



Lecture 13


정렬과 탐색

2018년도 2학기

컴퓨터프로그래밍2

김 영 국
충남대학교 컴퓨터공학과

이번 주에 학습할 내용

- 
- 정렬(Sorting)
 - 탐색(Searching)

정렬과 탐색의
기초적인
내용에 대하여
학습합니다.



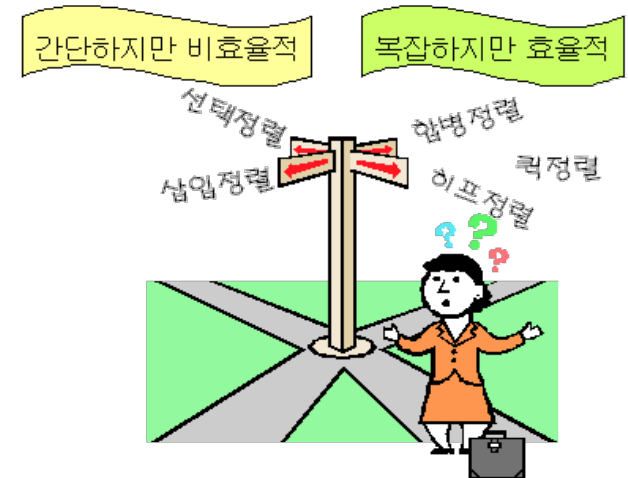


정렬(sort)이란?

- 정렬은 물건을 크기순으로 오름차순이나 내림차순으로 나열하는 것
- 정렬은 컴퓨터 공학분야에서 가장 기본적이고 중요한 알고리즘 중의 하나
- 정렬은 자료 탐색에 있어서 필수적이다.
 - (예) 만약 사전에서 단어들이 정렬이 안되어 있다면?

정렬 알고리즘 개요

- 많은 정렬 알고리즘이 존재
 - 모든 경우에 최적인 알고리즘은 없음
 - 응용에 맞추어 선택
- 정렬 알고리즘의 평가
 - 비교 횟수
 - 이동 횟수



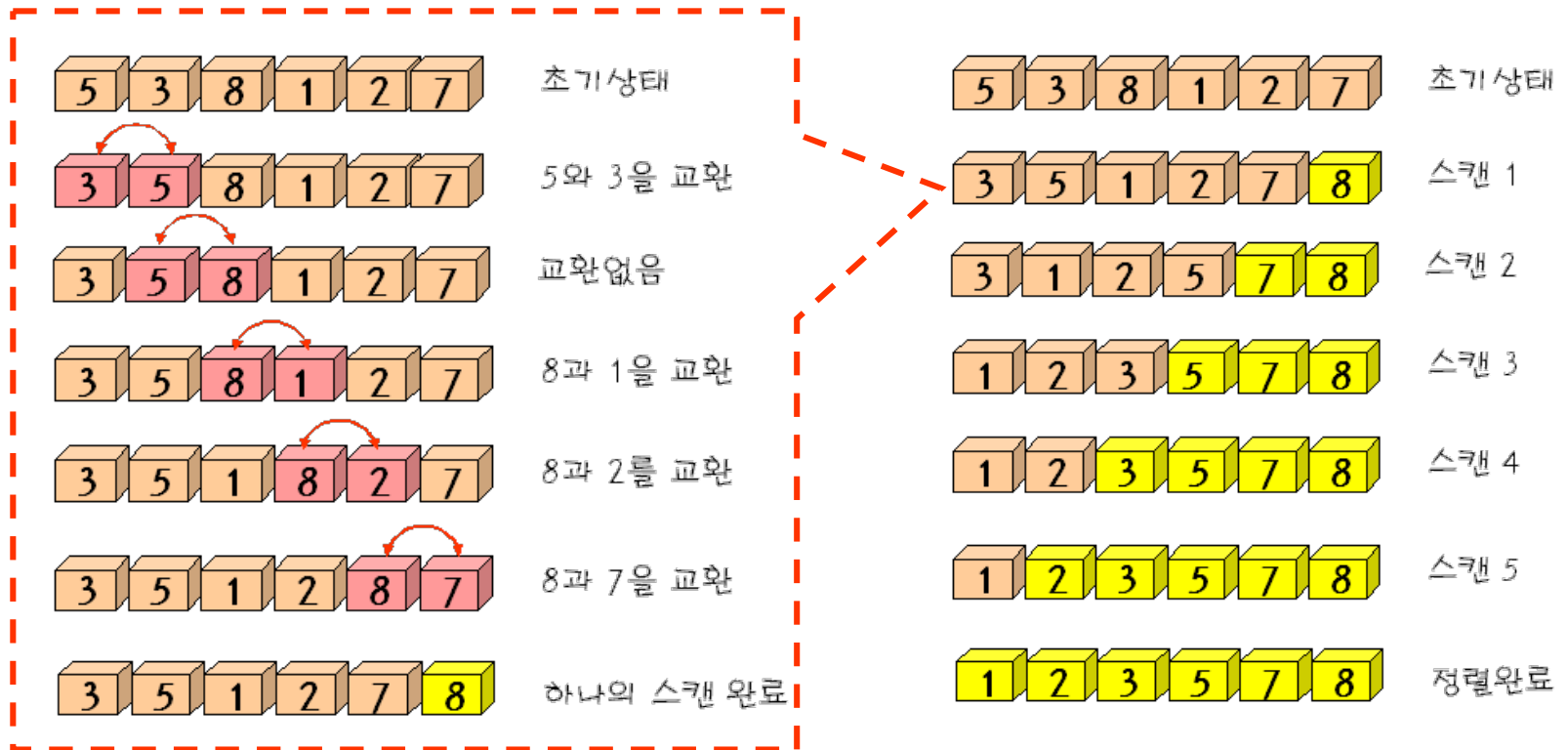


버블 정렬 (Bubble Sort)

- 배열에서 인접한 원소를 비교한 다음 제 위치에 있지 않을 경우 그것들을 서로 교환
- 순서에 따라 처음부터 마지막까지 원소 쌍 사이의 비교를 수행
- 각각의 비교가 끝나면 큰 원소는 한 칸씩 앞으로 이동하기 때문에 가장 큰 원소는 "거품이 올라오듯이(bubble up)" 배열의 끝으로 이동
- 하나의 완전한 과정이 끝나면 배열이 오름차순을 유지하는 경우 가장 큰 원소는 배열의 가장 끝으로 이동함.
- 이러한 과정을 $(n-1)$ 개의 정렬되지 않은 원소의 부분 배열에 대해 반복해서 수행하면 두 번째로 큰 원소가 제 자리로 이동함.

버블 정렬의 수행 과정

- 인접한 원소가 순서대로 되어 있지 않으면 교환
- 전체가 정렬될 때까지 비교/교환 계속





버블 정렬 알고리즘 및 구현

■ 버블 정렬 알고리즘

1. $i = n-1$ 에서 1까지 내려가면서 단계 2-3을 반복
2. $j = 0$ 에서 $i-1$ 까지 올라가면서 단계 3을 반복
3. 만일 $a[j] > a[j+1]$, 두 원소를 교환

■ Bubble Sort 구현

```
1 void sort(int[] a) {  
2     for (int i = a.length-1; i > 0; i--)  
3         for (int j = 0; j < i; j++)  
4             if (a[j] > a[j+1]) swap(a, j, j+1);  
5 }
```



버블 정렬의 테스트

```
1 public class TestBubbleSort {
2     public TestBubbleSort() {
3         int[] a = {88, 55, 22, 77, 11, 44, 33, 99, 66};
4         print(a);
5         sort(a);
6         print(a);
7     }
9     void sort(int[] a) {
10        for (int i = a.length-1; i > 0; i--)
11            for (int j = 0; j < i; j++)
12                if (a[j] > a[j+1]) swap(a, j, j+1);
13    }
15    public static void main(String[] args) {
16        new TestBubbleSort();
17    }
18 }
```




버블 정렬 성능 분석

■ 비교 횟수

- 버블 정렬의 비교 횟수는 최상, 평균, 최악의 어떠한 경우에도 항상 일정

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

■ 이동 횟수

- 최악: 역순 정렬 이동 = 3*비교
- 최상: 이미 정렬 이동 = 0
- 평균: **$O(n^2)$**



선택 정렬 (Selection Sort)

- n 원소 시퀀스에 대해 $(n-1)$ 패스를 수행하는데, 매번 나머지 정렬되지 않은 원소들 중에서 가장 큰 원소를 올바른 위치로 이동
- 그러나 이것은 각 패스에서 교환을 한 번만 수행하기 때문에 버블 정렬에 비해 약간 더 효율적임.
- 이것을 “**선택(selection)**” 정렬이라고 부르는 이유는 각 패스마다 정렬되지 않은 원소들 중에서 가장 큰 원소를 선택하여 그것을 올바른 위치로 이동시키기 때문임.

선택 정렬의 수행 과정

- 배열에서 최소값을 선택하여 첫번째 원소와 교환
- 첫번째 원소를 제외한 나머지 원소들 중에서 최소값을 선택하여 두번째 원소와 교환. 이 과정을 $(n-1)$ 번 반복





선택 정렬 알고리즘 및 구현

- 선택 정렬 알고리즘

1. $i = n-1$ 에서 1까지 내려가면서 단계 2를 반복
2. $a[i]$ 와 $\max\{a[0]..a[i]\}$ 를 교환

- Selection Sort 구현

```
1 void sort(int[] a) {  
2     for (int i = a.length-1; i > 0; i--) {  
3         int m = 0;  
4         for (int j = 1; j <= i; j++)  
5             if (a[j] > a[m]) m = j;  
6         swap(a, i, m);  
7     }  
8 }
```



선택 정렬 성능 분석

- 성능 분석
 - 최소값을 선택하는데 걸리는 시간: $O(n)$
 - 정렬할 숫자의 개수가 n 이면,
 - 전체 시간 복잡도: $O(n^2)$



삽입 정렬 (Insertion Sort)

- n 원소 시퀀스에 대해 $(n-1)$ 패스를 수행
- 각각의 패스에서 이것은 왼쪽에 있는 부분 배열을 정렬된 상태로 두면서 다음 원소를 이 부분 배열에 삽입
- 마지막 원소가 이러한 방법으로 "삽입되면(inserted)" 전체 배열이 정렬된 것임.

삽입 정렬의 수행 과정

- 삽입 정렬은 정렬되어 있는 부분에 새로운 원소를 적절한 위치에 삽입하는 과정을 반복





삽입 정렬 알고리즘과 구현 (1)

■ 삽입 정렬 알고리즘

1. $i = 1$ 에서 $n-1$ 까지 올라가면서 단계 2-5를 반복
2. 원소 $a[i]$ 를 임시 기억장소에 저장
3. $a[j] \geq a[i]$ 에 대해 $j \leq i$ 인 최소 인덱스에 위치함
4. 서브시퀀스 $\{a[j] \dots a[i-1]\}$ 을 $\{a[j+1] \dots a[i]\}$ 가 되도록 위로 하나씩 이동
5. $a[i]$ 의 저장된 값을 $a[j]$ 로 복사



삽입 정렬 알고리즘과 구현 (2)

■ Insertion Sort 구현

```
1 void sort(int[] a) {  
2     for (int i = 1; i < a.length; i++) {  
3         int ai = a[i], j = i;  
4         for (j = i; j > 0 && a[j-1] > ai; j--)  
5             a[j] = a[j-1];  
6         a[j] = ai;  
7     }  
8 }
```

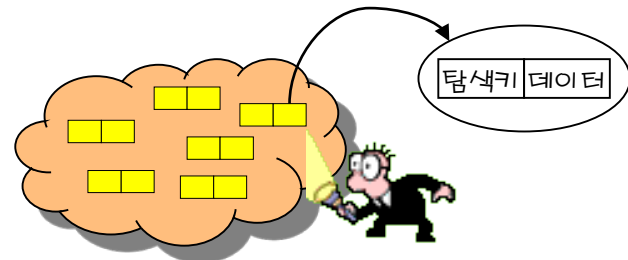


삽입 정렬 성능 분석

- 최상의 경우: 이미 정렬되어 있는 경우
 - 비교: $(n-1)$ 번
 - 이동: $2(n-1)$ 번
- 최악의 경우: 역순으로 정렬
 - 모든 단계에서 앞에 놓인 자료 전부 이동
 - 비교:
$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$
 - 이동:
$$\frac{n(n-1)}{2} + 2(n-1) = O(n^2)$$

탐색(Search)이란?

- **탐색(search):** 기본적으로 여러 개의 자료 중에서 원하는 자료를 찾는 작업
 - 컴퓨터가 가장 많이 하는 작업 중의 하나
 - 탐색을 효율적으로 수행하는 것은 매우 중요
- **탐색키(search key):** 항목과 항목을 구별해주는 키(key)
- 탐색을 위하여 사용되는 자료 구조
 - 배열, 연결 리스트, 트리, 그래프 등





순차 탐색 (Sequential Search)

- 순차 탐색 (선형 탐색 또는 직렬 탐색)
 - 주어진 목표 값을 찾아 리스트 앞에서부터 순차적으로 탐색
 - 목표가 발견된 첫 번째 위치를 리턴
 - 목표가 발견되지 않으면 음수를 리턴

순차 탐색의 수행 과정

• 8을 찾는 경우

(1) $9 \neq 8$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(2) $5 \neq 8$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(3) $8 = 8$ 이므로 탐색성공

9	5	8	3	7
---	---	---	---	---

(a) 탐색 성공의 경우

• 2을 찾는 경우

(1) $9 \neq 2$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(2) $5 \neq 2$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(3) $8 \neq 2$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(4) $3 \neq 2$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(5) $7 \neq 2$ 이므로 탐색계속

9	5	8	3	7
---	---	---	---	---

(6) 더 이상 항목이 없으므로
탐색실패

(b) 탐색 실패의 경우



순차 탐색 알고리즘 및 성능

■ 순차 탐색 알고리즘

입력: sequence와 목표 값 x

출력: 인덱스 값 i

후조건: $a[i] = x$ 또는 모든 $a[i] \neq x$ 일 때 $i = -n$

1. 0에서 $n-1$ 의 각 i 에 대해, 단계 2를 수행
2. $a[i] = x$ 이면, i 를 리턴
3. $-n$ 을 리턴

■ 시간 복잡도: $\Theta(n)$



순차 탐색 구현 및 테스트 (1)

■ Sequential Search 구현 및 테스트

```
1 public class TestSequentialSearch {  
3     public static void main(String[] args) {  
4         int[] a = {66, 44, 99, 33, 55, 22, 88, 77};  
5         System.out.println("search(a," + 55 + "): " + search(a,55));  
6         System.out.println("search(a," + 50 + "): " + search(a,50));  
7     }  
9     public static int search(int[] ar, int target) {  
10        for (int i = 0; i < ar.length; i++)  
11            if (ar[i] == target) return i;  
12        return -ar.length;  
13    }  
14 }
```



순차 탐색 구현 및 테스트 (2)

- 출력 결과

search(a,55): 4

search(a,50): -8

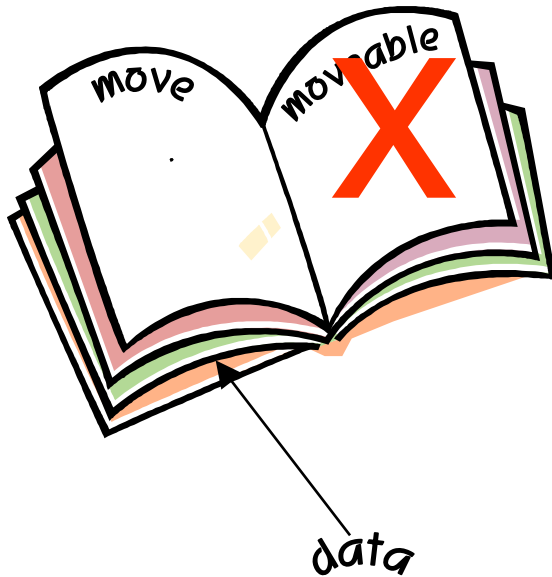


이진 탐색 (Binary Search)

- 정렬된 시퀀스의 탐색에는 **이진 탐색**(binary search)이 적합
- 시퀀스의 중앙에 있는 값을 조사하여 찾고자 하는 항목이 왼쪽 또는 오른쪽 부분 배열에 있는지를 알아내어 탐색의 범위를 반으로 줄임
- (예) 10억 명 중에서 이진 탐색을 이용하여 특정한 이름을 탐색
 - 이진 탐색은 단지 30번의 비교
 - 순차 탐색에서는 평균 5억 번의 비교

이진 탐색의 수행 과정

영어사전



• 5을 탐색하는 경우

7과 비교

1	3	5	6	7	9	11	20	30
---	---	---	---	---	---	----	----	----

5 < 7이므로 앞부분만을 다시 탐색

1	3	5	6	7	9	11	20	30
---	---	---	---	---	---	----	----	----

5를 3과 비교

1	3	5	6	7	9	11	20	30
---	---	---	---	---	---	----	----	----

5 > 3이므로 뒷부분만을 다시 탐색

1	3	5	6	7	9	11	20	30
---	---	---	---	---	---	----	----	----

5 == 5이므로 탐색성공

1	3	5	6	7	9	11	20	30
---	---	---	---	---	---	----	----	----



이진 탐색 알고리즘과 성능

■ 이진 탐색 알고리즘

입력: sequence와 목표 값 x

출력: 인덱스 값 i

선조건: sequence는 정렬되어 있음

후조건: $a[i] = x$; 또는 모든 $j < p$ 에 대해서 $a[j] < x$ 이고
모든 $j \geq p$ 에 대해서 $a[j] > x$ 일 때 $i = -p-1$

1. $p = 0, q = n-1$ 로 놓음
2. $p \leq q$ 이면 단계 2-5를 반복
3. $i = (p+q)/2$ 로 놓음
4. $a[i] = x$ 이면, i 를 리턴
5. $a[i] < x$ 이면, $p = i+1$ 로 놓음; 그렇지 않으면 $q = i-1$ 로 놓음
6. $-p-1$ 을 리턴

■ 시간 복잡도: $\Theta(\log n)$



이진 탐색의 구현 및 테스트 (1)

Binary Search 구현 및 테스트

```
1 public class TestBinarySearch {
3     public static void main(String[] args) {
4         int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
5         System.out.println("search(a," + 55 + "): " + search(a, 55));
6         System.out.println("search(a," + 50 + "): " + search(a, 50));
7     }
9     static int search(int[] ar, int x) {
10        int p = 0, q = ar.length-1;
11        while (p <= q) { // search the segment a[p..q]
12            int i = (p+q)/2; // index of element in the middle
13            if (ar[i] == x) return i;
14            if (ar[i] < x) p = i+1; // search upper half
15            else q = i-1; // search lower half
16        }
17        return -p-1; // not found
18    }
19 }
```



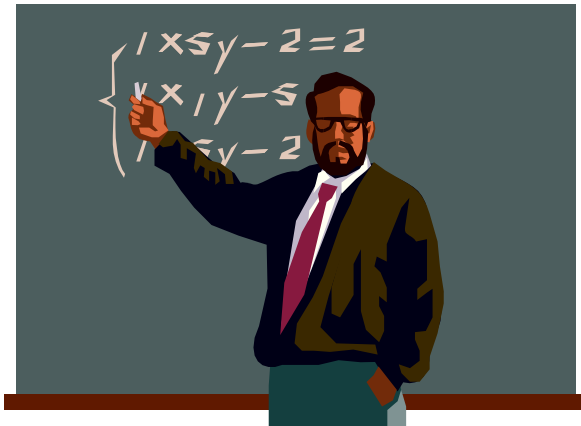
이진 탐색의 구현 및 테스트 (2)

- 출력 결과

search(a,55): 3

search(a,50): -4

Q & A





부록: int 배열의 print와 swap 메소드

```
public static void print(int[] a) {  
    System.out.print("{ " + a[0]);  
    for (int i = 1; i < a.length; i++)  
        System.out.print(", " + a[i]);  
    System.out.println("}");  
}
```

```
public static void swap(int[] a, int i, int j) {  
    int temp = a[i];  
    a[i] = a[j];  
    a[j] = temp;  
}
```