자바 프로그래밍2

- 4장-5장 보조 자료

삽입 정렬과 이진 탐색

sort

binary search

삽입 정렬 (Insertion Sort)

- 부분적으로 정렬된 다음의 정수 배열에서

  [**2 3 5 9** 4]

- 우선 마지막 요소가 정렬된 그룹 안에서 올바른 위치에 삽입되어야 한다. 이 숫자(**4**)를 적절한 위치로 배치하기 위하여,
  - 마지막 위치에 있는 값(4)을 삭제되지 않게 "안전"한 값이 되도록 임시 변수 temp에 복사한다. 즉, 값을 안전하게 보관하기 위하여 임시로 저장해 둔다.
  - 정렬된 부-배열(sub-array)의 마지막 위치에 있는 요소인 9와 4를 비교한다. 9는 4보다 크기 때문에, 정렬된 배열에서 4가 9보다 우선한다. 따라서 9를 오른쪽으로 한 위치 이동(shift)시킨다 (복사한다).

    [2 3 5 9 4] → [2 3 5 9 **9**].

    ❖ 4는 "삭제"된 것이 아니라 이미 temp에 저장되어 있다.

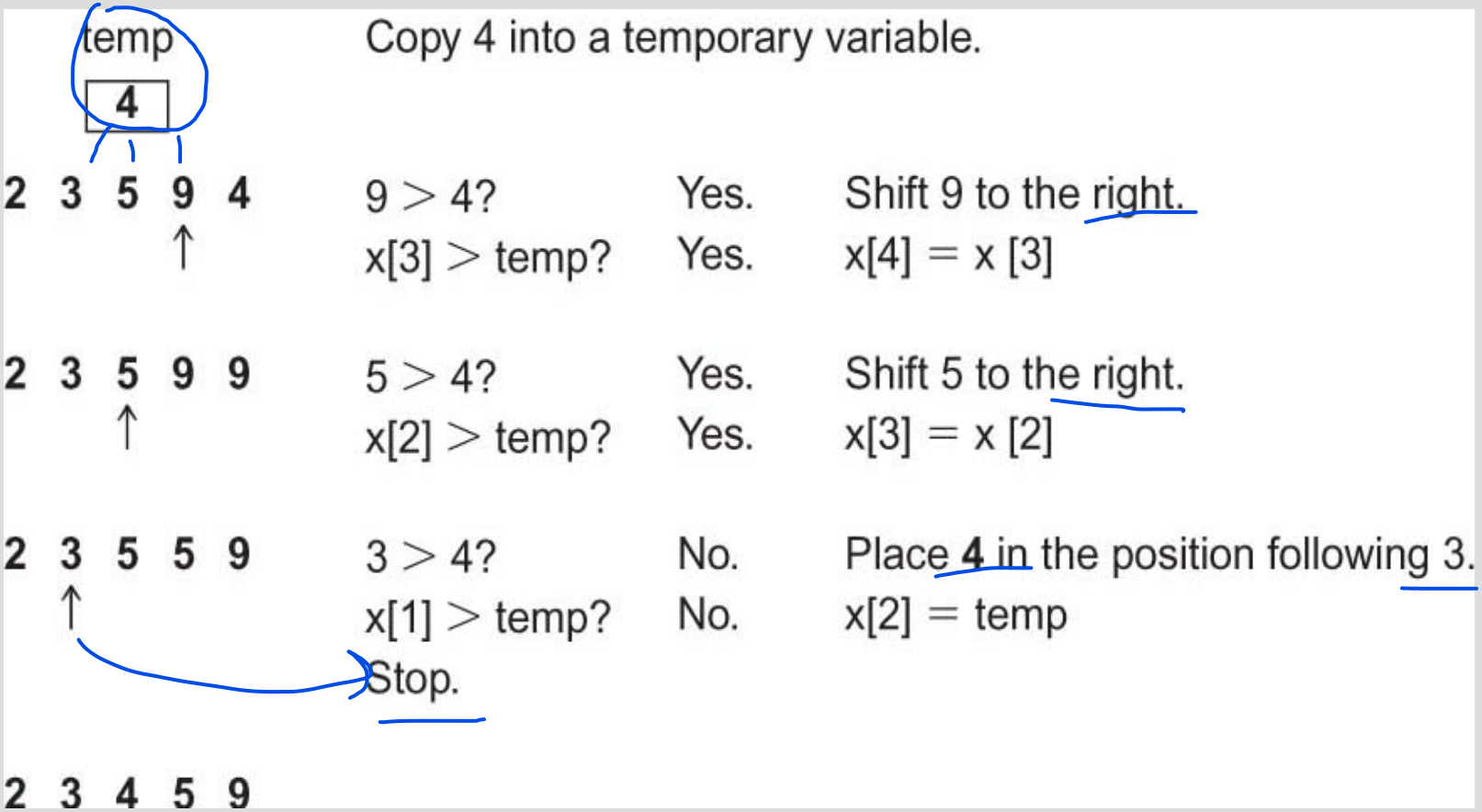
삽입 정렬

- 우선 마지막 요소가 정렬된 그룹 안에서 올바른 위치에 삽입되어야 한다. 이 숫자(**4**)를 적절한 위치로 배치하기 위하여, (cont.)
  - 4(temp)와 5를 비교한다. 5가 더 크기 때문에, 5를 오른쪽으로 한 위치 이동시킨다.

    [2 3 5 9 9] → [2 3 5 **5** 9]
  - 4(temp)와 3을 비교한다. 3은 4보다 작기 때문에 4의 올바른 위치가 결정된다. 이제 temp에 저장되어 있는 4를 즉시 3 다음의 위치에 복사하고 중지한다. 이로써 4는 3과 5 사이에 놓이게 된다.

    [2 3 5 5 9] → [2 3 **4** 5 9]

삽입 정렬



정렬된 리스트 내에서 올바른 위치로 숫자의 삽입

삽입 정렬

- 이 삽입 과정은 다음의 void 메소드로 구현이 가능하다.

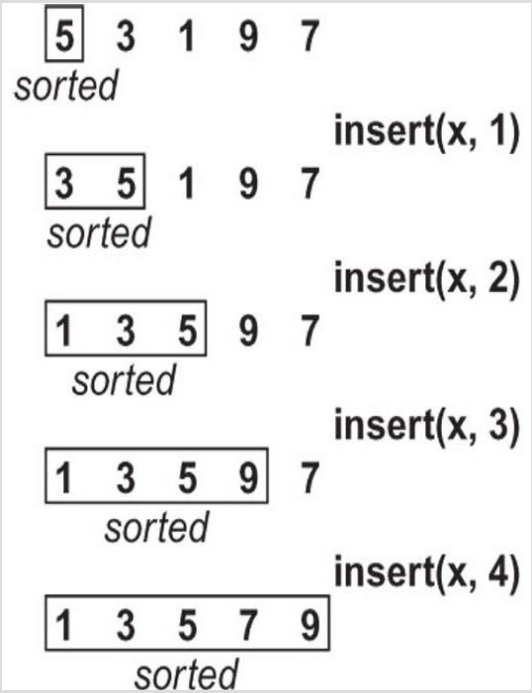
```
void insert(int[] x, int i)
```

- 이 메소드는 x의 i번째 값을 정렬된 값 x[0], x[1], ..., x[i-1] 사이의 적절한 위치에 놓게 되는데, 만약 필요하다면 이 숫자들을 오른쪽으로 위치 이동(shift)하게 된다.
- 메소드는 다음과 같이 동작한다.

1. x[i]를 임시 변수 temp에 복사
2. j를 정렬된 부-배열의 가장 큰 인덱스인 i-1로 초기화
3. while (j >= 0 and temp < x[j])
4. x[j]를 x[j+1]로 복사하고 j를 1 감소 // 위치 이동
5. temp를 x[j+1]에 복사

insert 메소드 작성

- *n*개의 요소가 있는 배열을 정렬하기 위하여, 삽입 정렬은 insert(...) 메소드를 *n-1*번 호출
- 삽입 정렬은 정렬된 부-배열, (x[0] x[1]), (x[0] x[1] x[2]), (x[0] x[1] x[2] x[3]), ... 마지막으로 (x[0] x[1] x[2] x[3] ... x[n-1])을 점진적으로 형성



삽입 정렬은 insert를 n−1번 호출한다.

삽입 정렬 구현

[문제 제시]

- 삽입 정렬을 구현하여라. 데이터 개수와 데이터 요소를 사용자에게 입력받는 main(...) 메소드를 포함시켜야 한다.

삽입 정렬 - 자바 해법

```
1.    import java.util.*;
2.    public class InsertionSort {

3.        // 정렬된 값 x[0], x[1], ..., x[i-1] 사이의 적절한 위치에 x[i]를 배치
4.        public static void insert(int[] x, int i) {
5.            int temp = x[i];              // 값 저장
6.            int j = i - 1;

7.            while ( j >= 0 && temp < x[j]) {  // temp를 배치해야 할 곳 결정
8.                x[j + 1] = x[j];      // 오른쪽으로 위치 이동
9.                j--;
10.           }
11.           x[j + 1] = temp;          // 올바른 위치에 temp(즉, 원래의 x[i])를 배치
12.       }

13.       // n은 x 배열에 저장된 데이터의 개수
14.       public static void insertionSort(int[] x, int n) {
15.           for (int i = 1; i < n; i++)
16.               insert(x, i);
17.       }
```

삽입 정렬 - 자바 해법 (cont.)

```
18.     public static void main(String[] args)  {
19.         Scanner input = new Scanner(System.in);
20.         int  size;                  // 데이터의 개수

21.         System.out.print("Enter the number of data: ");
22.         size = input.nextInt();
23.         int [] numbers = new int[size];
24.         System.out.print("Enter " + size + " integers: ");

25.         // 데이터 읽기
26.         for (int i = 0; i < size; i++)
27.             numbers[i] = input.nextInt();
28.         System.out.println();

29.         insertionSort(numbers, size);

30.         System.out.print("Sorted: ");
31.         for (int i = 0; i < size; i++)
32.             System.out.print(numbers[i] + " ");
33.         System.out.println();
34.     }
35. }
```

삽입 정렬 - 출력

Enter the number of data: 9
Enter 9 integers: 1 4 3 7 2 8 6 9 5
Sorted: 1 2 3 4 5 6 7 8 9

Enter the number of data: 5
Enter 5 integers: 1 4 3 2 5
Sorted: 1 2 3 4 5

이진 탐색 (Binary Search)

- *이진 탐색(binary search)*은 선형 탐색보다 훨씬 더 좋은 성능으로 수행되는 탐색 방법이다.
- 이진 탐색을 위해서는 배열이 미리 정렬되어 있어야 한다.
- 배열 x에서 키(key)를 탐색하기 위하여, 이진 탐색은 먼저 배열의 중앙에 있는 항목(x[mid])과 키를 비교한다.
  - 만약 키가 x[mid]와 같으면, 탐색은 성공적으로 종료된다.
  - 만약 키 < x[mid]이면, x[mid]와 x[mid]보다 큰 모든 항목들은 탐색으로부터 제외된다.
  - 만약 키 > x[mid]이면, x[mid]와 x[mid]보다 작은 모든 항목들은 탐색으로부터 제외된다.
- 따라서 하나의 위치만 검토한 후에, 배열에 있는 데이터의 절반은 탐색으로부터 제외될 수 있다.
- 이진 탐색은 아직 제외되지 않은 배열의 일부에 대하여 키를 발견하거나 또는 검사할 항목이 더 이상 없을 때까지 이 과정을 반복한다.

Start from middle

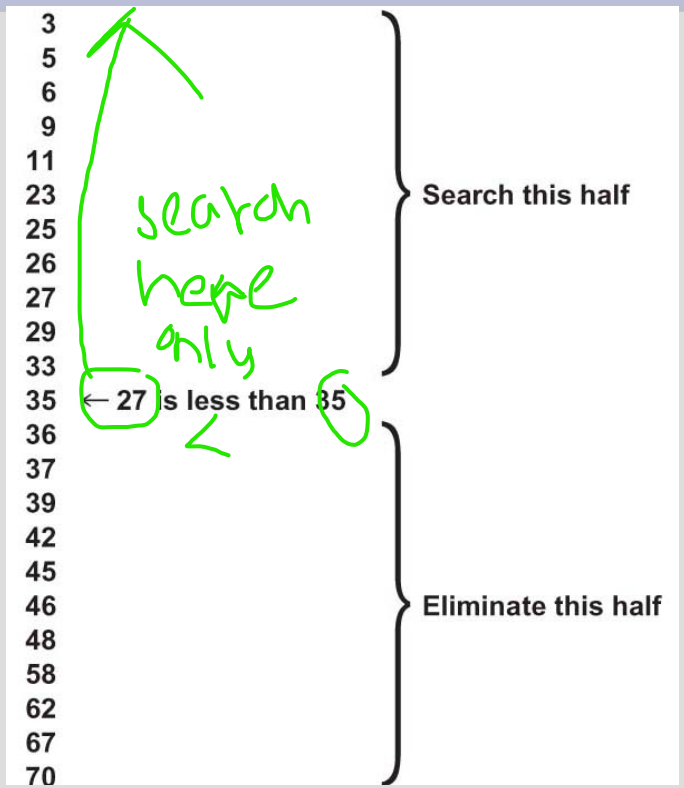
이진 탐색

- 다음의 정렬된 배열에서 키 27의 탐색을 살펴보자.

  [3 5 6 9 11 23 25 26 27 29 33 **35** 36 37 39 42 45 46 48 58 62 67 70]

- 이진 탐색에서는 먼저 키 27과 배열의 중앙 항목인 35를 비교한다.
- *배열이 정렬되어 있는 상태*이고 27이 배열의 중앙 항목인 35보다 작기 때문에, "부분-배열"인 [35 36 37 39 42 45 46 48 58 62 67 70]은 다음 탐색에서 제거된다.
- 만약 27이 리스트에 존재한다면, 27은 부분-배열 [3 5 6 9 11 23 25 26 27 29 33] 내에 존재해야 된다.

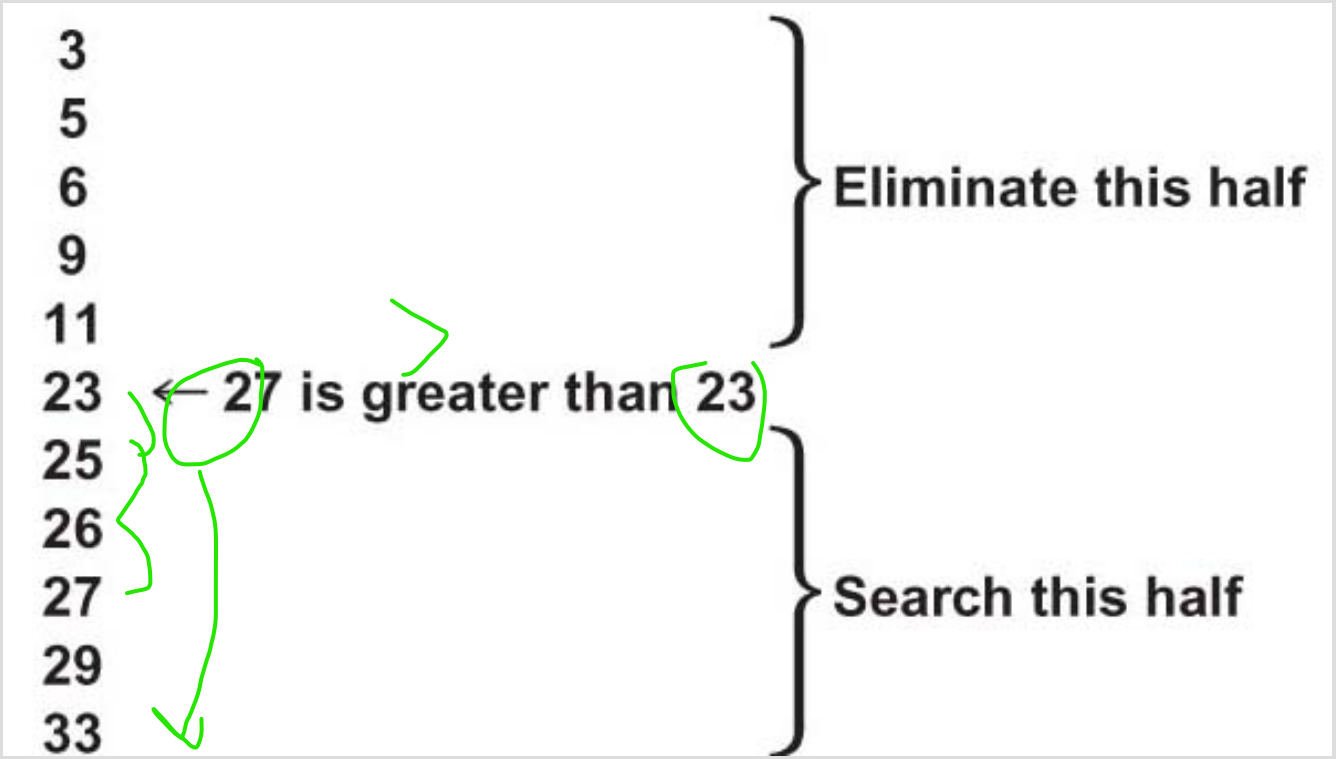
이진 탐색



- **한 번의 비교 후, 배열의 절반은 제외된다.**
- 다음으로, 이진 탐색은 이 부분-배열의 중앙 요소 값과 키(27)를 비교한다.

이진 탐색

- 두 번 비교 후의 이진 탐색



이진 탐색

- 27은 중앙 요소 값(23)보다 크기 때문에, 이진 탐색은 23보다 큰 값들을 탐색한다.

발견된 키

- 27의 키 값은 단지 세 번의 위치 검사 후에 발견되었다.

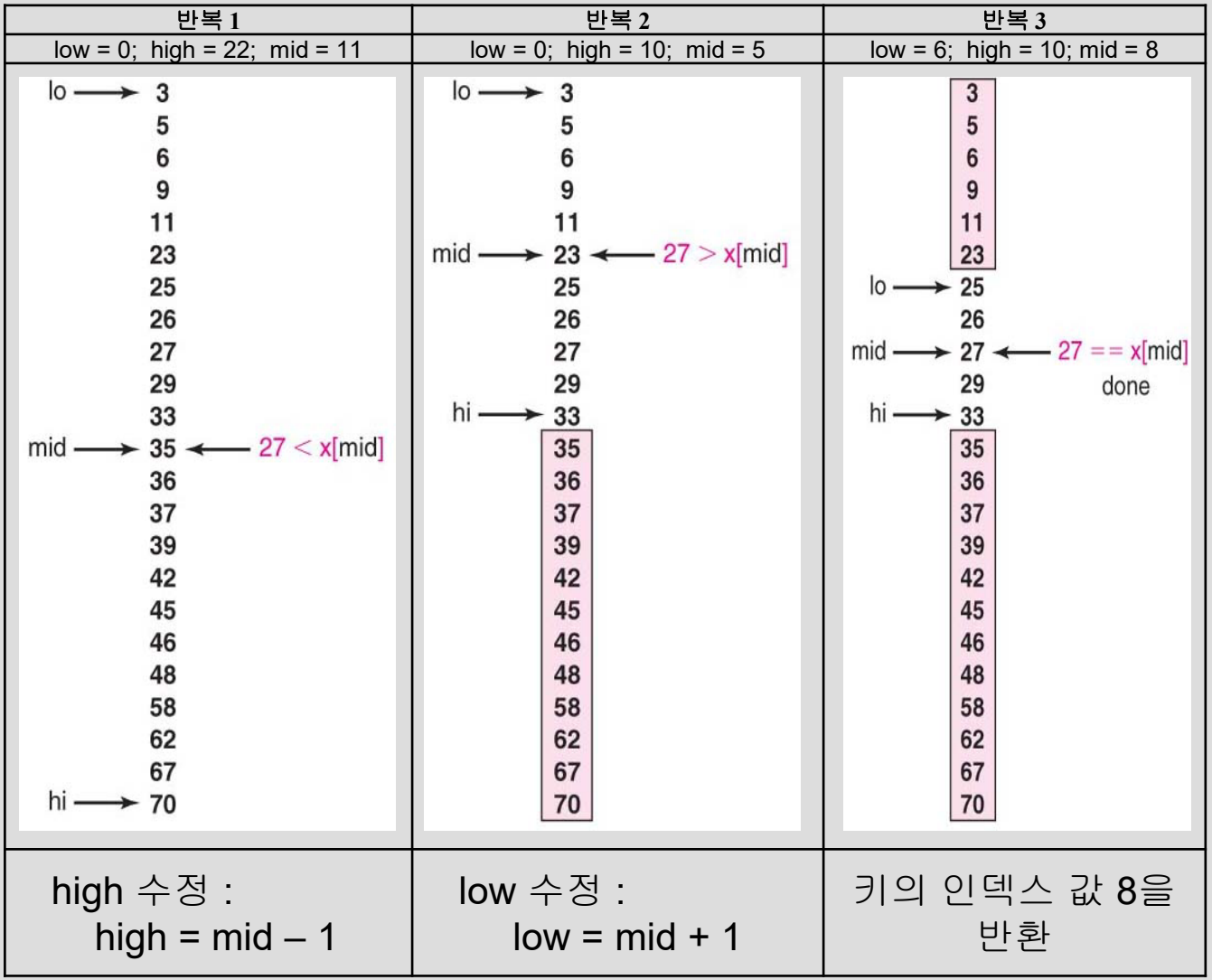
이진 탐색을 구현한 자바 메소드

```
1.    public static int bsearch(int[] x , int n, int key)  {
2.    // x는 n개의 정수가 정렬된 배열이다. 키는 정수 값을 가진다.
3.    // x는 오름차순으로 정렬되어 있다.
4.        int low = 0;                  // 배열의 최소 인덱스
5.        int  high = n - 1;            // 최대 인덱스
6.        int  mid;                     // 중앙 인덱스

7.        while (high >= low)  {
8.            mid  = (high + low) / 2;        // 중앙 인덱스 계산
9.            if (key == x[mid])
10.               return mid;                 // 키를 발견 -- 종료
11.           if (key < x[mid])
12.               high = mid - 1;             // x[mid]에서 x[high]까지 제외
13.           else
14.               low = mid + 1;              // x[low]에서 x[mid]까지 제외
15.       }
16.       return -1;      // 키를 발견하지 못함
17.   }
```

이진 탐색 동작

key = 27; 밝게 강조된 블록은 탐색으로부터 제외된다.



.