인공지능

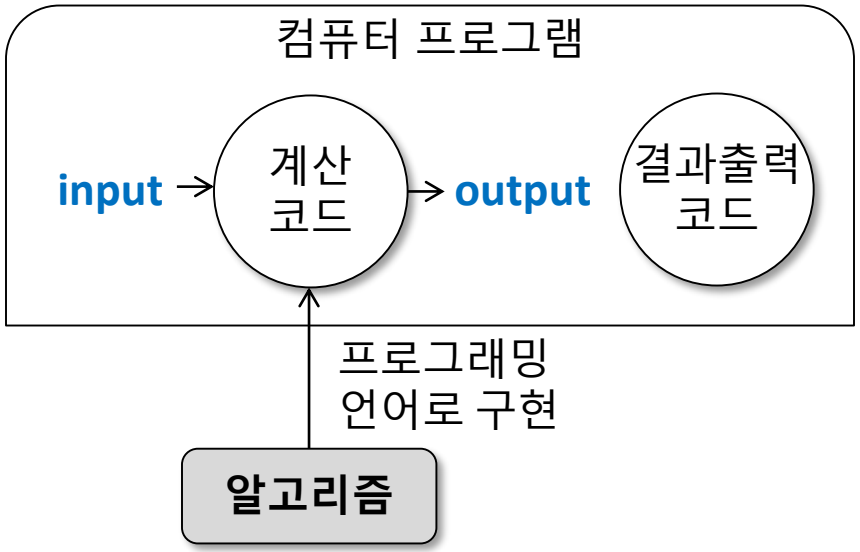
Week 1-1
알고리즘

시스템경영공학부 이지환 교수

프로그램(Program)

- **알고리즘이란**
  문제를 해결하기 위하여 특정한 형태의 입력(input)을 가지고 원하는 결과(output)을 얻어낼 수 있는 잘 정의된(well-defined) 절차

- **프로그램(Program)**
  알고리즘에 정의된 절차를 컴퓨터로 하여금 수행하도록 하기 위해 기술한 지시문



알고리즘의 조건

1. **입력**
   외부에서 제공되는 자료가 존재해야한다.

2. **출력**
   문제를 해결하기 위한 정확한 결과를 얻을 수 있어야 한다.

3. **명확성**
   수행 과정은 명확해야 하고 모호하지 않은 명령어로 구성되어야 한다.

4. **종결성**
   알고리즘은 반드시 종료될 수 있도록 작성되어야 한다.

5. **효율성**
   모든 과정은 명백하게 실행 가능(검증 가능)한 것이어야 한다.

알고리즘 작성의 의미

1. **단계적 계산 절차의 기술**
   프로그래밍 코드는 단순한 문장(변수=계산식)들로 이루어지기 때문에, 입력값으로부터 출력값을 얻어내는 과정이 단계적으로 기술되어야 한다.

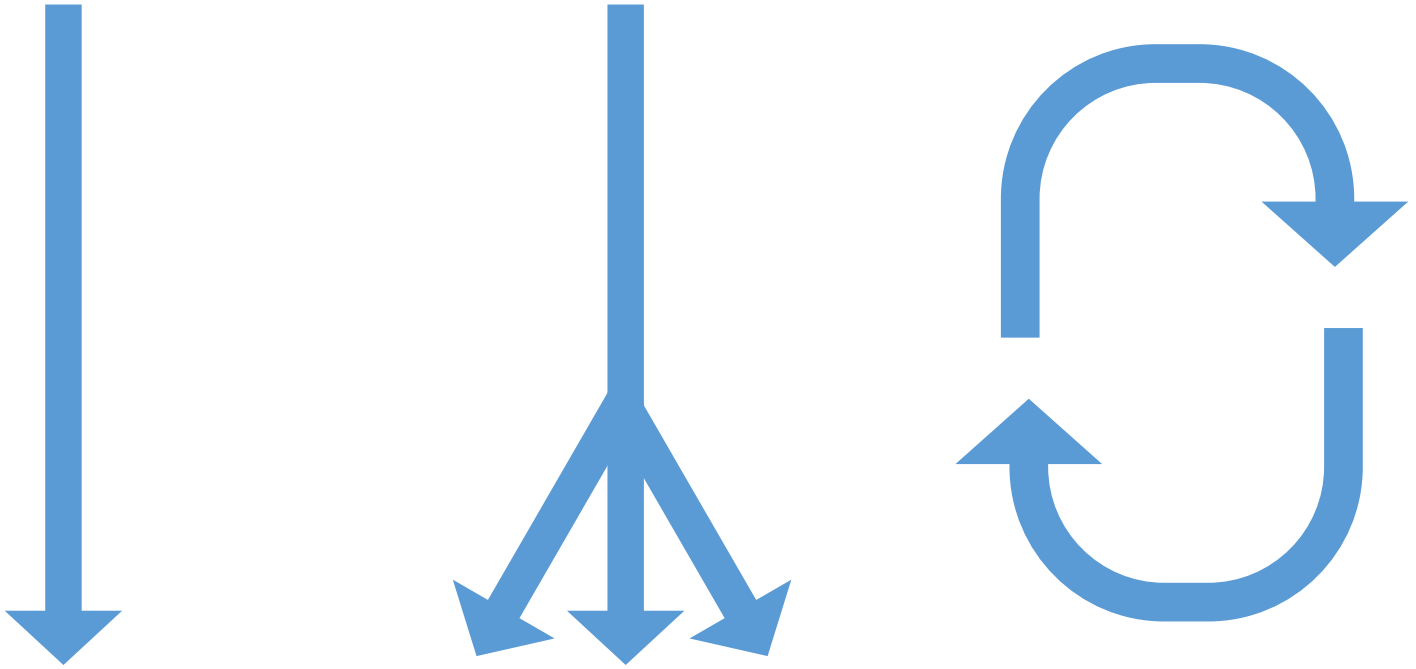
2. **프로세스의 정립**
   컴퓨터가 하는 계산 뿐 아니라 '**인간이 하는 일**'도 단계별 프로세스로 기술되어야 누구나 따라할 수 있다.
   ex) '밀가루 반죽을 만들어라' vs.
   '1) 밀가루 2컵을 붓고, 2)물 1컵을 부은 다음, 3)계란 1개를 넣고 4) 덩어리가 질 때까지 치대시오'

3. **효율성에 대한 고려**
   단순히 결과값을 얻어내는 것보다 얼마나 빨리, 얼마나 적은 메모리 공간을 사용하면서 결과값을 얻을 것이냐의 문제.
   → 기업의 문제는 수많은 입력값으로부터 결과를 얻어야 하기 때문에 **경영과학(O.R.)의 주요 관심사**

알고리즘의 구조

- **알고리즘을 구성하는 세가지 구조**
  알고리즘에 사용되는 절차는 세 가지 구조의 조합으로 구성되어 있다.

순차구조 선택구조 반복구조

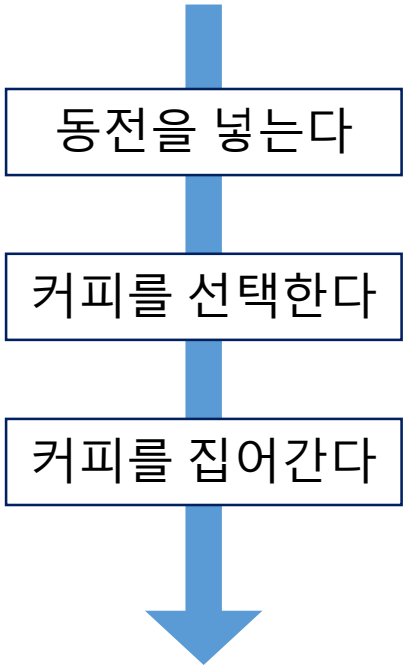


알고리즘의 구성 - 순차구조

- **순차구조: 처음부터 순서대로 작업을 처리**
  - 알고리즘은 기본적으로 순차적으로 작업을 진행해 나간다.

- **자판기에서 커피를 뽑기 위한 순차적 절차는 다음과 같다.**

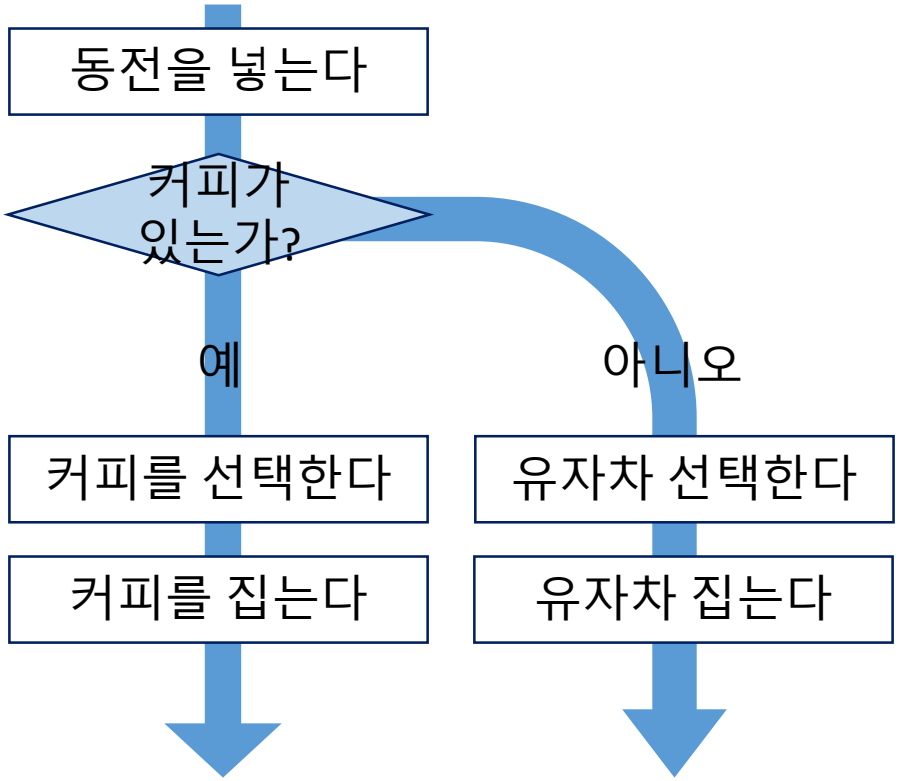
문제: 자판기에서 커피뽑기



알고리즘의 구성 - 선택구조

- **선택구조: 조건에 따라 실행할 작업을 전환**
  - 조건에 따라서 그 이후의 처리가 나누어 지는 경우가 빈번히 발생한다.

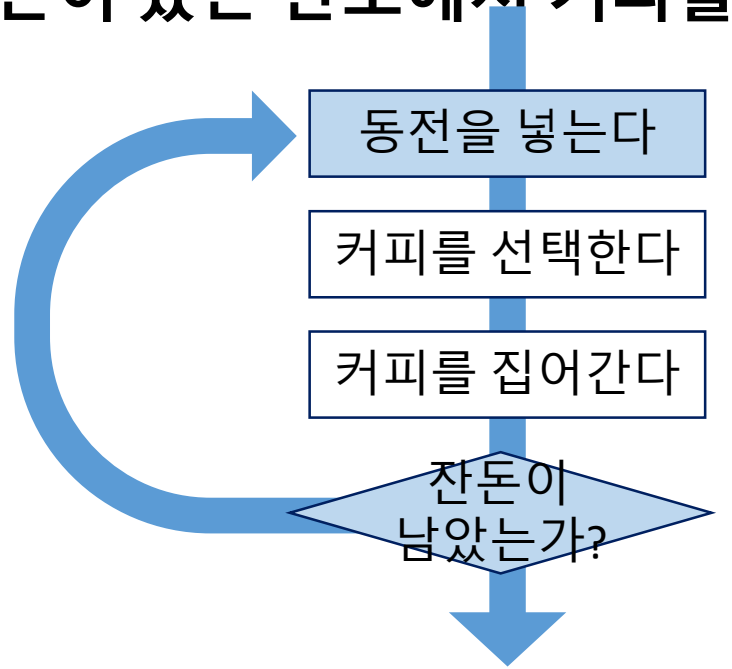
- **자판기에 커피가 없는 경우, 유자차를 뽑고 싶다.**



알고리즘의 구성 - 반복구조

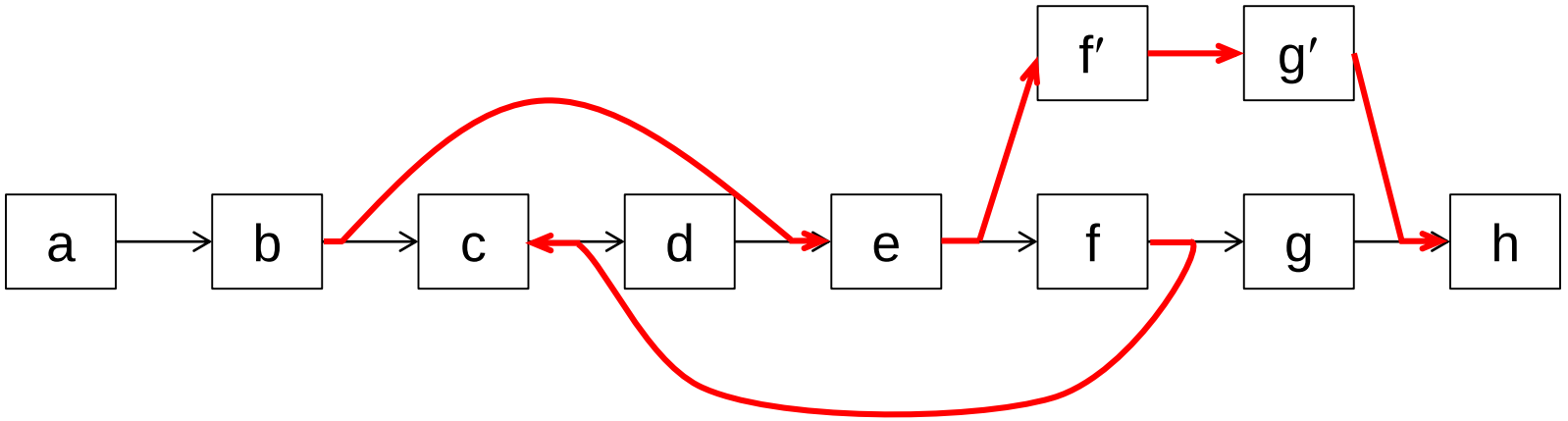
- **반복: 조건을 만족하는 동안 같은 처리를 반복**
  - 알고리즘에서는 또한 같은 작업을 지속적으로 반복하는 경우가 많이 있다.

- **내가 가진 돈이 있는 한도에서 커피를 여러 잔 뽑고 싶다.**



결국 알고리즘은

- 복잡한 일도 작은 단위로 쪼개면 단순한 작업의 들의 흐름이 된다.
- 알고리즘은 순차+선택+반복을 조합하여 선택과 반복복잡한 흐름은 순차적으로만은 표현할 수 없다.



실습0: 최소값 위치반환

- 문제: 숫자로 이루어진 리스트 안에서 최소값의 위치 반환

- 예시:
  입력: [3,2,1,4,5]
  출력: 2

- 알고리즘

탐색과 정렬 알고리즘

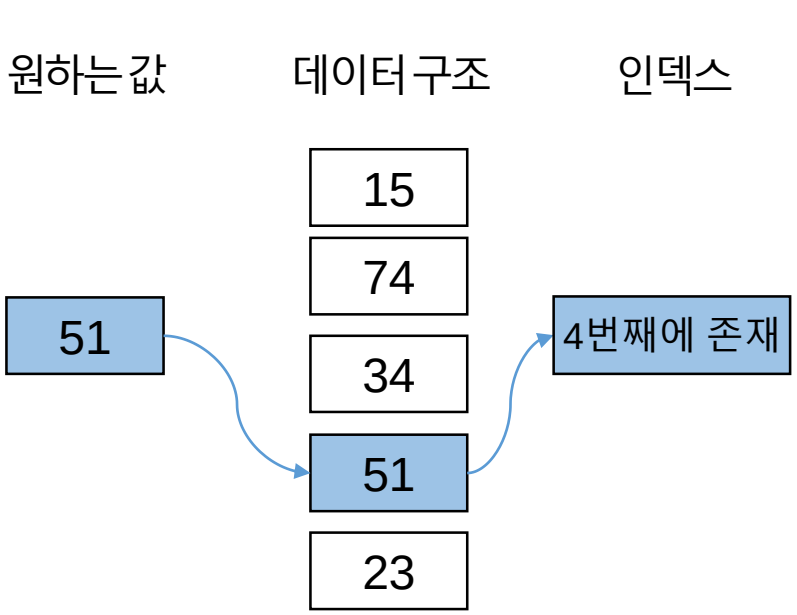
- 가장 기초적이고 대표적으로 활용되는 두 알고리즘: {탐색}과 {정렬}

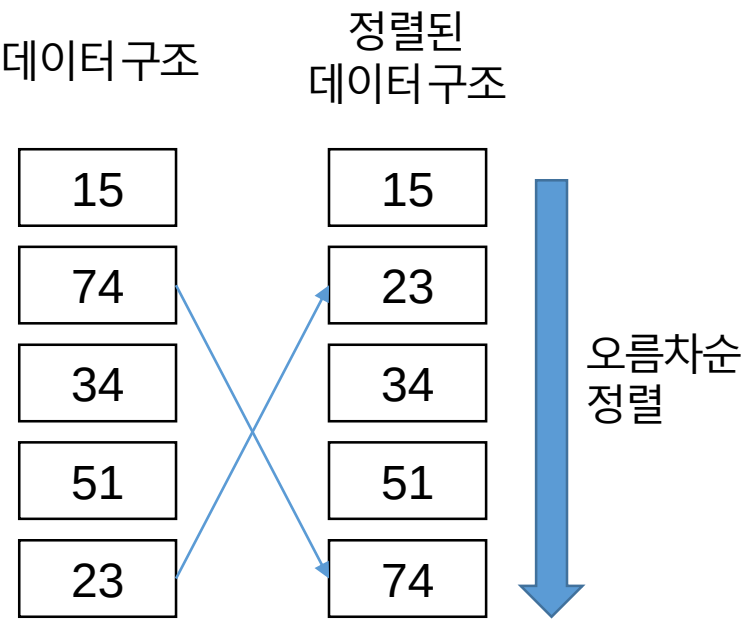
탐색

주어진 데이터 구조 안에 원하는 값이 있는지 찾아내는 문제

정렬

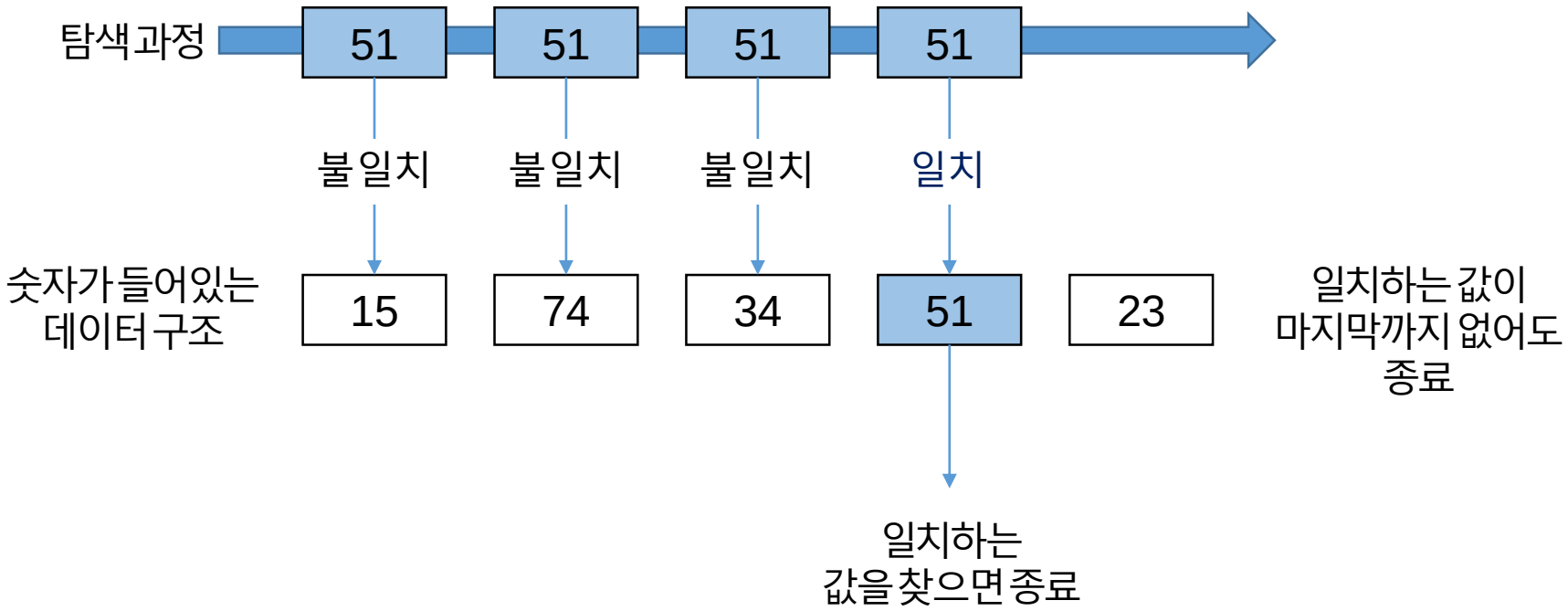
주어진 데이터를 일정한 기준에 따라 정렬하는 문제





실습1: 순차탐색

- 문제: 주어진 데이터 구조 안에 원하는 값이 있는지 찾아내는 문제
- 전략: 맨 앞부터 순서대로 원하는 숫자를 찾아나감



실습1: 선형탐색 알고리즘

- 예시: 숫자로 이루어진 리스트 중 주어진 값의 위치를 반환하는 알고리즘

```
Linear Search (List, X)

If List[0]==X:
    return 0

Elif:
    list[1]==X:
    return 1

Elif:
    list[2]==X:
    return 2
…
Else:
    return “None was found”
```

실습1: 선형탐색 알고리즘의 유사코드

- 예시: 숫자로 이루어진 리스트 중 주어진 값의 위치를 반환하는 알고리즘

```
Linear Search ( Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

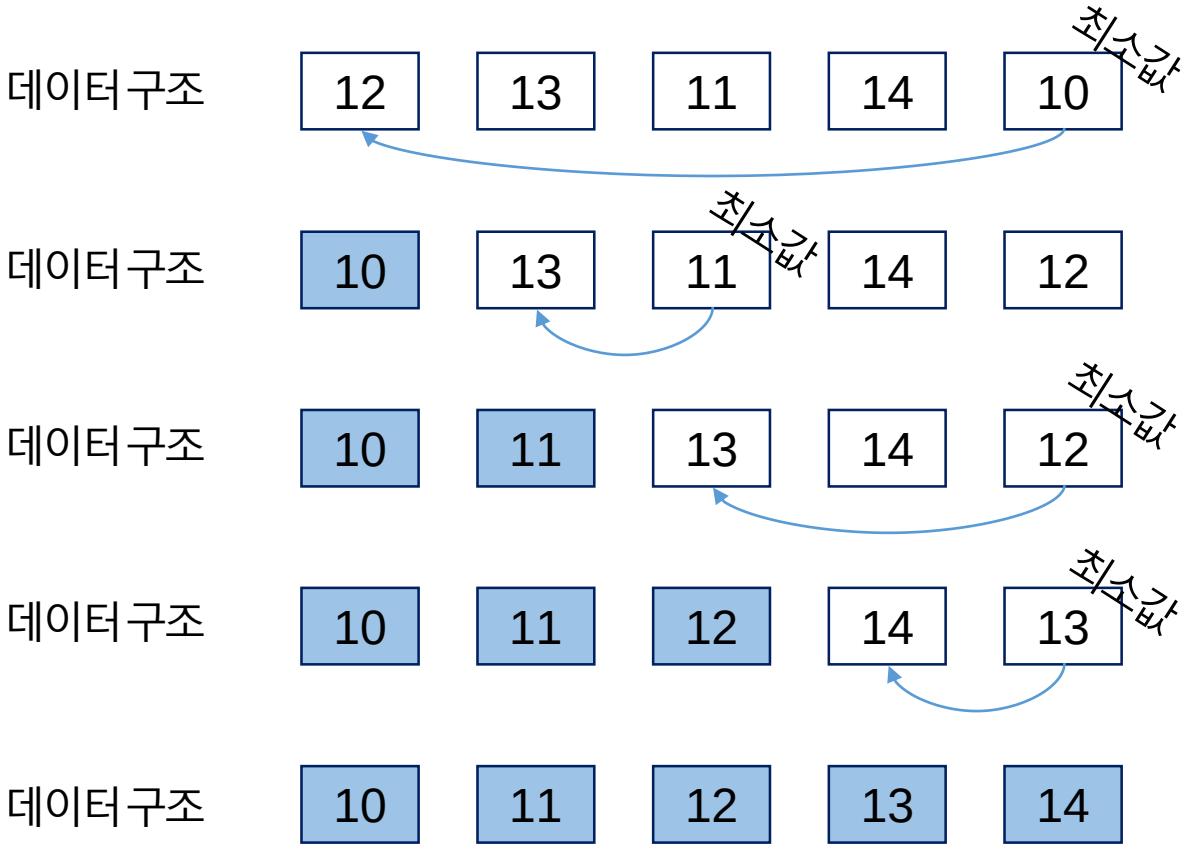
```
procedure linear_search (list, value)
   for each item in the list
      if match item == value
         return the item's location
      end if
   end for
   return “Element not found”
end procedure
```

실습1

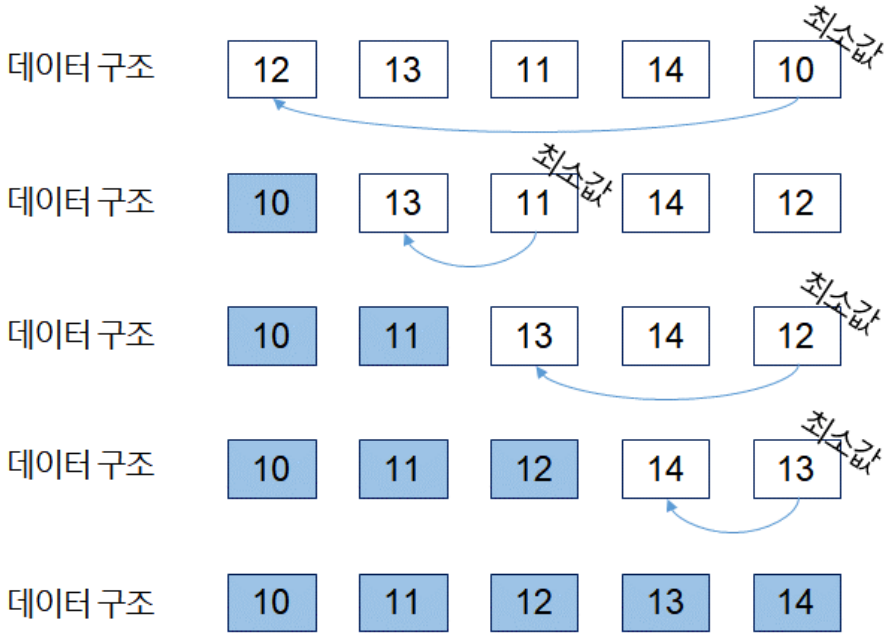
- 선형탐색 알고리즘을 파이썬 코드로 설계해 보시오

실습2: 선택정렬

- 문제: 주어진 데이터를 일정한 기준에 따라 정렬하는 문제
- 전략: 가장 작은 값을 선택하여 맨 앞부터 순서대로 정렬해나간다.

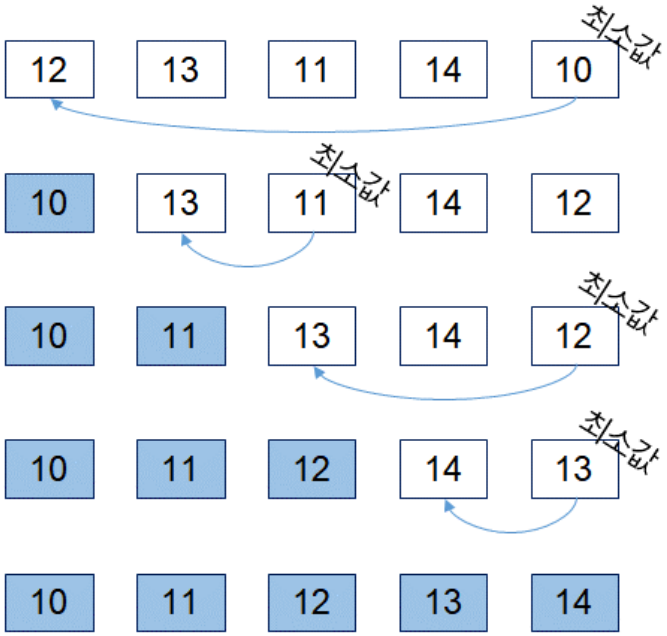


실습2: 선택정렬 알고리즘




```
Step 1 – Set POS to location 0
Step 2 – Search the minimum element in the list
Step 3 – Swap with value at location POS
Step 4 – Increment POS to point to next element
Step 5 – Repeat until list is sorted
```

실습2: 선택정렬 - 유사코드

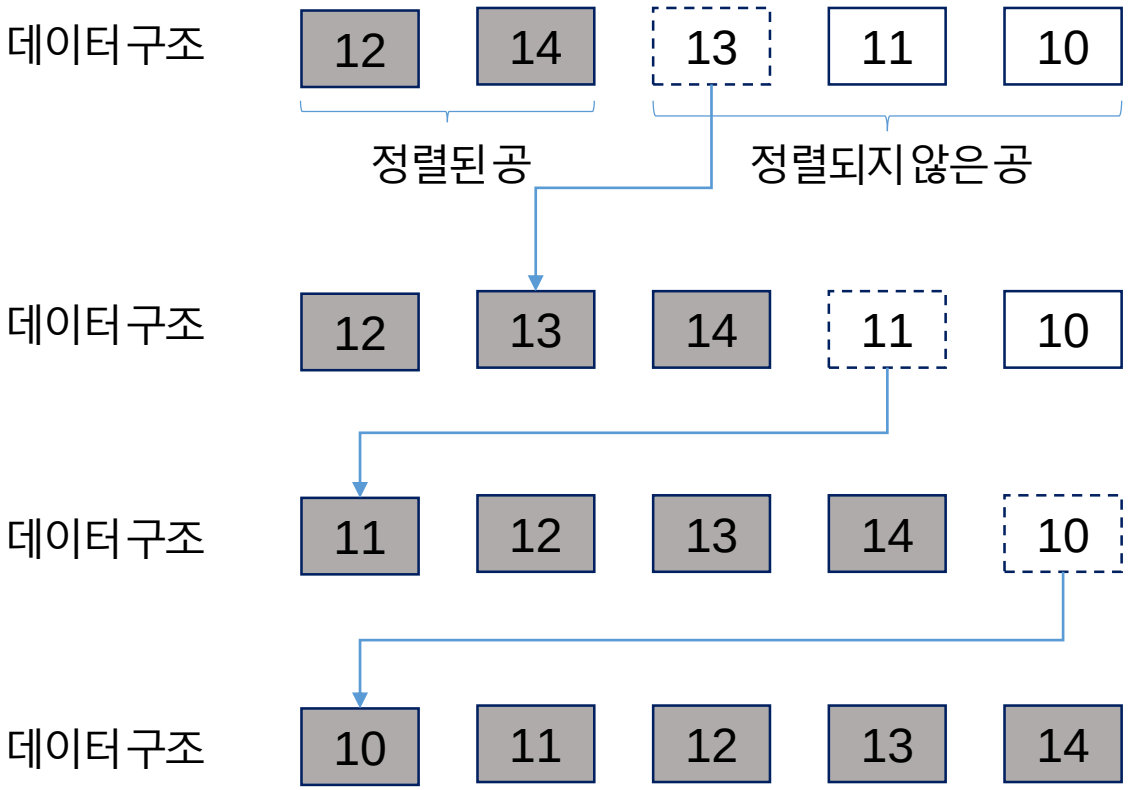



```
LIST=[12,13,11,14]
SORTED=[10]

LIST=[12,13,14]
SORTED=[10,11]

LIST=[13,14]
SORTED=[10,11,12]
..
```

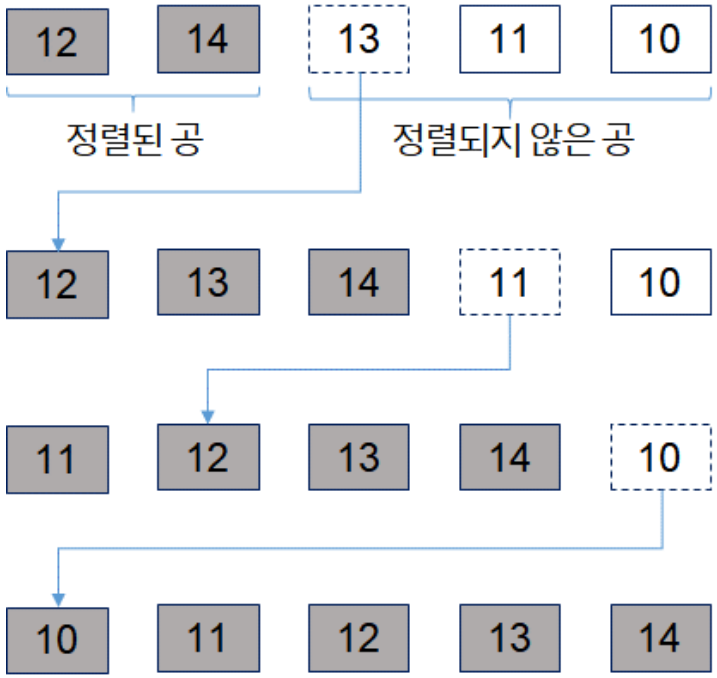
```
WHILE LIST is not EMPTY
    FIND MinNum from LIST
    REMOVE MinNum from List
    ADD MinNum to SORTED
RETURN SORTED
```

실습3: 삽입정렬

- 문제: 주어진 데이터를 일정한 기준에 따라 정렬하는 문제
- 전략: 값을 하나씩 올바른 위치에 삽입 해나감



실습3: 삽입정렬




```
[],[12,14,13,11,10]
[12], [14,13,11,10]
[12,14], [13,11,10]
[12,13,14], [11,10]
[11,12,13,14], [10]
[10,11,12,13,14] []
```

```
WHILE LIST is EMPTY
    Select element x from LIST
    Remove x from LIST
    Shift all elements of SORTED that is greater
than X
```

실습3: 삽입정렬

1) 기존 리스트와 새로운 숫자를 비교하여, 다음처럼 새로운 값보다 큰값들은 모두 오른쪽으로 밀어서 새로운 리스트를 만들어 보시오 (hint: python list내 insert 객체)

입력

출력

실습3: 파이썬 코드

실습4: 동명이인 찾기 알고리즘

- 주어진 리스트에서 동명이인이 있으면 모두 출력하는 알고리즘을 설계해 보시오. (힌트 딕셔너리 사용)

- Ex)

  ["Tom","Jerry","Mike","Tom"]

  →["Tom“]

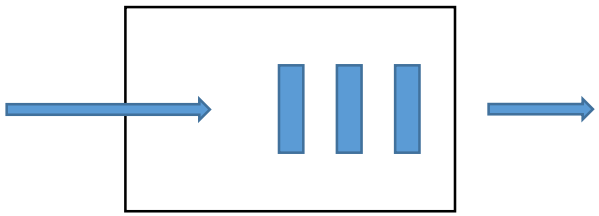
  ["Tom","Jerry","Mike","Tom”, “Mike”]

  →[”Tom”, “Mike”]

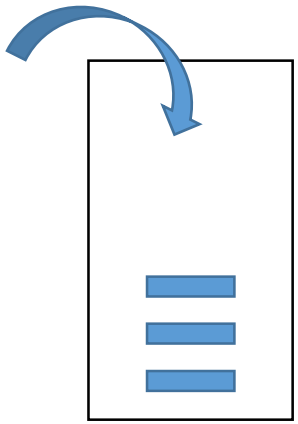
자료구조

- 알고리즘 계산을 위해 고안한 특별한 자료저장 구조

큐(queue):
First-in-first-out



스택(stack): last-in-first-out



실습5: 친구의 친구 찾기 알고리즘

- 다음은 친구들간의 Social Network를 나타내는 딕셔너리이다.
- 친구의 친구도 친구가 될 수 있다고 할때, 나의 모든 친구들을 (친구의 친구를 모두 연결) 찾아내는 알고리즘을 생각해 볼 것

```
fr_info={
    'Summer':['John','Justin','Mike'],
    'John':['Summer','Justin'],
    'Justin':['John','Summer','Mike','May'],
    'Mike':['Summer','Justin'],
    'May':['Justin','Kim'],
    'Kim':['May'],
    'Tom':['Jerry']
}
```

- 힌트: 자료구조를 이용할 것

자료구조를 이용한 알고리즘

```
fr_info={
    'Summer':['John','Justin','Mike'],
    'John':['Summer','Justin'],
    'Justin':['John','Summer','Mike','May'],
    'Mike':['Summer','Justin'],
    'May':['Justin','Kim'],
    'Kim':['May'],
    'Tom':['Jerry']
}
```

Queue	Friends
[Summer]	
[John, Justin, Mike]	[Summer]
[~~Summer,~~ ~~Justin~~, John, Justin]	[Mike, Summer]
[~~John, Summer, Mike, May~~, May, John]	[Justin, Mike, Summer]
[~~Summer, Justin~~, May]	[John, Justin, Mike, Summer]
[Kim]	[May, John, Justin, Mike, Summer]
[~~May~~]	[Kim, May, John, Justin, Mike, Summer]

의사코드

- 입력값 (친구목록, A)
- Queue에 A의 친구들 추가
- Queue에 친구들이 없어질 때 까지 다음을 반복
  - Queue에서 친구를 하나 꺼내어 Friends에 추가
  - 해당 친구의 친구들 목록에 찾아서 Queue에 추가
    - 다만, 이미 친구목록에 있거나, 본인(A)인 경우에는 제외.

파이썬 코드

실습6: 팰린드롬(회문) 찾기

- "토마토"나 "기러기"처럼 거꾸로 읽어도 똑같은 단어를 팰린드롬(palindrome)이라고 부릅니다. 문자열 word가 팰린드롬인지 확인하는 함수 is_palindrome를 쓰세요. is_palindrome은 word가 팰린드롬이면 True를, 팰린드롬이 아니면 False를 리턴합니다.

```
def is_palindrome(word):
        # 코드를 입력하세요.

    # 테스트
print(is_palindrome("racecar"))
print(is_palindrome("stars"))
print(is_palindrome("토마토"))
print(is_palindrome("kayak"))
print(is_palindrome("hello"))
```

```
True
False
True
True
False
```

주의 사항

- 반드시 for문을 사용하셔야 합니다.
- **append**, **insert** 메소드와 **del** 함수를 사용하면 안됩니다.
- 자동 채점 과제이기 때문에, 문제의 조건에 정확히 따라주시기 바랍니다. 띄어쓰기도 일치해야 합니다.