[TA 2-2]

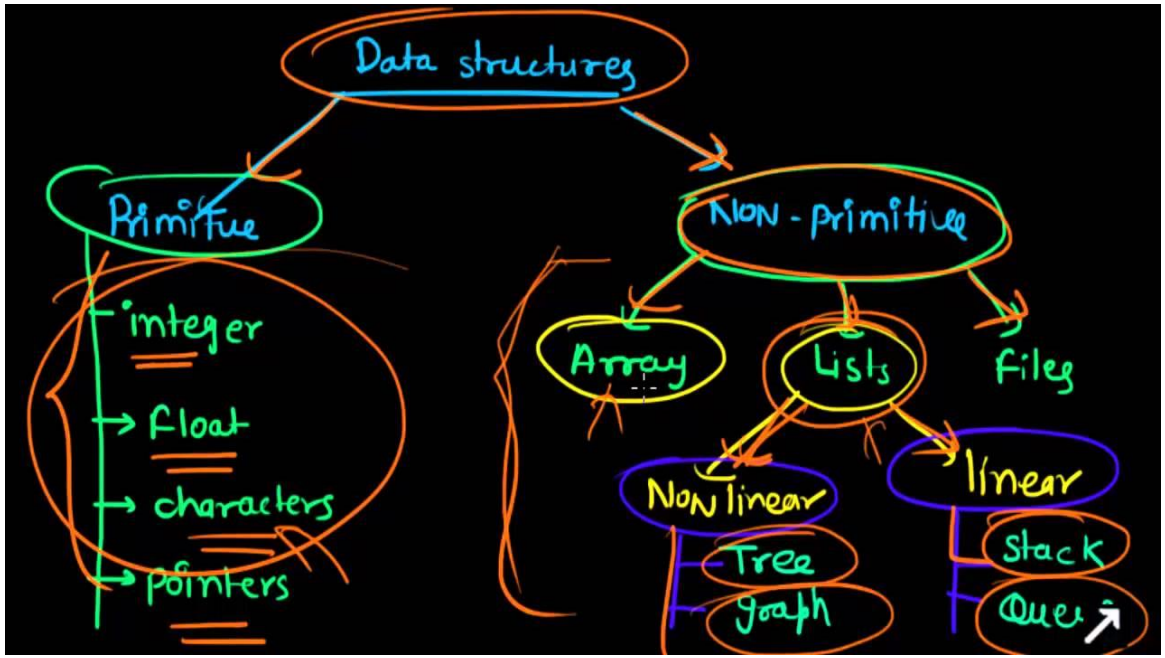
Data Structure and Algorithms - Basics

빅데이터 프로그래밍 기초

박진수 교수

Big Data Institute,
Seoul National University

자료구조와 알고리즘


Data structures
Primitive
Non-primitive
integer
float
characters
pointers
Array
Lists
Files
Non linear
linear
Tree
Graph
Stack
Queue

자료구조와 알고리즘

· 자료 구조(Data Structure)

- 자료가 효율적으로 활용될 수 있도록 컴퓨터 내에 체계적으로 구조화하는 방법

- 대용량의 데이터베이스나 검색 쿼리 등 대규모의 데이터 관리를 효율적으로 이루어지게 함

- 리스트(lilst), 스택(stack), 큐(queue), 트리(tree), 그래프(graph) 등

자료구조와 알고리즘

· 자료형(Data Type)

- 값(value)의 형태

■ 논리 자료형(Boolean)

■ 수치 자료형(Integer, float, long)

■ 문자열 자료형(String, char)

■ 배열 자료형(Array)

자료구조와 알고리즘

· 추상 자료형(Abstract Data Type)

- "소프트웨어(software)"로서의 자료형의 구현

- 자료들과 그 자료들에 대한 연산을 표현한 것

- "a logical description of how we view the data and the operations that are allowed without regard to how they will be implemented"

- 캡슐화(encapsulation): (코드를 통해) 명시적으로 구현하지 않음

■ 계산기 ADT

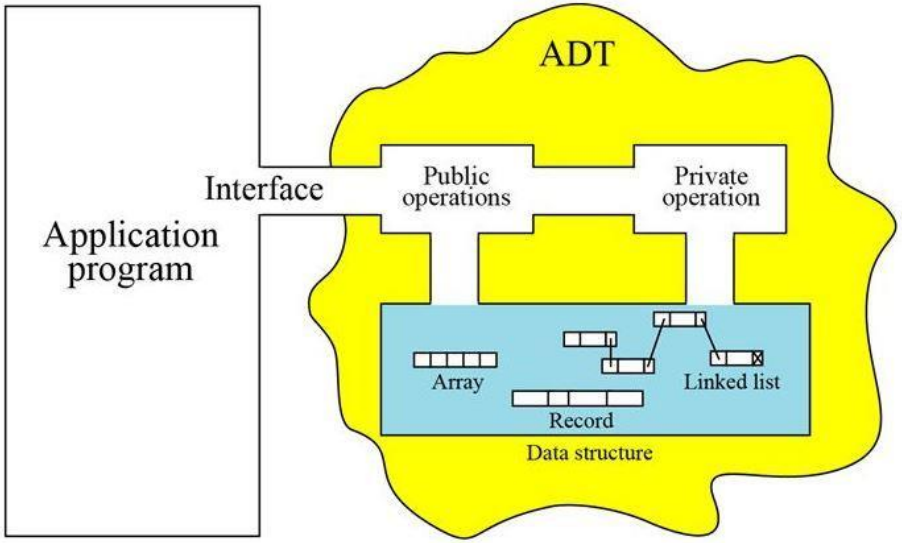
```
def add(v1, v2):
    # v1과 v2를 더한 값을 리턴
def subtract(v1, v2):
    # v1과 v2 중 큰 값에서 작은 값을 뺀 값을 리턴
def multiply(v1, v2):
    # v1과 v2를 곱한 값을 리턴
def divide(v1, v2):
    # v1과 v2 중에서 큰 값에서 작은 값을 나눈 값을 리턴
    # v1과 v2가 둘 다 0일 경우 none 리턴
```

자료구조와 알고리즘

· 자료 구조와 추상 자료형

- 자료 구조는 추상 자료형을 코드로 구현(implement)한 것

- “an ADT sits on top of the data structure, and the rules that govern the use of the data define **the interface between the data structure and the ADT**”



자료구조와 알고리즘

· 알고리즘(Algorithms)

- 단계적으로 수행되는 여러 작업들의 모음

- 연산, 데이터 처리, 최적화 등을 위한 다양한 목적을 위한 수많은 알고리즘이 존재한다

- 알고리즘은 다음의 조건들을 만족하여야 한다

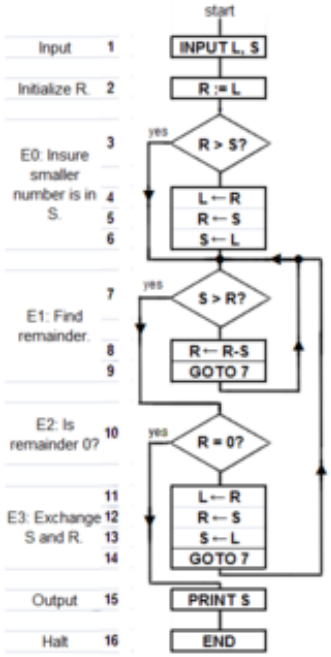
  ■ 입력: 외부에서 제공되는 자료가 0개 이상 존재한다

  ■ 출력: 적어도 1개 이상의 서로 다른 결과를 내어야 한다
  (즉, 모든 입력에 하나의 출력이 나오면 안됨)

  ■ 명확성: 수행 과정은 명확하고 모호하지 않은 명령어로 구성되어야 한다

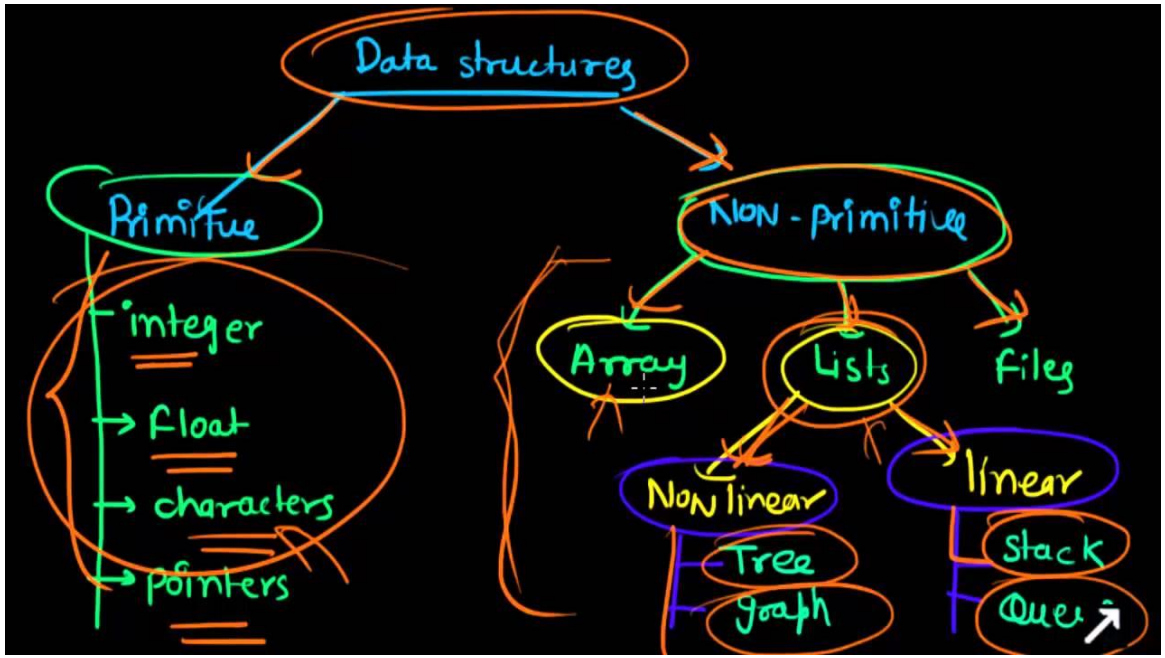
  ■ 유한성: 유한 번의 명령어를 수행 후에 종료한다

  ■ 효율성: 모든 과정은 명백하게 실행(검증) 가능한 것이어야 한다



"Inelegant"

스택과 큐 자료구조

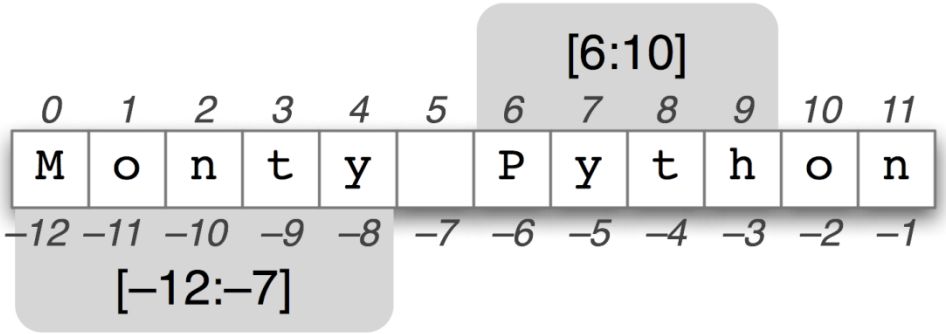

Data structures
Primitive
Non-primitive
integer
float
characters
pointers
Array
Lists
Files
Non linear
linear
Tree
Graph
Stack

스택(Stack)과 큐(Queue)

· 선형 자료 구조(Linear Data Structures)

- 요소(item)들이 어떻게 추가되거나 삭제되었는지에 따라 순서(상대적 위치)가 정해지는 자료 구조

- 리스트, 스택, 큐 등



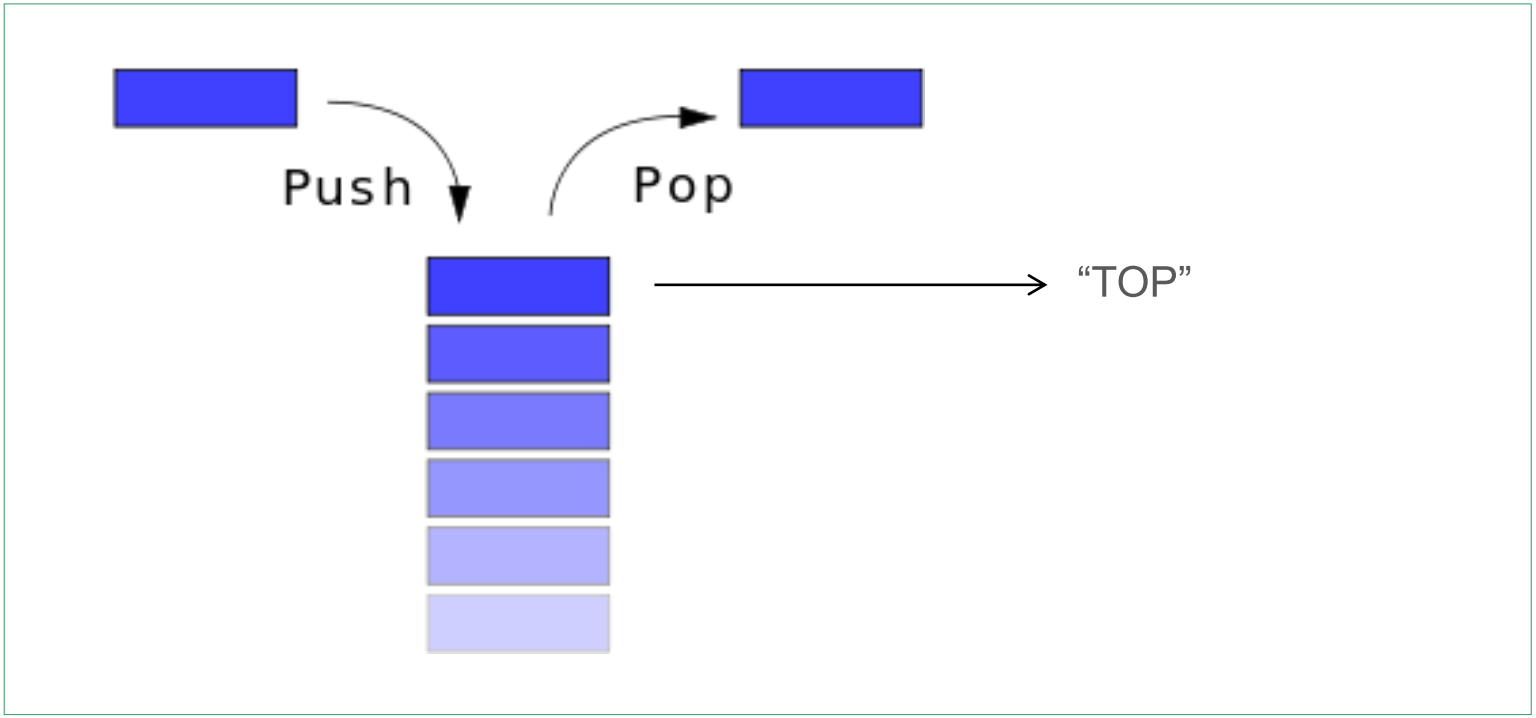
- 스택과 큐: 리스트와 거의 유사하나 **입력과 출력 방법을 단순화해** 효율을 극대화시킨 것

스택(Stack)과 큐(Queue)

· 스택(Stack)

- 데이터의 추가와 삭제가 동일한 위치(top)에서만 이루어지는 자료구조

- 후입선출법(LIFO; Last In First Out)



스택(Stack)과 큐(Queue)

· 스택 추상 자료형(Stack ADT)

- is_empty(STACK): 스택이 비었는지 아닌지를 검사
  - ■ 비었으면 False를, 아니면 True를 반환한다
- push(STACK, item): 스택의 top 위치에 요소를 추가한다
- pop(STACK): 스택의 top 위치에 있는 요소를 제거한다
  - ■ 제거한 요소의 값을 반환한다
- top_value(STACK): 스택의 top 위치에 있는 요소를 반환한다
  - ■ 요소를 제거하지 않고 반환만 한다
- get_size(STACK): 스택의 사이즈(요소의 개수)를 반환한다

스택(Stack)과 큐(Queue)

· 실습 2-2-1. 스택 자료구조 구현

- 스택 ADT를 참고하여 스택 자료구조를 구현해본다

■ 리스트로 스택을 생성하고 각 함수를 정의한다

■ 수행 예시

In [4]:
```
stack = [1, 2, 3, 4, 5]
print(is_empty(stack))
push(stack, 6)
print(stack)
pop(stack)
print(stack)
print(top_value(stack))
print(get_size(stack))
```

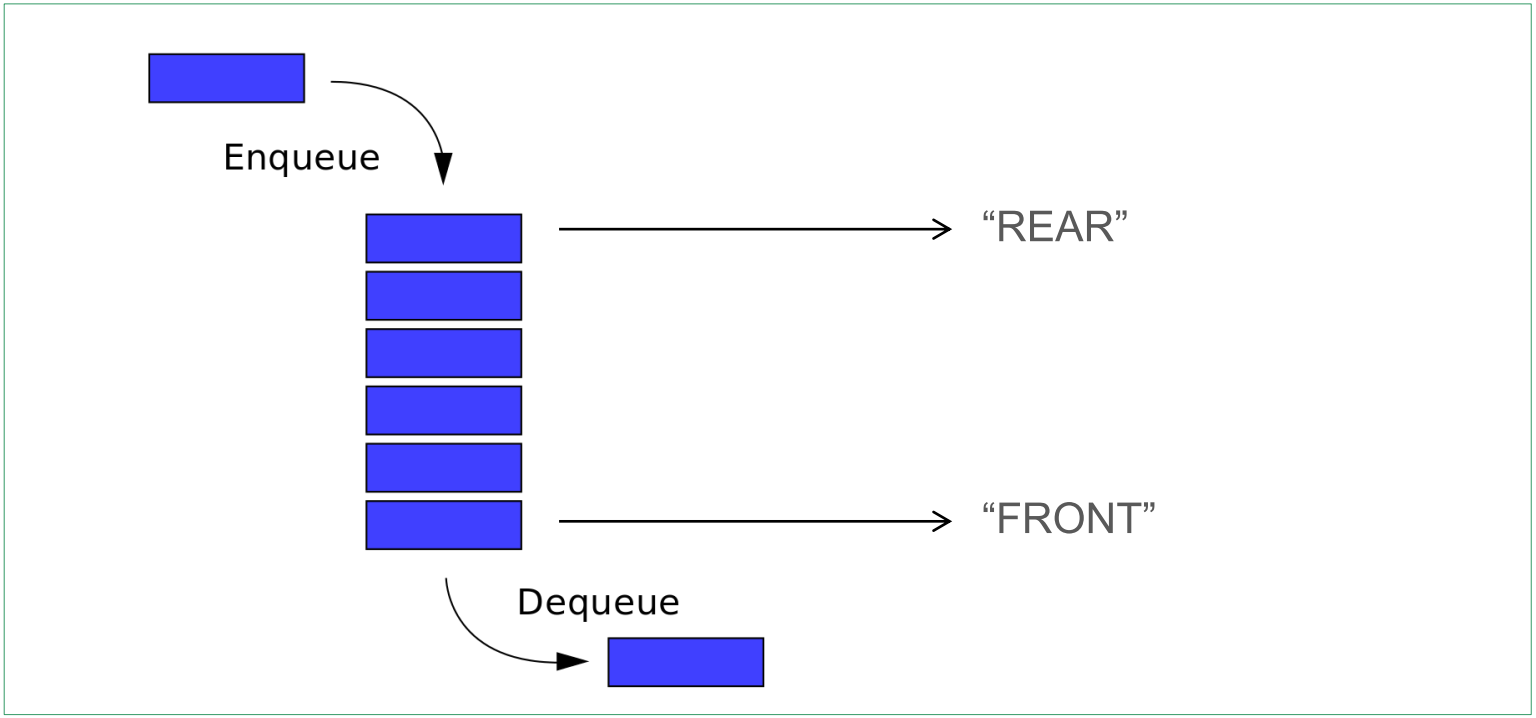
```
True
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5]
5
5
```

스택(Stack)과 큐(Queue)

· 큐(Queue)

- 데이터의 추가와 삭제가 이루어지는 위치가 서로 다르지만 일정하게 유지되는 자료구조

- 선입선출법(FIFO; First In First Out)



스택(Stack)과 큐(Queue)

· 큐 추상 자료형(Queue ADT)

- is_empty(QUEUE): 큐가 비었는지 아닌지를 검사
  - ■ 비었으면 False를, 아니면 True를 반환한다
- enqueue(QUEUE, item): 큐의 rear 위치에 요소를 추가한다
- dequeue(QUEUE): 큐의 front 위치에 있는 요소를 제거한다
  - ■ 제거한 요소의 값을 반환한다
- get_size(QUEUE): 스택의 사이즈(요소의 개수)를 반환한다

스택(Stack)과 큐(Queue)

· 실습 2-2-2. 큐 자료구조 구현

- 큐 ADT를 참고하여 큐 자료구조를 구현해본다

■ 리스트로 큐를 생성하고 각 함수를 정의한다

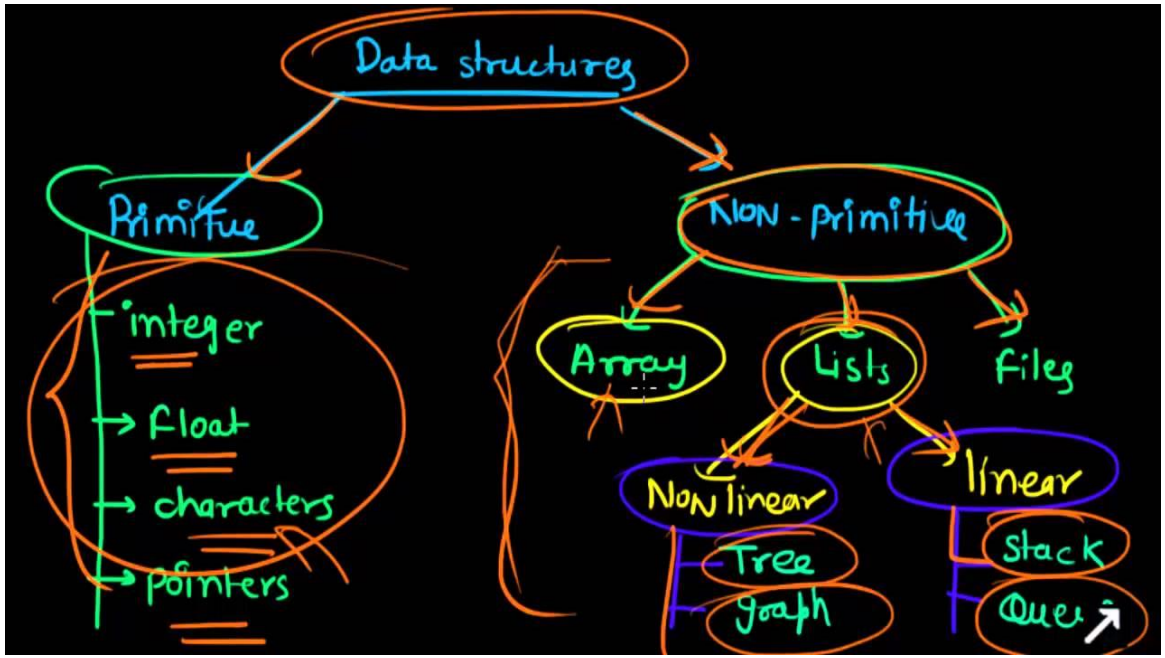
■ 수행 예시

```
In [16]: queue = [1, 2, 3, 4, 5]
         print(is_empty(queue))
         enqueue(queue, 6)
         print(queue)
         dequeue(queue)
         print(queue)
         print(get_size(queue))

         True
         [1, 2, 3, 4, 5, 6]
         [2, 3, 4, 5, 6]
         5
```

■ front와 rear를 어디로 설정할 것인가?! (리스트의 끝 혹은 시작)

정렬 알고리즘


Data structures
Primitive
Non-primitive
integer
float
characters
pointers
Array
Lists
Files
Non linear
linear
Tree
Graph
Stack
Queue

정렬 알고리즘(Sorting algorithm)

· 정렬(sorting): 컴퓨터 작업 중 가장 많이 수행되는 작업 중 하나

- 정렬을 효율적으로 수행하기 위한 수많은 알고리즘이 고안됨

  ■ insertion, bubble, selection, quick, shell, merge, …

- 파이썬 리스트의 sort() 함수를 생각하면 이해가 쉽다

```
In [25]: l = [9, 4, 8, 2, 3, 5, 1]
         l.sort()
         print(l)

         [1, 2, 3, 4, 5, 8, 9]
```

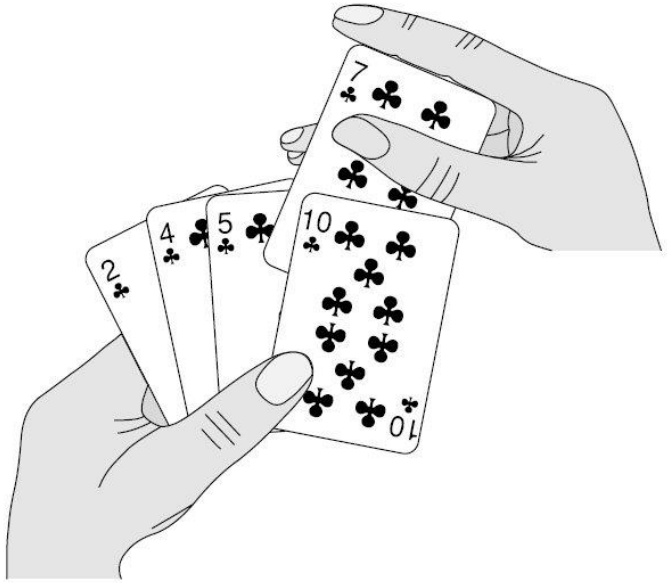
- 정렬 알고리즘을 익히고 그 장단점을 파악하는 것은 알고리즘 디자인과 분석을 종합적으로 이해하는데 큰 도움이 된다

정렬 알고리즘(Sorting algorithm)

· 삽입정렬

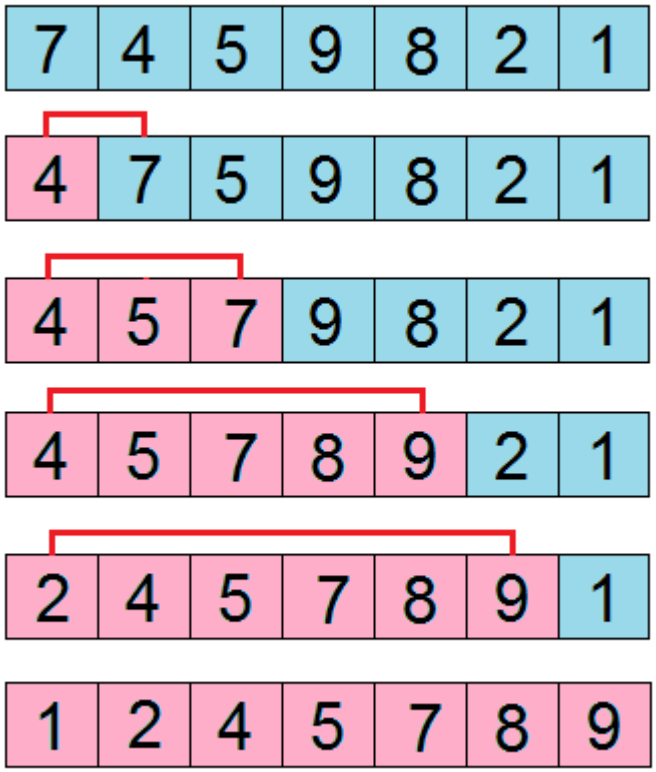
- 가장 직관적이고 간단한 알고리즘

- i를 1부터 n-1(n은 원소의 개수)까지 1씩 증가시키면서 첫 (i+1)개의 원소를 정렬해 나간다



정렬 알고리즘(Sorting algorithm)

· 삽입정렬 예시



정렬 알고리즘(Sorting algorithm)

· 실습 2-2-3. 삽입정렬 구현

- 리스트를 입력으로 받아 정렬된 리스트를 리턴해주는 insertion_sort() 함수를 구현한다

■ 삽입정렬을 구현하는 데에는 수많은 방법이 있다 – 가장 효율적인 방법을 창의적으로 생각해 본다

■ 수행 예시

```
In [28]: insertion_sort([9, 4, 8, 2, 3, 5, 1])
Out[28]: [1, 2, 3, 4, 5, 8, 9]
```

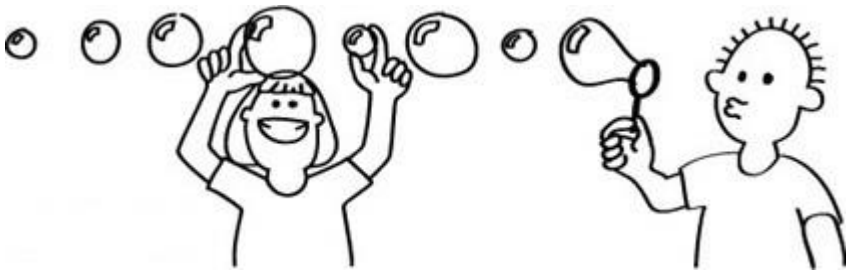
정렬 알고리즘(Sorting algorithm)

· 거품정렬

- 두 인접한 원소를 검사하여 정렬하는 방법

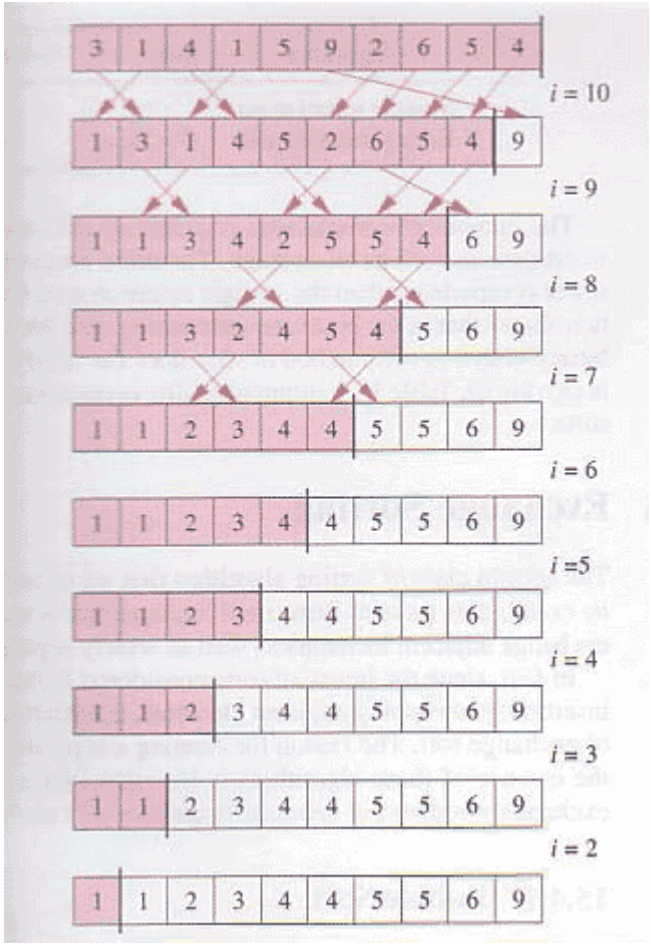
- 상당히 느린 정렬이라 현실에서 사용되는 일은 거의 없다

- i를 1부터 n(n은 원소의 개수)까지 증가시키며 가장 작은 i개 원소를 제 자리를 찾아가게 한다



정렬 알고리즘(Sorting algorithm)

· 거품정렬 예시



정렬 알고리즘(Sorting algorithm)

· 실습 2-2-4. 거품정렬 구현

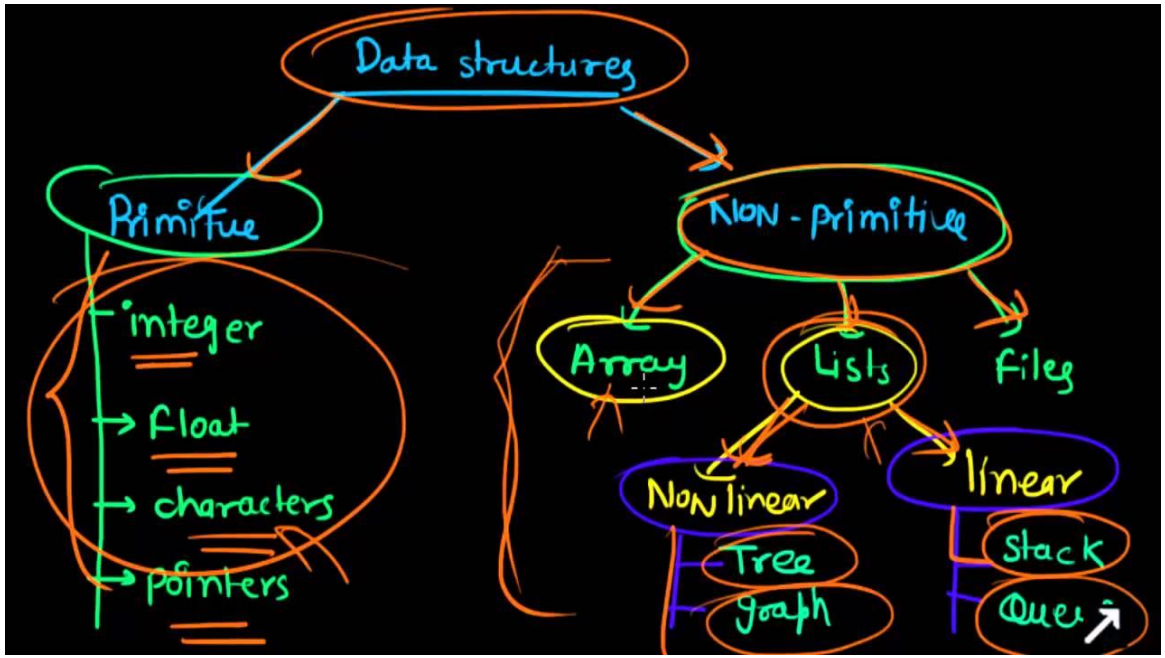
- 리스트를 입력으로 받아 정렬된 리스트를 리턴해주는 bubble_sort() 함수를 구현한다

■ 거품정렬을 구현하는 데에는 수많은 방법이 있다 – 가장 효율적인 방법을 창의적으로 생각해 본다

■ 수행 예시

```
In [33]: bubble_sort([9, 4, 8, 2, 3, 5, 1])
Out[33]: [1, 2, 3, 4, 5, 8, 9]
```

이진 탐색 알고리즘


Data structures
Primitive
Non-primitive
integer
float
characters
pointers
Array
Lists
Files
Non linear
linear
Tree
Graph
Stack

탐색 알고리즘(Search algorithm)

· 탐색(searching): 탐색 범위(search space) 내에서 특정 정보를 골라 뽑아내기 위한 알고리즘

- 예시: 드라이브에서 특정 이름을 가진 파일 혹은 폴더를 검색

· 탐색의 엄밀한 정의

- n개의 기록을 가지는 탐색 범위 $L = \{(k_1, I_1), \dots, (k_n, I_n)\}$에 대하여, $k_j = K$를 만족하는 기록 (k_j, I_j)를 찾는 것
- 정확히 일치 쿼리(exact query): 특정 키 값과 일치하는 기록을 탐색
- 범위 쿼리(range query): 키 값이 어느 범위에 들어가는 기록을 탐색

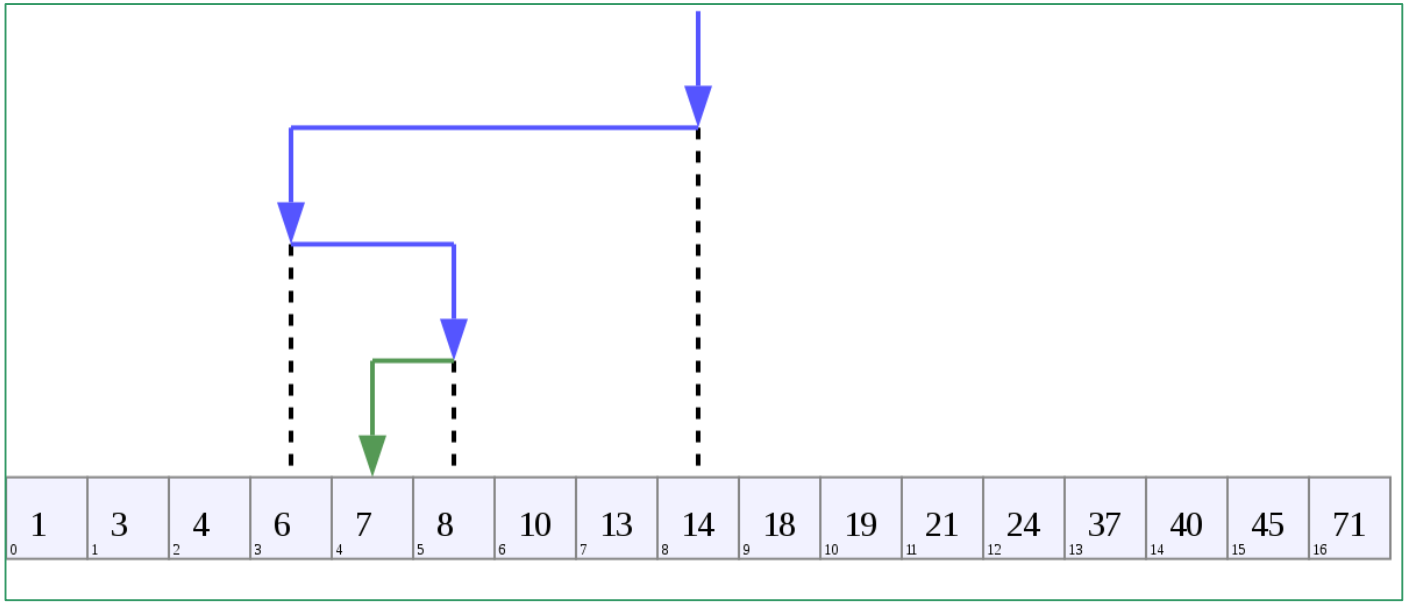
탐색 알고리즘(Search algorithm)

· 이진 탐색(Binary search algorithm)

- 정렬된 리스트(sorted list)에서 특정한 값의 위치를 찾는 알고리즘

- 처음에 중간의 값을 선택하여, 그 값과 찾고자 하는 값의 크고 작음을 비교하는 작업을 재귀적으로 반복

- 중간의 값이 찾고자 하는 값보다 크면 앞의 반으로, 작으면 뒤의 반으로 가서 탐색을 다시 수행한다



탐색 알고리즘(Search algorithm)

· 실습 2-2-5. 이진탐색 알고리즘 구현

- 정렬된 리스트와 특정 값을 입력 받아 그 값이 리스트에서 위치한 인덱스를 반환해주는 binary_search() 함수를 구현한다

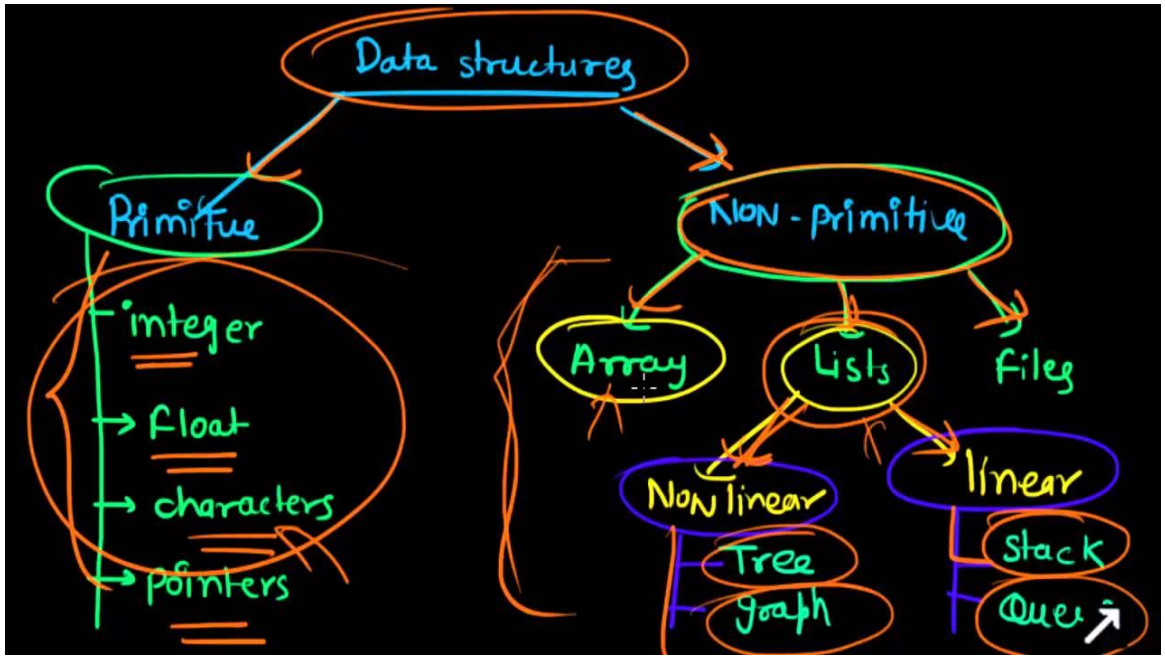
- 그 값이 리스트에 없을 경우 None을 반환한다

- 수행 예시

In [6]:
```
# 이진 탐색의 인풋은 정렬된 리스트여야 함
print(binary_search([1,5,8,10], 5))
print(binary_search([1,5,8,10], 0))
print(binary_search([1,2,9,10,12,15,19], 9))
print(binary_search([1,2,9,10,12,15,19], 20))
```

```
1
None
2
None
```

그래프(graphs)

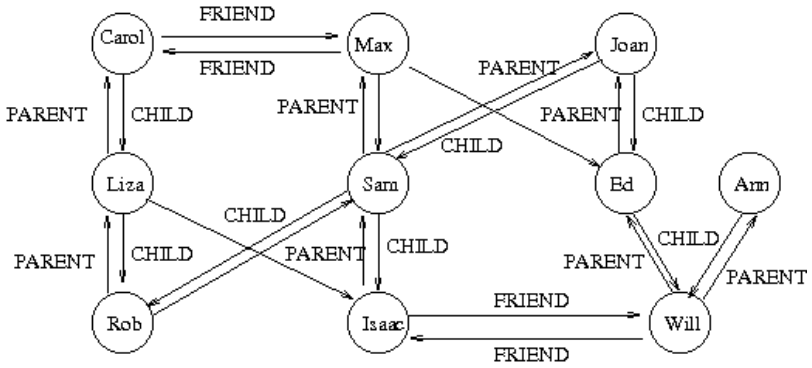

Data structures
Primitive
Non-primitive
integer
float
characters
pointers
Array
Lists
Files
Non linear
linear
Tree
Graph
Stack
Queue

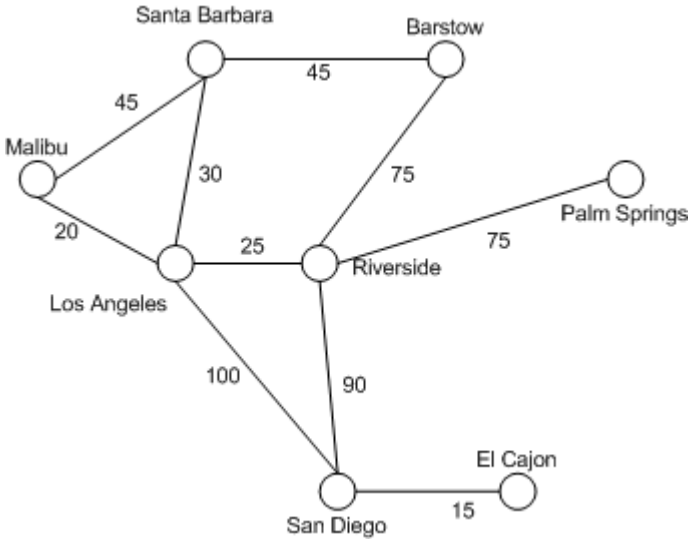
그래프(graphs)

· 각각의 모서리(E; edge or link)가 꼭지점(V; vertex or node) 한 쌍을 연결하는 꼭지점 V의 집합과 모서리 E의 집합을 칭한다

- $G = (V, E)$

· 구조 내의 일부 객체들 간의 “관계(relationship)”을 설명하기 위한 목적으로 주로 활용됨





그래프(graphs)

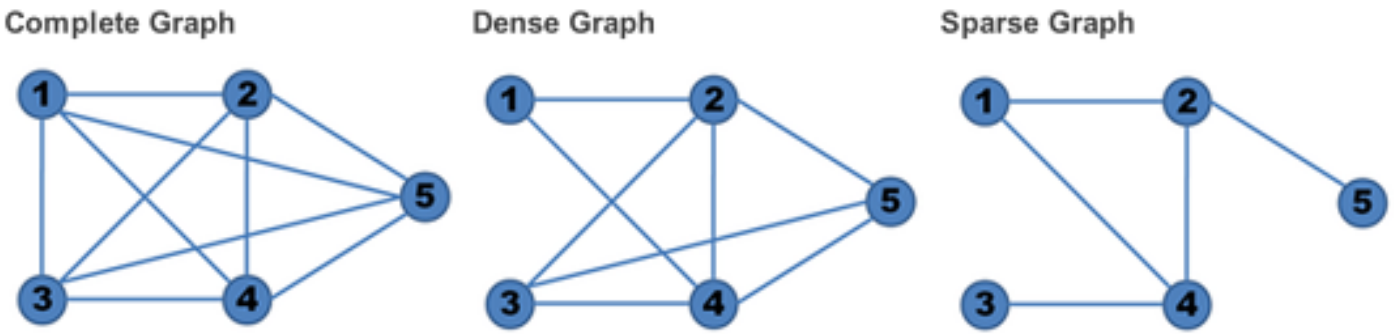
· 그래프의 분류

- 모서리의 개수에 따라

■ 성긴 그래프(sparse graph): 모서리가 상대적으로 적은 그래프

■ 빽빽한 그래프(dense graph): 모서리가 상대적으로 많은 그래프

■ 완전한 그래프(complete graph): 모서리가 꽉 찬 그래프(모든 꼭지점 쌍에 모서리가 존재)



그래프(graphs)

· 그래프의 분류

- 모서리의 방향 유무에 따라

■ 방향 있는 그래프(directed graph): 한 꼭지점에서 다른 꼭지점으로의 모서리의 방향이 있는 그래프

■ 방향 없는 그래프(undirected graph): 모든 모서리의 방향이 없는 그래프

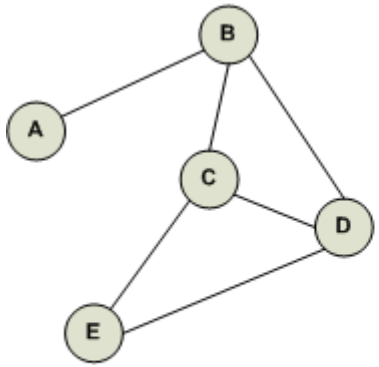


Fig 1. Undirected Graph

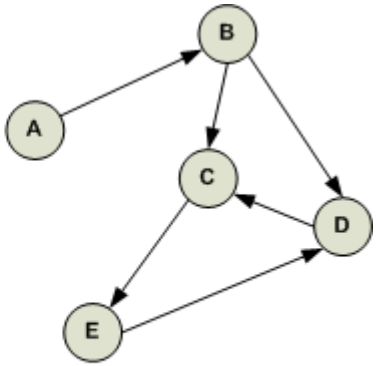


Fig 2. Directed Graph

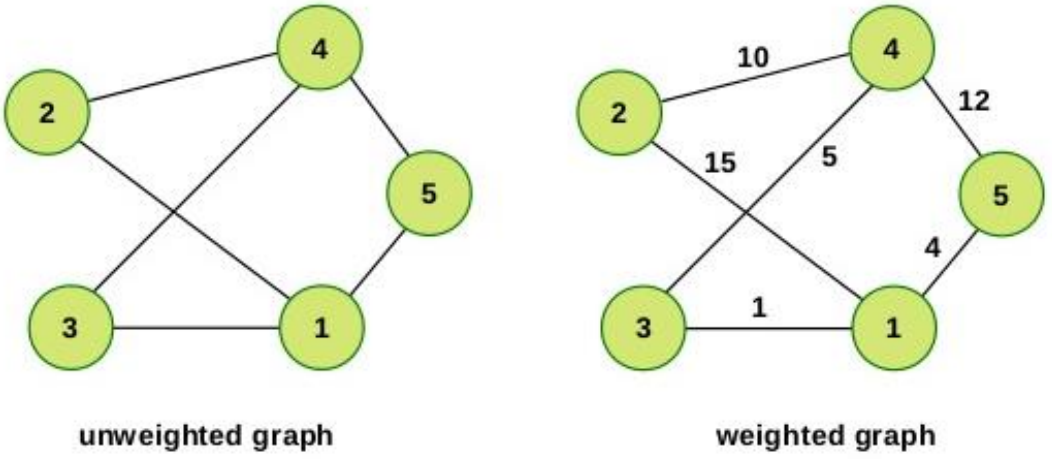
그래프(graphs)

· 그래프의 분류

- 모서리 가중치(weight)의 유무에 따라

■ 가중 그래프(weighted graph): 모서리에 가중치가 있는 그래프

■ 무가중 그래프(unweighted graph): 모서리에 가중치가 없는 그래프



그래프(graphs)

· Python으로 그래프 그리기

- Python의 networkx 모듈을 활용하면 그래프를 쉽게 그릴 수 있다

■ 그래프 생성하기

In [14]:
```
%matplotlib inline

import networkx as nx
import matplotlib.pyplot as plt
G = nx.Graph()
```

■ 꼭지점 더하기

In [15]:
```
G.add_node('Tony')
```

In [16]:
```
G.add_nodes_from(['Johnny', 'Jake', 'Jane', 'Eva', 'Patrick'])
```

그래프(graphs)

· Python으로 그래프 그리기

- Python의 networkx 모듈을 활용하면 그래프를 쉽게 그릴 수 있다

■ 모서리 더하기

In [17]:
```
G.add_edge('Tony', 'Jane')
```

In [18]:
```
G.add_edges_from([('Johnny', 'Jane'),('Eva', 'Patrick'),('Jake', 'Patrick')])
```

■ 레이블 생성하기

In [25]:
```
label_dict = {}
for node in G.nodes():
    label_dict[node] = node
```

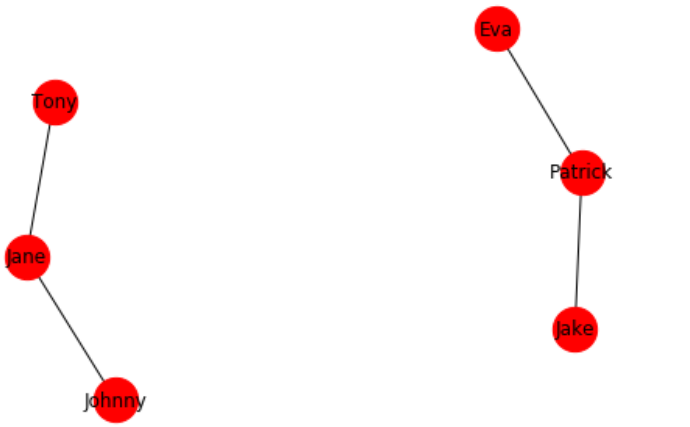
그래프(graphs)

· Python으로 그래프 그리기

- Python의 networkx 모듈을 활용하면 그래프를 쉽게 그릴 수 있다

■ 그래프 그리기

```
In [26]: nx.draw(G, node_size = 800, labels = label_dict, with_lables = True)
```



그래프(graphs)

· 실습 2-2-6. 그래프 그리기

- 아래 데이터를 바탕으로, 미국 주요 도시 간 거리를 표현하는 그래프를 만들어 보자

■ Python의 networkx 패키지를 활용한다

	Atlanta	Boston	Chicago	Dallas	Denver	Houston	Las Vegas	Los Angeles	Miami	New Orleans	New York	Phoenix	San Francisco	Seattle	Washington
Atlanta		1095	715	805	1437	844	1920	2230	675	499	884	1832	2537	2730	657
Boston	1095		983	1815	1991	1886	2500	3036	1539	1541	213	2664	3179	3043	44
Chicago	715	983		931	1050	1092	1500	2112	1390	947	840	1729	2212	2052	695
Dallas	805	1815	931		801	242	1150	1425	1332	504	1604	1027	1765	2122	1372
Denver	1437	1991	1050	801		1032	885	1174	2094	1305	1780	836	1266	1373	1635
Houston	844	1886	1092	242	1032		1525	1556	1237	365	1675	1158	1958	2348	1443
Las Vegas	1920	2500	1500	1150	885	1525		289	2640	1805	2486	294	573	1188	2568
Los Angeles	2230	3036	2112	1425	1174	1556	289		2757	1921	2825	398	403	1150	2680
Miami	675	1539	1390	1332	2094	1237	2640	2757		892	1328	2359	3097	3389	1101
New Orleans	499	1541	947	504	1305	365	1805	1921	892		1330	1523	2269	2626	1098
New York	884	213	840	1604	1780	1675	2486	2825	1328	1330		2442	3036	2900	229
Phoenix	1832	2664	1729	1027	836	1158	294	398	2359	1523	2442		800	1482	2278
San Francisco	2537	3179	2212	1765	1266	1958	573	403	3097	2269	3036	800		817	2864
Seattle	2730	3043	2052	2122	1373	2348	1188	1150	3389	2626	2900	1482	817		2755
Washington D.C.	657	440	695	1372	1635	1443	2568	2680	1101	1098	229	2278	2864	2755	

그래프(graphs)

· 실습 2-2-6. 그래프 그리기

- 수행 예시: 아래의 그래프와 비슷한 형태의 그래프를 그려 본다