

Fastcampus

Frontend Dev SCHOOL

Algorithm & Data structure

Recap

- Computer? Calculator?
- CPU, Memory, OS
- Computational Thinking

Goal

- 의사코드를 이해하고 본 코드를 작성하기 전 생각의 흐름을 의사코드로 구현할 수 있다.
- 알고리즘에 대해 이해할 수 있다.
- 자료구조에 대해 이해할 수 있다.

Pseudocode

Pseudocode

== 의사코드 != Doctorcode

So, pseudocode is,

가짜코드: 프로그램이나 알고리즘이 수행할 내용을 인간이 이해할 수 있는 언어로 표현하는 것

Let's make fizzbuzz

fizzbuzz

1부터 n 까지 반복하면서,

3의 배수는 "fizz"

5의 배수는 "buzz"

15의 배수는 "fizzbuzz"

나머지는 숫자

...

pseudocode는 프로그램을 설계할 때 밑그림의 역할을 하게 됩니다!

목적과 수행과정이 명확해 코드 수정과 분해가 편리합니다!

pseudocode가 comment의 역할을 수행할 수 있습니다.

How to write pseudocode?

nonstandard..

자신이 작성할 언어의 스타일에 맞춰 작성하면 끝!

Let's go back to yesterday

I'm still hungry

1. hunger가 true가 됨
2. 현재위치가 집일때,
 1. 밥솥에 밥이 있다면, 해결한다.
 2. 굽는다.
3. 현재위치가 밖일때,
 1. 현금이 10만원 초과라면, 레스토랑을 간다.
 2. 현금이 10만원 이하라면, 편의점을 간다.

Let's make fizzbuzz again

fizzbuzz

1부터 n 까지 반복하면서,

3의 배수는 "fizz"

5의 배수는 "buzz"

15의 배수는 "fizzbuzz"

나머지는 숫자

| 영어로 작성하면 더 좋지만.. 한글로!

fizzbuzz-Kor

1. 사용자에게 정수 하나를 입력받아 num에 선언한다. i는 1임.
2. 1 부터 num까지 반복하면서
3. 만약 그 수(i)가 3의 배수라면 "fizz"를 출력
4. 만약 그 수가 5의 배수라면 "buzz"를 출력
5. 만약 그 수가 15의 배수라면 "fizzbuzz"를 출력
6. 3,4,5의 경우를 만족하지 못한 나머지 경우 그 수를 출력

fizzbuzz - Eng

1. `get integer from user ==> num, i == 1`
2. `WHILE i is less than or equal to num`
3. `if i is divisible by 3, print "fizz"`
4. `if i is divisible by 5, print "buzz"`
5. `if i is divisible by 15, print "fizzbuzz"`
6. `else, print i`

fizzbuzz - python

```
num = input("get number for fizzbuzz: ")
for i in range(1,100+1):
    if i % 3 == 0:
        print("fizz")
    elif i % 5 == 0:
        print("buzz")
    elif i % 15 == 0:
        print("fizzbuzz")
    else:
        print(i)
```

그럼, 부가가치세를 계산해봅시다.

VAT

- Korea: 10%
- Japan: 8%
- USA: 주마다 다름
- UK: 20%

제품 가격과 나라에 따라 다른 부가가치세를 계산해 그 가격을 보여주도록!

VAT - answer

1. **get** price of item ==> item_price
2. **set** tax rate (kor == **0.1**, jap == **0.08**, usa == "depend on state")
3. **get** country code(example: kor, jap, usa, uk) ==> country_code
4. tax_rate **is** matched price **with** country_code
5. sales tax **is** item_price times tax rate
5. total price **is** item_price plus sales tax

Data Structure & Algorithm

Algorithm

Algorithm

Algorithm is a series of contained steps which you follow in order to achieve some goal, or to produce some output

목표를 달성하거나 결과물을 생산하기 위해 필요한 과정들

1부터 100까지 더해보아요

누군가는

$$1+2+3+4+\dots$$

누군가는

$$1+100=101$$

$$2+99=101$$

...

$$101*50=5050$$

누군가는

$$n(n+1)/2$$

There are many ways to solve problem.

Algorithm is focus on clarity

Conditions

- has external input
- has 1 or more result
- clarity
- finite trial
- simplicity

Clarity

time complexity == big O notation

| 자료의 수 (n)이 증가할 때 시간의 증가 패턴을 나타낸 것

big O notation

1

$\log n$

n

$n \log n$

n^2

n^3

2^n

$n!$

O(1) : constant

- 값에 대한 키 또는 인덱스를 알고 있을 경우

```
minsu_exam_result = {"kor":95,"math":40}
```

```
minsu_exam_result['kor']
```

```
result = 0  
n = 100  
result = n*(n+1)/2
```

$O(\log n)$: logarithmic

- 배열에서 값을 접근할 때 앞 또는 뒤에서 접근 선택이 가능할 경우

```
animals = ['cat', 'dog', 'fox', 'giraffe', 'hippo',  
'koyote', ...]
```

$O(n)$: linear

- 자료의 수와 시도횟수가 1:1 관계인 경우

```
result = 0
for i in range(1,100+1):
    result += i
print(result)
```

$O(n^2)$: quadratic

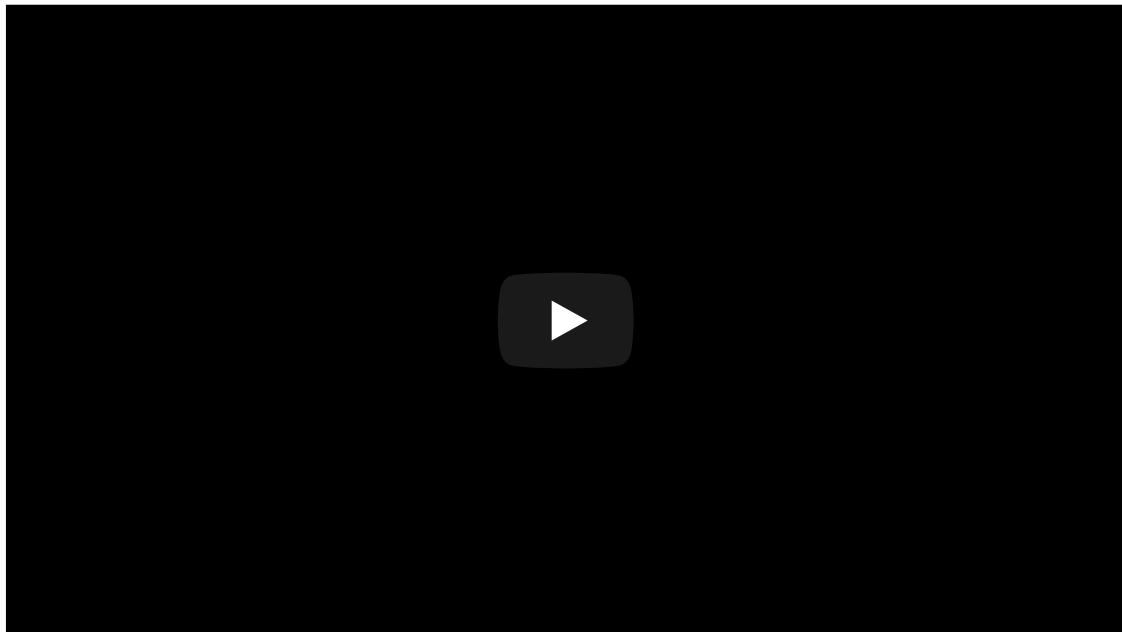
- 자료의 참조를 이중으로 하게 될 경우

```
result = 0
for i in range(1,10+1):
    for j in range(1,j+1):
        result += j
```

Sort algorithms

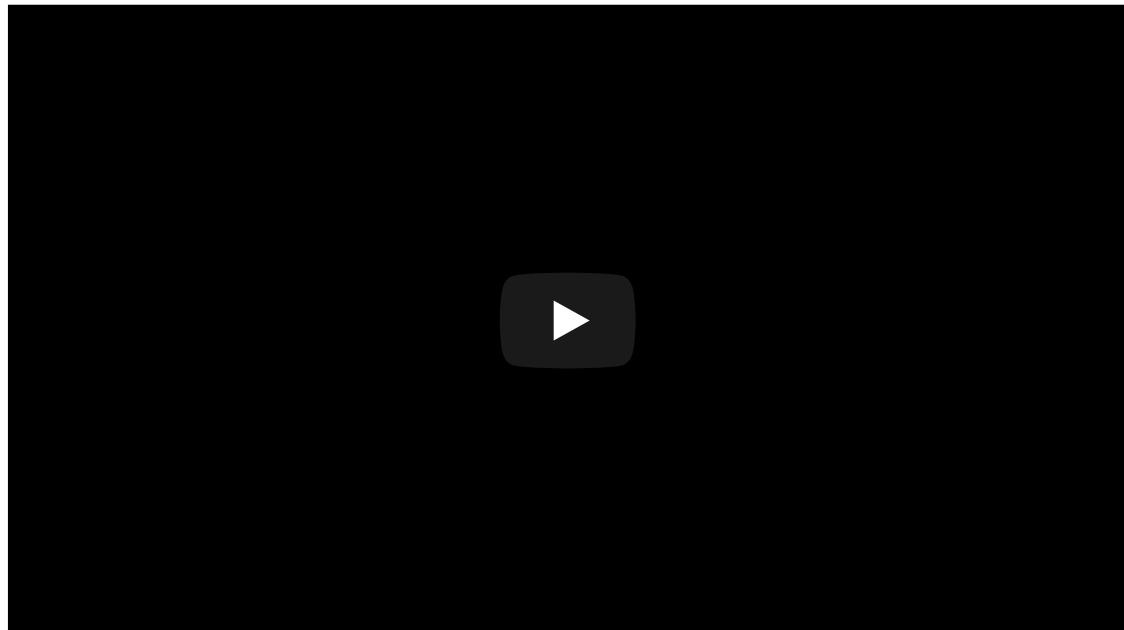
- $O(n^2)$
 - Bubble sort
 - Selection sort
 - Insertion sort
- $O(n \log n)$
 - Merge sort
 - Heap sort
 - Quick sort

Bubble sort



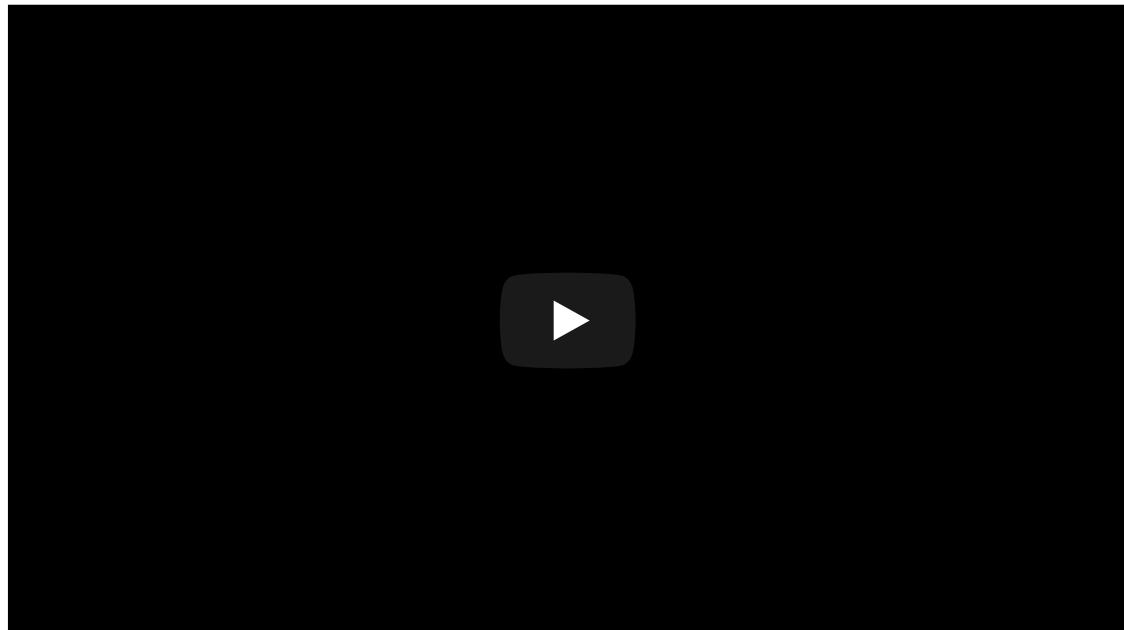
- 1:1로 $n(n-1)/2$ 번 수행하는 방법
- 최악..

Selection sort



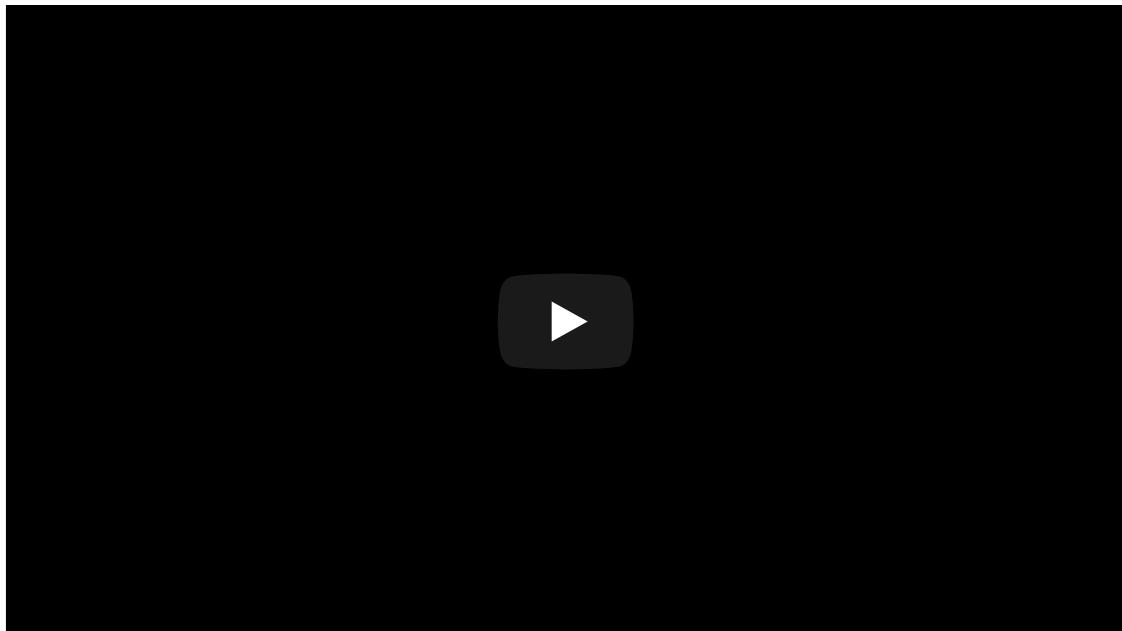
- 가장 작은 값부터 순서대로 정렬
- 인간과 가장 가까운 정렬법

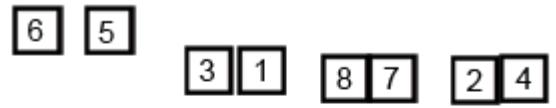
Insertion sort



- n번째 요소를 처음부터 n-1번째 까지 비교하면서 값을 끼워넣는 법
- 윗 방법들보단 빠름

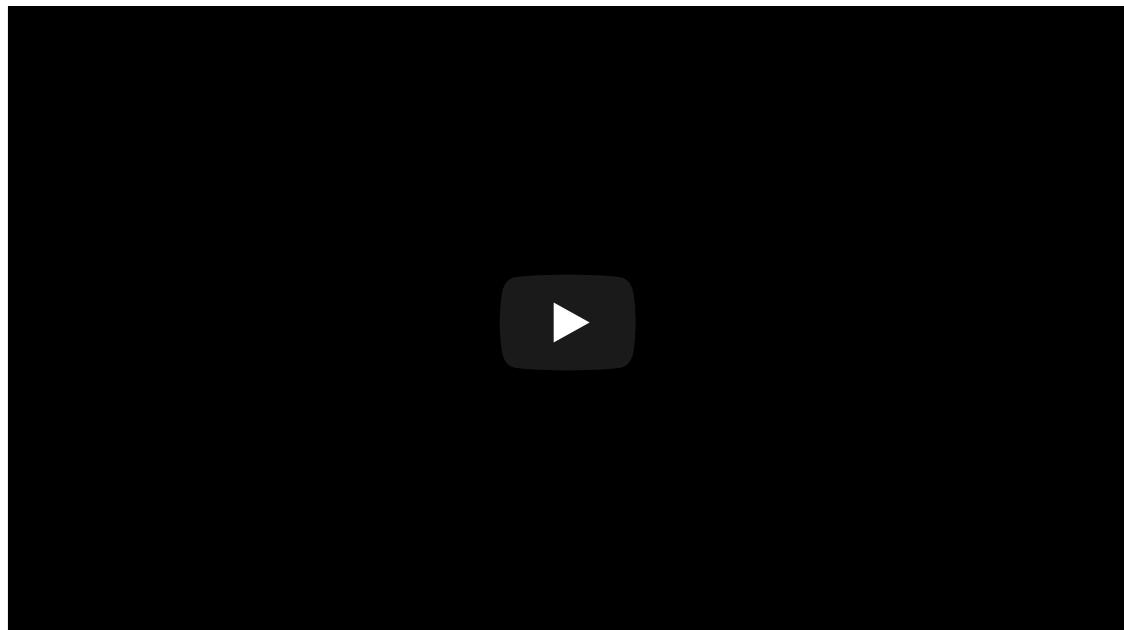
Merge sort



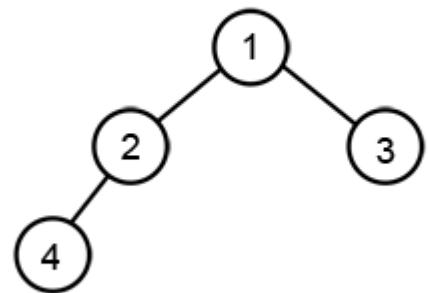


- 두개씩 쪼개 각각을 비교하며 정렬하는 방법
- 데이터 상태에 큰 영향을 받지 않음

Heap sort

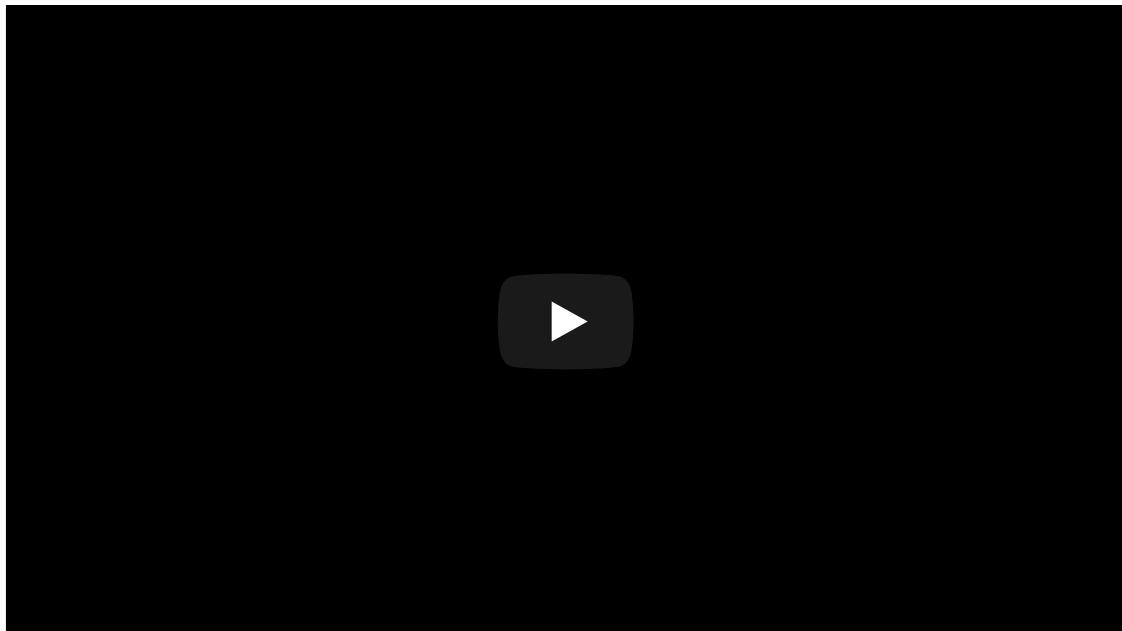


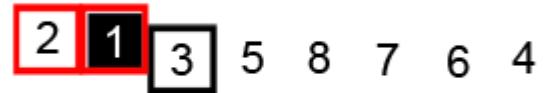
1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---



- 데이터를 힙에 넣은 뒤 최대값(루트)을 출력하고 힙에서 제거

Quick sort





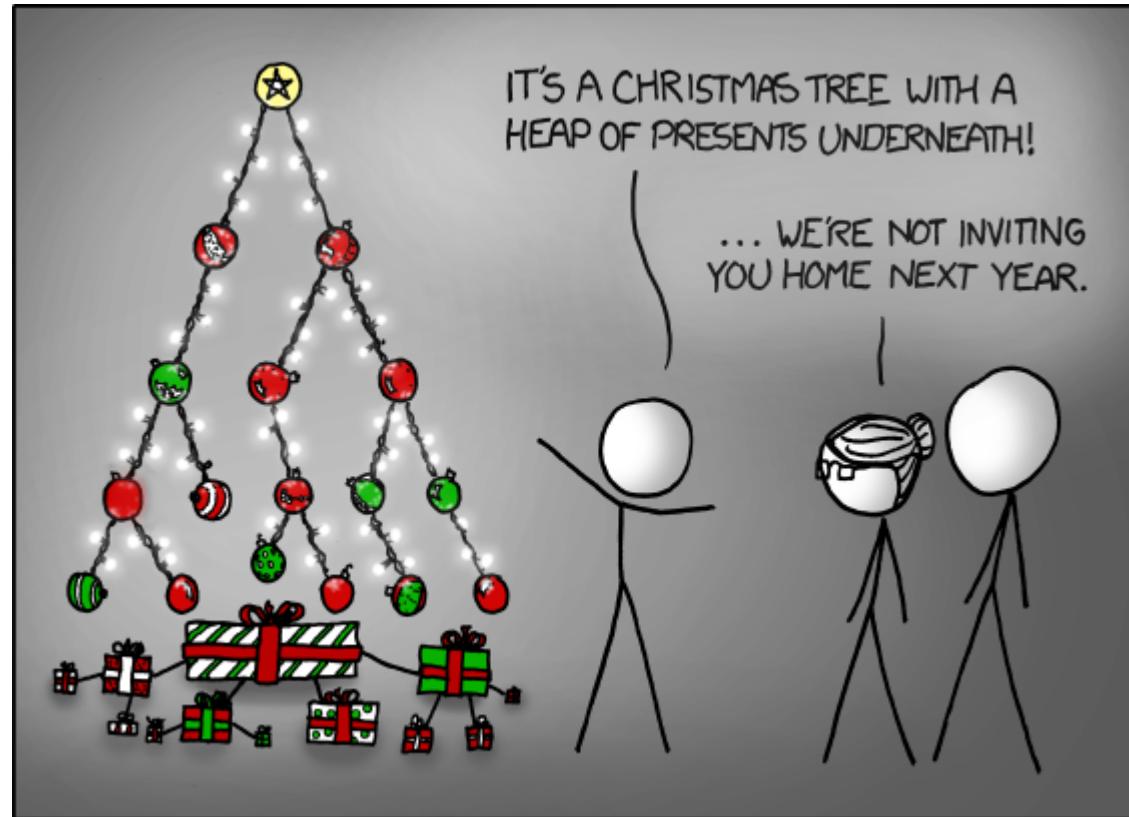
- 피벗을 기준으로 큰값 작은 값을 나눈 뒤, 피벗을 옮겨 다시 수행하는 방법
- 평균적으로 가장 빠른 방법

Data Structure

Data Structure

Data structure is a particular way of organizing data in a computer so that it can be used efficiently.

So, Data Structure is..



Data Structures in Web Development

Array & Hash(Dictionary) - indexing post

```
in RDB  
[articleId, title, body, userId, view]
```

```
[  
  {  
    userId: 1,  
    articleId: 1,  
    view: 100,  
    title: "sunt aut facere repellat provident occaecati ex"  
    body: "quia et suscipit suscipit recusandae consequuntur  
  },  
  ...  
]
```

Data Structures in Web Development

Tree - DOM rendering performance, reply

```
<html>
<head></head>
<body>
<h1></h1>
<p></p>
</body>
</html>
```

Data Structures in Web Development

- Binary Tree Search
 - Queue(BFS, Breadth First Search)
 - Stack(DFS, Depth First Search)

We'll Learn about..

- Stack
- Queue

Stack



Stack

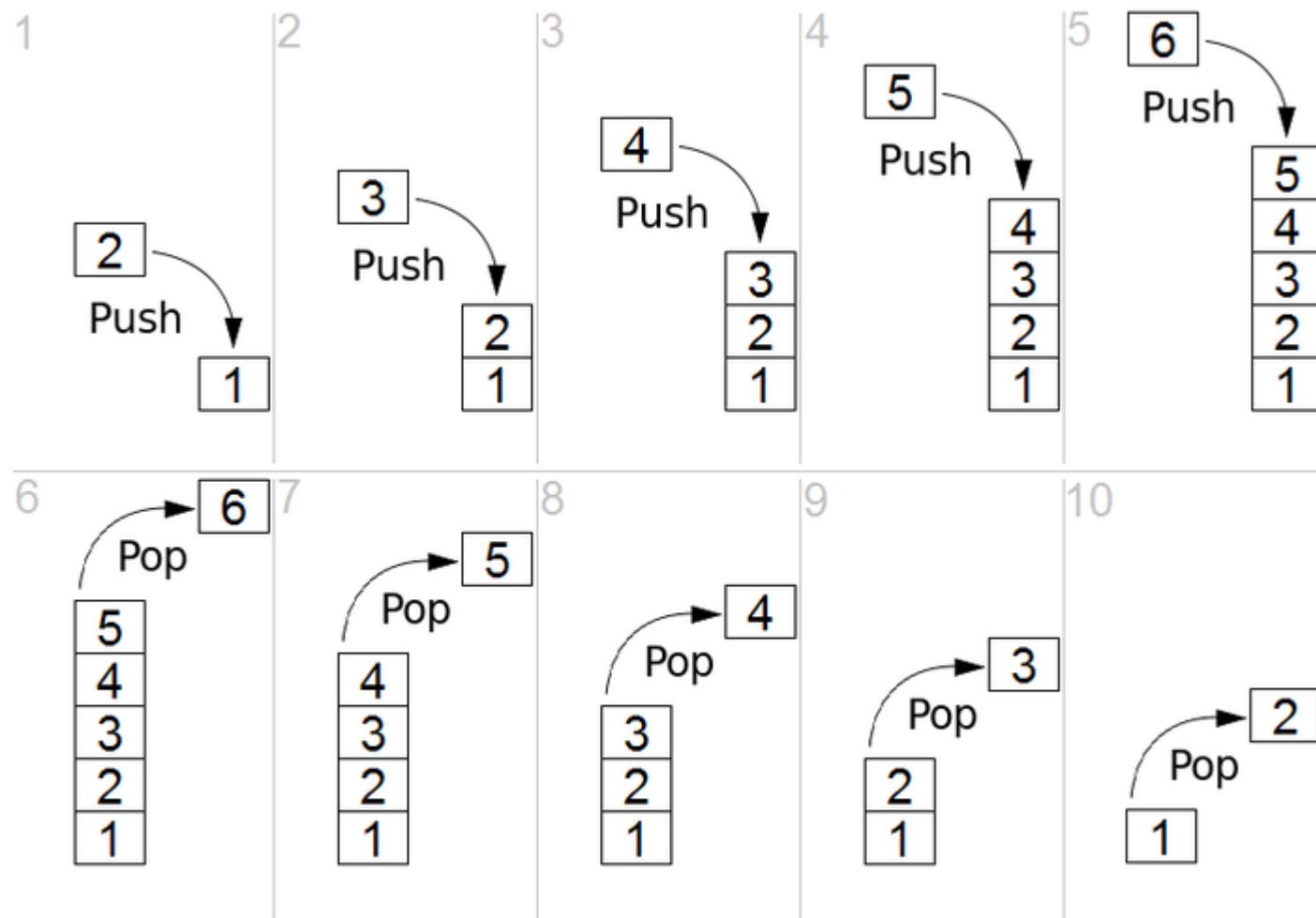
A stack is an abstract data type that serves as a collection of elements, with two principal operations.

- push: which adds an element to the collection
- pop: which removes the most recently added element that was not yet removed

LIFO

Last In, First Out





Let's Create Stack class

```
function Stack() {  
    //properties, methods  
    var items = [];  
}
```

push & pop

```
function Stack() {  
    //properties, methods  
    var items = [];  
  
    this.push = function(element){  
        return items.push(element);  
    };  
  
    this.pop = function(){  
        return items.pop();  
    };  
}
```

peek & isEmpty

```
function Stack() {
    //underneath push & pop
    ...
    this.peek = function(){
        return items[items.length-1];
    };
    this.isEmpty = function(){
        return items.length == 0;
    };
}
```

size & clear & print

```
function Stack() {
    //underneath peek & isEmpty
    ...
    this.size = function(){
        return items.length;
    };

    this.clear = function(){
        items = [];
    };

    this.print = function(){
        console.log(items.toString());
    };
}
```

Let's push with Stack class

```
> var stack = new Stack();
> console.log(stack.isEmpty());

> stack.push(5);
> stack.push(2);
> stack.push(8);

> console.log(stack.peek());

> stack.push(11);

> console.log(stack.size());
> console.log(stack.isEmpty());
```

Let's pop with Stack class

```
> stack.pop();
> stack.pop();

> console.log(stack.size());
> stack.print();
```

Queue



Queue

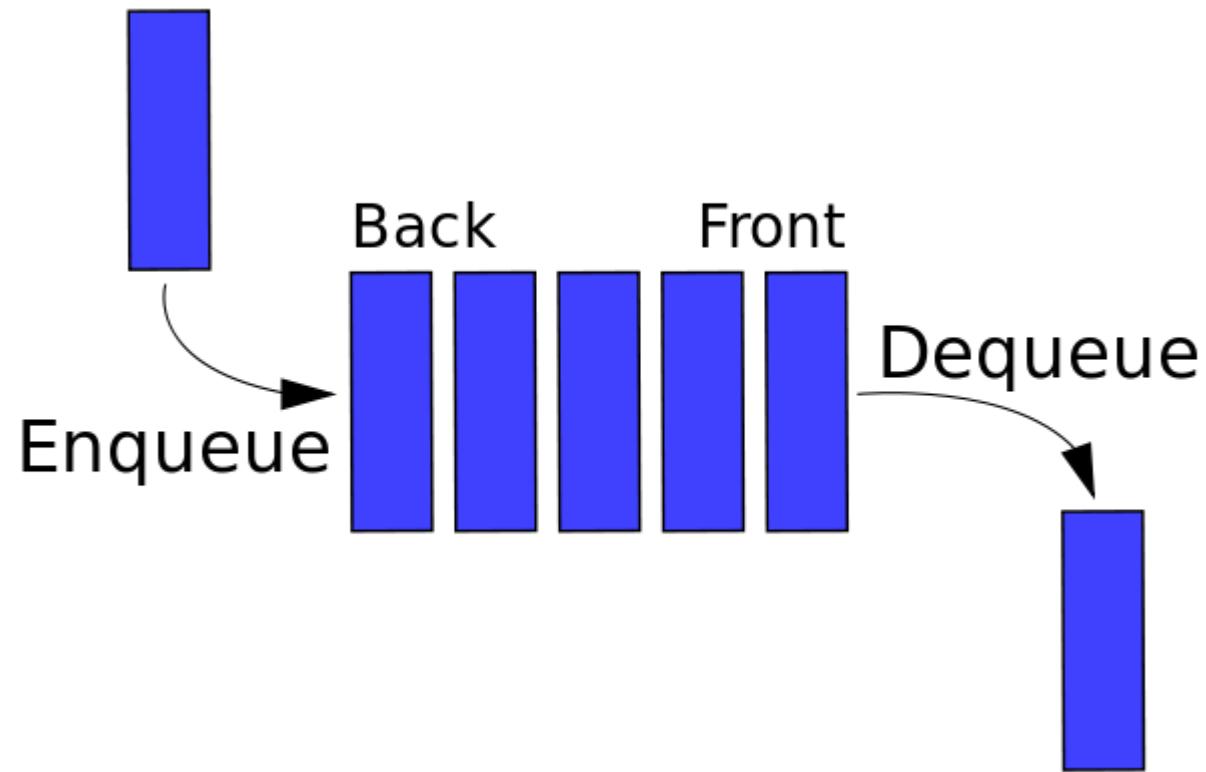
a queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order.

FIFO

First In First Out

Enqueue & Dequeue

- Enqueue: addition of entities to the rear terminal position
- Dequeue: removal of entities from the front terminal position



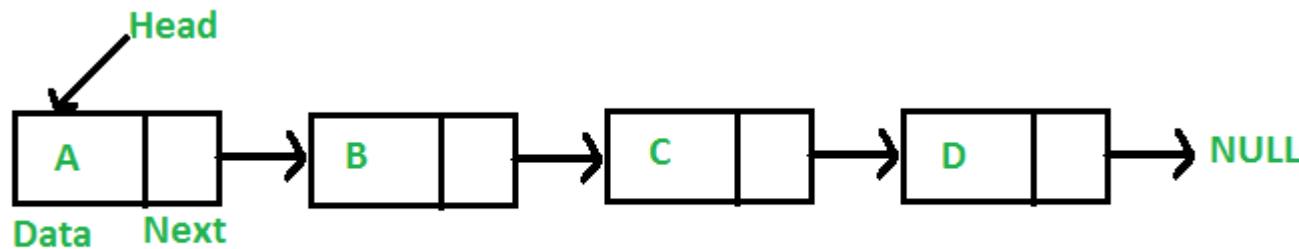
Linked List

Linked List

A linked list is a linear collection of data elements, in which linear order is not given by their physical placement in memory.

Linked List

- Can be used to store linear data of similar types.



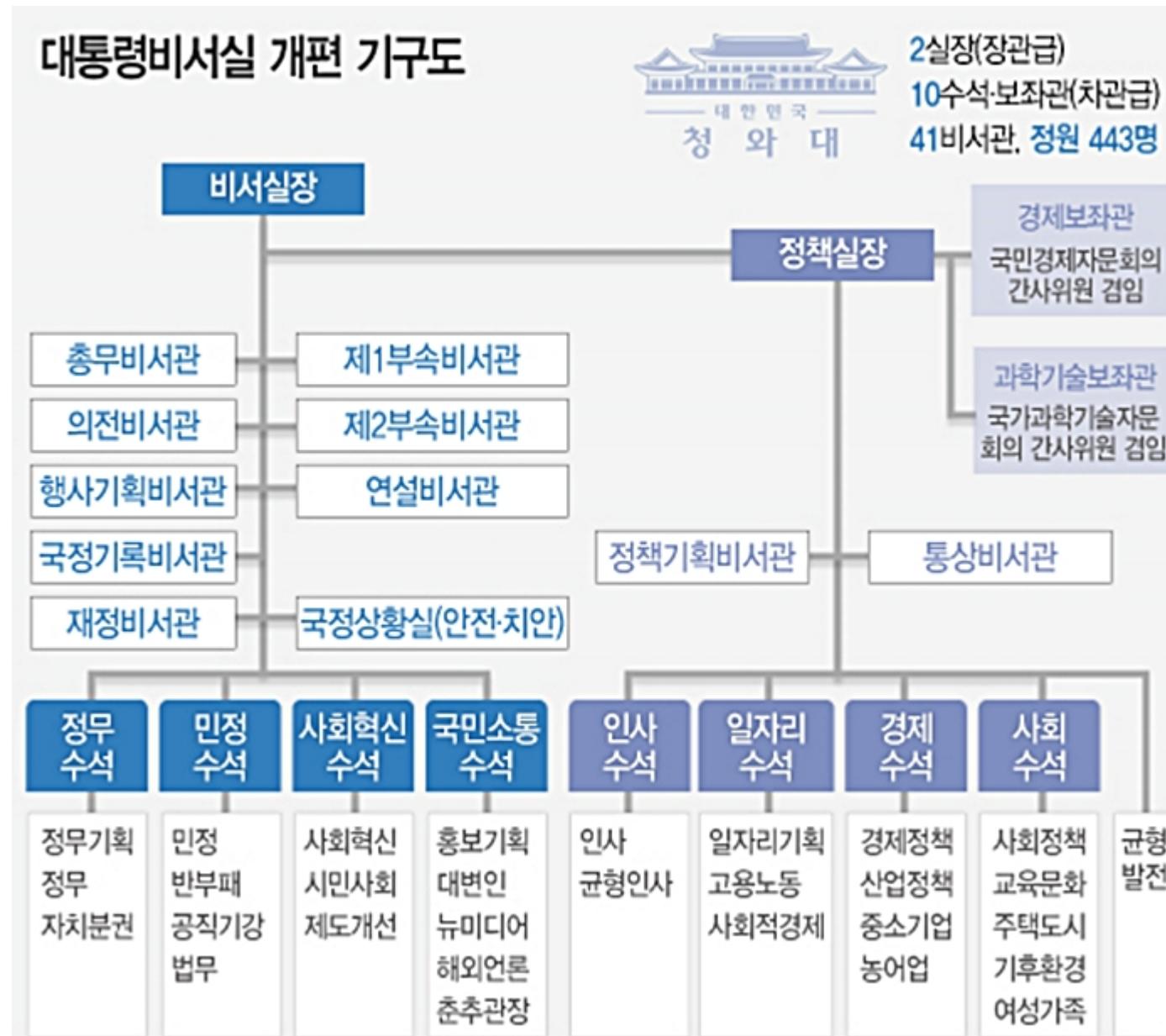
Tree

Tree

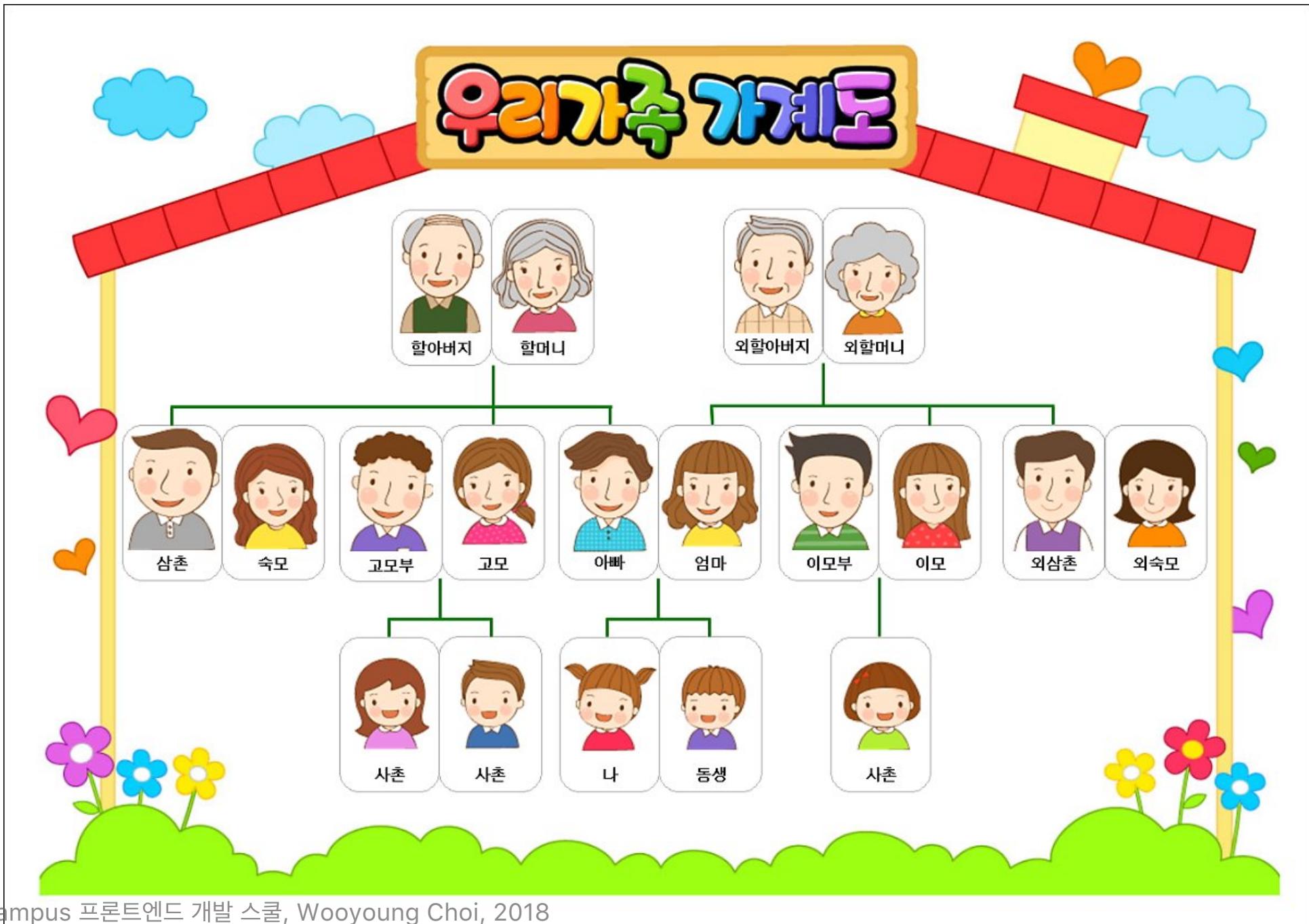
A tree is an abstract model of a hierarchical structure.

- hierarchical: arranged in order of rank.

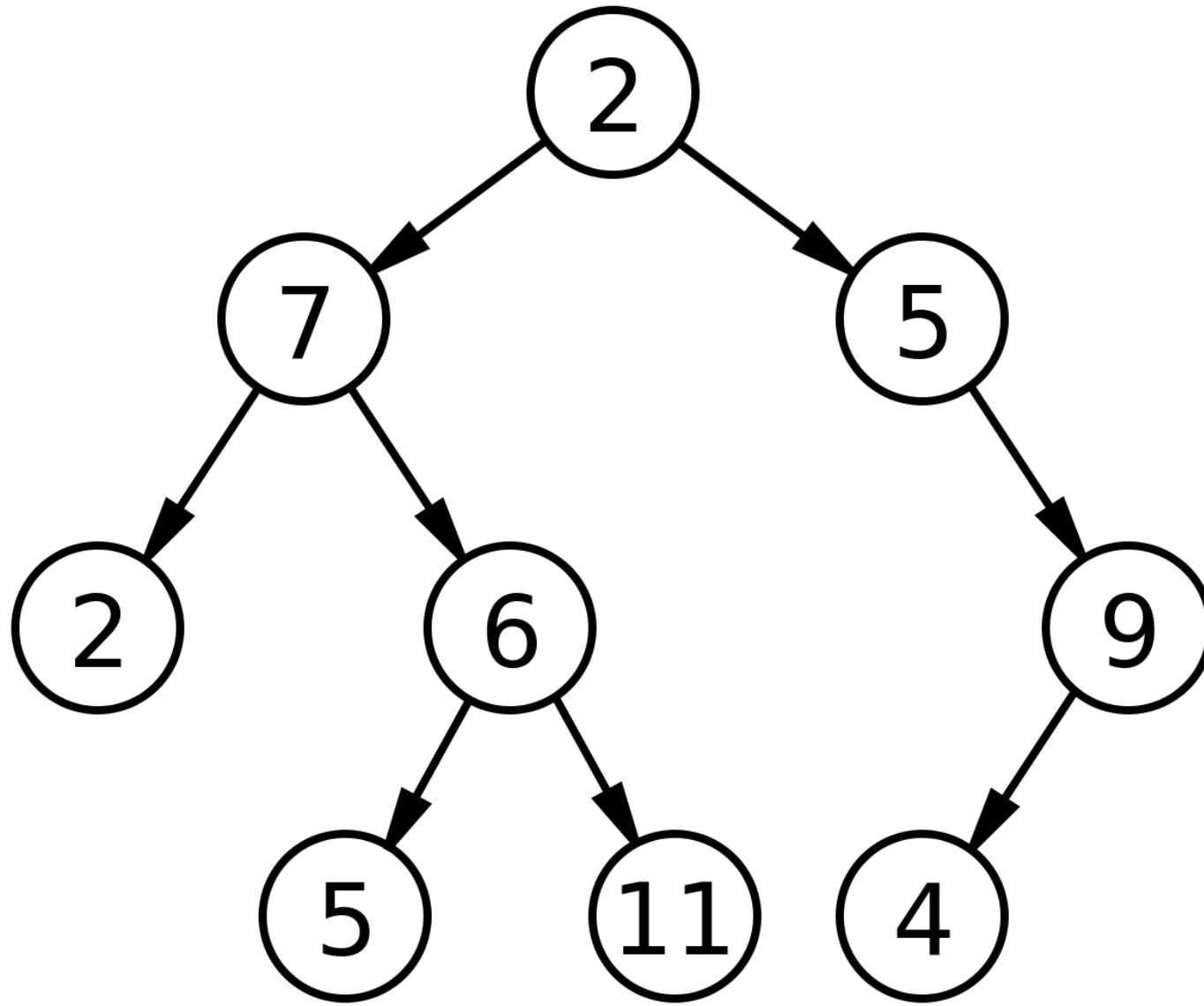
Tree



Tree



Tree



Tree

- root: 2
- level: (0 ~ 3)
- child of 2: 7,5
- subtree: 6,5,11
- Node: (9)
- edge: (8)

Binary Search Tree

A node in a binary tree has at most two children: left child, right child

- if `root == null`, `node = newNode`
- `left child < right child`