정렬 (Sorting)

- 간단한 정렬 알고리즘
  선택정렬, 버블정렬, 삽입정렬
- 분할과 정복 (Divide and Conquer) 방법
- 재귀(Recursion, 순환)
- 분할과 정복에 의한 정렬 알고리즘
  병합정렬과 퀵정렬
- 힙정렬
- Radix 정렬

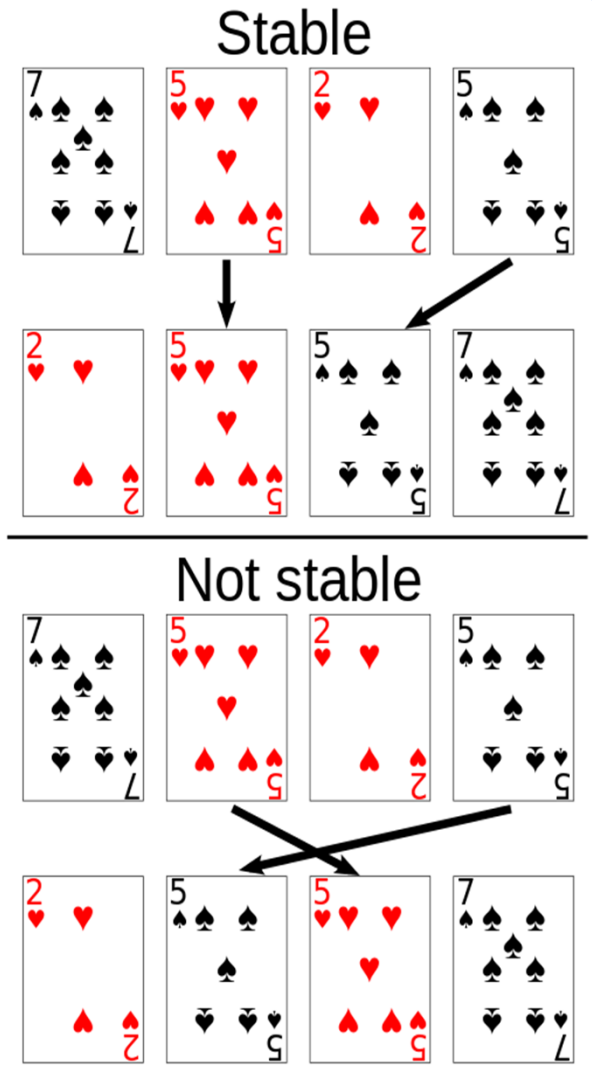
2. 정렬 (Sorting)

- 정렬: 자료들을 크기 순서대로 (오름차순으로) 나열하는 것
  - 입력 예: 7 9 6 2 10 4 5 9 8
    정렬: 2 4 5 6 7 8 9 9 10
- 정렬은 여러 응용분야 사용되는 매우 기본적인 문제
- 정렬알고리즘
  - 간단한 정렬 알고리즘: 선택정렬/버블정렬/삽입정렬
  - 고급 정렬 알고리즘
    (1) 분할과 정복 알고리즘 –
    병합정렬(merge sort)/퀵정렬(quick sort)
    (2) 힙정렬(heapsort) – 우선순위큐 (priority queue)와 힙
    힙정렬
    (3) 기수정렬(radix sort)
    …

정렬알고리즘 분류

- Stable 정렬 알고리즘
  - 정렬할 자료들(입력 자료) 중 동일한 두 자료의 상대적인 위치가 정렬 후에도 유지되는 정렬 알고리즘

- In-place 정렬 알고리즘
  - 입력(정렬할 자료)을 저장하는 메모리 공간이외에 추가로 사용하는 메모리 공간이 O(1)인 정렬 알고리즘



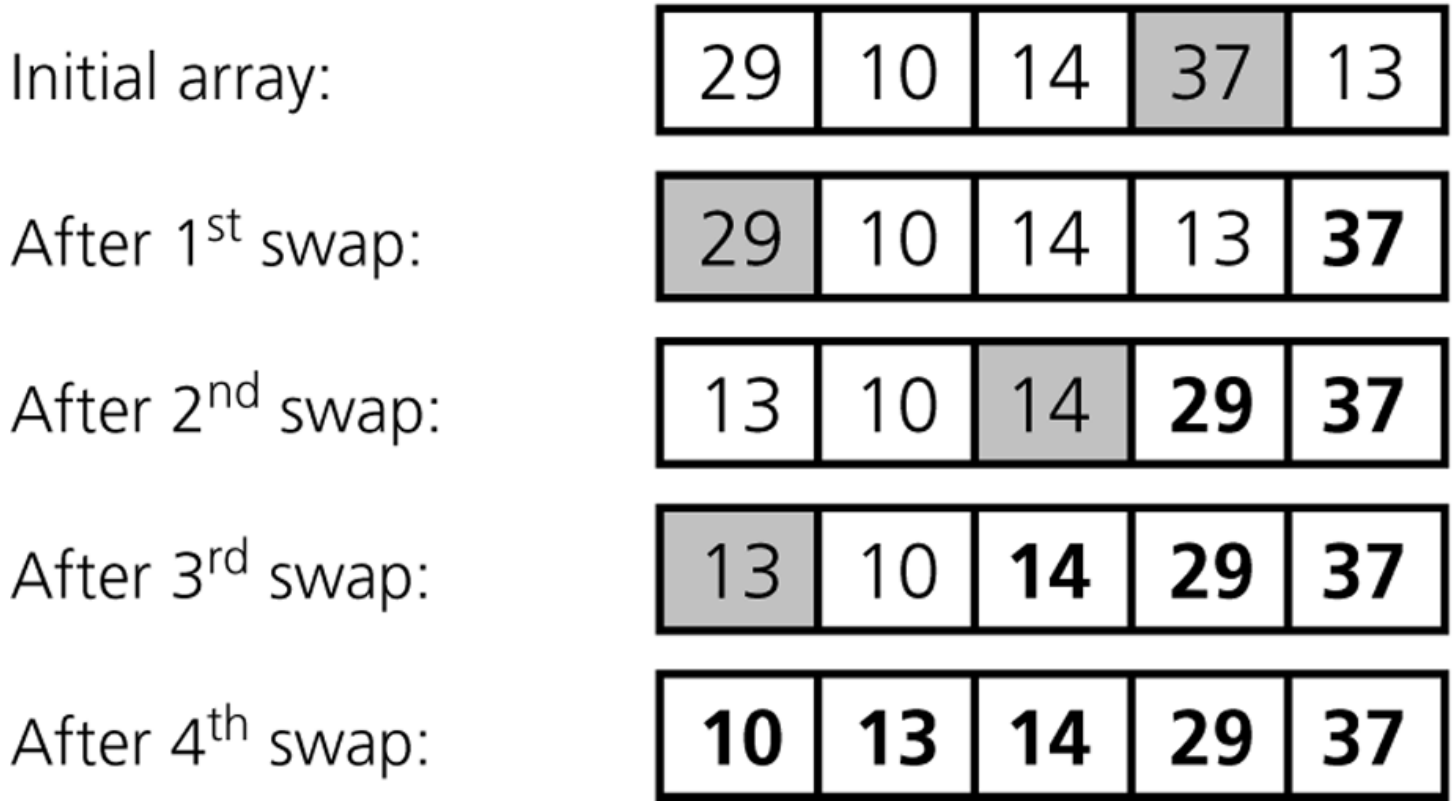
2.1 선택 정렬(Selection Sort)-간단한 알고리즘

◆ 기본적인 아이디어

- 각 루프마다
  - 최대(최소) 원소를 찾는다
  - 최대(최소) 원소와 마지막(가장 앞에 있는) 원소를 교환한다
  - 마지막(가장 앞에 있는) 원소를 제외한다
- 하나의 원소만 남을 때까지 위의 루프를 반복

2.1 선택 정렬 - 예

Shaded elements are selected;
boldface elements are in order.



2.1 선택 정렬 – 프로그램 및 시간 분석

◆ 프로그램

```
Algorithm selectionSort(a, n)
   for i = n-1 downto 1      // ------------ ①
//   a[0]부터 a[i]까지 중 가장 큰 원소의 위치
//   index를 찾는다.
        index = 0
        for j = 1 to  i          // ------------- ②
            if (a[index] < a[j]) then
                index = j
// a[index]와 a[i]를 교환한다.      // ------------ ③
       int temp = a[index]
       a[index] = a[i]
       a[i] = temp
```

◆시간분석

수행시간 분석

①의 for 루프는 n-1번 반복

②에서 가장 큰 수를 찾기 위한 원소 비교횟수: i

③의 교환은 상수 시간 작업

전체 원소 비교회수:
n-1 + n-2 + …+ 2 + 1 = n(n-1)/2

최악의 경우 시간복잡도(원소 비교횟수= n-1 + n-2 + …+ 2 + 1 = n(n-1)/2

$O(n^2)$

= 가장 좋은 경우(best case) 시간 복잡도
= 평균적인 경우 시간복잡도

2.2 버블 정렬 (bubble sort)-간단한 알고리즘

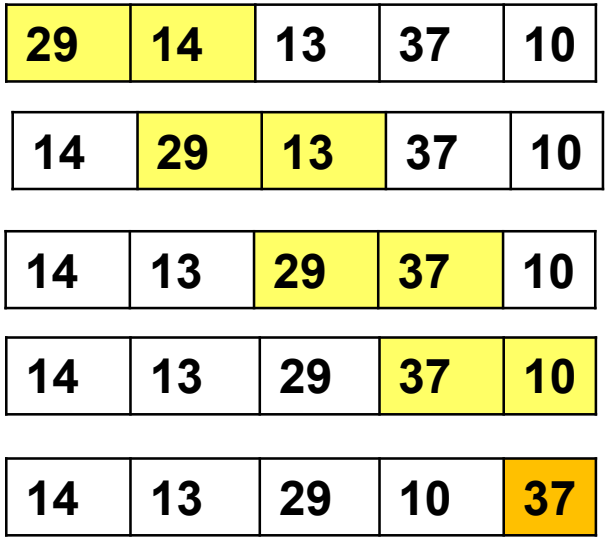
- 기본적인 아이디어

  - 매 단계마다 처음부터 마지막까지 차례대로 인접한 두 원소를 비교하여 뒤에 있는 원소가 앞의 원소보다 작은 경우 두 원소를 교환
  - 각 단계 수행 후 최대값이 마지막으로 이동하므로, 마지막 원소는 정렬대상에서 제외
  - (만약 끝까지 가면서 교환이 일어나지 않았으면 정렬이 되어 있는 것임)

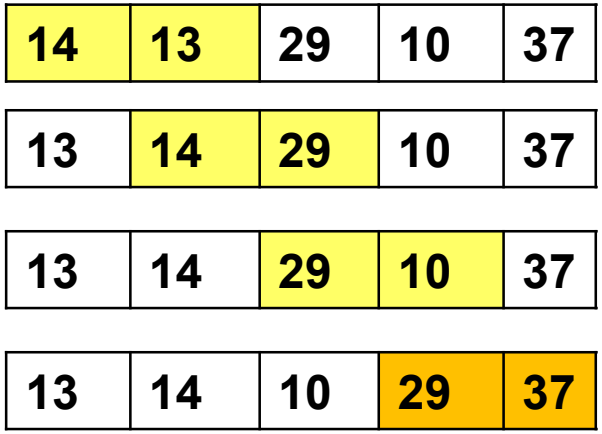
2.2 버블 정렬- 예

- 예 1: 초기 배열

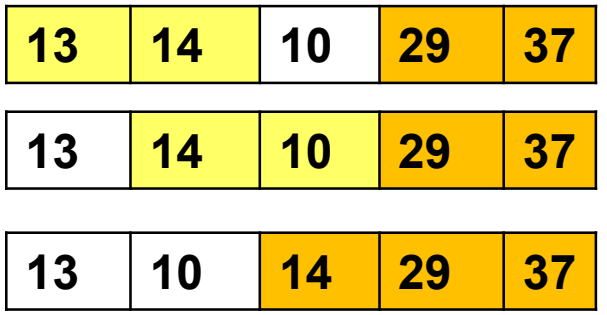
– 단계 1



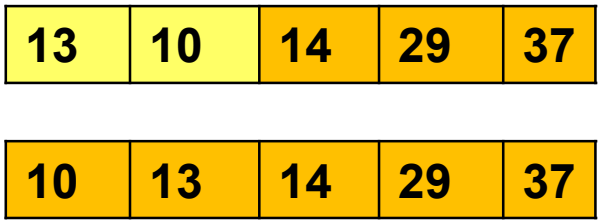
- 단계 2



– 단계 3



- 단계 4



2.2 버블 정렬 – 시간 분석

◆ 알고리즘

```
Algorithm bubbleSort(a, n)
   for i = n – 1 downto 1 // i는 정렬하고자 하는 원소들의 마지막 위치
      for j = 0 to i-1
         if(a[j] > a[j+1])  then
            a[j]와 a[j+1]을 교환
```

✓ 수행시간:
원소 비교횟수 = $(n-1)+(n-2)+\cdots+2+1 = O(n^2)$

◆ 시간분석

✓ 최악의 경우 시간복잡도 = 가장 좋은 경우 시간복잡도 = 평균적인 경우 시간복잡도: $O(n^2)$

2.2 개선된 버블 정렬 알고리즘

- 예2

70	60	10	20	30	40	50

  - 단계 1

70	60	10	20	30	40	50

60	70	10	20	30	40	50

60	10	70	20	30	40	50

60	10	20	70	30	40	50

60	10	20	30	70	40	50

60	10	20	30	40	70	50

60	10	20	30	40	50	70

2.2 개선된 버블정렬 알고리즘 – 예2 (계속)

- 예2

70	60	10	20	30	40	50

  – 단계 2

60	10	20	30	40	50	70

10	60	20	30	40	50	70

10	20	60	30	40	50	70

10	20	30	60	40	50	70

10	20	30	40	60	50	50

10	20	30	40	50	60	50

2.2 개선된 버블 정렬 알고리즘 – 예2 (계속)

- 예2

70	60	10	20	30	40	50

  - 단계 3

10	20	30	40	50	60	70
10	20	30	40	50	60	70
10	20	30	40	50	60	70
10	20	30	40	50	60	70
10	20	30	40	50	60	70

  - 단계 3에서 교환이 일어나지 않았으므로, 이는 정렬되어 있음을 의미한다. 정렬과정이 끝난다.

2.2 개선된 버블 정렬 알고리즘

◆ 알고리즘

```
Algorithm bubbleSort(a, n)
    for i = n – 1 downto 1
        sorted = true
        for j = 0 to i-1
            if(a[j] > a[j+1])  then
                a[j]와 a[j+1]을 교환
                sorted = false // 정렬이 되어 있지 않은 상태로 바꿈
        if  (sorted)
            return
```

✓ 수행시간:
원소 비교횟수 ≤ $(n-1)+(n-2)+\cdots+2+1 = O(n^2)$

◆ 시간분석

✓ 최악의 경우 시간 W(n) : $O(n^2)$ ⇨ 역순으로 정렬되어 있는 입력
✓ 가장 좋은 경우 시간 B(n) : O(n) ⇨ 정렬되어 있는 입력

2.3 삽입정렬 (Insertion Sort)-간단한 알고리즘

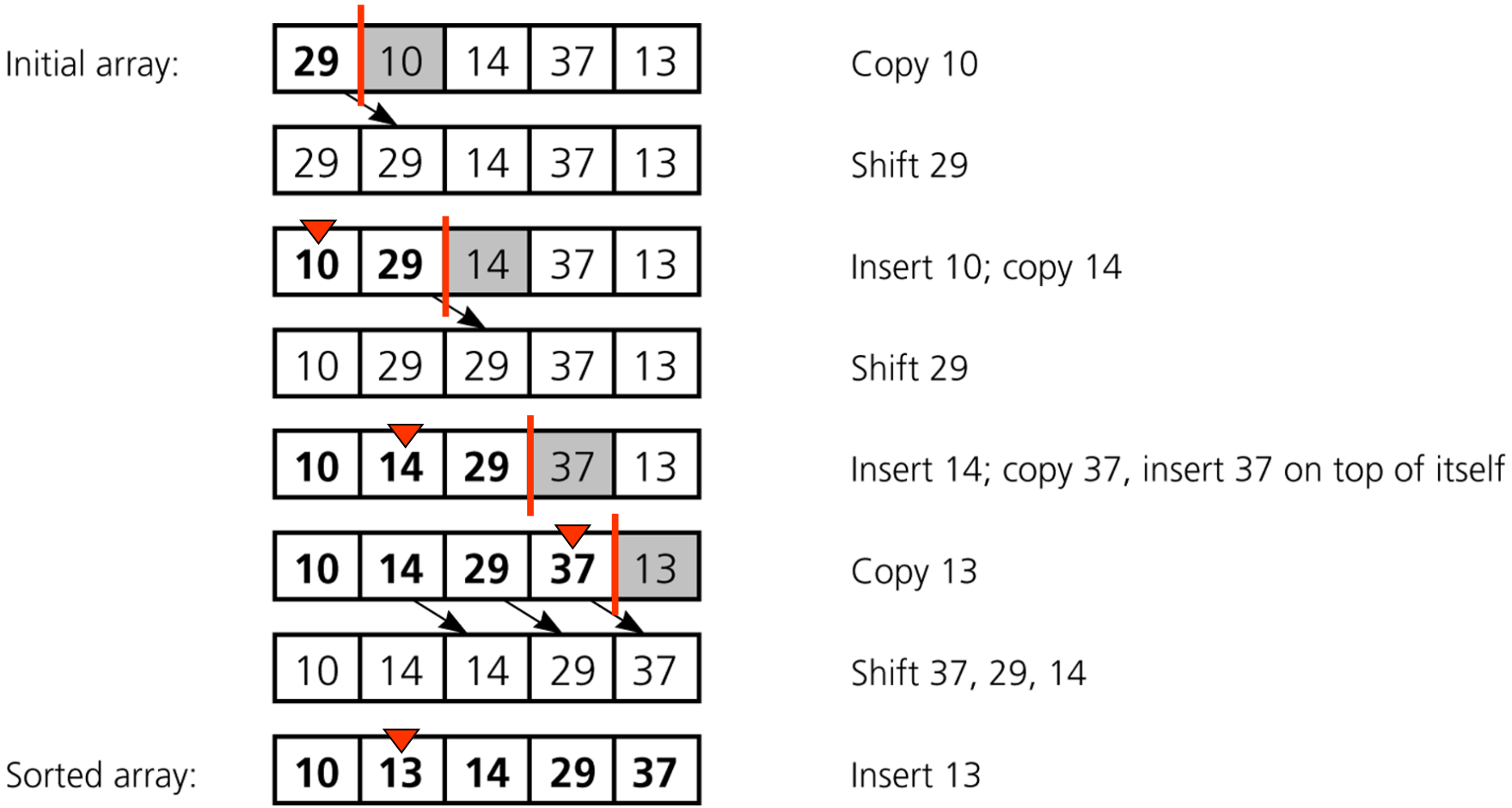
- 기본적인 idea
  - a[0]부터 a[i-1]까지 정렬되어 있을 때 a[i]까지 포함하여 정렬하는 알고리즘을 이용

- 알고리즘

```
Algorithm insertionSort(a, n)
  for i = 1 to n-1  // a[1]부터 차례대로 삽입하여 정렬함
      temp = a[i]
      for  j = i-1 downto 0  // 정렬되어 있는 a[0..i-1]에 a[i]를 삽입함
          if (a[j]> temp) then
             a[j+1] = a[j]
          else
             break
    // for loop을 끝까지 수행한 경우에는 j = -1이라 가정
      a[j+1] = temp
```

가장 좋은 경우 입력이 정렬되어 있는 경우

2.3 삽입 정렬 - 예


Initial array:
29 10 14 37 13
Copy 10
29 29 14 37 13
Shift 29
10 29 14 37 13
Insert 10; copy 14
10 29 29 37 13
Shift 29
10 14 29 37 13
Insert 14; copy 37, insert 37 on top of itself
10 14 29 37 13
Copy 13
10 14 14 29 37
Shift 37, 29, 14
Sorted array:
10 13 14 29 37
Insert 13

2.3 삽입 정렬 – 시간 분석

◆ 시간 분석

- ✓ 최악의 경우 시간복잡도
  W(n) : $O(n^2)$ ⇒ 역순으로 정렬되어 있는 입력
- ✓ 가장 좋은 경우 시간복잡도
  B(n) : O(n) ⇒ 정렬되어 있는 입력
- ✓ 평균적인 경우 시간복잡도
  A(n) : $O(n^2)$

간단한 정렬알고리즘 Summary

장점: 이해하기 쉽다.
코딩하기 쉽다.
단점: 입력이 클 경우 시간이 많이 걸린다.

	Best Case	Worst Case	Average Case	Stable?	In Place ?
선택정렬	$O(n^2)$	$O(n^2)$	$\Theta(n^2)$	No	Yes
버블정렬	$O(n^2)$ => $O(n)$	$O(n^2)$	$\Theta(n^2)$	Yes	Yes
삽입정렬	$O(n)$	$O(n^2)$	$\Theta(n^2)$	Yes	Yes

•Stability Property: Relative positions of two equal elements remain the same in the output as in the input.