubble_sort ( int n, int a[] )
{
    int i, j;
    for ( i = 0; i < n-1; i++ ) {
        for ( j = n-1; j > i; j-- ) {
            if ( a[j-1] > a[j] )
                swap ( a[j-1], a[j] );
        }
    }
}
```

(n – 1)번째부터 (i+1)번째까지 진행 → 밑에서부터 올라옴

바로 위에 있는 원소와 비교해서 아래의 원소가 더 작으면, swap

6.2 버블 정렬 (bubble sort)

3) 알고리즘 구현

– swap ( ) 함수

- 두 매개변수의 값을 바꾸는 함수
- Call-by-reference 이용

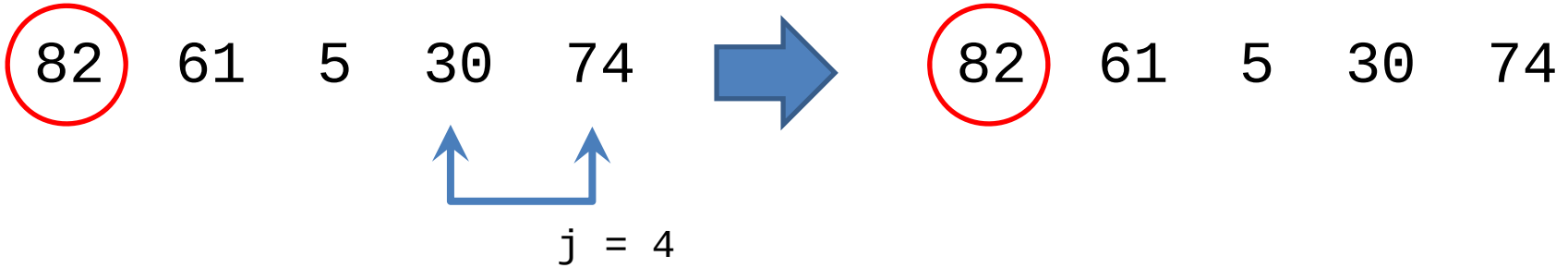
```
void swap (int & a, int & b)
{
    int x;
    x = a;
    a = b;
    b = x;
}
```

6.2 버블 정렬 (bubble sort)

4) 수행 예 (1)

```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 0
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 4
                        if ( a[j-1] > a[j] ) -----------→ a[3] < a[4]
                                swap ( a[j-1], a[j] );  -----→ No SWAP
                }
        }
}
```

i = 0

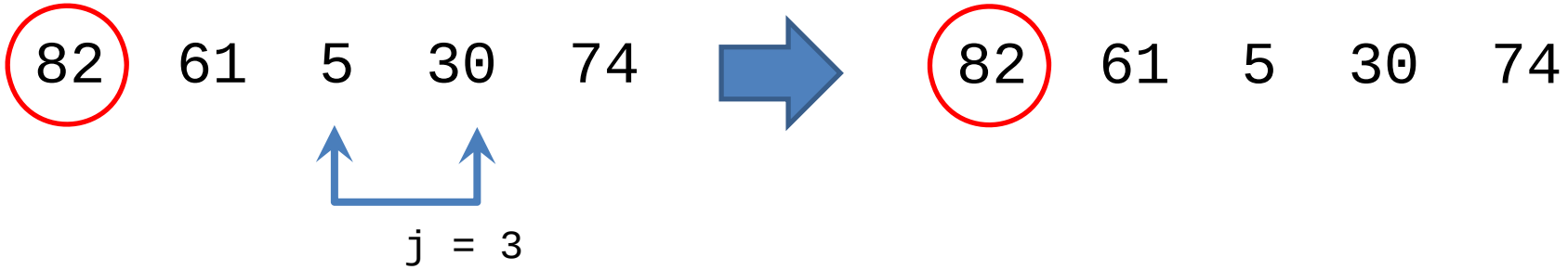


6.2 버블 정렬 (bubble sort)

4) 수행 예 (2)

```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 0
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 3
                        if ( a[j-1] > a[j] ) -----------→ a[2] < a[3]
                                swap ( a[j-1], a[j] );  -----→ No SWAP
                }
        }
}
```

i = 0



6.2 버블 정렬 (bubble sort)

4) 수행 예 (3)

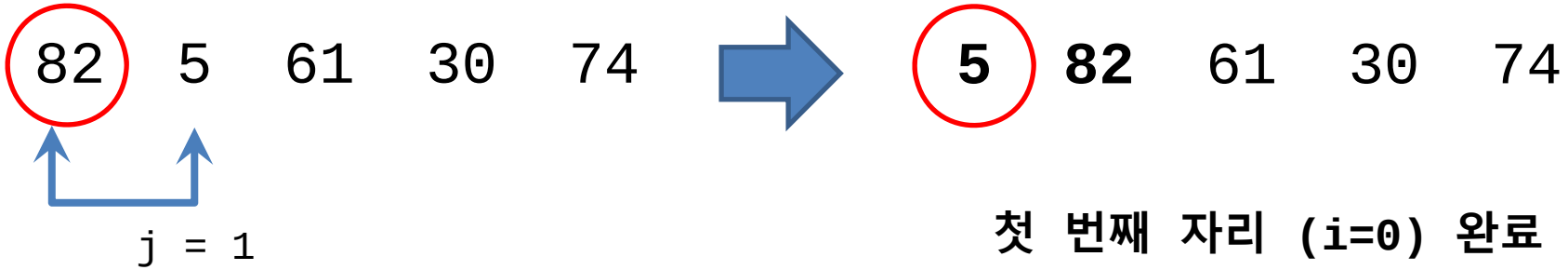
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 0
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 2
                        if ( a[j-1] > a[j] ) -----------→ a[1] > a[2]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```

6.2 버블 정렬 (bubble sort)

4) 수행 예 (4)

```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 0
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 1
                        if ( a[j-1] > a[j] ) -----------→ a[0] > a[1]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```

i = 0

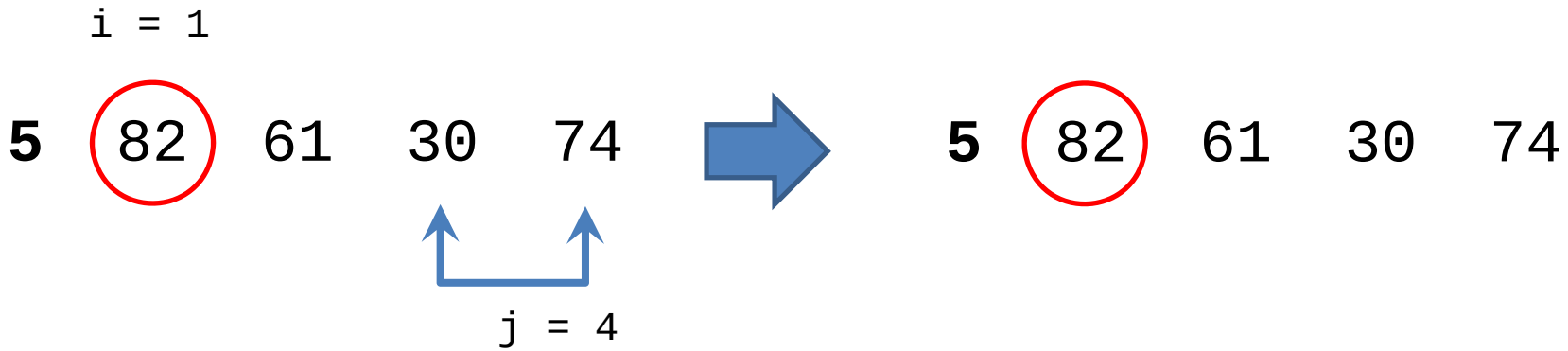


첫 번째 자리 (i=0) 완료

6.2 버블 정렬 (bubble sort)

4) 수행 예 (5)

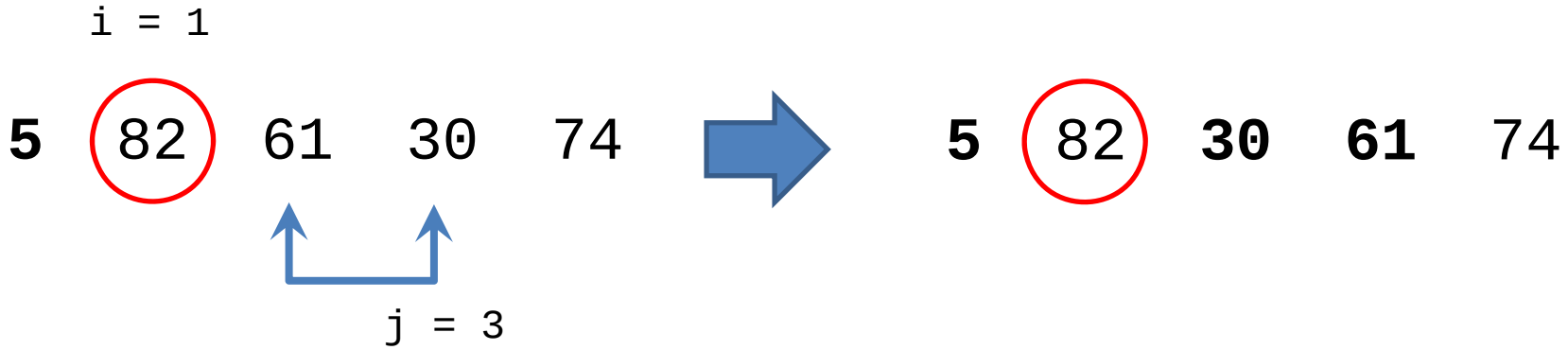
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 1
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 4
                        if ( a[j-1] > a[j] ) -----------→ a[3] < a[4]
                                swap ( a[j-1], a[j] );  -----→ No SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (6)

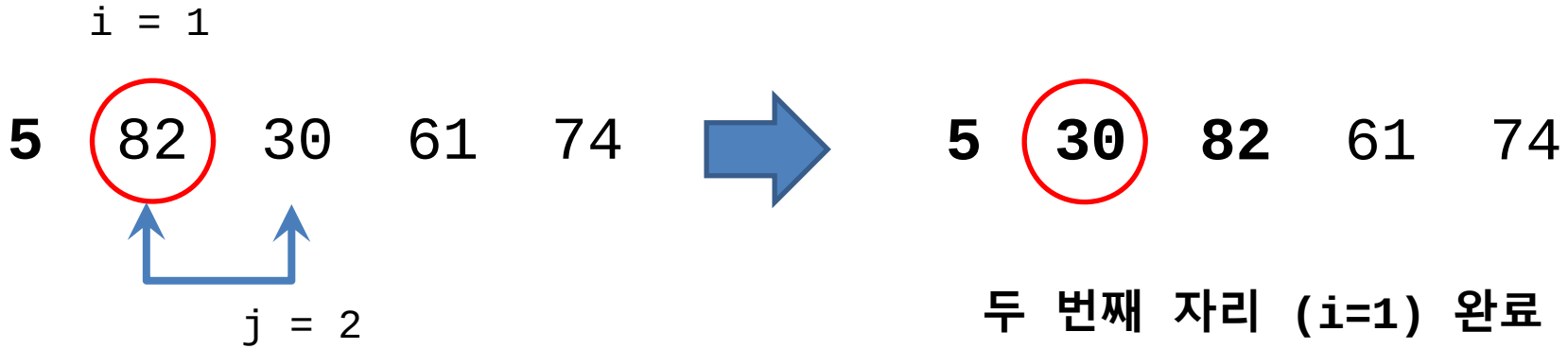
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 1
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 3
                        if ( a[j-1] > a[j] ) -----------→ a[2] > a[3]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (7)

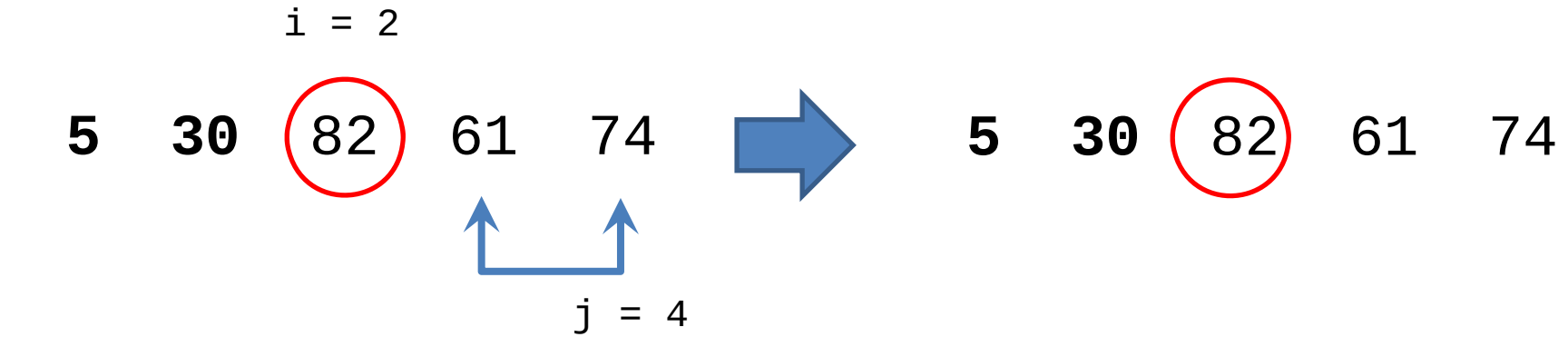
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 1
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 2
                        if ( a[j-1] > a[j] ) -----------→ a[1] > a[2]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (8)

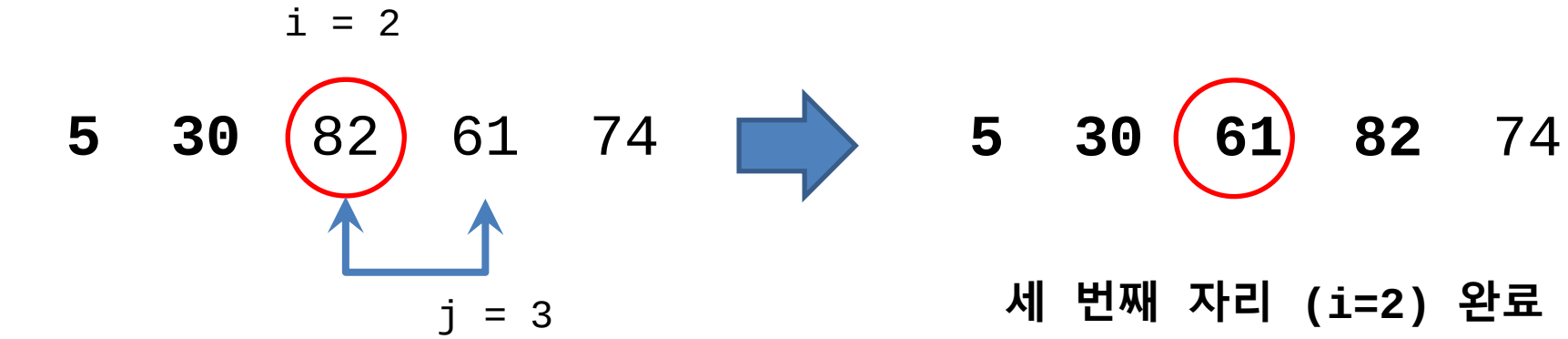
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 2
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 4
                        if ( a[j-1] > a[j] ) -----------→ a[3] < a[4]
                                swap ( a[j-1], a[j] );  -----→ No SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (9)

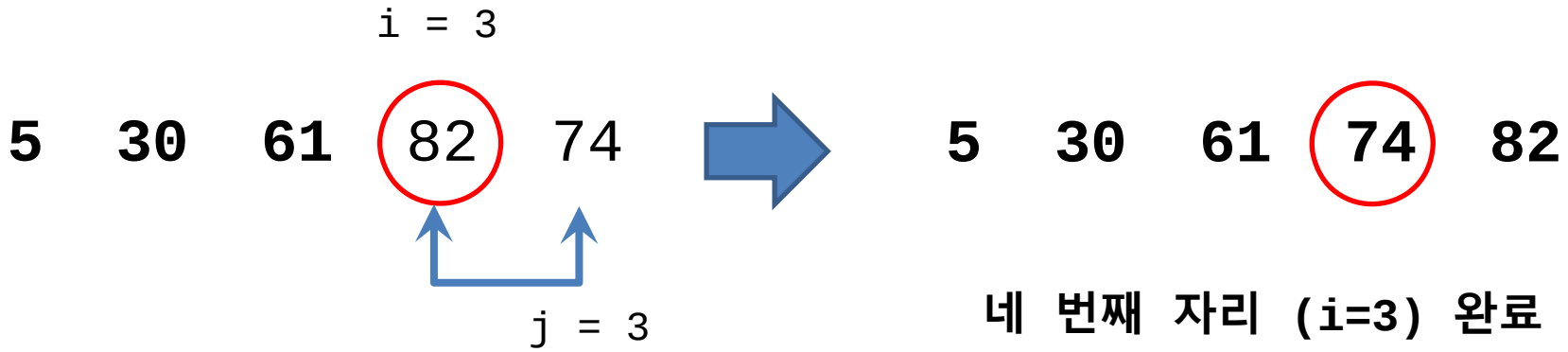
```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 2
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 3
                        if ( a[j-1] > a[j] ) -----------→ a[2] > a[3]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (10)

```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 3
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 4
                        if ( a[j-1] > a[j] ) -----------→ a[3] > a[4]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```



6.2 버블 정렬 (bubble sort)

4) 수행 예 (11)

```
void bubble_sort ( int n, int a[] )
{       int i, j;
        for ( i = 0; i < n-1; i++ ) { ------------------------→ i = 3
                for ( j = n-1; j > i; j-- ) { ------------ ---→ j = 4
                        if ( a[j-1] > a[j] ) -----------→ a[3] > a[4]
                                swap ( a[j-1], a[j] );  -----→ SWAP
                }
        }
}
```

5 30 61 74 82

정렬 완료

6.2 버블 정렬 (bubble sort)

5) 성능 분석

– 안쪽 for-loop의 수행 횟수 = $n(n-1)/2 = O(n^2)$

```
void bubble_sort ( int n, int a[] )
{
    int i, j;
        for ( i = 0; i < n-1; i++ ) {
            for ( j = n-1; j > i; j-- ) {
               if ( a[j-1] > a[j] )
                   swap ( a[j-1], a[j] );
            }
        }
}
```

i = 0 → n – 1 times
i = 1 → n – 2 times
……
i = n – 2 → 1 time

6.3 삽입 정렬 (insertion sort)

1) 기본 개념 (1)

- 배열의 가장 앞자리로부터 정렬된 부분 배열을 생성하고 그 크기를 증가시킴
  - 부분 배열은 처음에는 배열의 첫 원소로부터 시작
  - 마지막으로 부분 배열이 배열과 일치하게 되면 정렬 완료
- 새로운 원소를 정렬된 부분 배열의 적당한 위치에 삽입하여 부분 배열의 정렬을 유지함

6.3 삽입 정렬 (insertion sort)

1) 기본 개념 (2)

- 정렬된 배열에 새로운 원소를 삽입하는 연산이 필요함
- 삽입 후에도 배열의 정렬이 유지되어야 함

```
{9   14   30   52   61   -1   -1   -1}  +  44

{9   14   30   44   52   61   -1   -1}
```

```
void insert ( int x, int n, int b[] )
{

}
```

6.3 삽입 정렬 (insertion sort)

1) 기본 개념 (3)

```
void insert ( int x int n, int b[] )
{
        for ( int i = 0; i < n; i++ ) {
                if ( b[i] >= x ) {
                        for ( j = n-1; j >= i; j-- )
                                b[j] = b[j-1];
                        b[i] = x;
                        break;
                }
        }
}
```

6.3 삽입 정렬 (insertion sort)

2) 알고리즘 설계

- 배열의 원소를 하나씩 부분 배열에 삽입하면서 정렬을 수행
  - 부분 배열: b[ ]
- 부분 배열 b[ ]의 초기화
  - b[0] = a[0], b[1..n] = -1
- 부분 배열 b[ ]가 b[0..i-1]까지 형성된 상태에서 a[i]를 b[0..i-1]에 삽입
  → 부분배열 b의 개수 증가

6.3 삽입 정렬 (insertion sort)

3) 알고리즘 구현

```
void insertion_sort ( int b[], int n, int a[] )
{
    int i;
    for ( i = 1, b[0] = a[0]; i < n; i++ )
        b[i] = -1;

    for ( i = 1; i < n; i++ )
        insert ( a[i], n, b );
}
```

부분배열 초기화
- b[0] = a[0]
- b[1] ~ b[n-1] = -1

i번째 원소를 부분 배열에 삽입

6.3 삽입 정렬 (insertion sort)

```
void insert ( int x int n, int b[] )
{       int i, j;
        for ( i = 0; i < n; i++ ) {
                if ( b[i] >= x ) {
                        for ( j = n-1; j >= i; j-- )
                                b[j] = b[j-1];
                        b[i] = x;
                        break;
                }
        }
}
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ ) {
                insert ( a[i], n, b );
        }
}
```

6.3 삽입 정렬 (insertion sort)

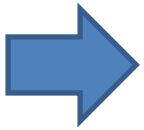
4) 수행 예 (초기화)

```
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ )
                insert ( a[i], n, b );
}
```

a[]:

82	61	5	30	74

b[]:



a[]:

82	61	5	30	74

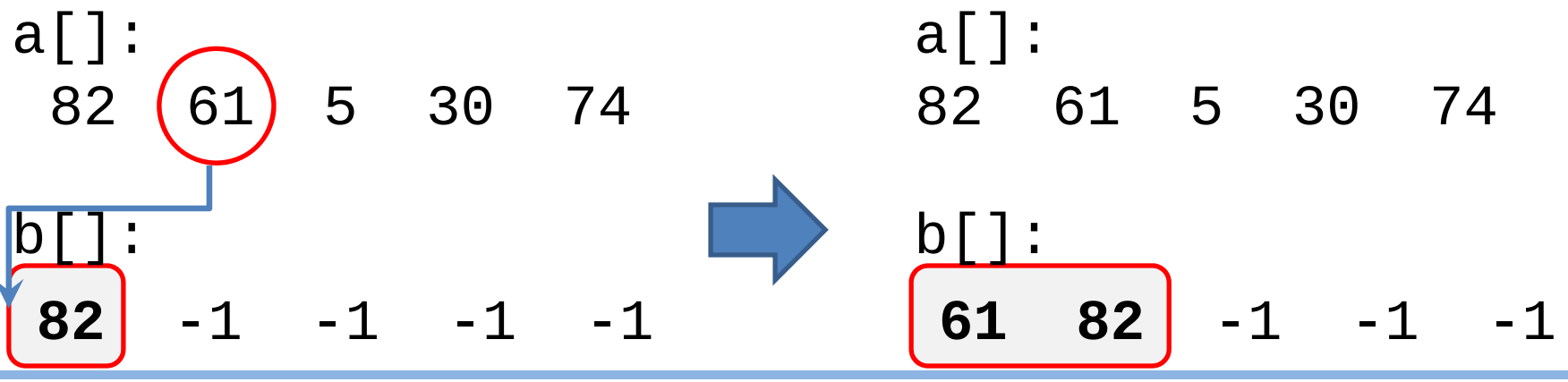
b[]:

82	-1	-1	-1	-1

6.3 삽입 정렬 (insertion sort)

4) 수행 예 (i = 1)

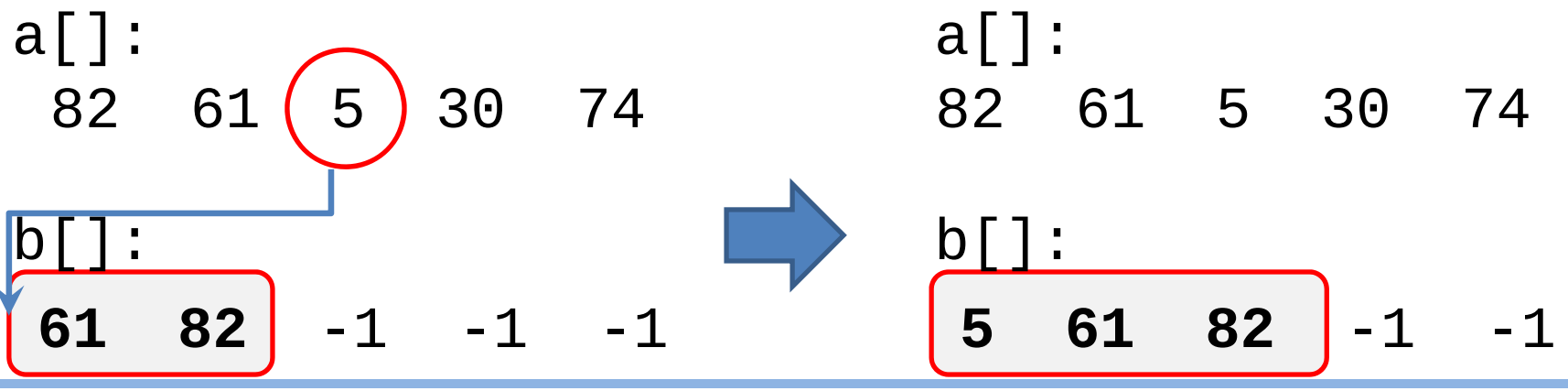
```
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ )
                insert ( a[i], n, b );
}
```



6.3 삽입 정렬 (insertion sort)

4) 수행 예 (i = 2)

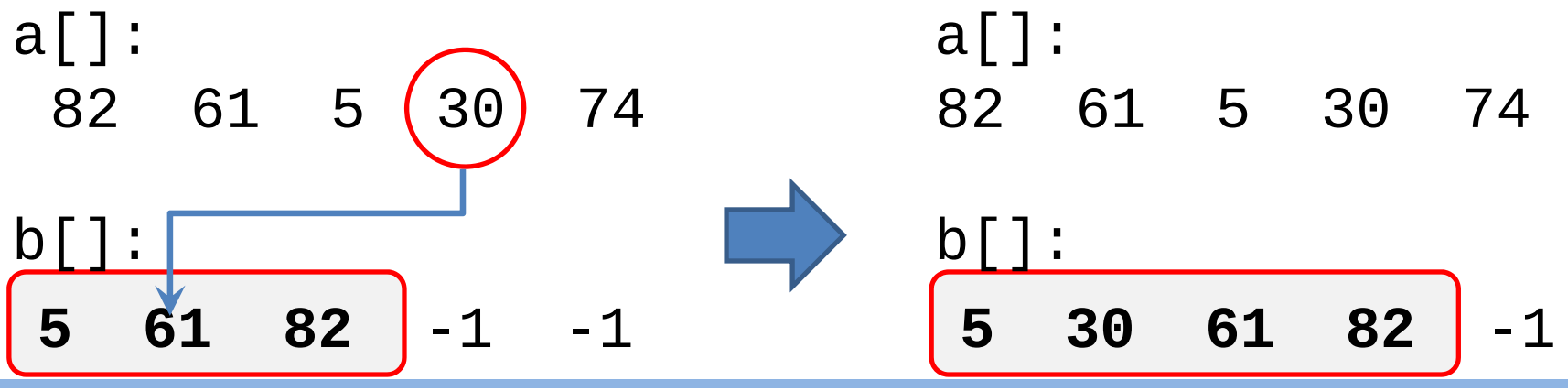
```
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ )
                insert ( a[i], n, b );
}
```



6.3 삽입 정렬 (insertion sort)

4) 수행 예 (i = 3)

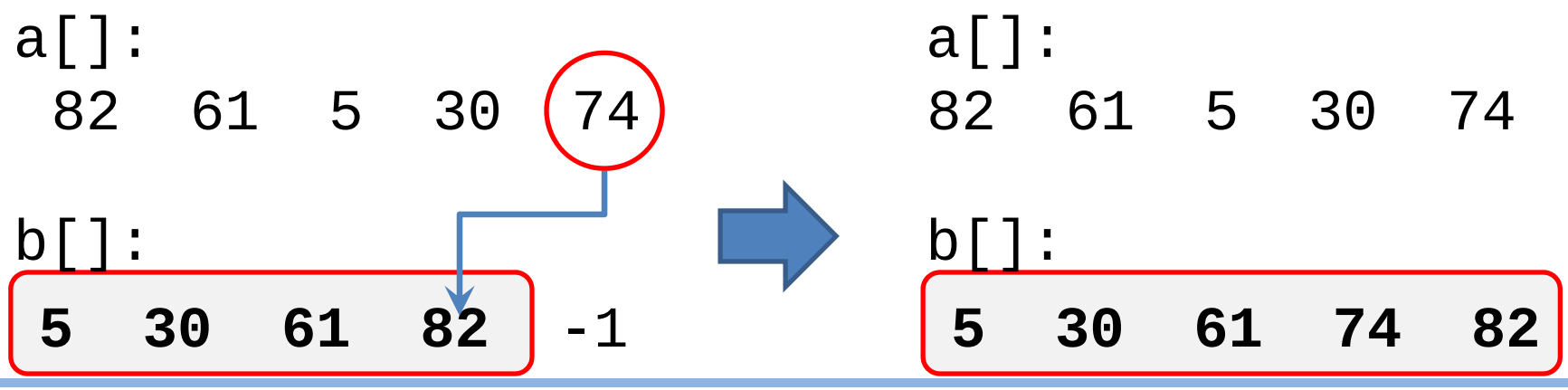
```
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ )
                insert ( a[i], n, b );
}
```



6.3 삽입 정렬 (insertion sort)

4) 수행 예 (i = 4) → 삽입정렬 완료

```
void insertion_sort ( int b[], int n, int a[] )
{       int i;
        for ( i = 1, b[0] = a[0]; i < n; i++ )
                b[i] = -1;
        for ( i = 1; i < n; i++ )
                insert ( a[i], n, b );
}
```



6.3 삽입 정렬 (insertion sort)

5) 성능 분석

– insert ( )의 수행 시간 → O(n)

– insert ( )를 호출하는 회수 → O(n)

– 결과: $O(n^2)$

6.4 선택 정렬 (selection sort)

1) 기본 개념

- 배열의 첫 번째 자리에 최소값을 옮김
  - 배열에서 가장 작은 값과 첫 번째 원소와 교환
- 배열의 두 번째 자리에 두 번째 작은 값을 옮김
  - 배열에서 두 번째 작은 값과 두 번째 원소와 교환
- 이 과정을 반복함

- 버블 정렬과 다른 점은? → select_min ( ) 함수

6.4 선택 정렬 (selection sort)

1) 기본 개념

– 기본 연산

- 배열에서 s번째 원소와 e번째 원소 사이에서 가장 작은 원소(최소값)의 인덱스를 찾는 연산

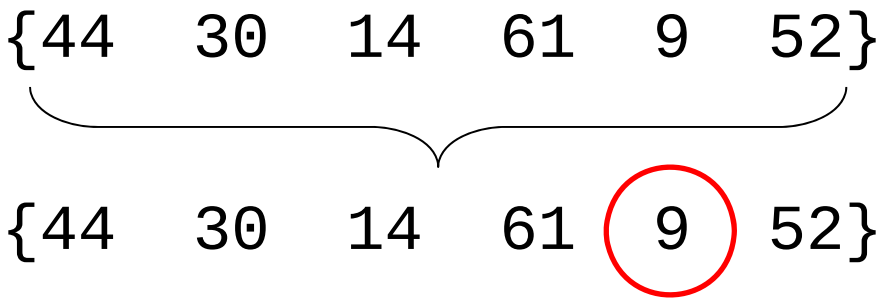
```
int select_min ( int s, int e, int b[] )
{
    int min_idx = s;
    for ( int i = s+1; i <= e; i++ ) {
        if ( b[i] < b[min_idx] )
            min_idx = i;
    }
    return min_idx;
}
```

6.4 선택 정렬 (selection sort)

1) 기본 개념

– 기본 연산

- 배열에서 s번째 원소와 e번째 원소 사이에서 가장 작은 원소(최소값)의 인덱스를 찾는 연산



min_idx = 4

min = b[min_idx] = 9

```
k = select_min ( 0, 5, b );
```

6.4 선택 정렬 (selection sort)

2) 알고리즘 설계

- 정렬하고자 하는 배열의 첫 번째 원소부터 차례로 방문
- i 번째 원소의 처리
  - a[i] ~ a[n-1] 중에서 최소값을 찾을 것
    → min_idx는 최소값의 인덱스를 가리킴
  - 최소값과 a[i]의 값을 교환할 것
    → a[i]에 a[i] ~ a[n-1]까지의 최소값이 오도록 함

6.4 선택 정렬 (selection sort)

3) 알고리즘 구현

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

6.4 선택 정렬 (selection sort)

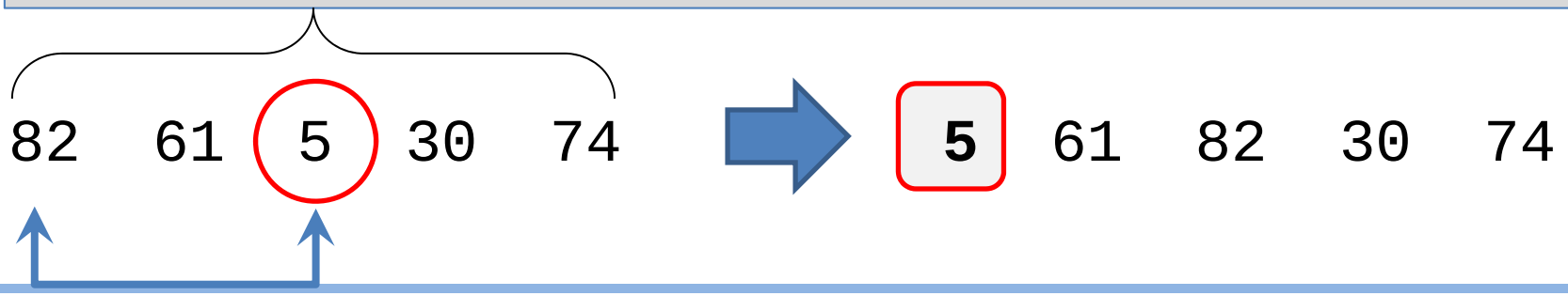
```
int select_min ( int s, int e, int b[] )
{
    int min_idx = s;
    for ( int i = s+1; i <= e; i++ ) {
        if ( b[i] < b[min_idx] )
            min_idx = i;
    }
    return min_idx;
}

void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

6.4 선택 정렬 (selection sort)

4) 수행 예 (i = 0)

– a[0] ~ a[n-1]에서 최소값을 찾아서 a[0]와 교환

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

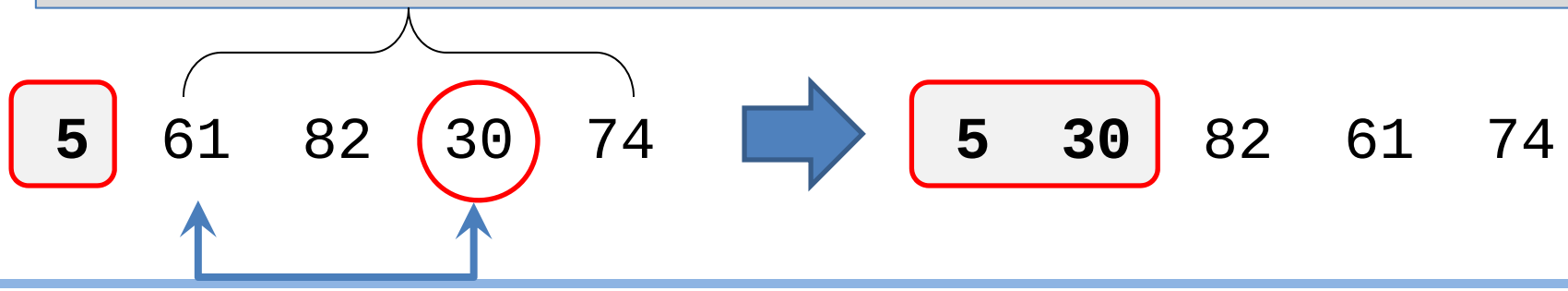


6.4 선택 정렬 (selection sort)

4) 수행 예 (i = 1)

– a[1] ~ a[n-1]에서 최소값을 찾아서 a[1]와 교환

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

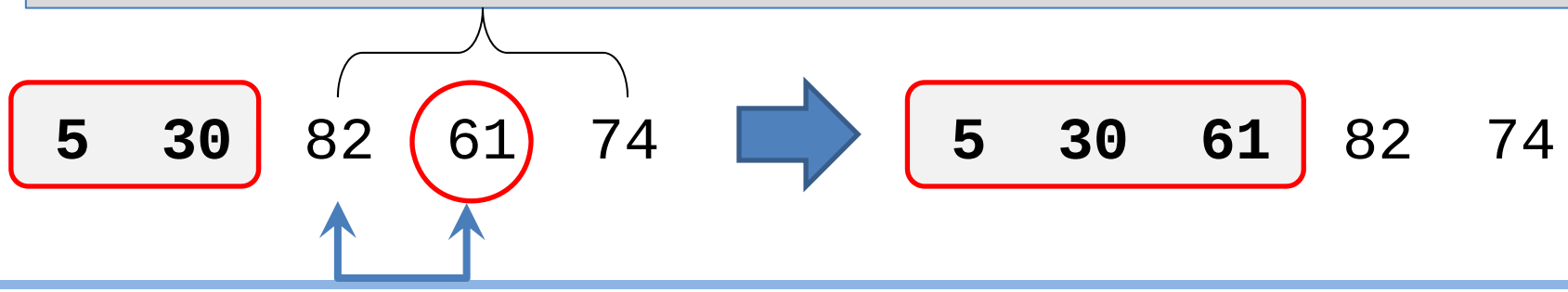


6.4 선택 정렬 (selection sort)

4) 수행 예 (i = 2)

– a[2] ~ a[n-1]에서 최소값을 찾아서 a[2]와 교환

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

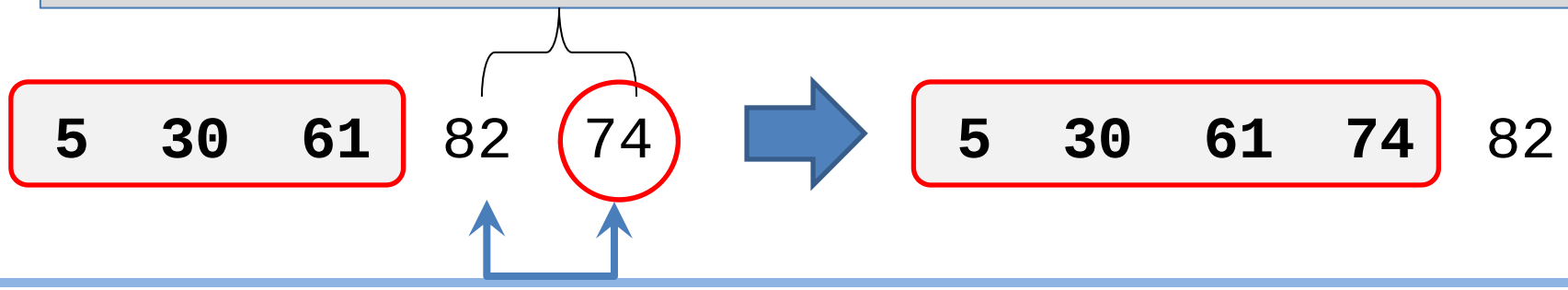


6.4 선택 정렬 (selection sort)

4) 수행 예 (i = 3)

– a[3] ~ a[n-1]에서 최소값을 찾아서 a[3]와 교환

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```



6.4 선택 정렬 (selection sort)

4) 수행 예 (마지막)

```
void selection_sort ( int n, int a[] )
{
    int i;
    int min_idx;
    for ( i = 0; i < n-1; i++ ) {
        min_idx = select_min ( i, n-1, a );
        swap ( a[i], a[min_idx] );
    }
}
```

6.4 선택 정렬 (selection sort)

5) 성능 분석

- select_min ( ) → n개에 대해서 수행 → O(n)
- select_min ( ) 함수를 O(n)번 수행
- 결과: $O(n^2)$

6.5 기타 정렬들

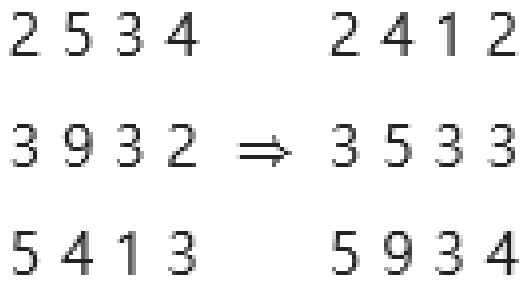
- 내부 정렬 VS 외부 정렬
  - 내부 정렬 (internal sorting)
    - 정렬할 자료를 메인 메모리에 올려서 정렬하는 방식
  - 외부 정렬 (external sorting)
    - 대용량의 보조 기억 장치를 이용해서 정렬하는 방식
- 쉘 (shell) 정렬
- 기수 (radix) 정렬

6.5 기타 정렬들

- 쉘 (shell) 정렬
  - 일반화된 삽입 정렬
  - 배열을 여러 개의 부분 배열로 분할하여 각각 삽입 정렬을 수행
  - 정렬된 부분 배열을 통합하면서 삽입 정렬을 반복해서 수행
  - 예)
    - 2, 5, 3, 4, 3, 9, 3, 2, 5, 4, 1, 3

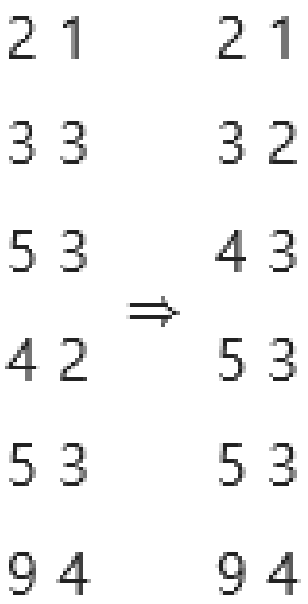
6.5 기타 정렬들

- 쉘 (shell) 정렬
  - 1단계: 크기 3의 부분 배열 4개로 분할하여 정렬



6.5 기타 정렬들

- 쉘 (shell) 정렬
  - 2단계: 크기 6의 부분 배열 2개로 분할하여 정렬



6.5 기타 정렬들

- 쉘 (shell) 정렬
  - 3단계: 하나의 행렬로 통합하여 정렬

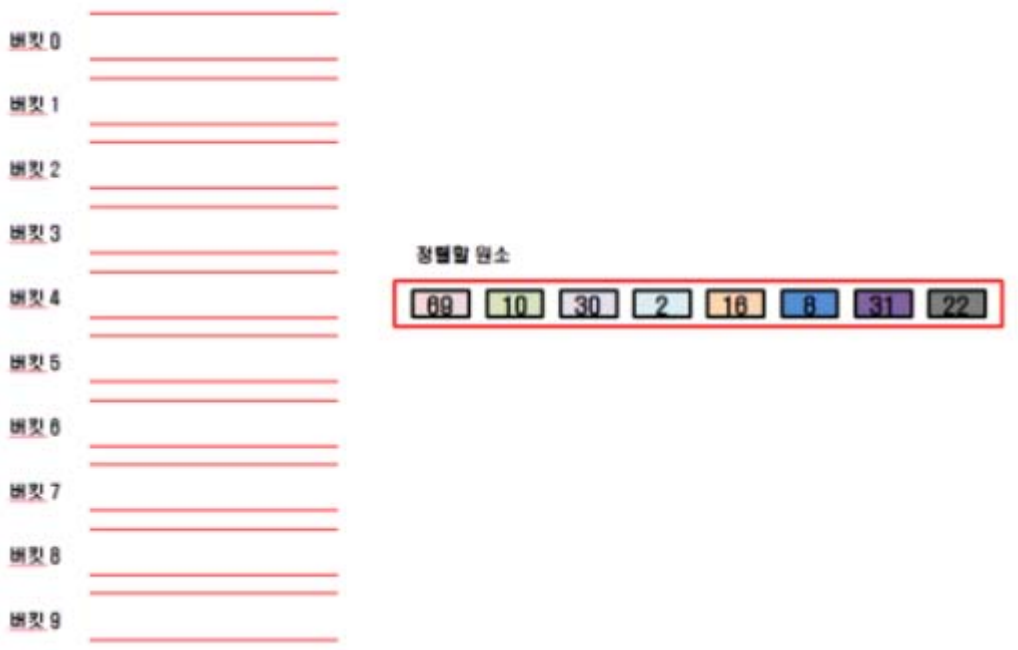
2 3 4 5 5 9 1 2 3 3 3 4 ⇒ 1 2 2 3 3 3 3 4 4 5 5 9

6.5 기타 정렬들

- 기수 (radix) 정렬
  - 버킷 (bucket)을 이용하는 분배 방식의 정렬
  - 정렬할 원소를 버킷에 분배하는 과정을 반복해서 정렬을 수행함
  - 예)
    - 69, 10, 30, 2, 16, 8, 31, 22

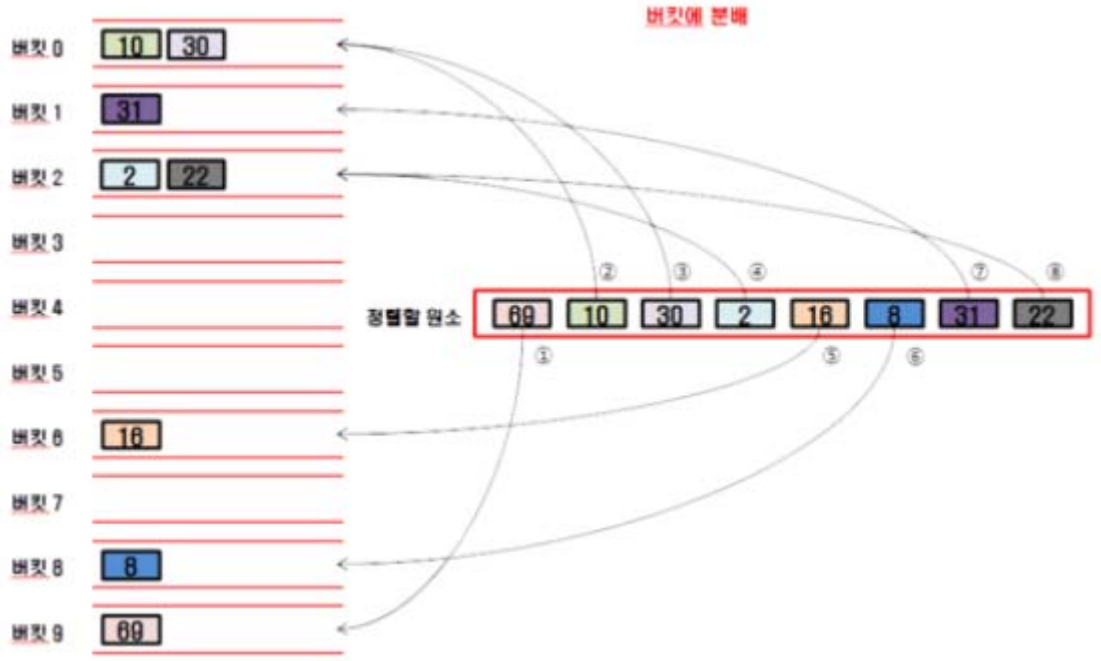
6.5 기타 정렬들

- 기수 (radix) 정렬
  - 초기화: 10개의 버킷을 준비



6.5 기타 정렬들

- 기수 (radix) 정렬
  - 1단계: 1의 자리수에 따라서 버킷에 저장

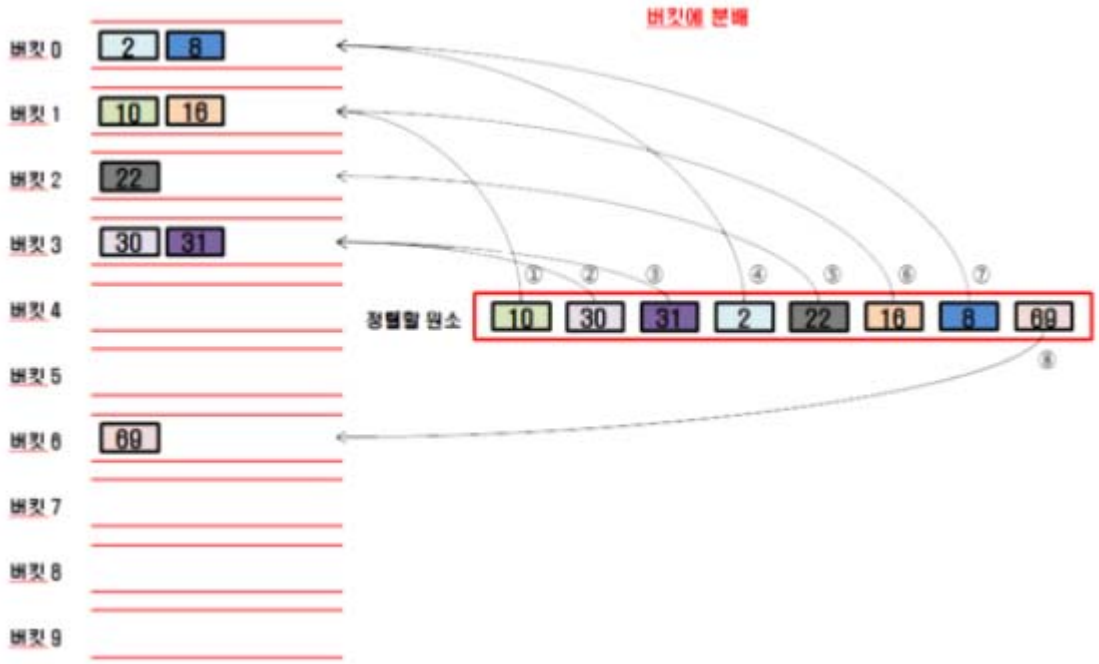


6.5 기타 정렬들

- 기수 (radix) 정렬
  - 2단계: 버킷에 저장된 수를 정렬

6.5 기타 정렬들

- 기수 (radix) 정렬
  - 3단계: 10의 자리수에 따라서 버킷에 저장



6.5 기타 정렬들

- 기수 (radix) 정렬
  - 4단계: 버킷에 저장된 수를 정렬

6.6 결론

- 단순한 정렬 알고리즘: O (n^2) 성능
  - 버블 정렬, 삽입 정렬, 선택 정렬, …
- 분할정복 정렬 알고리즘: O(n log n) 성능
  - 합병 정렬, 쾌속 정렬, …
- 성능의 차이가 크기 때문에 분할정복 정렬 알고리즘을 사용할 것

내용

6.1 소개

6.2 버블 정렬

6.3 삽입 정렬

6.4 선택 정렬

6.5 기타 정렬들

6.6 결론

Contents