스크립트 프로그래밍

11 다차원 리스트

2016 2학기 (02분반)

강승우

10 리스트

리스트 검색하기

- 검색
  - 리스트에서 특정 원소를 찾는 과정
  - 프로그래밍에서 흔한 작업
    - 많은 검색 알고리즘이 개발되어 왔음
  - 예
    - 선형 검색 (linear search)
    - 이진 검색 (binary search)

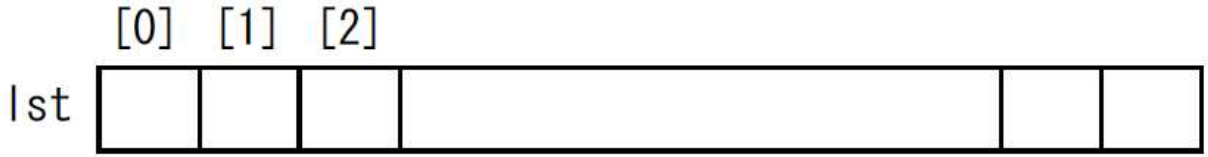
- list 클래스
  - 검색 관련 메소드
    - index(x): x라는 오브젝트가 리스트에 있는지 검색하여 있으면 그의 인덱스를 반환
    - in / not in: 어떤 원소가 리스트에 있는지 없는지 판단

선형 검색

- 찾고자 하는 원소(key)를 리스트의 각 원소와 순차적으로 비교한다
- Key와 일치하는 원소를 리스트에서 찾을 때까지 혹은 일치된 원소가 없이 리스트의 모든 원소를 전부 비교할 때까지 반복
  - 일치하는 원소를 찾으면, 이 원소의 인덱스 반환
  - 찾지 못 하면 -1 반환

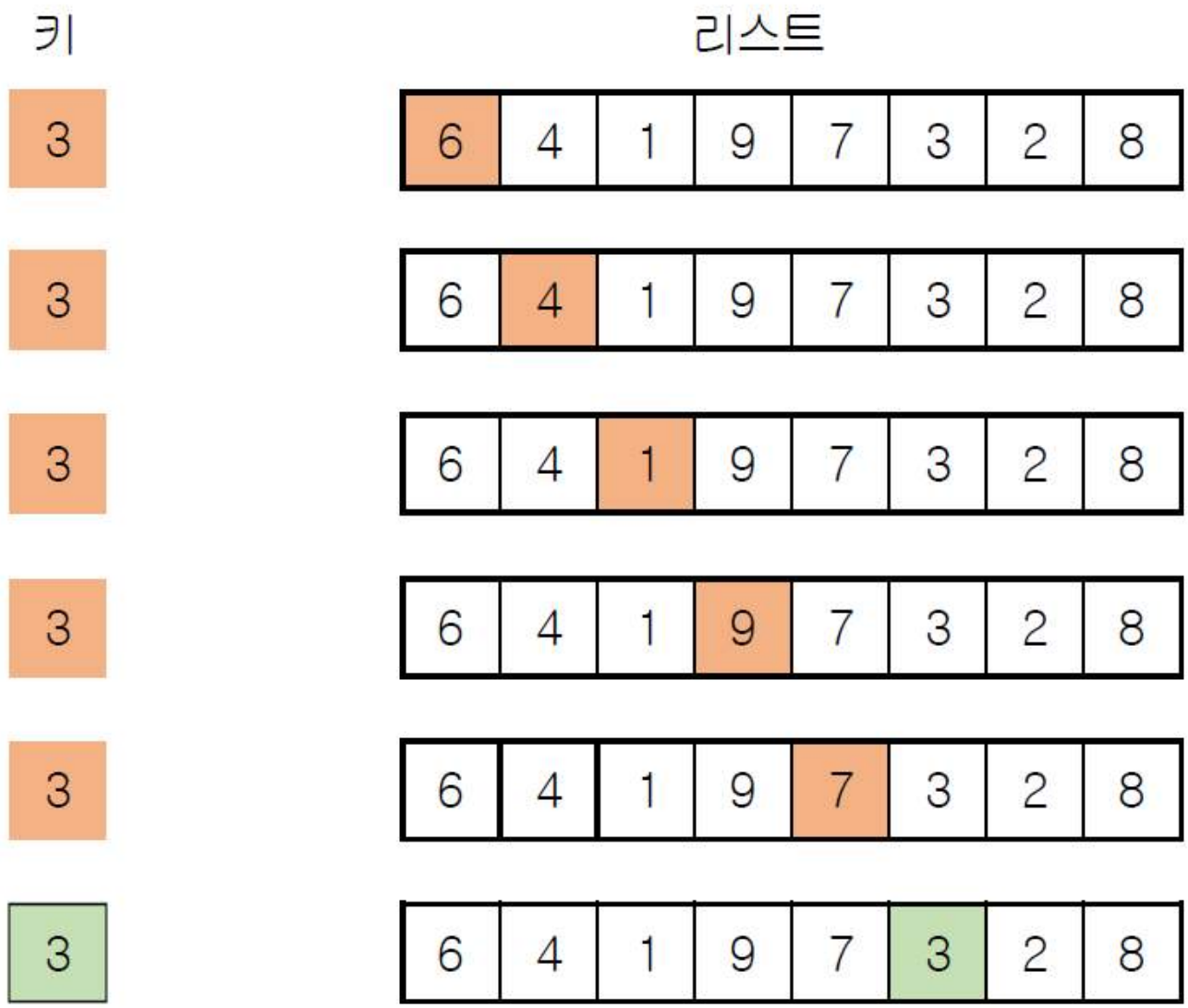
선형 검색 예제

```
#리스트에서 key를 찾는 함수

def linearSearch(lst, key):
    for i in range(0, len(lst)):
        if key == lst[i]:
            return i

    return -1
```



key i = 0, 1, …에 대해 lst[i]와 key를 비교한다.

선형 검색



선형 검색 알고리즘

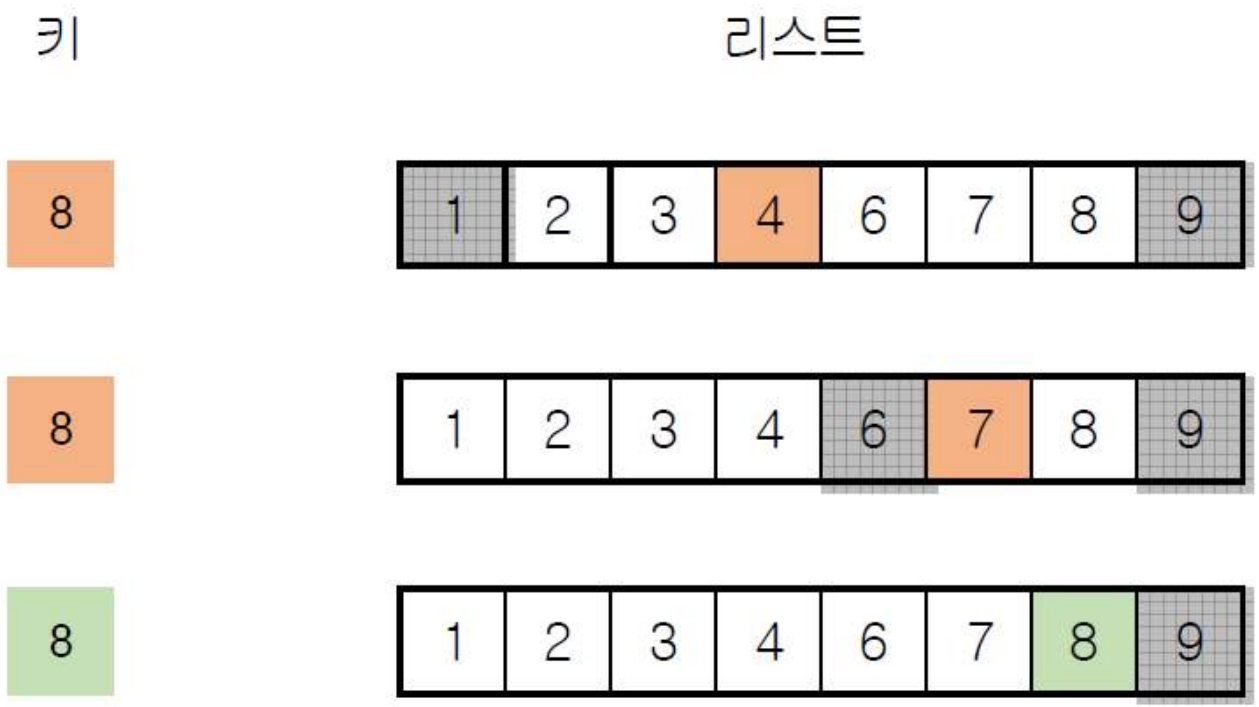
- ✓ 평균적으로 리스트 절반을 검색해야 함
- ✓ 검색 실행 시간은 리스트 원소의 개수가 증가할 수록 선형적으로 증가
- ✓ 큰 리스트의 경우 비효율적임

이진 검색

- 한 번에 리스트의 반씩 쪼개어 가면서 검색해야 하는 부분을 줄임
- 리스트의 원소가 정렬이 되어 있어야 함

- 오름차순 정렬된 리스트를 가정하면,
- 리스트의 중간 원소와 키를 비교
  - 키가 중간 원소보다 작으면, 리스트의 첫 번째 절반만을 대상으로 다시 키 검색
  - 키가 리스트의 중간 원소와 같으면, 인덱스를 반환하고 검색 종료
  - 키가 리스트의 중간 원소보다 크면, 리스트의 두 번째 절반만을 대상으로 다시 키 검색

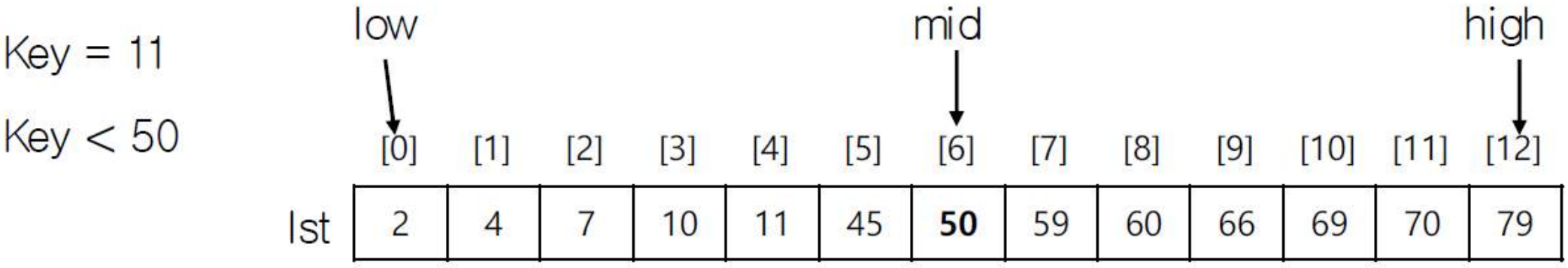
이진 검색 예제



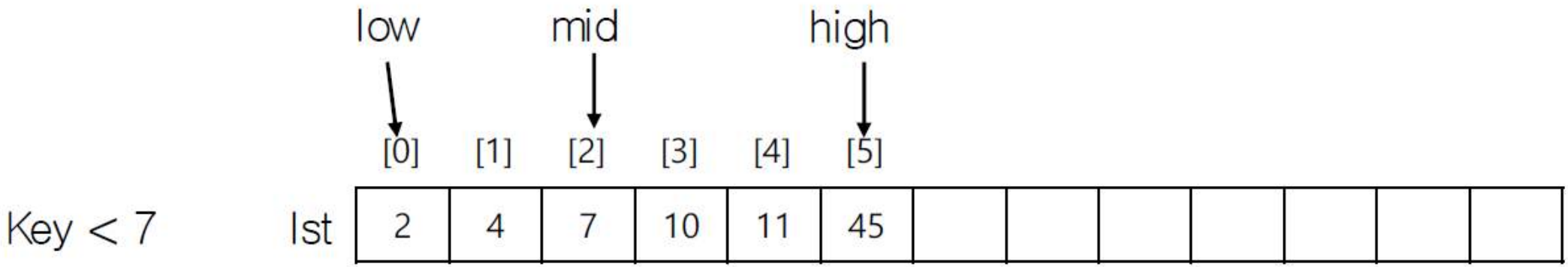
이진 검색 예제 – 검색 원소가 존재 (1/2)

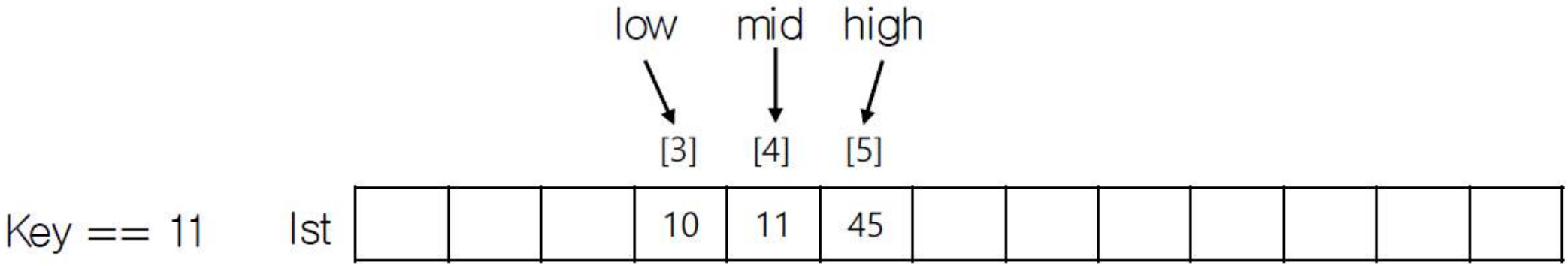
- 리스트
  - 2 4 7 10 11 45 50 59 60 66 69 70 79

- 검색하고자 하는 key = 11



이진 검색 예제 – 검색 원소가 존재 (2/2)

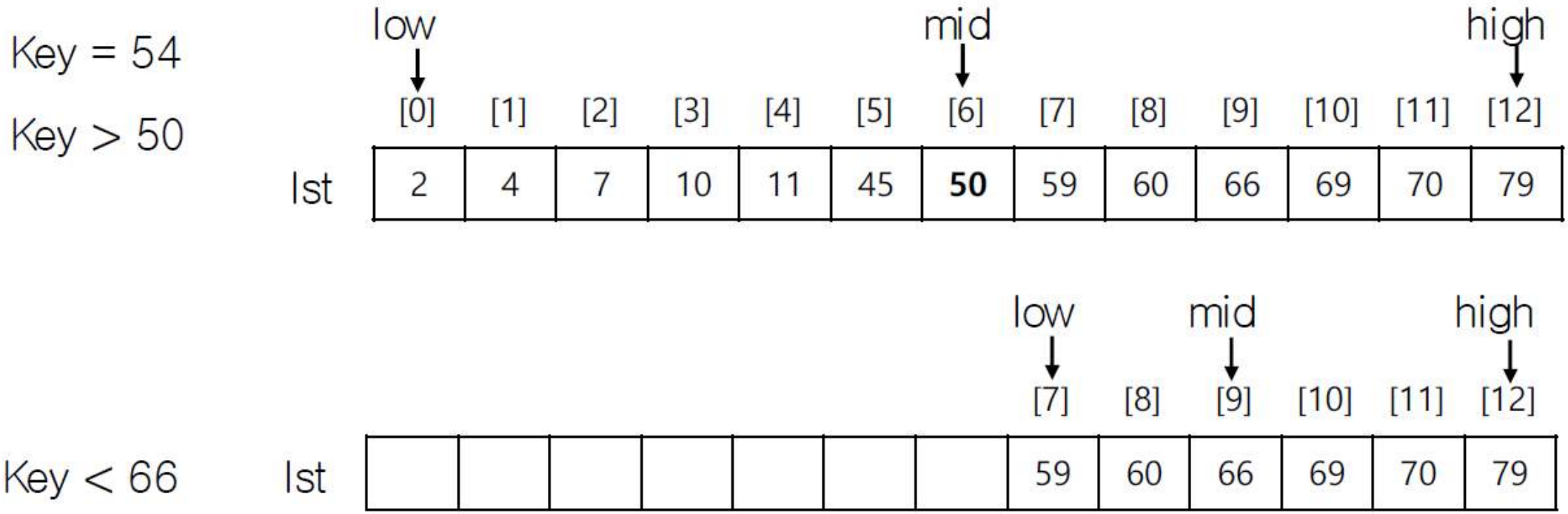




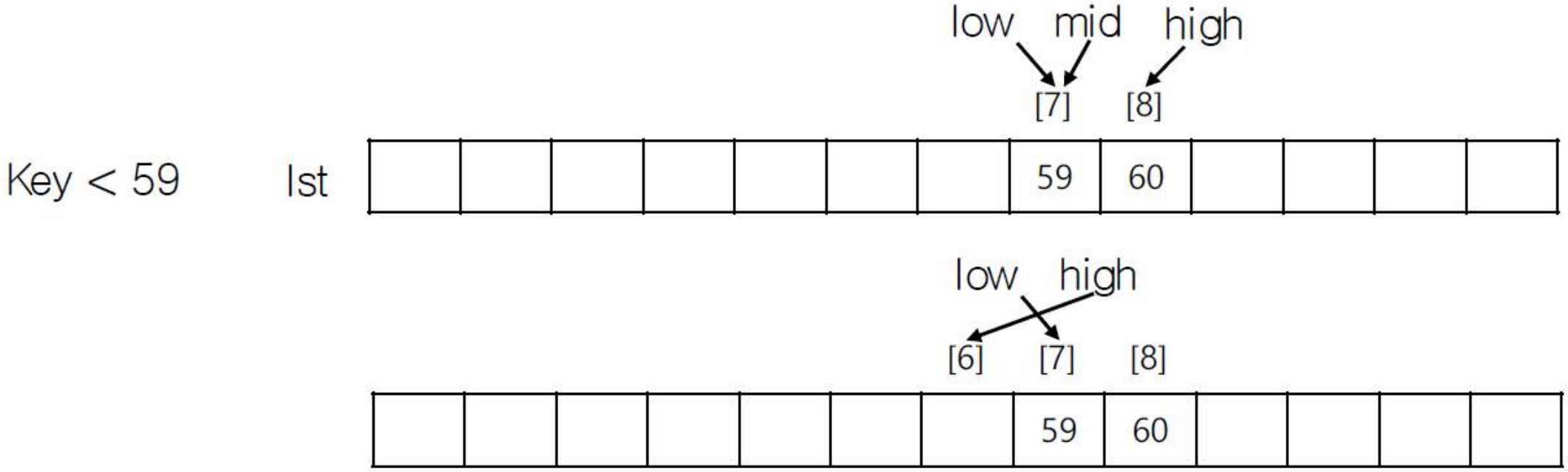
✓ 원소 11의 인덱스 4 반환

이진 검색 예제 – 검색 원소가 미존재 (1/2)

- 없으면 해당 키가 리스트에 삽입되어야 할 위치 반환



이진 검색 예제 – 검색 원소가 미존재 (2/2)



✓ low – 1 ➔ 6 반환

이진 검색 구현 코드

```
# 리스트에서 키를 찾기 위해 이진 검색을 사용한다.
def binarySearch(lst, key):
    low = 0
    high = len(lst) - 1

    while high >= low:
        mid = (low + high) // 2
        if key < lst[mid]:
            high = mid - 1
        elif key == lst[mid]:
            return mid
        else:
            low = mid + 1

    return - low - 1 # 현재 high < low이므로, 키는 발견되지 않음
```

리스트 정렬하기

- 리스트의 원소들을 일정 순서로 나열하는 것
  - 오름차순
- 다양한 정렬 알고리즘 존재
  - 선택 정렬 (Selection sort), 삽입 정렬(Insertion sort), quick sort, bubble sort, heap sort

- list 클래스
  - sort() 메소드
    - 리스트 내의 원소를 오름차순 정렬

선택 정렬

- 리스트 내에서 가장 작은 원소를 찾고 그것을 첫번째 원소와 교환
- 남은 원소 중에서 가장 작은 원소를 찾고 그것을 남은 원소들 중에 첫번째 원소와 교환
- 이 과정을 한 개의 원소가 남을 때까지 반복

- 예
  - 리스트 [2, 9, 5, 4, 8, 1, 6]

선택 정렬 예제 (1/2)

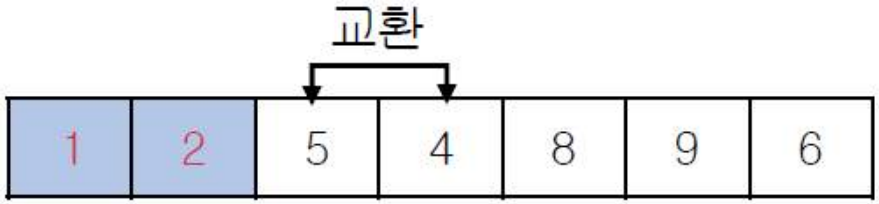
- 우선 리스트의 첫번째 원소를 가장 작은 원소로 가정하고, 이를 제외한 나머지 부분의 원소들 중에서 이보다 작은 것이 있는지 확인, 있다면 서로 교환

리스트에서 1(가장 작은)을 선택하고 2(첫 번째에 위치한)와 교환한다.

숫자 1은 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.

남은 리스트에서 2(가장 작은)를 9(첫 번째)와 교환한다.

숫자 2는 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.



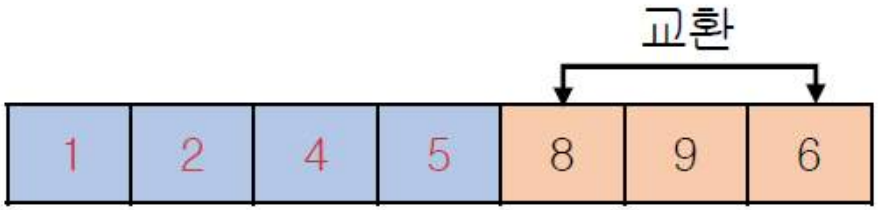
남은 리스트에서 4(가장 작은)를 5(첫 번째)와 교환한다.

선택 정렬 예제 (2/2)

숫자 4는 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.

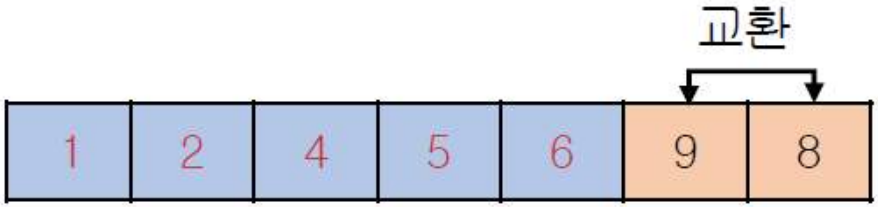
5는 가장 작고 올바른 위치에 놓여 있다. 교환은 필요없다.

숫자 5는 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.



남은 리스트에서 6(가장 작은)을 8(첫 번째)과 교환한다.

숫자 6은 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.



남은 리스트에서 8(가장 작은)을 9(첫 번째)와 교환한다.

숫자 8은 현재 정확한 위치에 놓여있기 때문에 더 이상 고려될 필요가 없다.

남은 리스트에 원소가 한 개밖에 없으므로 정렬이 완료된다.

선택 정렬 구현 코드

```
# 원소를 오름차순으로 정렬하기 위한 함수
def selectionSort(lst):
    for i in range(len(lst) - 1):
        # lst[i : len(lst)]에서 가장 작은 원소를 찾는다.
        currentMin, currentMinIndex = lst[i], i

        for j in range(i + 1, len(lst)):
            if currentMin > lst[j]:
                currentMin, currentMinIndex = lst[j], j

        # 필요한 경우, lst[i]와 lst[currentMinIndex] 를 교환한다.
        if currentMinIndex != i:
            lst[currentMinIndex], lst[i] = lst[i], currentMin
```

11 다차원 리스트

학습 목표

- 2차원 리스트가 어떻게 2차원 데이터를 표현할 수 있는지 이해할 수 있다(§11.1절).
- 행과 열 인덱스를 사용하여 2차원 리스트의 원소에 접근할 수 있다(§11.2절).
- 리스트 출력, 전체 원소의 합계, 가장 작은(min) 또는 가장 큰(max) 원소 검색, 랜덤 섞기 및 정렬 등 2차원 리스트의 기본 연산을 프로그래밍할 수 있다(§11.2절).
- 함수에 2차원 리스트를 전달할 수 있다(§11.3절).
- 2차원 리스트를 사용하여 객관식 문제를 평가하는 프로그램을 작성할 수 있다 (§11.4절)
- 2차원 리스트를 사용하여 가장 가까운 짝을 찾는 문제를 해결할 수 있다(§11.5-§11.6절).
- 2차원 리스트를 사용하여 스도쿠 풀이를 검사할 수 있다(§11.7-§11.8절).
- 다차원 리스트를 사용할 수 있다(§11.9절).

2차원 리스트

- 리스트의 원소가 또 다른 리스트로 이루어진 리스트
  - 행렬 또는 표와 같은 2차원 데이터를 저장하기 위해서 2차원 리스트 사용 가능
  - 행: row index / 열: column index

```
matrix = [
  [1, 2, 3, 4, 5],
  [6, 7, 0, 0, 0],
  [0, 1, 0, 0, 0],
  [1, 0, 0, 0, 8],
  [0, 0, 9, 0, 3],
]
```

	[0]	[1]	[2]	[3]	[4]
[0]	1	2	3	4	5
[1]	6	7	0	0	0
[2]	0	1	0	0	0
[3]	1	0	0	0	8
[4]	0	0	9	0	3

```
matrix[0] 은 [1, 2, 3, 4, 5]이다
matrix[1] 은 [6, 7, 0, 0, 0]이다
matrix[2] 은 [0, 1, 0, 0, 0]이다
matrix[3] 은 [1, 0, 0, 0, 8]이다
matrix[4] 은 [0, 0, 9, 0, 3]이다

matrix[0][0]은 1이다
Matrix[4][4]은 3이다
```

랜덤 값으로 리스트 초기화하기

```
import random
matrix = [] # 비어있는 리스트를 생성한다.

numberOfRows = eval(input("행의 개수를 입력하세요: "))
numberOfColumns = eval(input("열의 개수를 입력하세요: "))
for row in range(0, numberOfRows):
    matrix.append([]) # 새로운 빈 행을 추가한다.
    for column in range(0, numberOfColumns):
        matrix[row].append(random.randrange(0, 100))

print(matrix)
```

리스트 출력하기

- row, column index를 이용하여 리스트를 순회하면서 각 원소를 출력

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] #리스트는 주어진다.
for row in range(0, len(matrix)):
    for column in range(0, len(matrix[row])):
        print(matrix[row][column], end = " ")
    print() # 새로운 행을 출력한다.
```

모든 원소 합계 구하기

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
total = 0
for row in range(0, len(matrix)):
    for column in range(0, len(matrix[row])):
        

print("Total is " + str(total)) # 합계를 출력한다.
```

합계가 가장 큰 행 찾기

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
maxRow = sum(matrix[0]) # maxRow에 첫 번째 행의 합계를 저장
indexOfMaxRow = 0

for row in range(1, len(matrix)):
    # 현재 행의 합계가 maxRow보다 크면
    # 현재 행의 합계를 maxRow에 저장하고, 현재 행을 indexOfMaxRow에 저장
```

```
print(indexOfMaxRow, "번째 행의 합계", maxRow, "가 가장 큽니다.")
```

합계가 가장 큰 행 찾기

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] # Assume a list is given
maxRow = sum(matrix[0]) # maxRow에 첫 번째 행의 합계를 저장
indexOfMaxRow = 0

for row in range(1, len(matrix)):
    # 현재 행의 합계가 maxRow보다 크면
    # 현재 행의 합계를 maxRow에 저장하고, 현재 행을 indexOfMaxRow에 저장
    if sum(matrix[row]) > maxRow:
        maxRow = sum(matrix[row])
        indexOfMaxRow = row

print(indexOfMaxRow, "번째 행의 합계", maxRow, "가 가장 큽니다.")
```

다음 코드 실행 결과는?

```
matrix = []
matrix.append(3*[1])
matrix.append(3*[1])
matrix.append(3*[1])
matrix[0][0] = 2
print(matrix)
```

```
matrix = []
matrix.append(3*[1])
matrix.append(3*[1])
matrix.append(3*[1])
matrix[0] = 3
print(matrix)
```

사례 연구: 객관식 문제 평가하기

질문에 대한 학생의 답

	0	1	2	3	4	5	6	7	8	9
학생0	A	B	A	C	C	D	E	E	A	D
학생1	D	B	A	B	C	A	E	E	A	D
학생2	E	D	D	A	C	B	E	E	A	D
학생3	C	B	A	E	D	C	E	E	A	D
학생4	A	B	D	C	C	D	E	E	A	D
학생5	B	B	E	C	C	D	E	E	A	D
학생6	B	B	A	C	C	D	E	E	A	D
학생7	E	B	E	C	C	D	E	E	A	D

질문에 대한 정답:

	0	1	2	3	4	5	6	7	8	9
정답	D	B	D	C	C	D	A	E	A	D

- 각 학생 별로 정답 문항 개수를 출력하는 프로그램을 작성해보자

사례 연구: 객관식 문제 평가하기

- 학생 답
  - 2차원 리스트
- 정답
  - 1차원 리스트

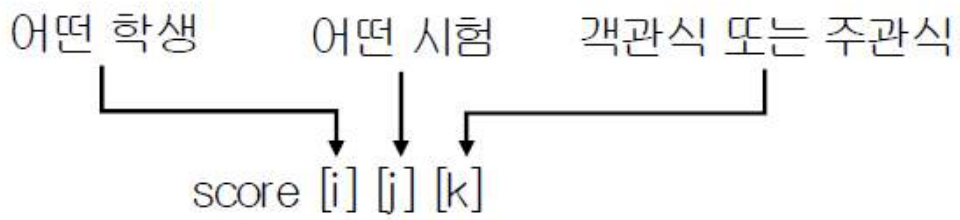
- 위 두 개의 리스트가 주어졌을 때,
- 학생 답 2차원 리스트의 각 행의 원소가 정답 리스트의 원소와 일치하는 검사하여 일치하는 개수를 계산한다
- 각 행의 비교가 끝나면 개수를 출력한다

다차원 리스트

- 리스트이 각 원소가 또 다른 리스트인 리스트
- 3차원 리스트
  - 2차원 리스트의 리스트
- 예
  - 한 반에 포함된 6명의 학생에 대한 5번의 시험 그리고 2종류(객관식, 주관식)로 구성된 시험 점수를 저장하는 리스트

다차원 리스트

```
scores = [
  [[9.5, 20.5], [9.0, 22.5], [15, 33.5], [13, 21.5], [15, 2.5]],
  [[4.5, 21.5], [9.0, 22.5], [15, 34.5], [12, 20.5], [14, 9.5]],
  [[6.5, 30.5], [9.4, 10.5], [11, 33.5], [11, 23.5], [10, 2.5]],
  [[6.5, 23.5], [9.4, 32.5], [13, 34.5], [11, 20.5], [16, 9.5]],
  [[8.5, 26.5], [9.4, 52.5], [13, 36.5], [13, 24.5], [16, 2.5]],
  [[9.5, 20.5], [9.4, 42.5], [13, 31.5], [12, 20.5], [16, 6.5]]]
```



- scores[0][1][0]
- 각 학생의 5번의 시험에 대한 객관식 점수의 평균과 주관식 점수의 평균을 계산하는 프로그램을 작성해보자