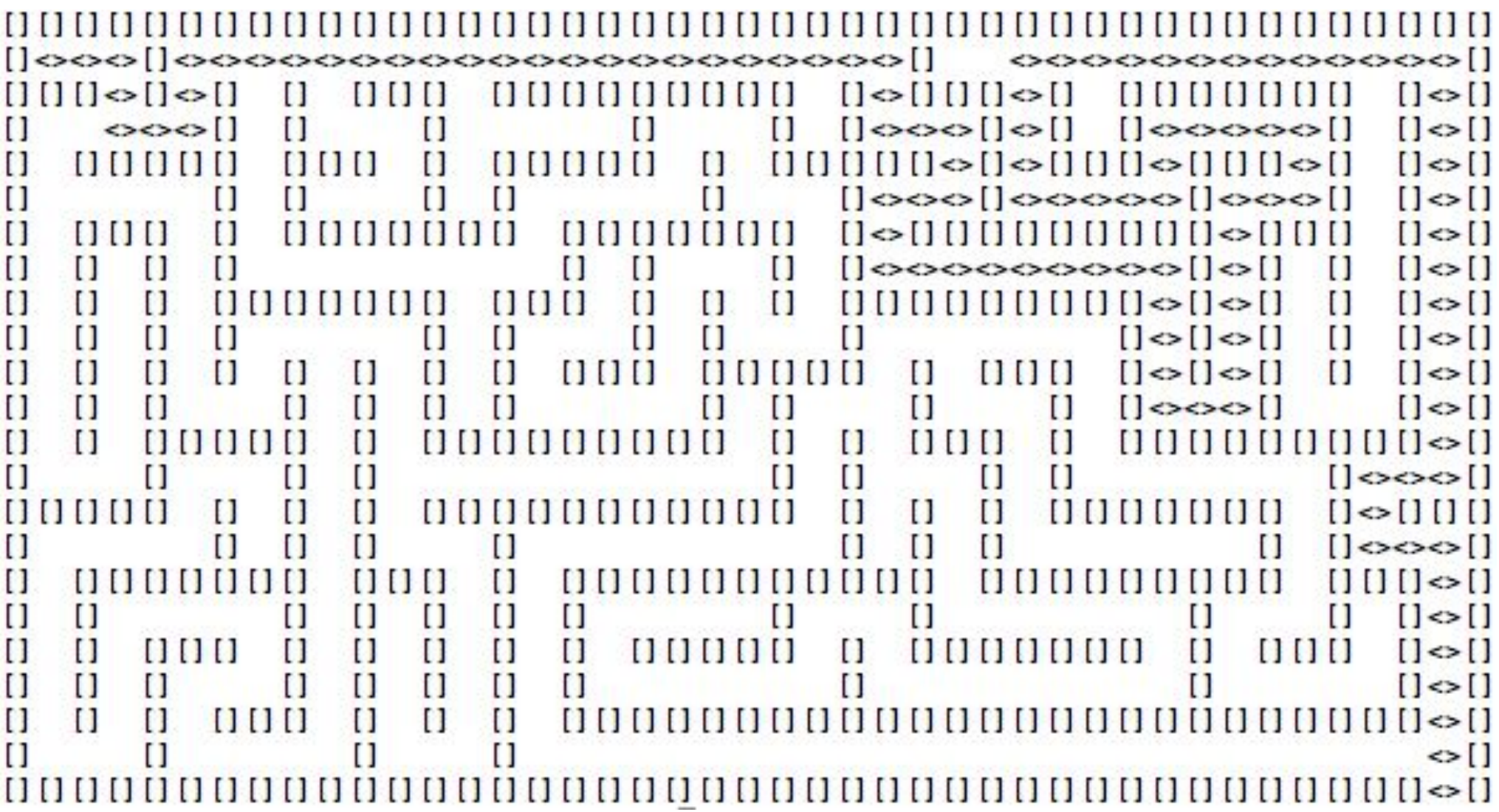
Data Structure & Algorithm

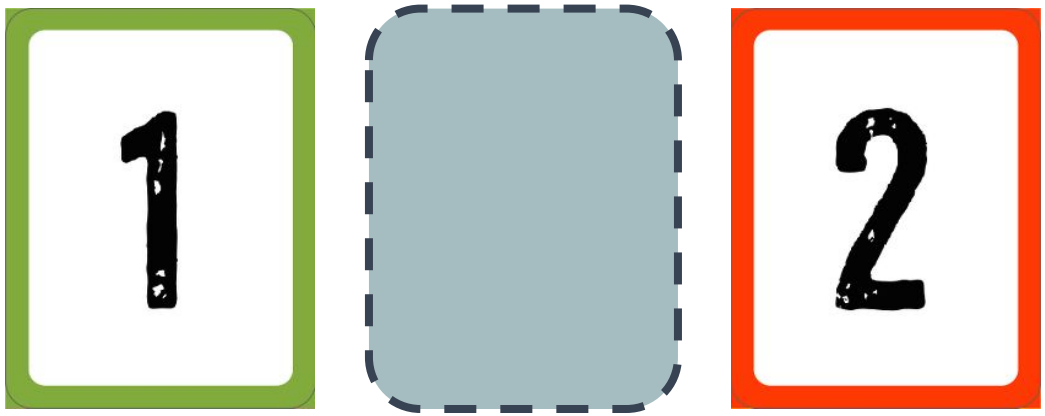
*

저녁이 있는 프로젝트
오상훈
6 Hours, 1 Month

Goal

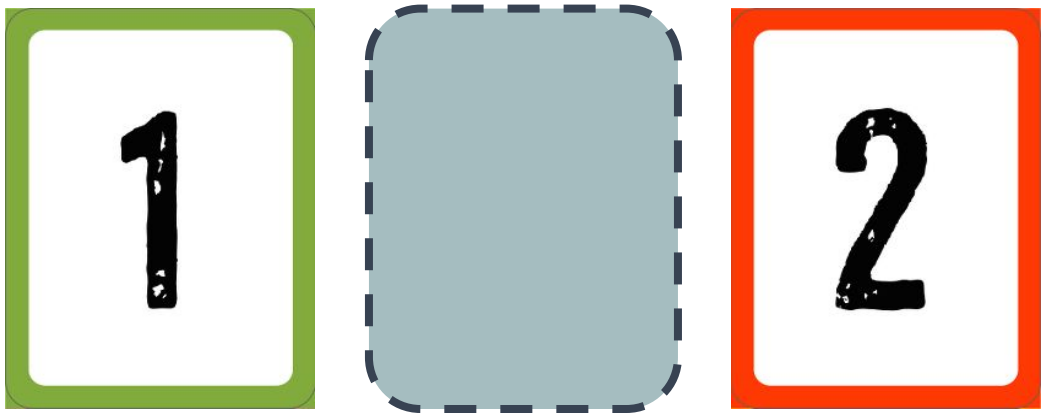


Making Game by One Hand

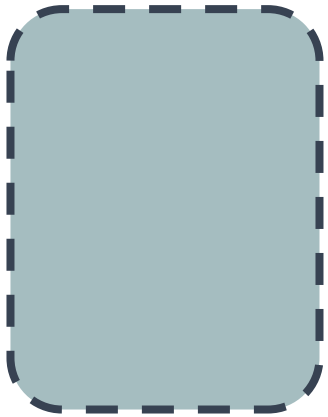


- One Hand And One Step.
- Can’t Jump Any Card.

Making Game by One Hand

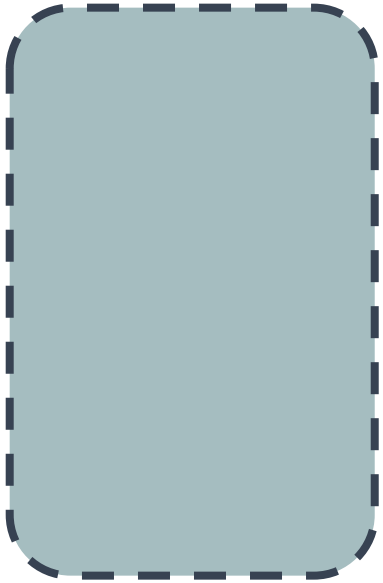


- One Hand And One Step.
- Can’t Jump Any Card.



Thinking About

- One Hand And One Step.
- Can’t Jump Two Card.



당신은 프로그래머 ?

Flowchart(1)

Terminator
Indicates the beginning or end of a program flow in your diagram.

Process
Indicates any processing function.

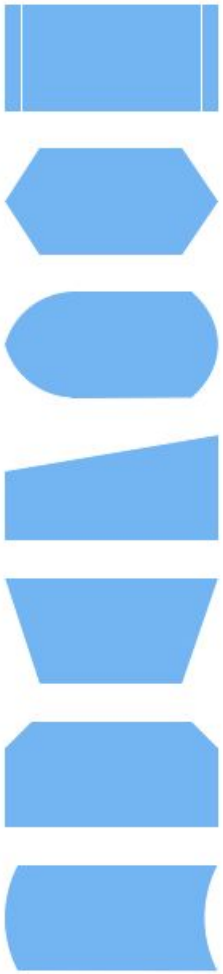
Decision
Indicates a decision point between two or more paths in a flowchart.

Delay
Indicates a delay in the process.

Data
Can represents any type of data in a flowchart.

Document
Indicates data that can be read by people, such as printed output.

Multiple documents
Indicates multiple documents.



Subroutine
Indicates a predefined (named) process, such as a subroutine or a module.

Preparation
Indicates a modification to a process, such as setting a switch or initializing a routine.

Display
Indicates data that is displayed for people to read, such as data on a monitor or projector screen.

Manual input
Indicates any operation that is performed manually (by a person).

Manual loop
Indicates a sequence of commands that will continue to repeat until stopped manually.

Loop limit
Indicates the start of a loop. Flip the shape vertically to indicate the end of a loop.

Stored data
Indicates any type of stored data.

Connector
Indicates an inspection point.

Off-page connector
Use this shape to create a cross-reference and hyperlink from a process on one page to a process on another page.

Off-page connector

Off-page connector

Off-page connector

Or
Logical OR

Summing junction
Logical AND

Collate
Indicates a step that organizes data into a standard format.

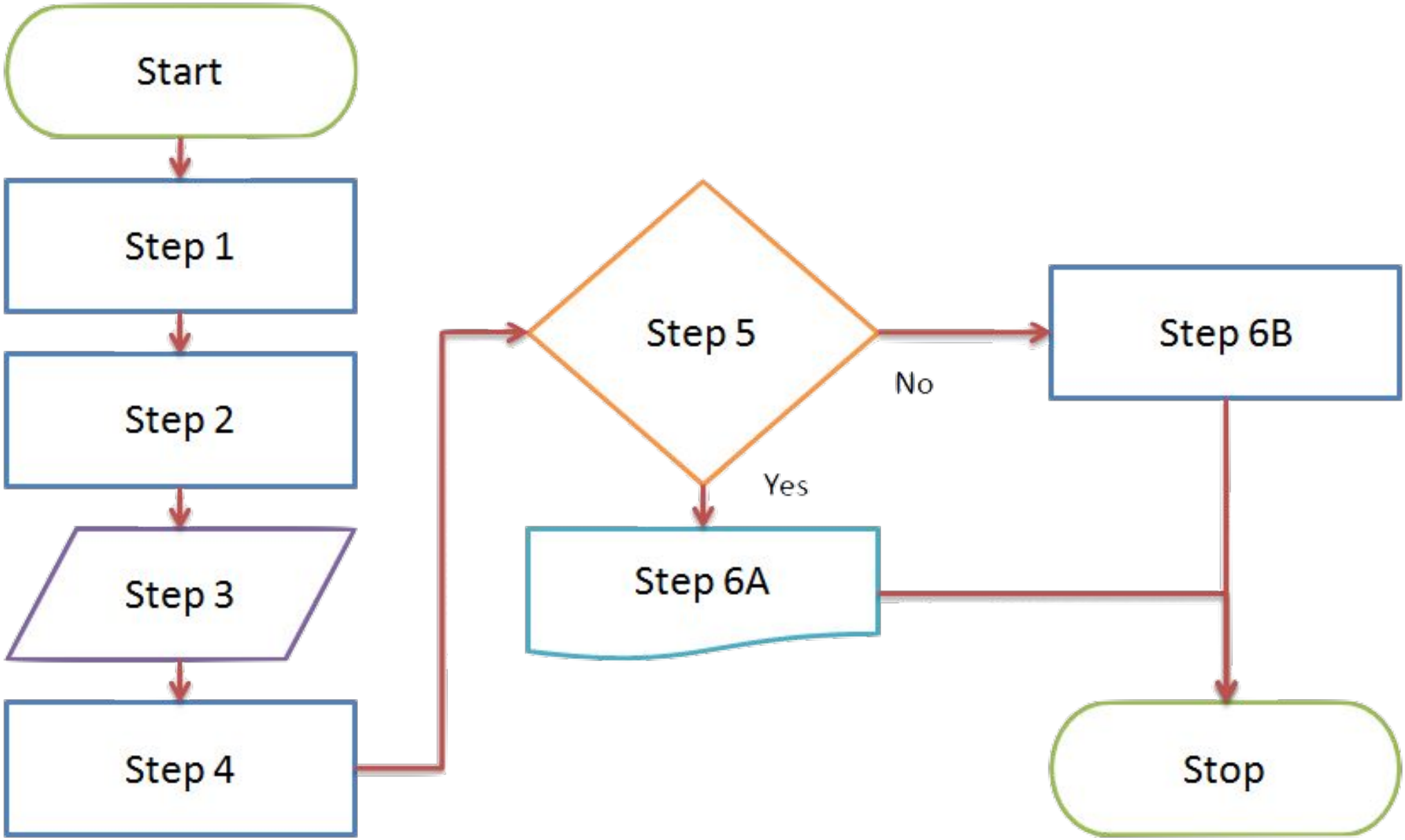
Sort
Indicates a step that organizes items list sequentially.

Merge
Indicates a step that combines multiple sets into one.

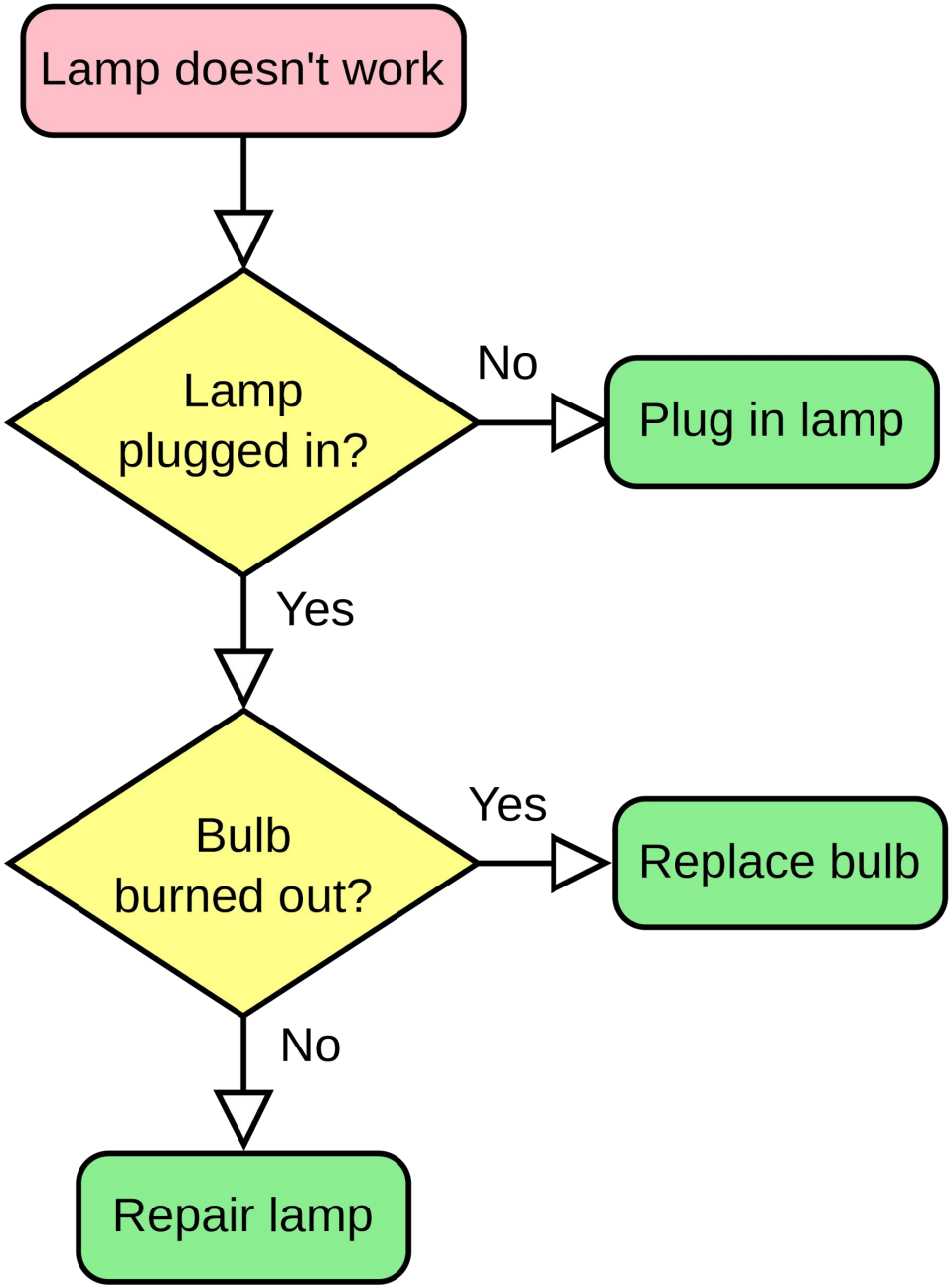
Database
Indicates a list of information with a standard structure that allows for searching and sorting.

Internal storage
Indicates an internal storage device.

Flowchart(2)


Start
Step 1
Step 2
Step 3
Step 4
Step 5
No
Yes
Step 6A
Step 6B
Stop

Flowchart(3)


Lamp doesn't work
Lamp plugged in?
No
Plug in lamp
Yes
Bulb burned out?
Yes
Replace bulb
No
Repair lamp

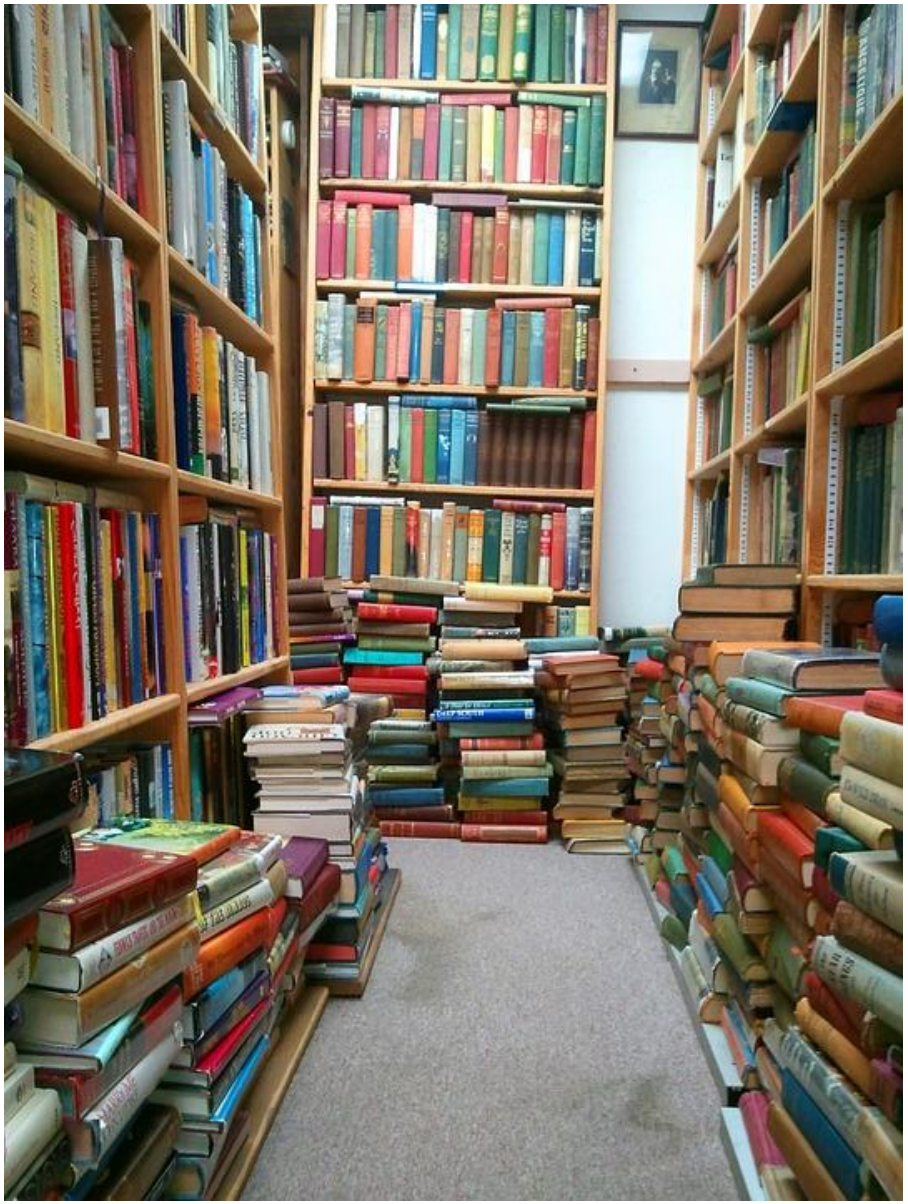
Trying(1) - flowchart 화

- 해보기
  - 값 0이 들어간 변수 A, B, C, D 만들다
  - A = 1, B = 3, C = 5, D = 7 입력
  - B 가 3이면 A = 10 입력, 아니면 C = 5
  - D 가 9 아니면 B = 5 입력, 아니면 B = 200
  - A 가 10이면, C = 50입력, 아니면 D = 30

Trying(2) - flowchart 화

- 해보기
  - 값 0이 들어간 변수 A, B, C, D 만들다
  - A = 138, B = 232, C = 495, D = 982 입력
  - B 가 100 보다 크면 A = B - 392 입력, 아니면 C = D - 392
  - D 가 394 보다 작지 않으면, C = B - 283 입력, 아니면 A = A + 200
  - A 가 800 보다 작으면, C = C - 50입력, 아니면 D = D + 30
  - 기호 사용
    - !(연산식)
    - <=, >=

일상 속 자료구조와 알고리즘

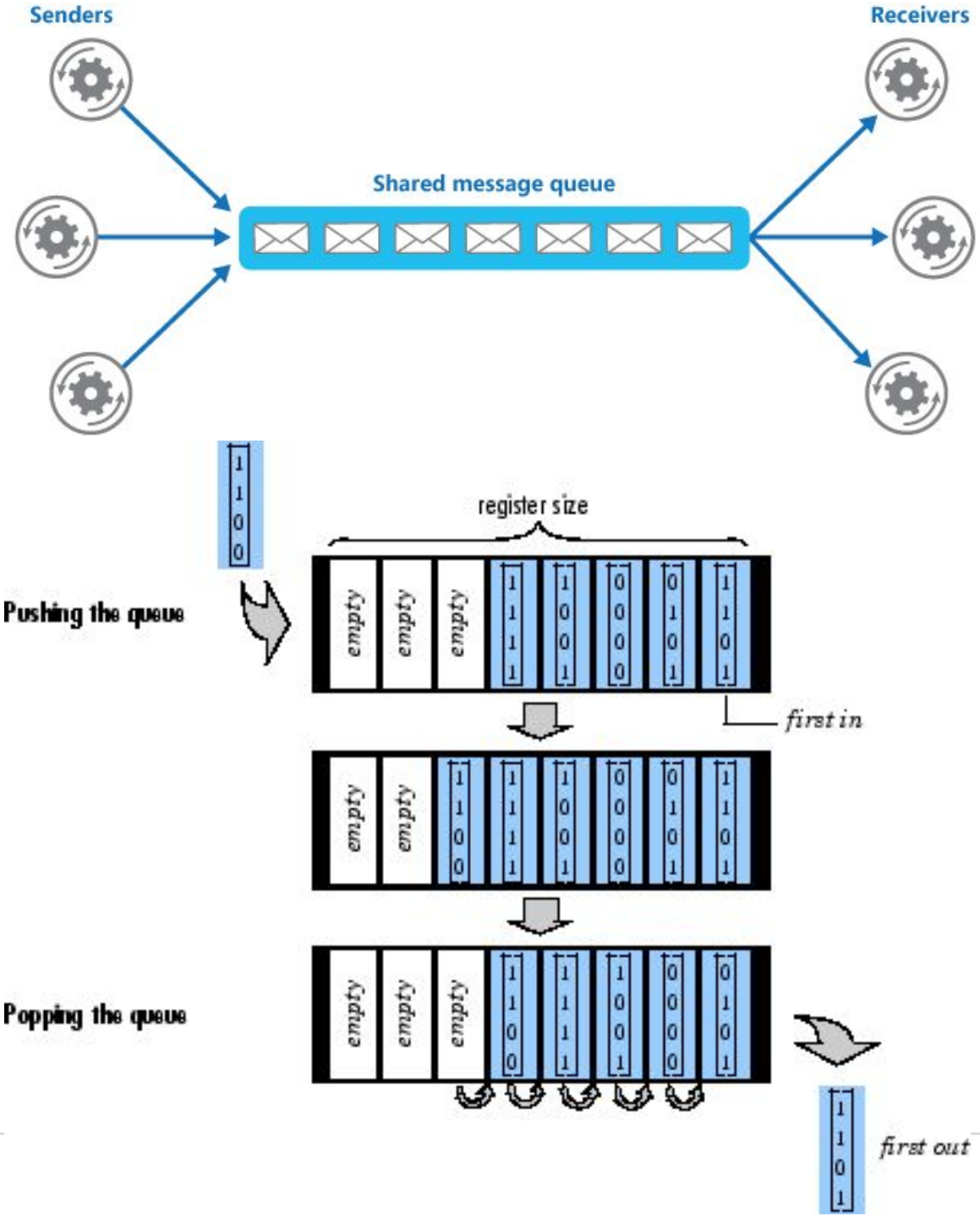


일상 속 자료구조


12:38 PM
MISSION DISTRICT
13.1 miles
62 bpm
Francesco
@evleaks
12:38 PM
Samsung Link
Maps
S Health
Video
S Memo
Gallery
Gmail
Play Books
Samsung Hub
Play Store
@evleaks


Mail box

일상 속 알고리즘


Senders
Receivers
Shared message queue
register size
Pushing the queue
empty
empty
empty
first in
Popping the queue
first out

자료구조 + 알고리즘 = 프로그래밍

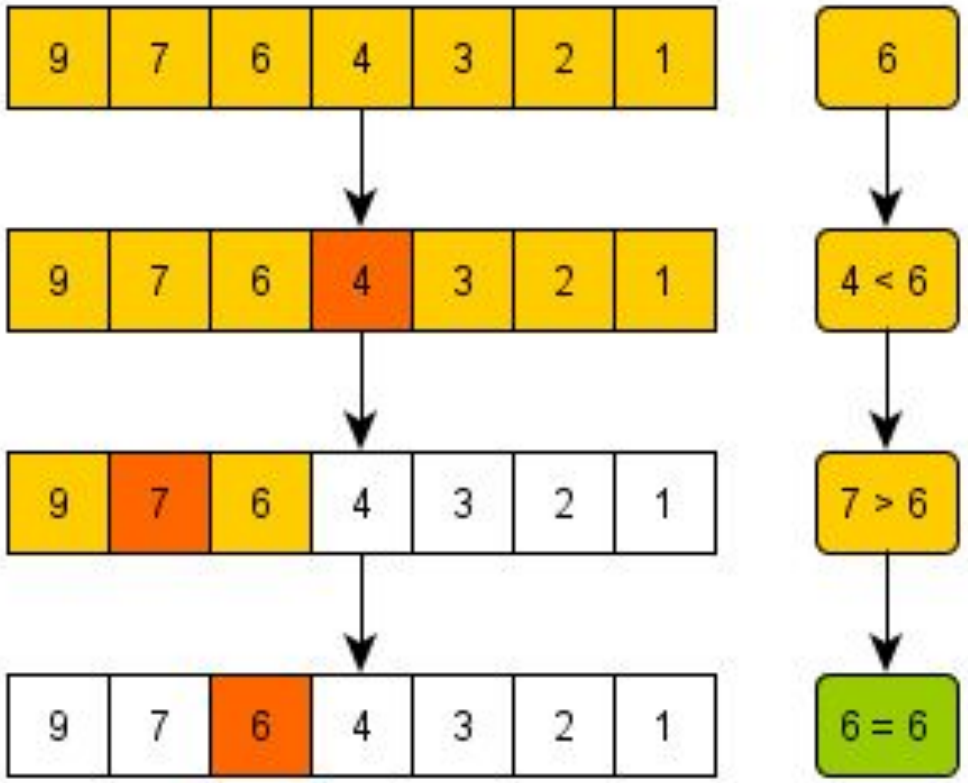
- 자료구조 : 데이터 표현 및 저장 방법.
- 알고리즘 : 문제 해결 방법.

```
#include <stdio.h>
int main(){
    // 자료구조
    int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ;

    int sum = 0;
    // 알고리즘
    for(int index=0; index < 10; index++){
        sum += array[index] ;
    }
}
```

탐색 알고리즘 관점 시간복잡도

❖ 순차 탐색

❖ 이진 탐색(정렬 필수)



Trying

- 구현 목적
  - 순차탐색 프로그래밍과 복잡도 이해
- 구현 순서
  - 입력 : 시작과 끝 번호 입력 - scanf()
  - 구현
    - 배열 생성 : array[100]
    - 탐색 번호 입력 받음 : scanf()
  - 출력 : 검색 번호 여부
- 실행결과

```
Input Start Number of elements in array : 10
Input End Number of elements in array : 50
Input Start number:  10, End number:  50
Input number to search : 32
32 is present at location 23.
```

- 궁금해 하기
  - int LinearSearch(int [], int, int);

Trying

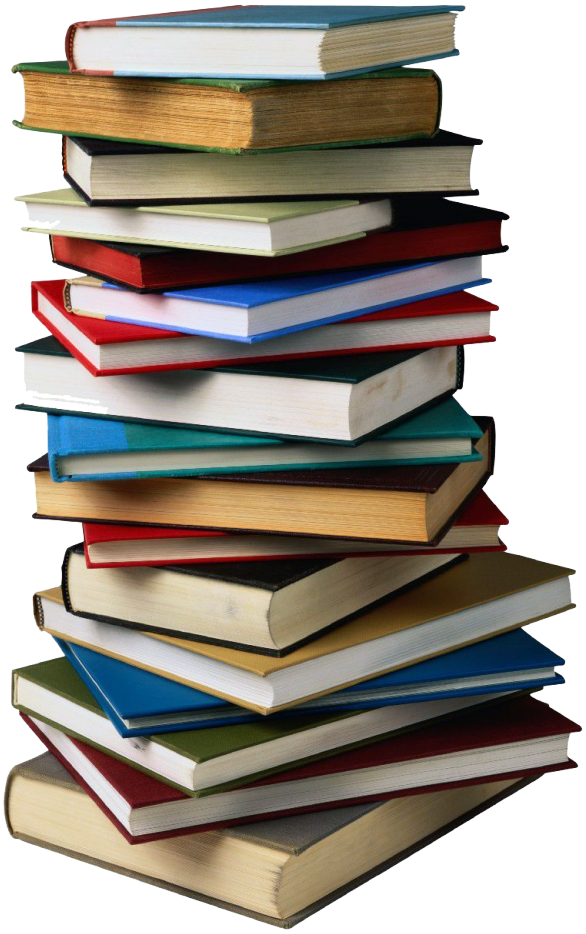
- 구현 목적
  - 이진탐색 프로그래밍과 복잡도 이해
- 구현 순서
  - 입력 : 시작과 끝 번호 입력 - scanf()
  - 구현
    - 배열 생성 : array[100]
    - 탐색 번호 입력 받음 : scanf()
  - 출력 : 검색 번호 여부
- 실행결과

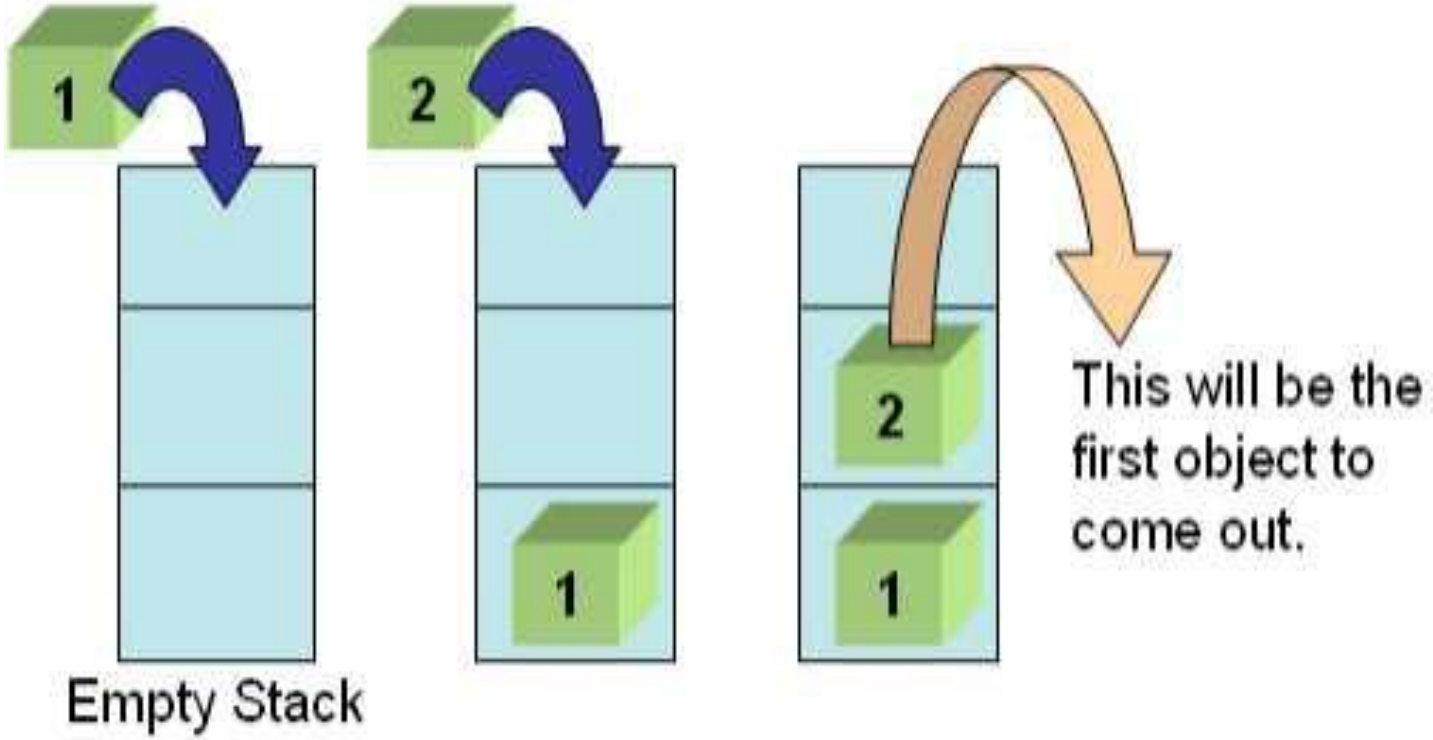
```
Input Start Number of elements in array : 29
Input End Number of elements in array : 50
Input Start number:  29, End number:  50
Input number to search : 20
20 is not present in array.
```

- 궁금해 하기
  - 램덤 입력 시 동작 구현
  - int BinarySearch(int [], int, int);

스택(Stack)

- LIFO(Last-In, First-Out)
- Push, Pop, peek





Trying

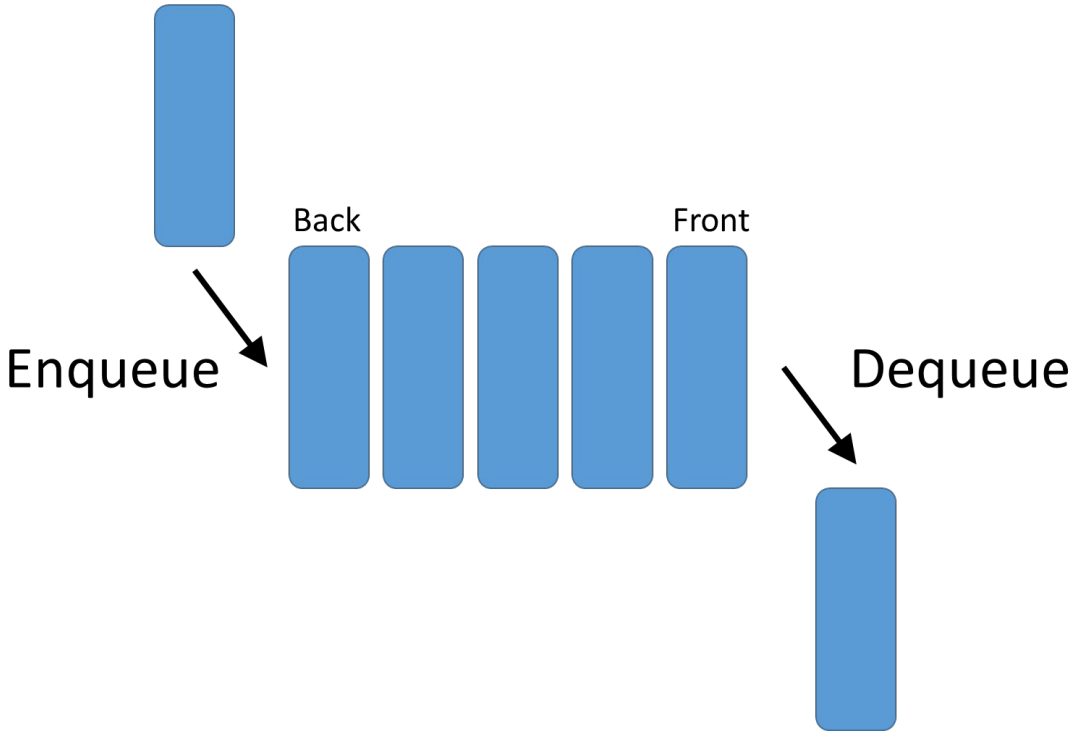
- 구현 목적
  - 스택 구현과 이해
- 구현 순서
  - 입력 : 3, 5, 9, 1, 12, 15
  - 구현 : int stack[8];
  - 출력 : 15, 12, 1, 9, 5, 3와 Stack 상태 출력
- 실행결과

```
Push Elements:
3   5   9   1   12 15
Element at top of the stack: 15
Pop Elements:
15 12 1   9   5   3
Stack full: false, Stack empty: true
```

- 궁금해 하기
  - Push, Pop 선택적 방식 구현 - scanf()

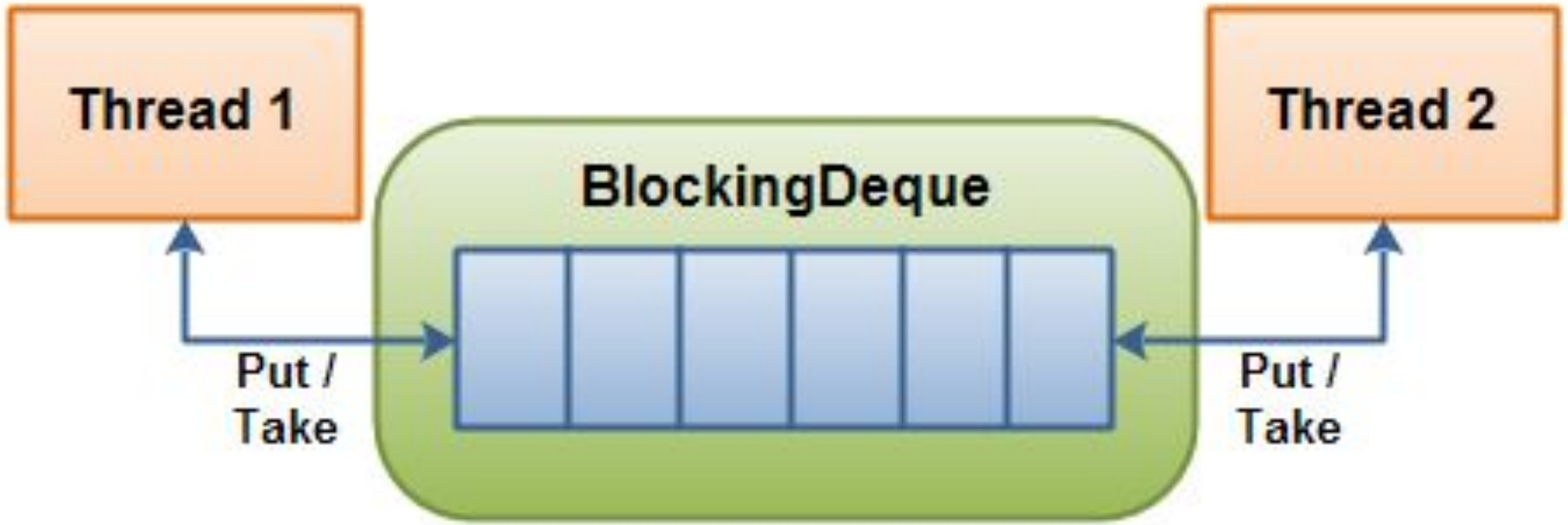
큐(Queue)

- FIFO(First-In First-Out)
- enqueue, dequeue



덱(Deque)

- Double-ended queue
- Stack + Queue

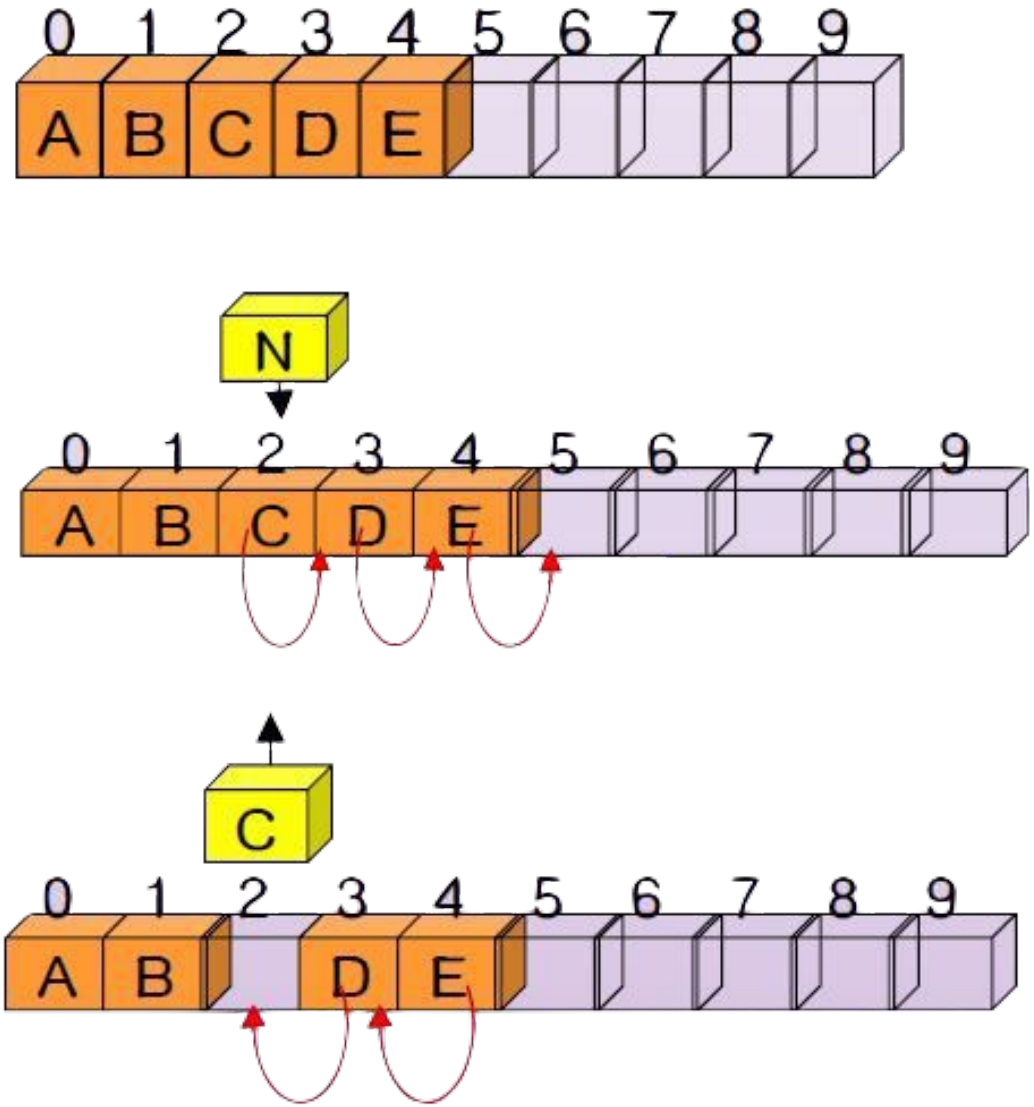


리스트(List)

❖ 순차리스트(Sequential List) : 배열 기반

❖ 연결리스트(Linked List) : 메모리 동적 할당 기반

❖ 순차리스트 : 배열 기반



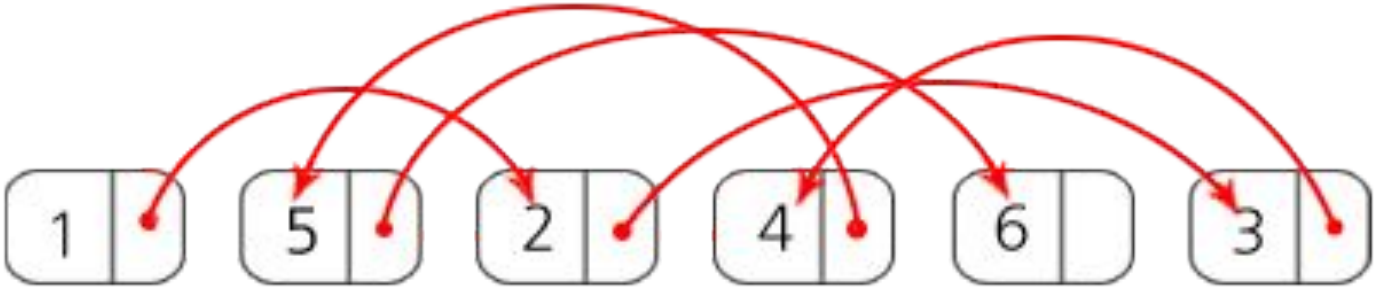
Trying

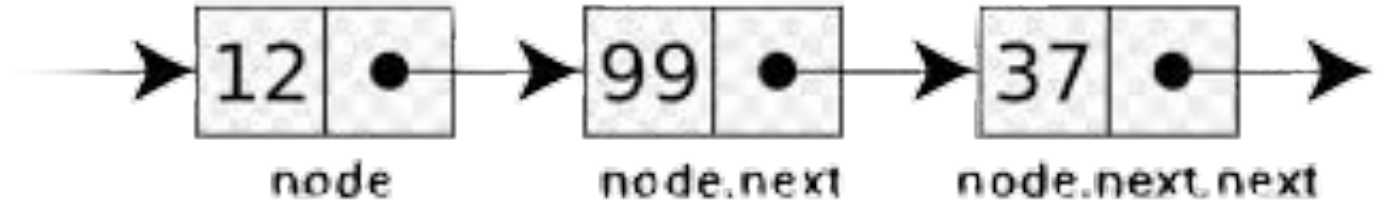
- 구현 목적
  - Sequential List 구현과 이해
- 구현 순서
  - 입력 : arraylist [10]
  - 구현 : 7, 9, 3, 8, 0 - rand()%10;
  - 결과 출력
- 실행결과?

```
1
1 2
...
1 2 3 4 5 6 7 8 9 10
deleting index 7 -> 1 2 3 4 5 6 7 9 10
deleting index 9 -> 1 2 3 4 5 6 7 9 10
deleting index 8 -> 1 2 3 5 6 7 9 10
deleting index 0 -> 2 3 5 6 7 9 10
…
deleting index 9 -> 2 3 6 10
```

```
typedef struct {
    int* array;
    int size;
} arraylist;
```

❖ 연결리스트(Linked List) : 메모리 동적 할당 기반





Trying(1)

- 구현 목적
  - ➢ Linked List 구현과 이해
- 따라해 보기

```
#include <stdlib.h>      #include <stdio.h>
int main(void){
        struct NODE  *head = NULL;
        head = malloc(sizeof(struct NODE));
        head -> value = 587;          head -> next = NULL;
        struct NODE *temp01 = NULL;
        temp01 = malloc(sizeof(struct NODE));
        temp01 -> value = -472;     temp01 -> next = NULL;     head -> next = temp01;
        …
        printf("\nPrint the whole list: ");          int count = 0;
        while (head != NULL){
                printf("%d ", head -> value);       head = head -> next;          count++;
        }
        while (head != NULL){
                free(head -> next);
        }
        printf("\n memory count : %d\n", count);
        return EXIT_SUCCESS;
}
```

```
struct NODE{
        struct NODE *next;
        int value;
};
```

Trying(2)

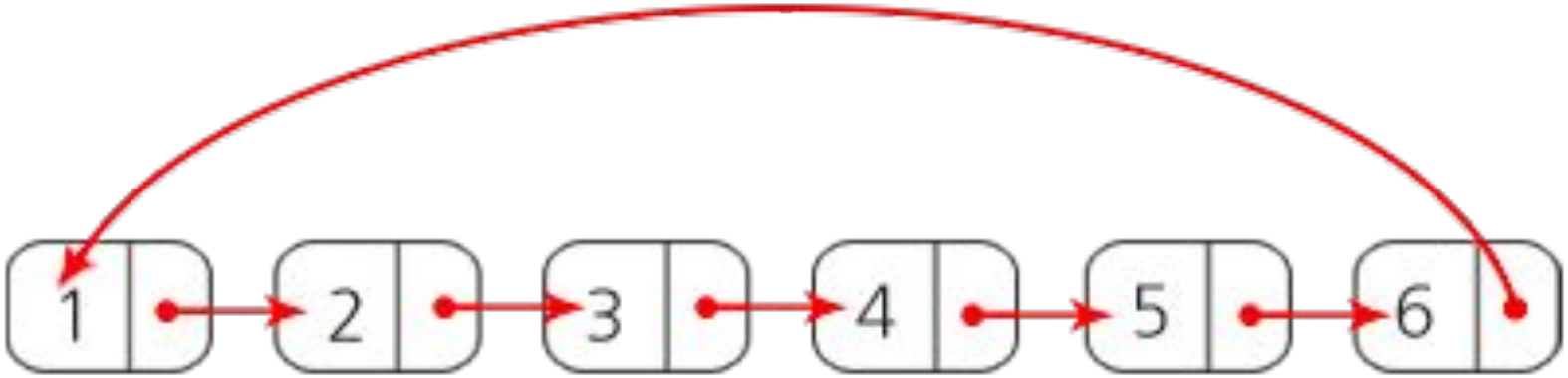
- 구현 목적
  - Linked List 구현과 이해
- 구현 순서
  - 입력 : 917, -504, 326, 138, -64, 263
  - 결과 출력
- 실행결과

insert 917 insert -504 insert 326
Print the whole list: 326 -504 917
delete : -504
Print the whole list: 326 917
insert 138 insert -64 insert 263
Print the whole list: 263 -64 138 326 917
138 is in the list
987 is not in the list
delete : 263
Print the whole list: -64 138 326 917

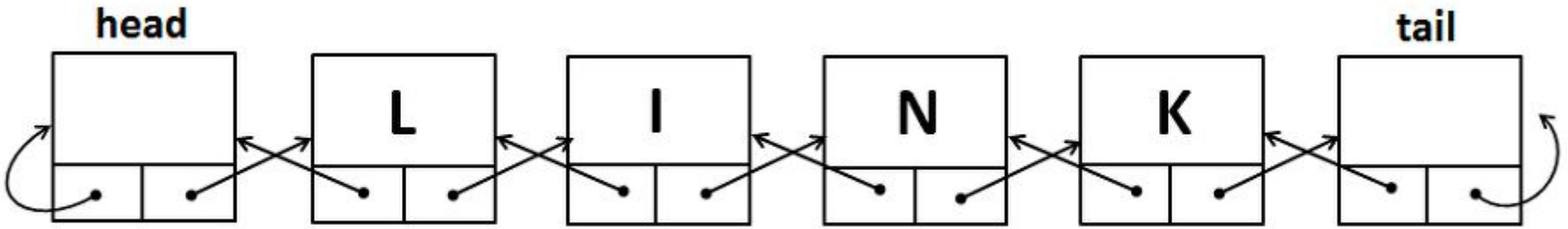
```
struct NODE
{
    struct NODE *next;
    int value;
} *node;
```

리스트(List)

❖ 원형 연결리스트



❖ 양방향 연결 리스트



버블정렬(Bubble Sort)

❖ 인접한 두 개 데이터 비교 정렬.

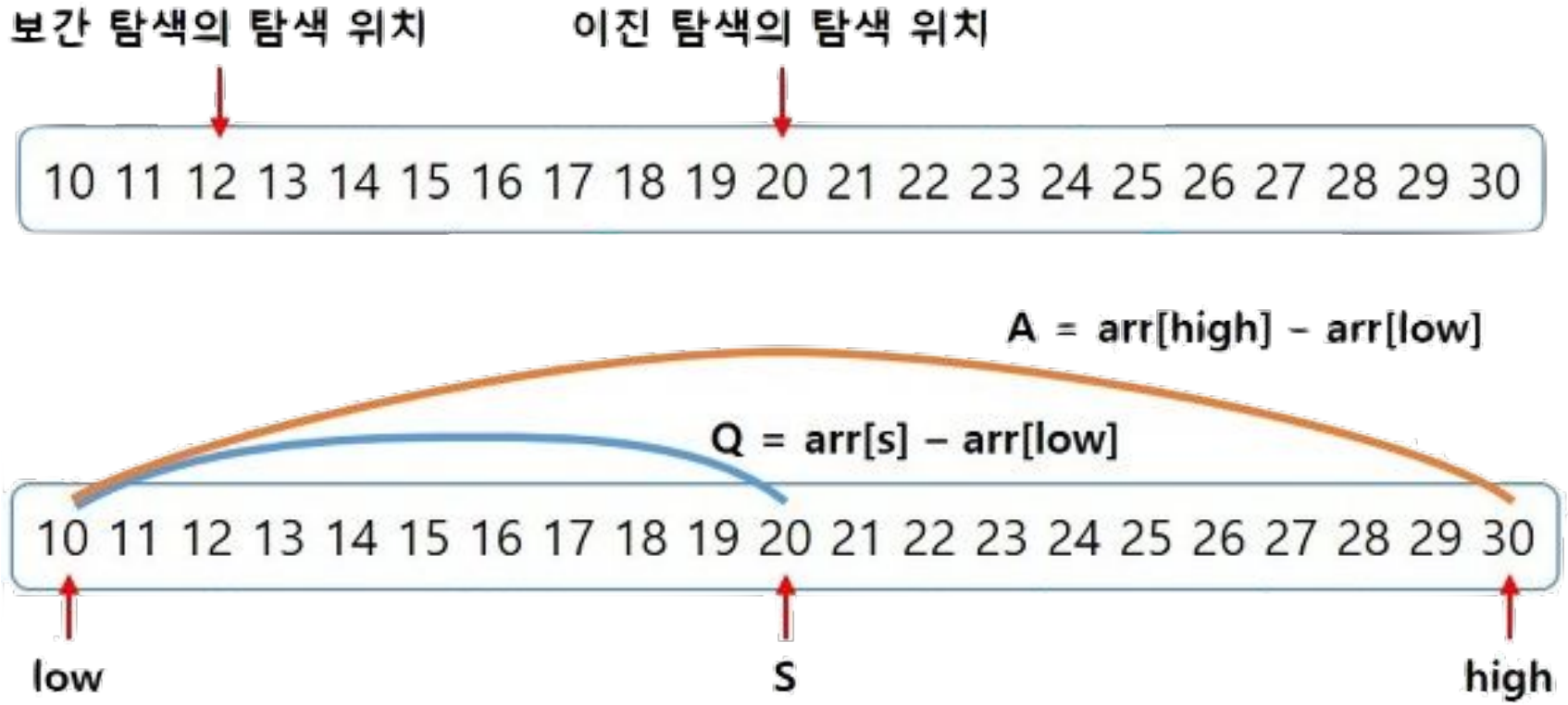
Trying

- 구현 목적
  - Bubble Sort 구현과 이해
- 구현 순서
  - 입력 : int list[10] = {1,8,4,6,0,3,5,2,7,9};
  - 출력 : 결과 반환
- 실행결과

Input Array: [1 8 4 6 0 3 5 2 7 9]
Items compared: [1, 8] => not swapped
Items compared: [8, 4] => swapped [4, 8]
...
Items compared: [8, 9] => not swapped
Iteration 1#: [1 4 6 0 3 5 2 7 8 9]
...
Items compared: [4, 5] => not swapped
Iteration 5#: [0 1 2 3 4 5 6 7 8 9]
...

보간 탐색(interpolation Search)

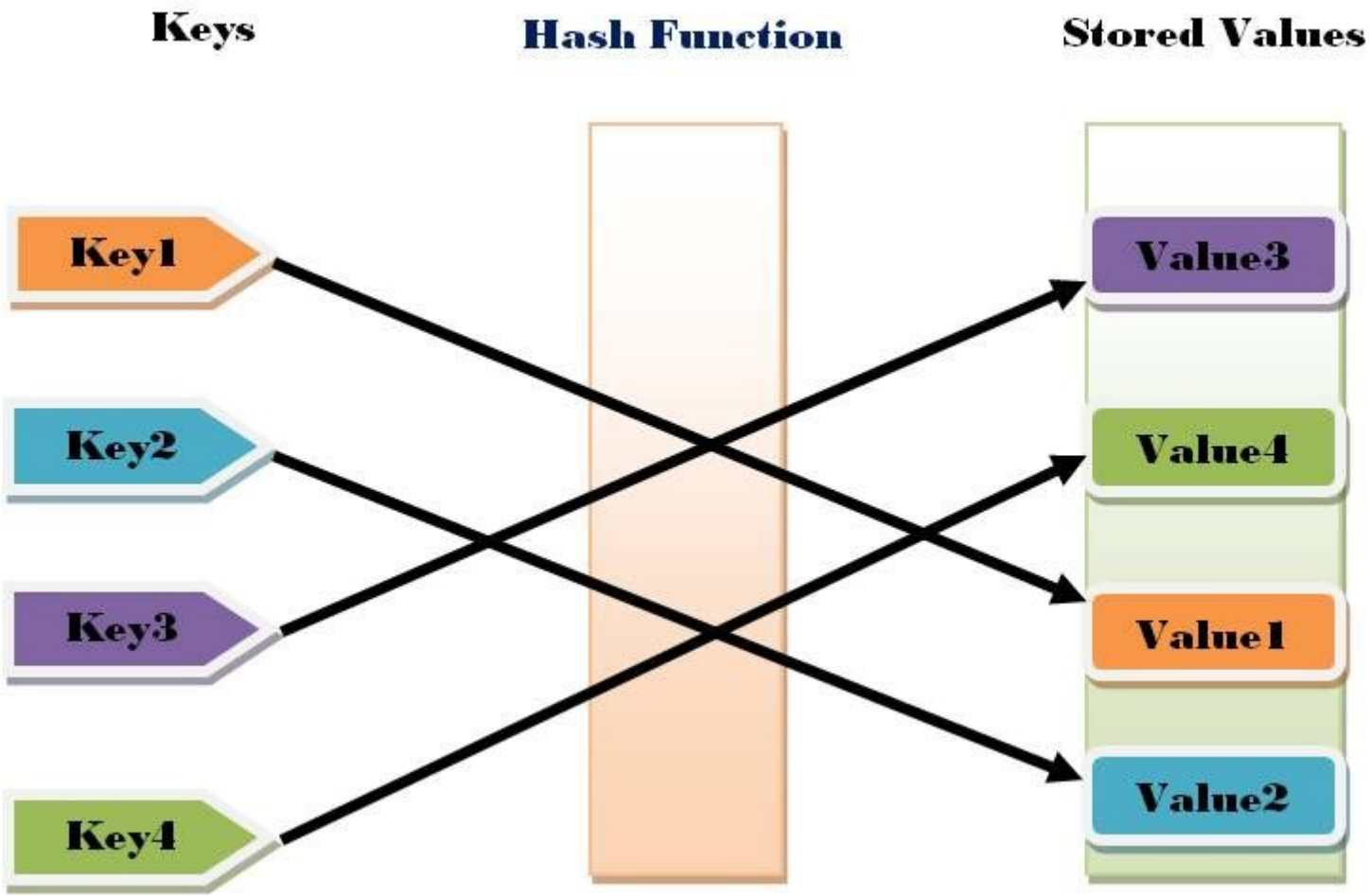
- ❖ 대상이 상대적으로 앞에 존재하면 앞에서 검색.
- ❖ Key and Data : Key have to Unique.



$$S = \frac{x - arr[low]}{arr[high] - arr[low]}(high - low) + low$$

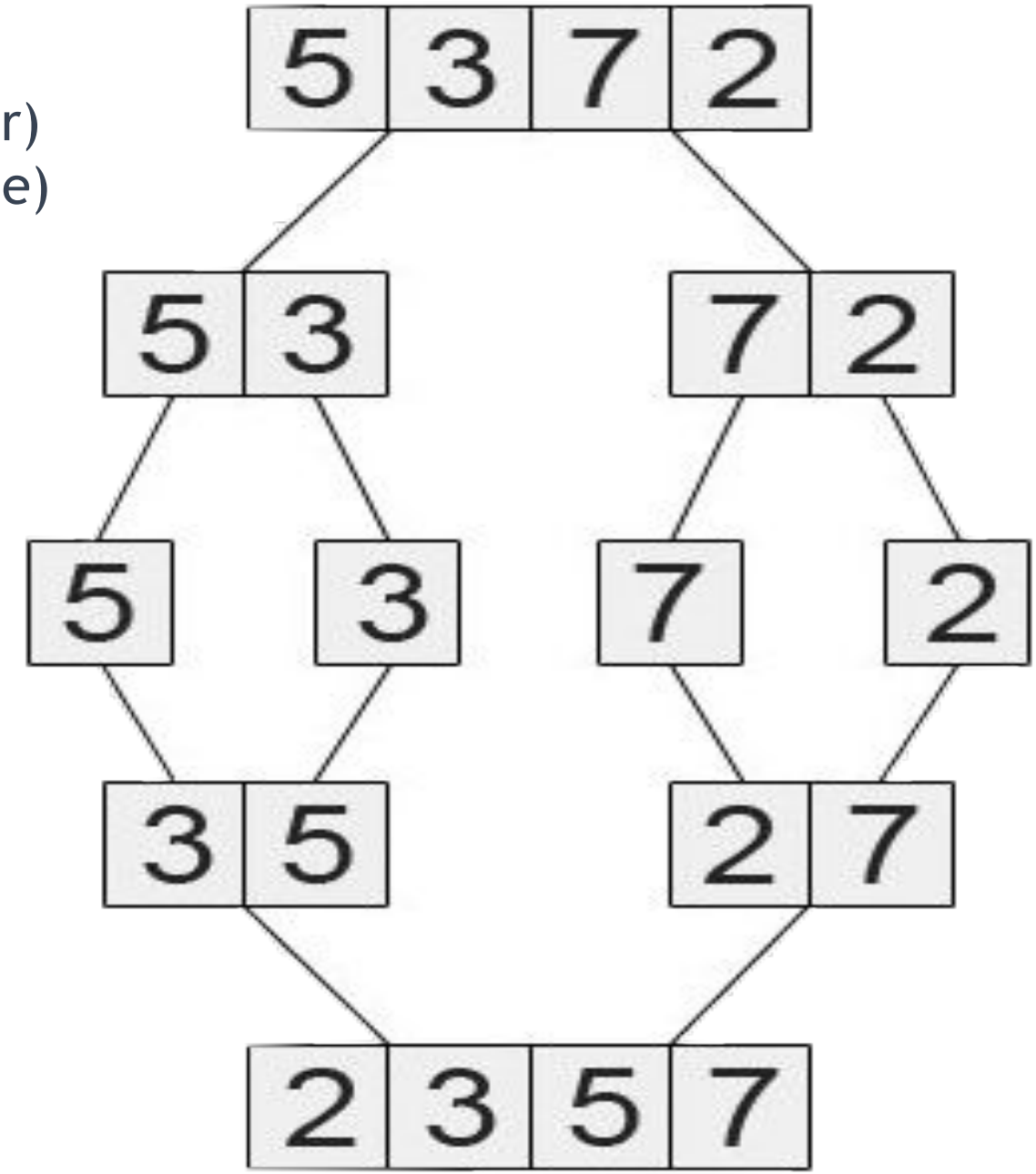
Hash Table

- 시간복잡도 O(1)
- Data = Key + Value



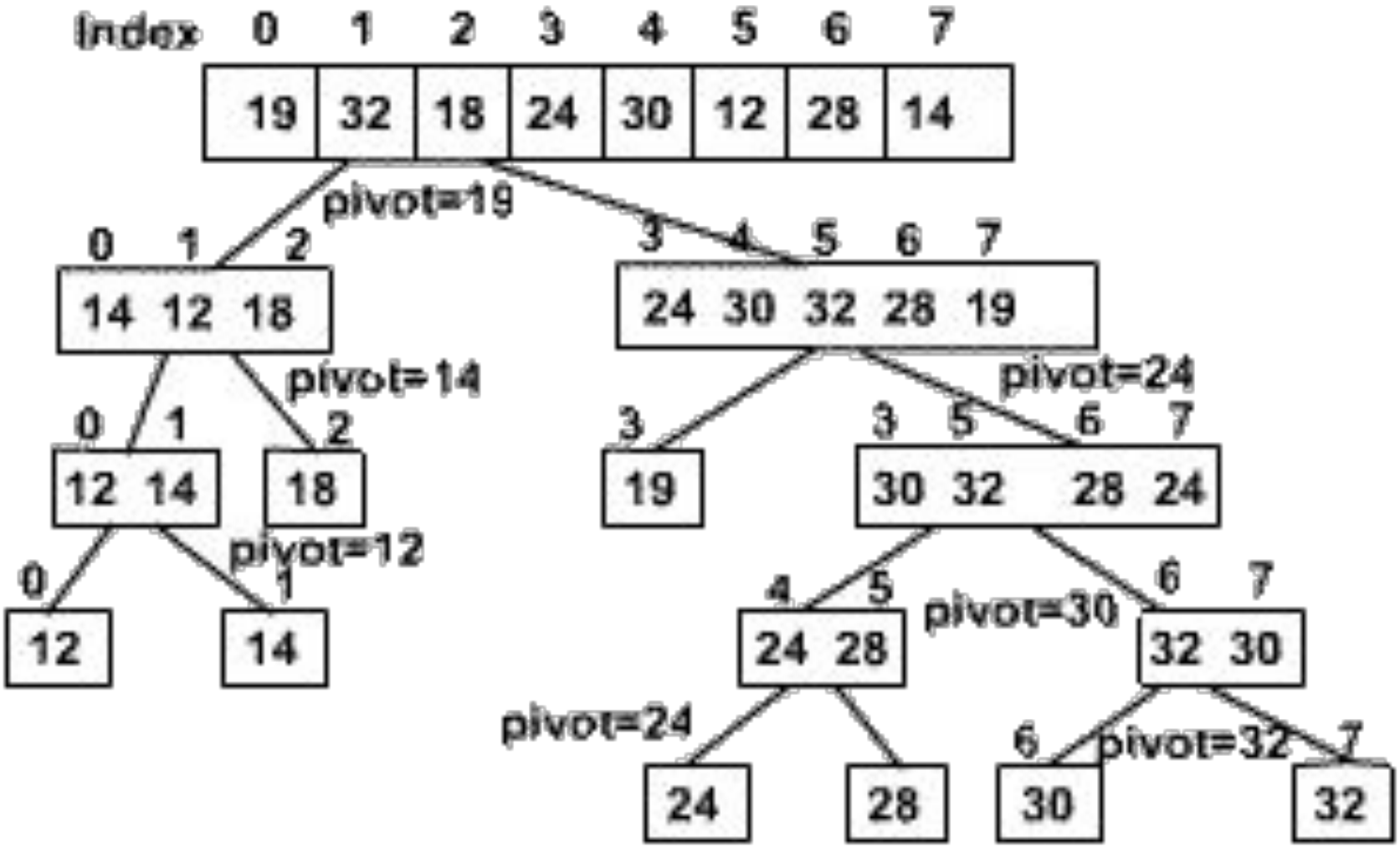
병합 정렬(Merge Sort)

- 1단계 : 분할(Divide)
- 2단계 : 정복(Conquer)
- 3단계 : 결합(Combine)



퀵 정렬(Quick Sort)

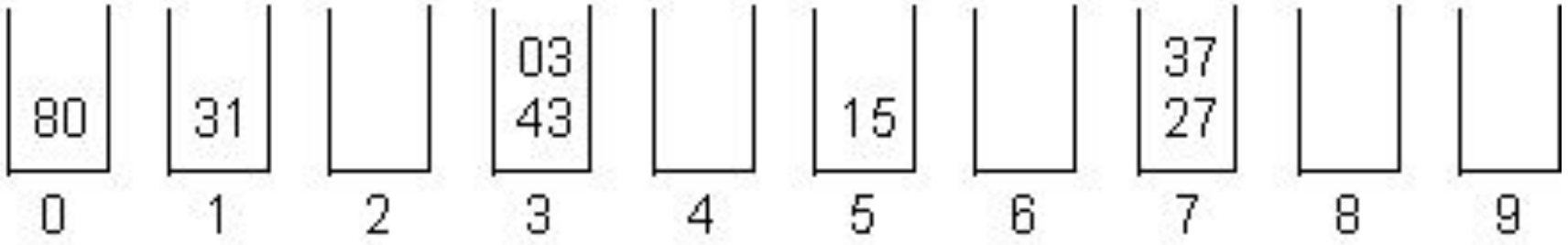
- Pivot
- Left, Right
- Low, High



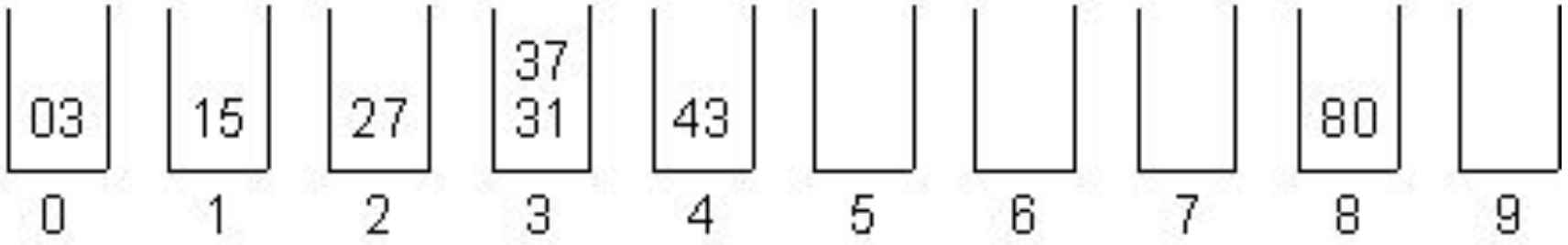
기수 정렬(Radix Sort)

❖ 비교 연산이 없음.

43 27 31 15 37 80 03



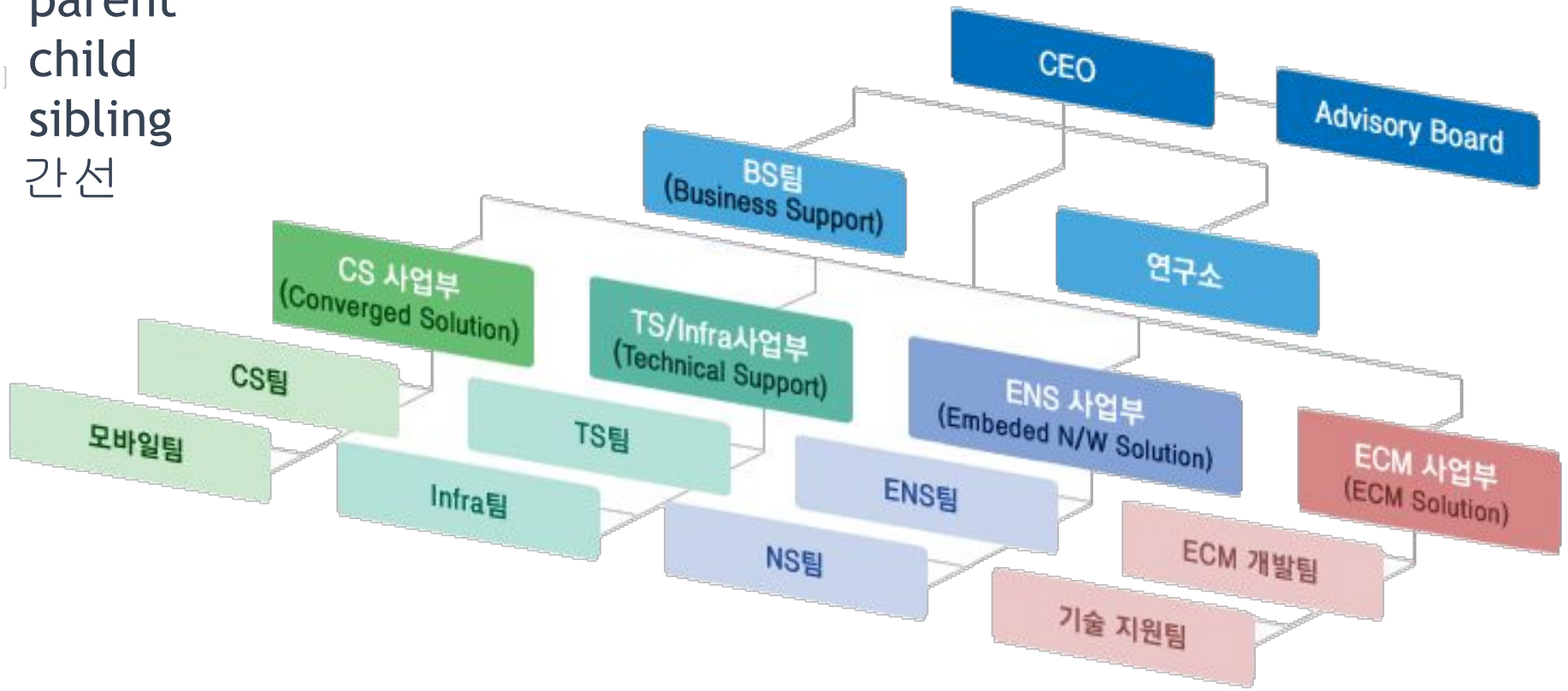
80 31 43 03 15 27 37



03 15 27 31 37 43 80

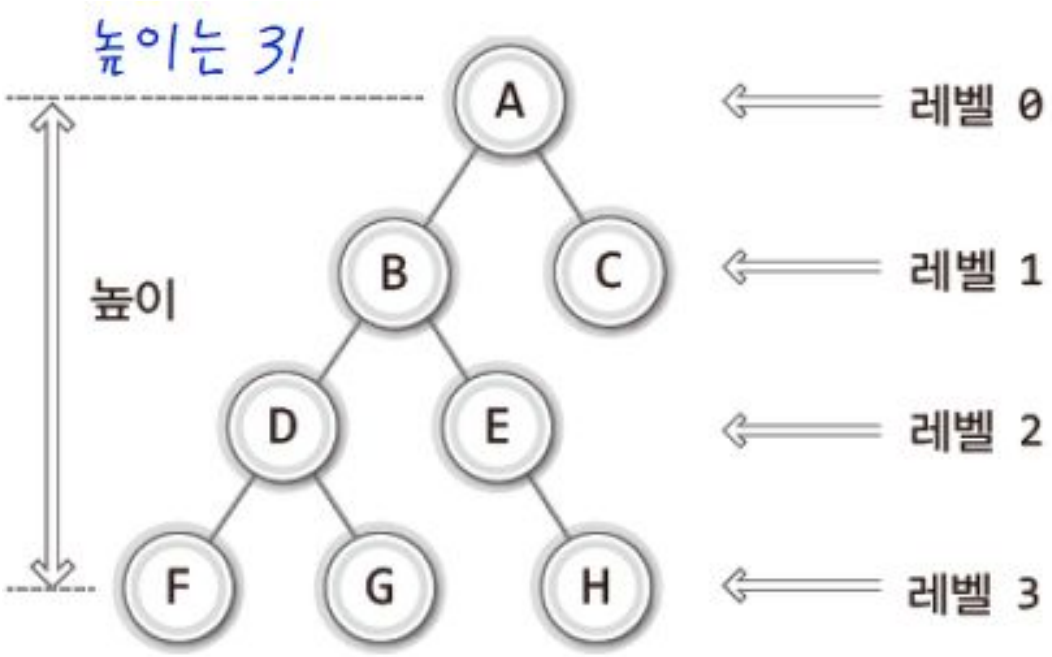
트리(Tree)

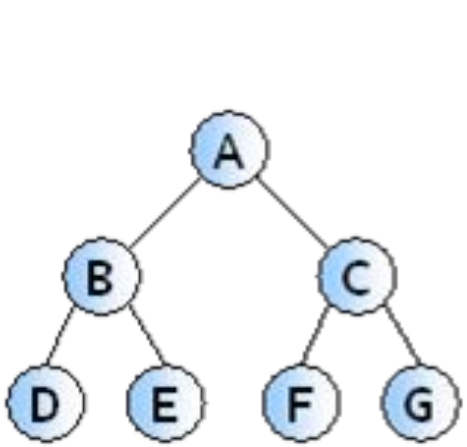
- Hierarchical Relationship
- node : 요소
  - root
  - terminal or leaf
  - internal
  - 관계
    - parent
    - child
    - sibling
- edge : 간선



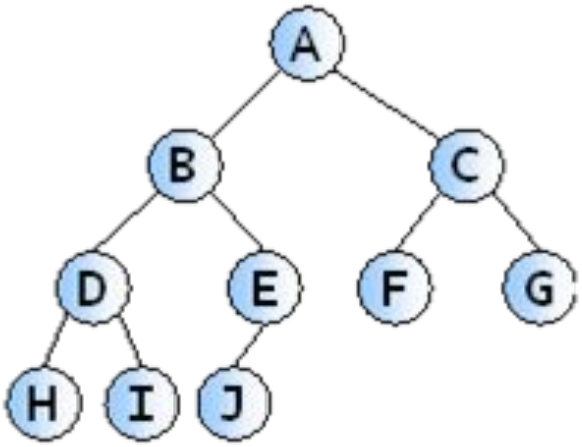
이진트리(Binary Tree)

- 두 개 노드 가짐.
- 레벨, 높이
- 분류
  - Full Binary Tree
  - Complete Binary Tree
- 노드 삭제
  - 단말 노드 상황
  - 하나의 서브 트리 있는 상황
  - 두 개 서브 트리 있는 상황

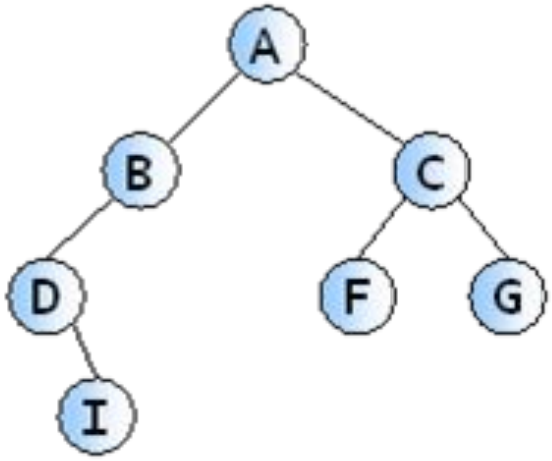




포화이진트리



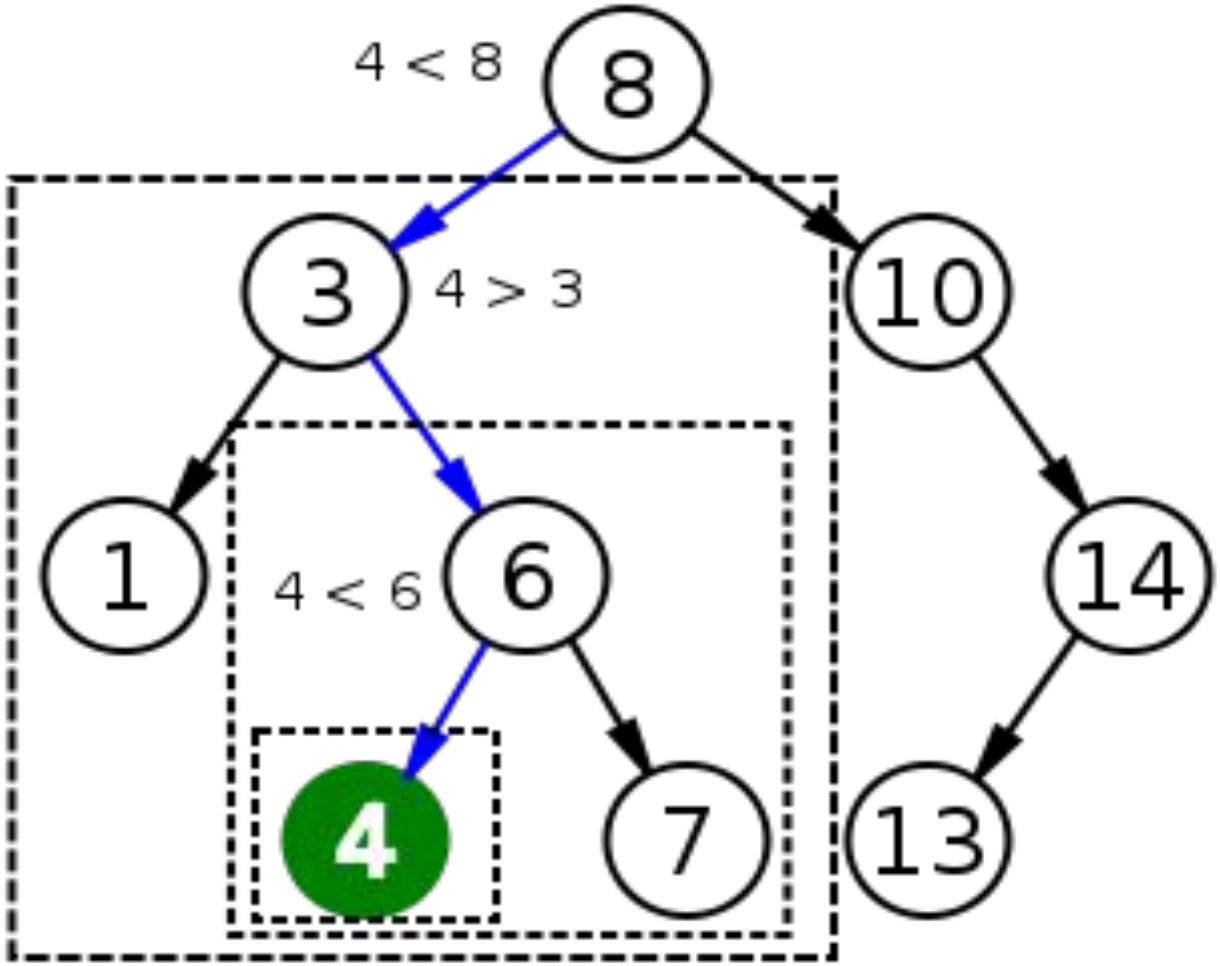
완전이진트리



기타이진트리

이진탐색 트리(Binary Search Tree)

❖ 데이터 저장 규칙
- ➢ 노드 Key는 유일.
- ➢ 루트 노드 Key는 서브 트리 Key보다 크다.



Trying

- 구현 목적
  - 계산기 프로그램 구현
    - 중위 표기법 : left -> parent -> right
    - 전위 표기법 : parent -> left -> right
    - 후위 표기법 : right -> left -> parent
- 구현 순서
  - 입력 : scanf()
  - 출력 : 결과 반환
- 실행결과

Enter the expression in postfix form (ex.426*+)

warning: this program uses gets(), which is unsafe.

4249+*-

The value of the postfix expression you entered is -22

The inorder traversal of the tree is

4-2*4+9

Trying

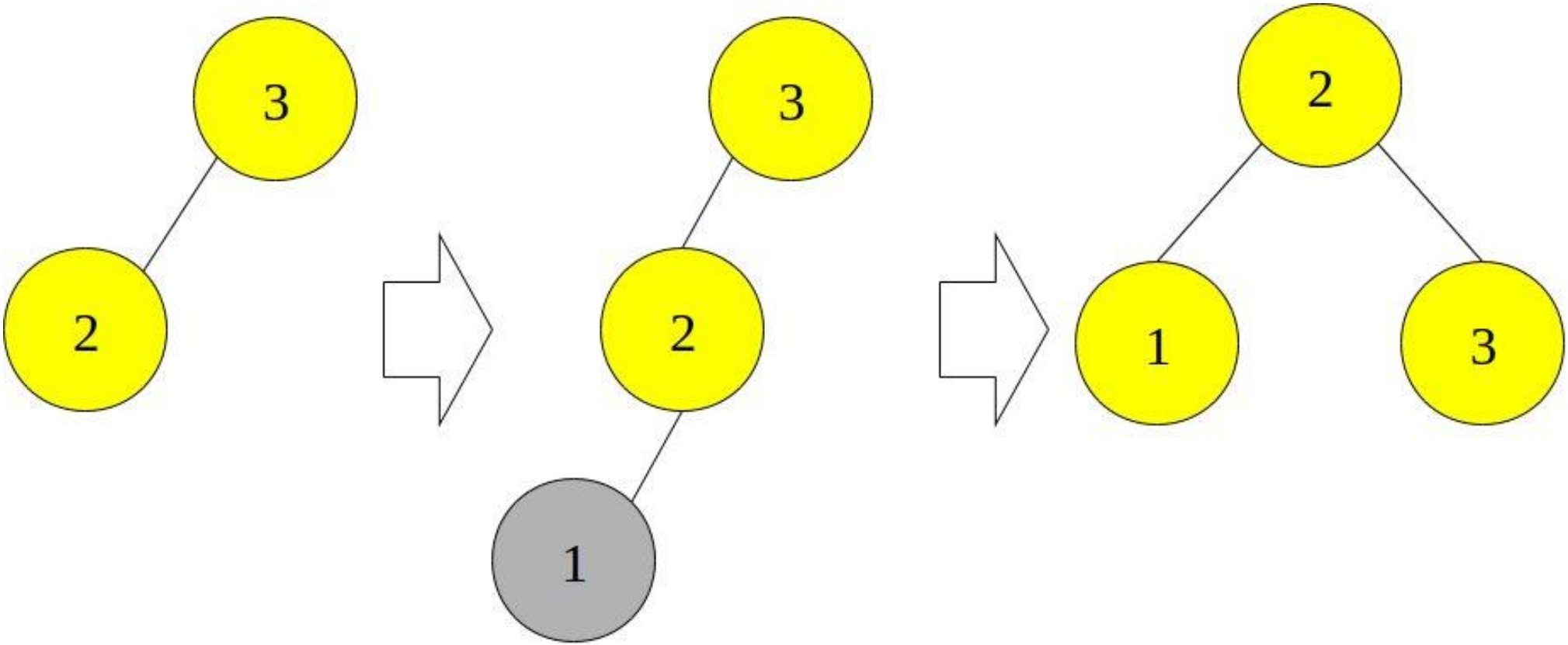
- 구현 목적
  - ➢ 이진트리 각종 정렬 구현과 이해
- 구현 순서
  - ➢ 입력 : 9, 4, 15, 6, 12, 17, 2
  - ➢ 출력 : 전위, 중위, 후위 방식 표시
- 실행결과

InPut Number 9, 4, 15, 6, 12, 17, 2
Pre Order Display
9 4 2 6 15 12 17
In Order Display
2 4 6 9 12 15 17
Post Order Display
2 6 4 12 17 15 9
Searched node=9

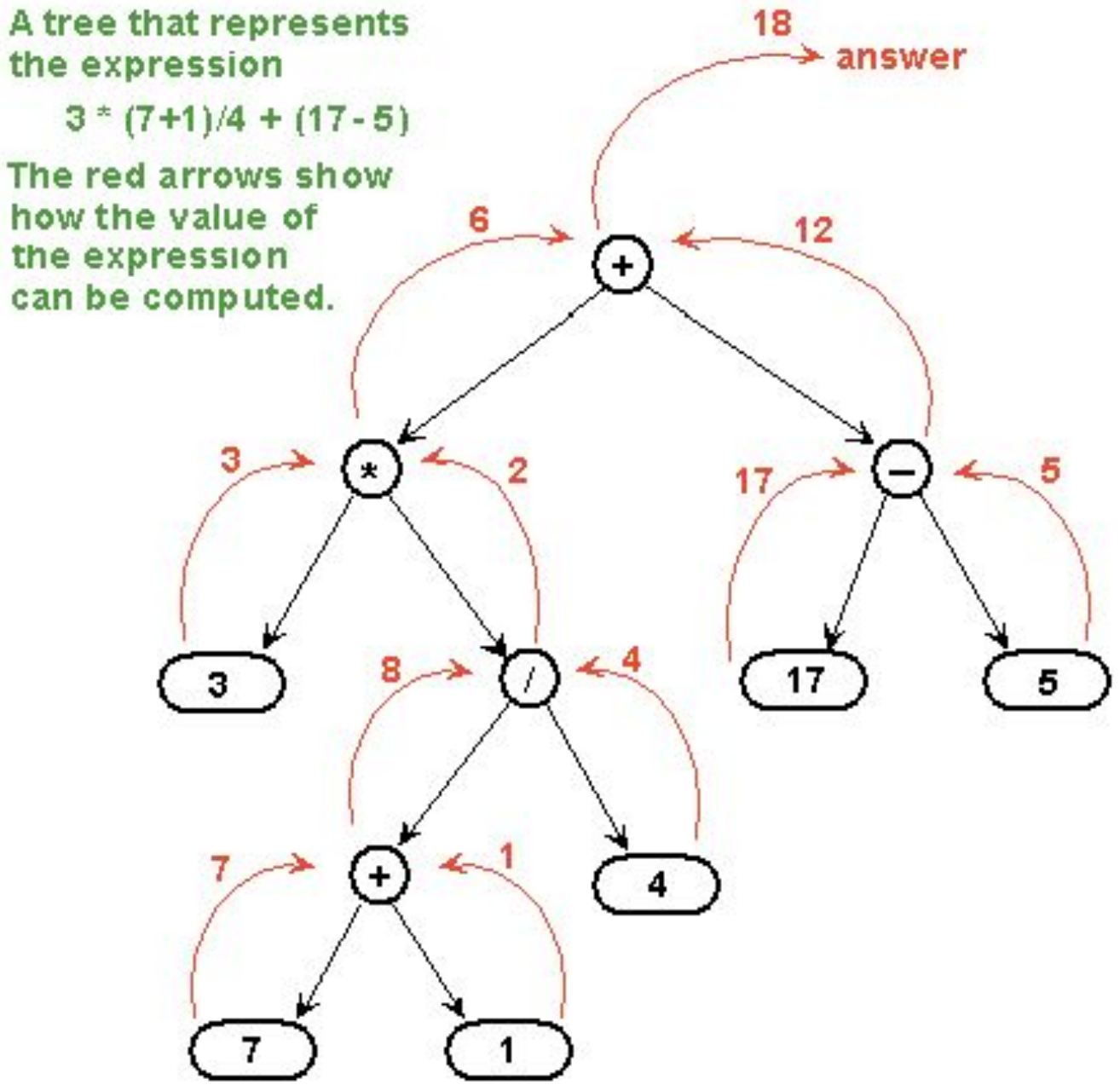
```
struct bin_tree {
    int data;
    struct bin_tree * right, * left;
};
```

AVL 트리

- 이진 트리 단점 보완.
  - 2-3 트리
  - 2-3-4 트리
  - Red-Black 트리
  - B 트리
- 균형인수 = 왼쪽 서브 트리 높이 - 오른쪽 서브 트리 높이

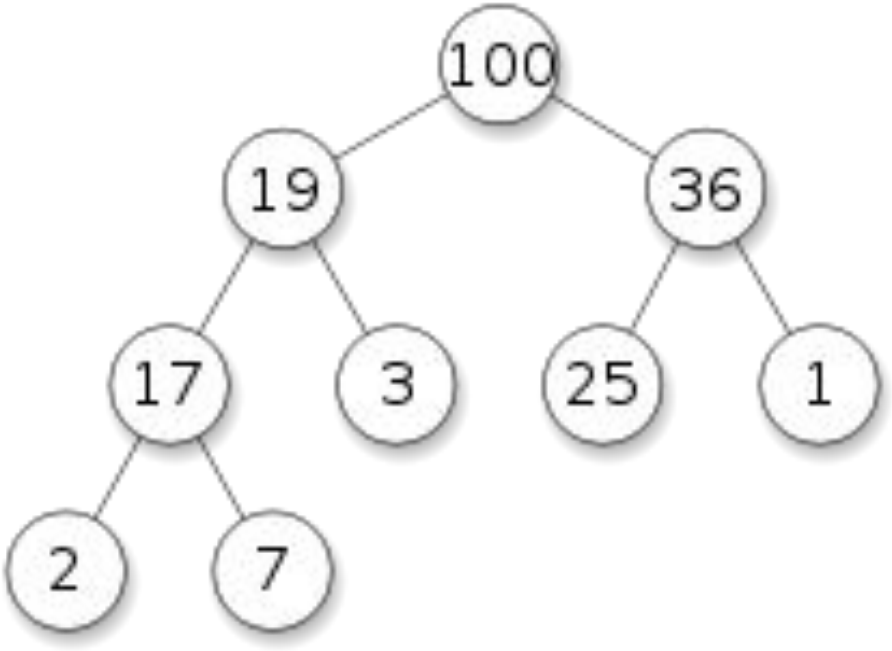


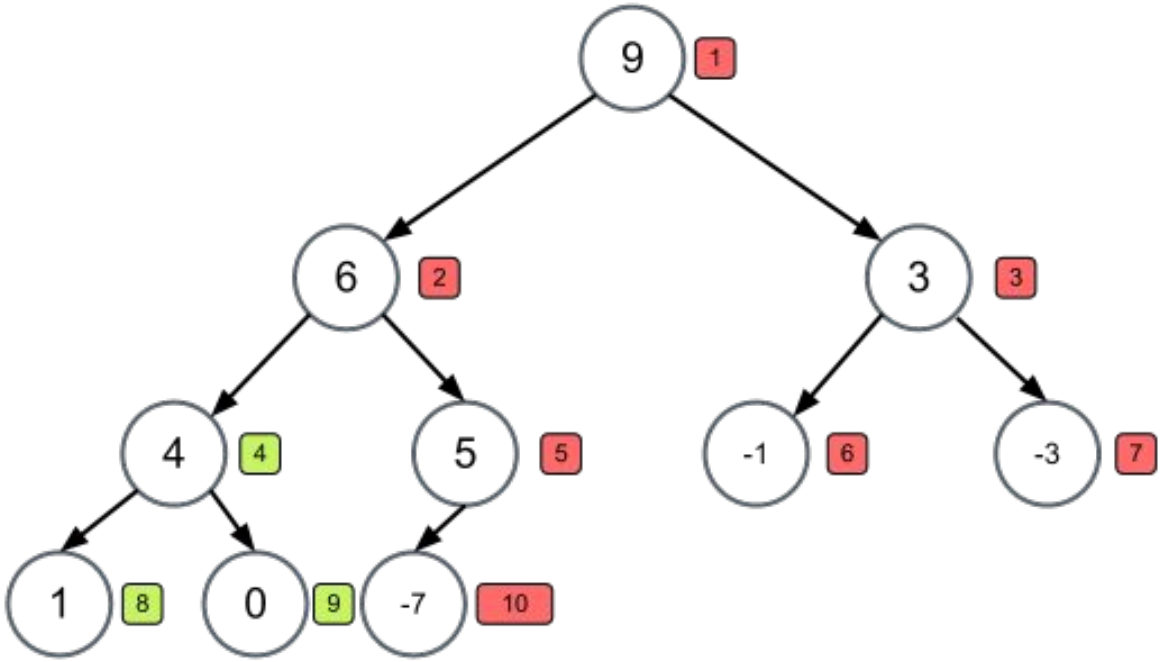
수식트리(Expression Tree)



힙(Heap)

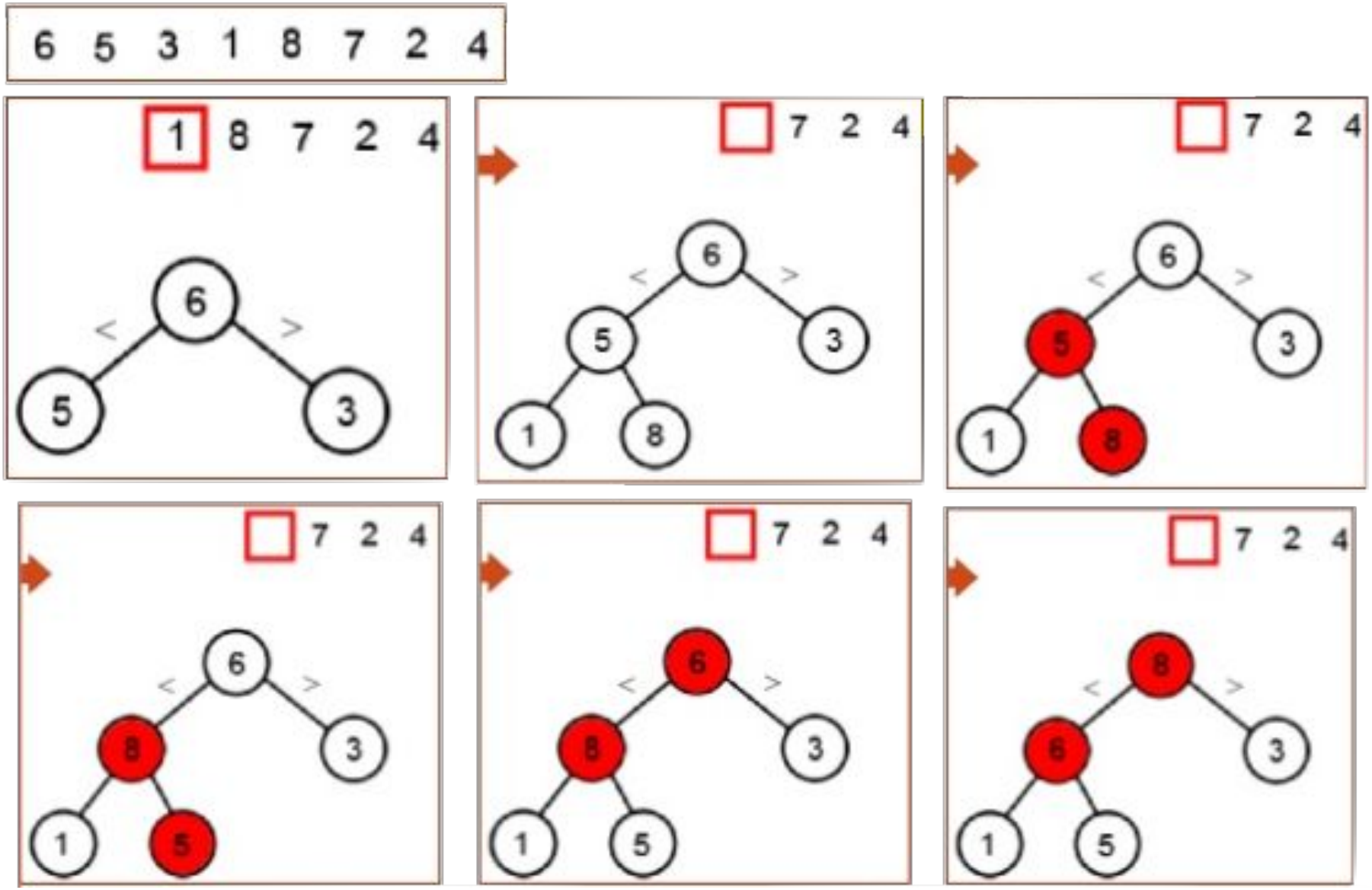
- 완전 이진 트리
- Parent node >= Child node
- Max heap, Min heap





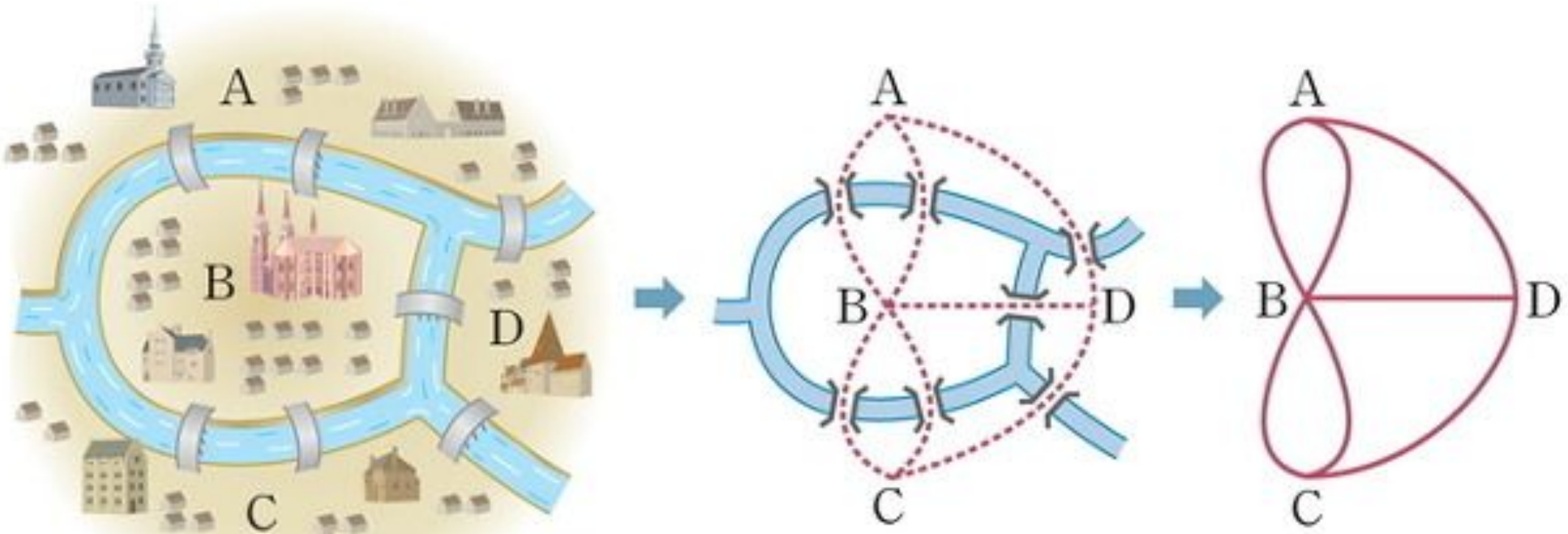
힙정렬(Heap Sort)

❖ 루트 노드에 저장된 값이 정렬순서상 가장 앞섬.



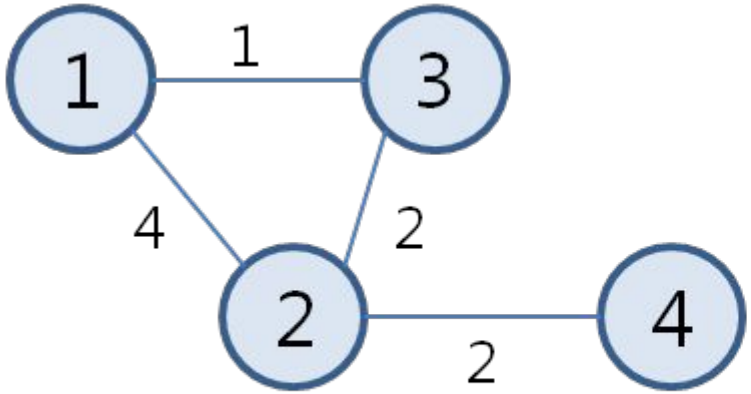
쾨니히스베르의 다리 문제

- 오일러가 1936년 풀기 위해 그래프 이론 적용.
- 한붓그리기 : 홀수점 = 0 or 2
- 해밀턴 회로 발전

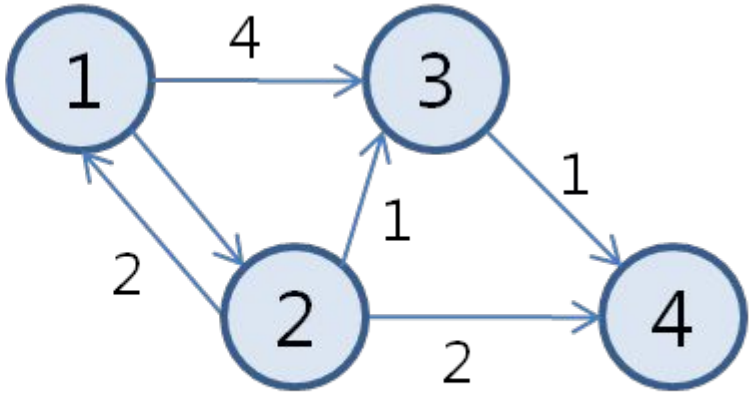


그래프 종류

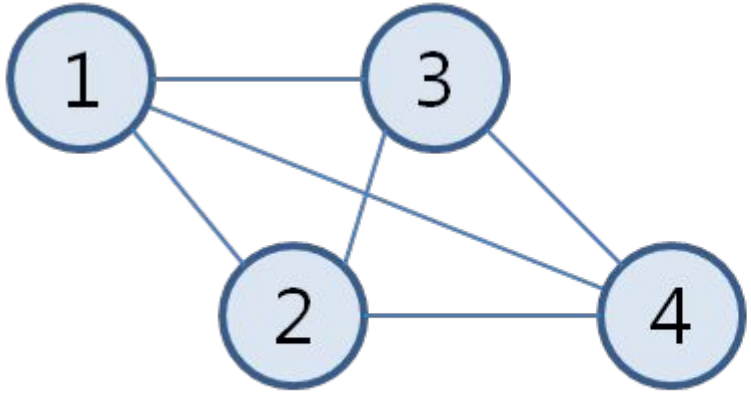
❖ 정점과 간선.



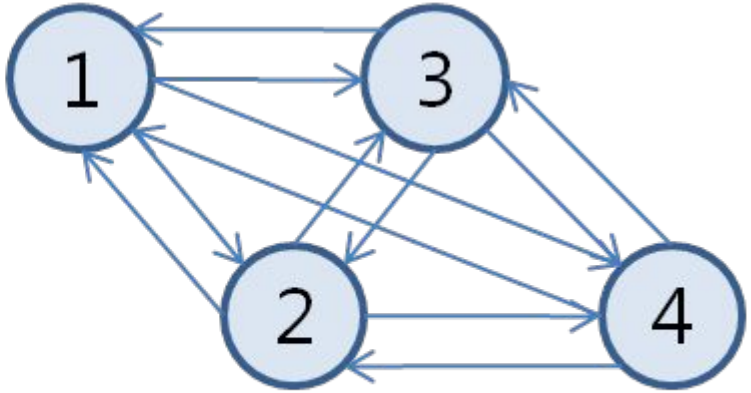
무방향 가중 그래프



방향 가중 그래프(=네트워크)



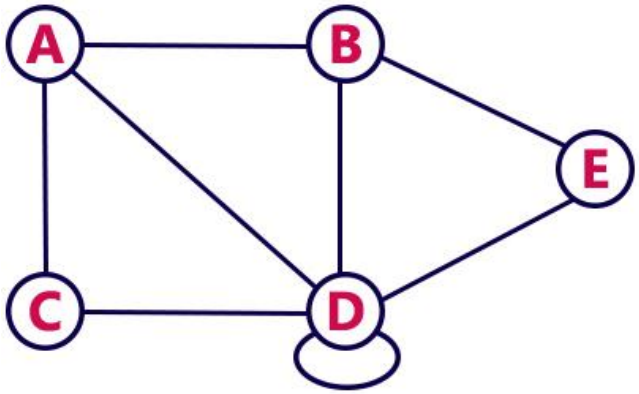
무방향 완전 그래프



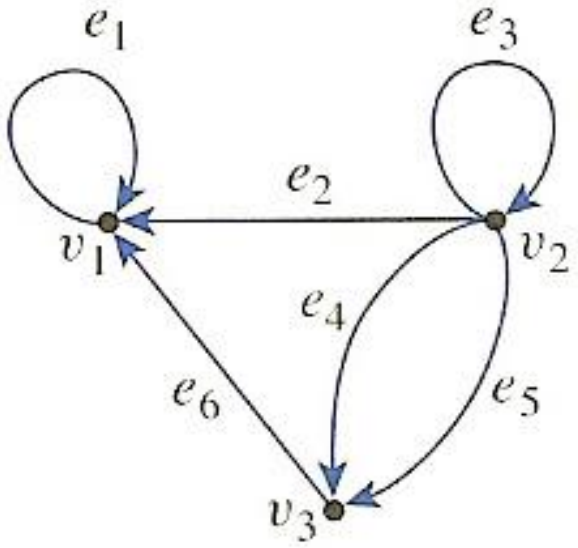
방향 완전 그래프

그래프 구현(1)

❖ 인접행렬(Adjacent matrix) - 정방 행렬 활용



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0



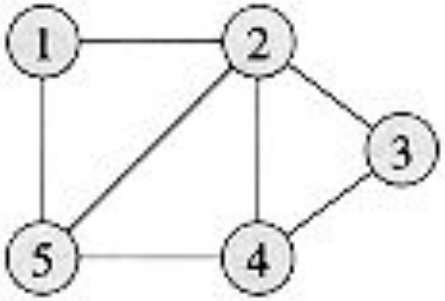
Directed Graph G

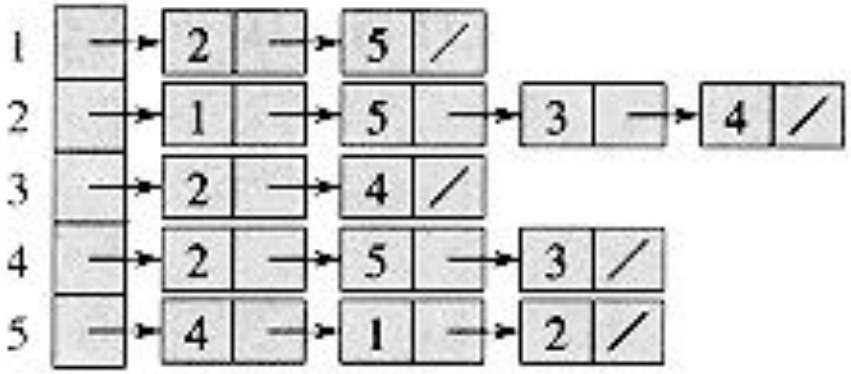
$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

Adjacency Matrix

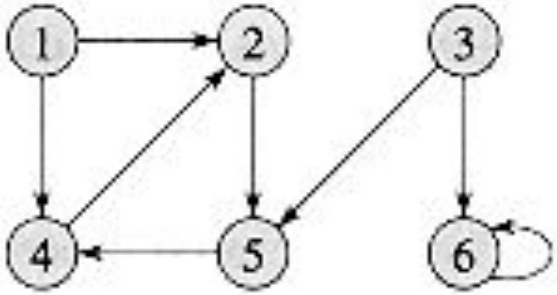
그래프 구현(2)

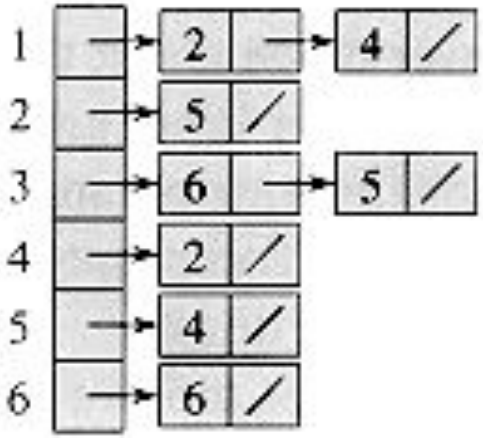
❖ 인접 리스트(Adjacent list) - 연결 리스트 활용





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





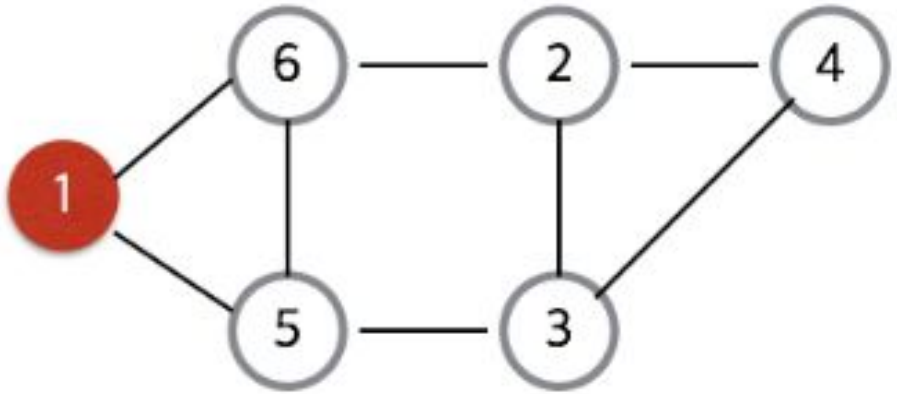
	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

깊이 우선 탐색(DFS:Depth First Search)

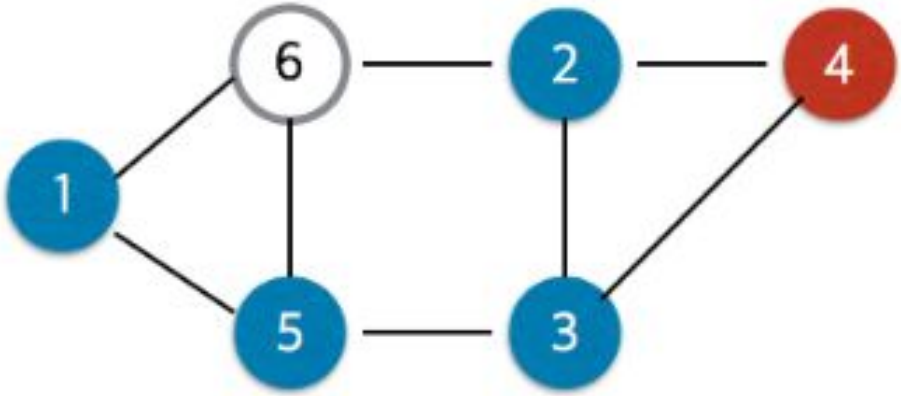
- 맹목적 탐색방법
- 접근 막히면 미 접근 정점

현재 위치 이미 방문한 정점

순서	1					
스택	1					



순서	1	5	3	2	4	
스택	1	5	3	2	4	

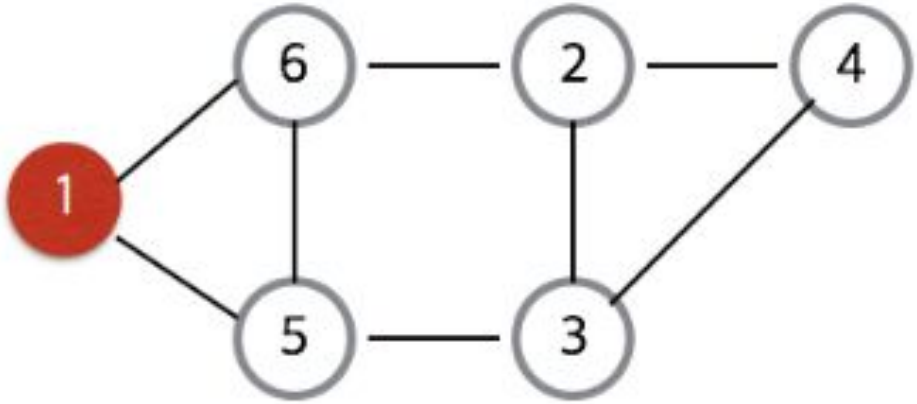


넓이 우선 탐색(BFS:Breadth First Search)

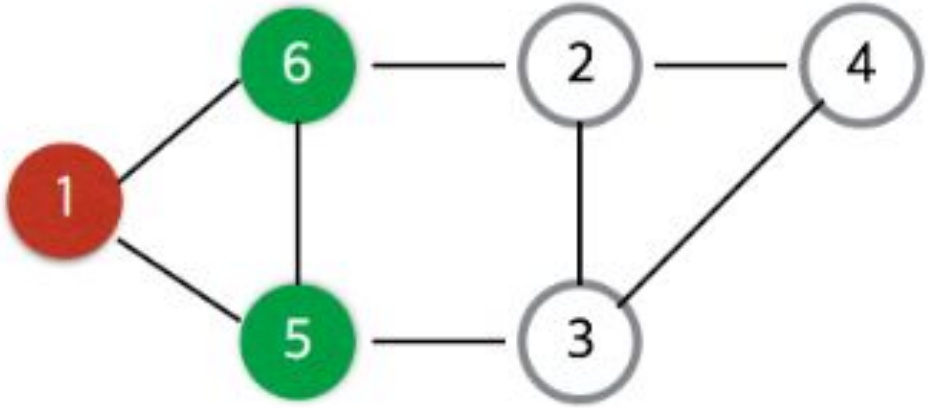
❖ 시작 정점에 인접한 모든 정점들을 우선 방문하는 방법

현재 위치 이미 방문 방문 가능

큐	1					
결과						

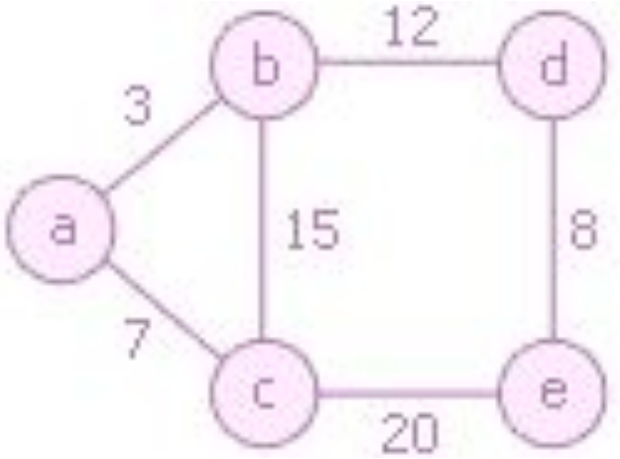


큐		5	6			
결과	1					

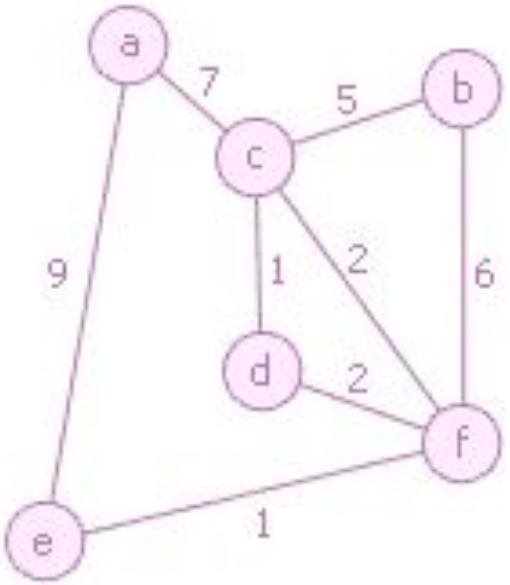


최소비용 신장 트리

❖ 크루스칼(Kruskal) 알고리즘 : 최소 비용 신장 트리
  ➢ 연결선에 가중치를 부여



❖ 프림(Prim) 알고리즘
  ➢ 가장 작은 가중치를 갖는 연결선을
  선택하여 최소 비용 신장 트리 MST 추가



알고리즘 성능분석

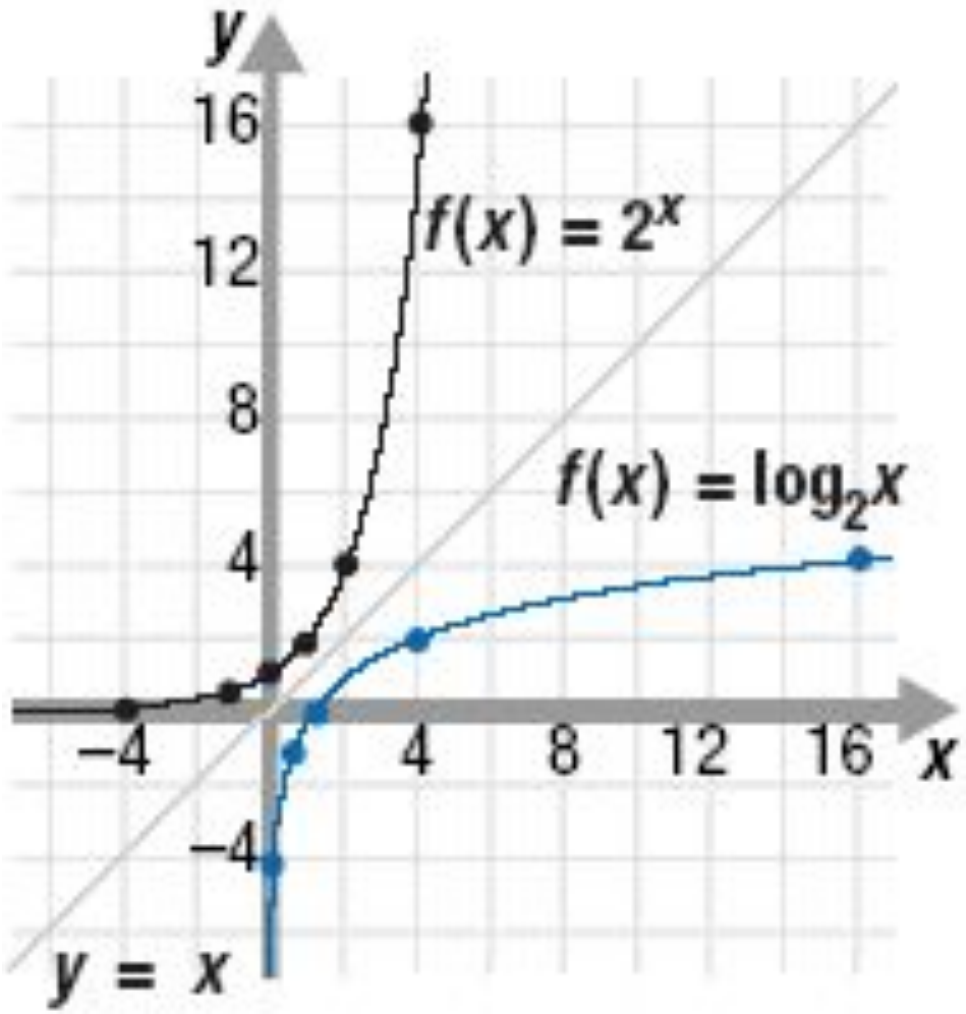
- 정확도(correctness)
- 단순성(simplicity)
- 복잡도(complexity)

```
# include <stdio.h>
int main(void){
    printf("Hello \n");
    printf("Hello \n");
    printf("Hello \n");
    printf("Hello \n");
    printf("Hello \n");
    printf("Hello \n");
    printf("Hello \n");

    return 0;
}
```

```
# include <stdio.h>
int main(void){
    int i ;
    for(i=0; i<7; i++){
        printf("Hello \n");
    }

    return 0;
}
```

알고리즘 복잡도 분석

- 시간 복잡도(Time Complexity) : Computation Time
- 공간 복잡도(Space Complexity) : Storage requirement , space complexity



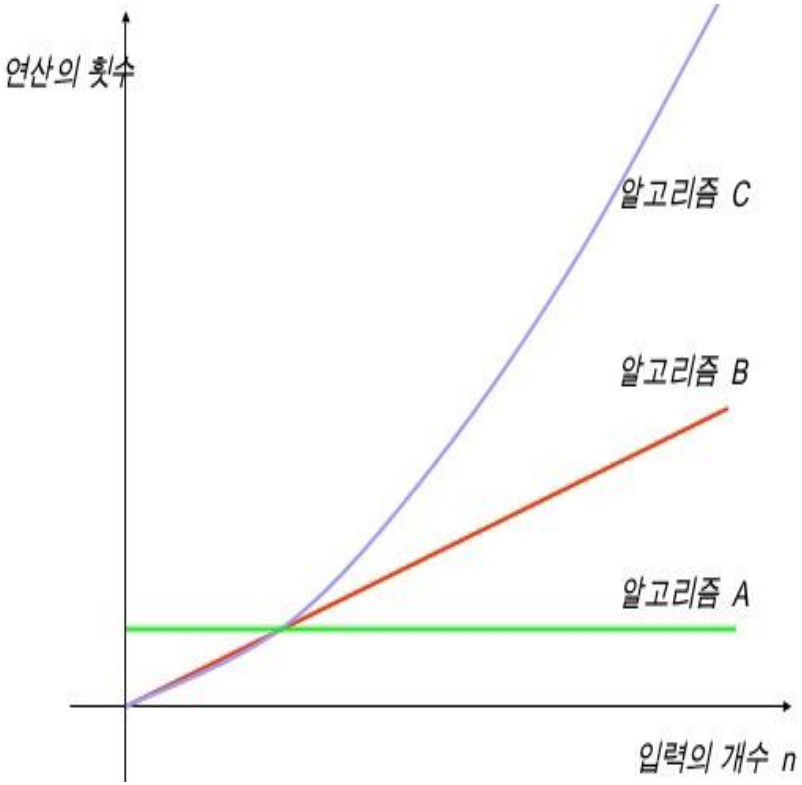
시간 복잡도 분석

- 연산 횟수 표시
- 산술, 대입, 비교, 이동의 기본적인 연산
- 연산 개수 계산하여 두 알고리즘 비교
- T(n)으로 표시

알고리즘 A	알고리즘 B	알고리즘 C

Sum <- n*n	Sum<-0
	For i<- to n do
	sum<-sum+n;
		Sum <-0
		For i<- 1 n to do
		for j <- 1 to n do
		sum<-sum +1;

	알고리즘 A	알고리즘 B	알고리즘 C
대입 연산	1	N+1	N*N+1
덧셈 연산		N	N*N
곱셈 연산	1		
나눗셈 연산			
전체 연산 수	2	2N+1	2N^+1

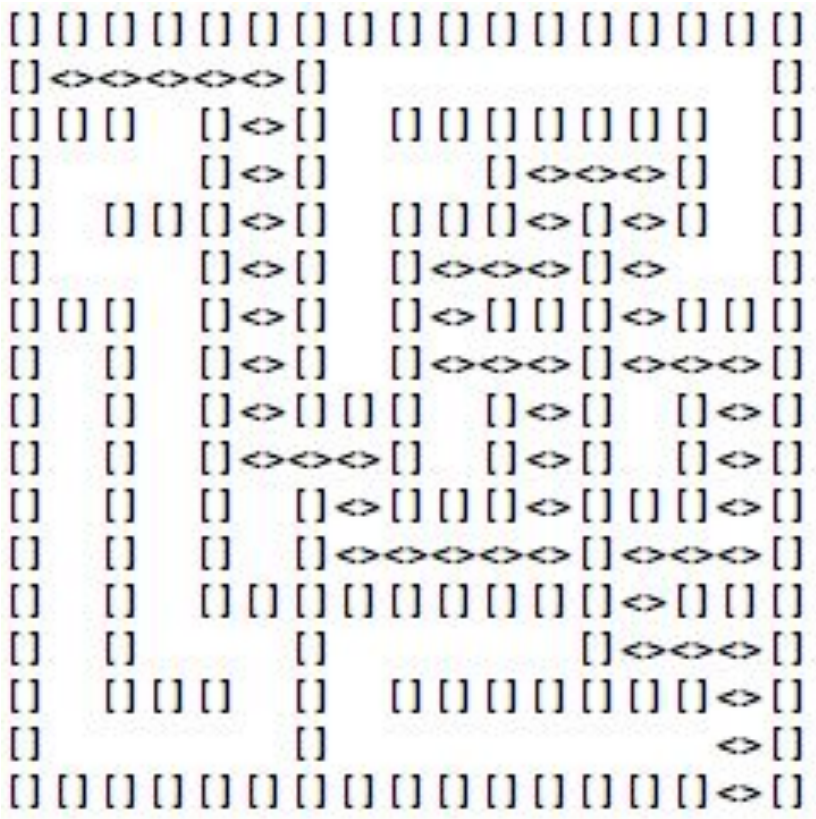


빅오 표기법(big-oh notation)

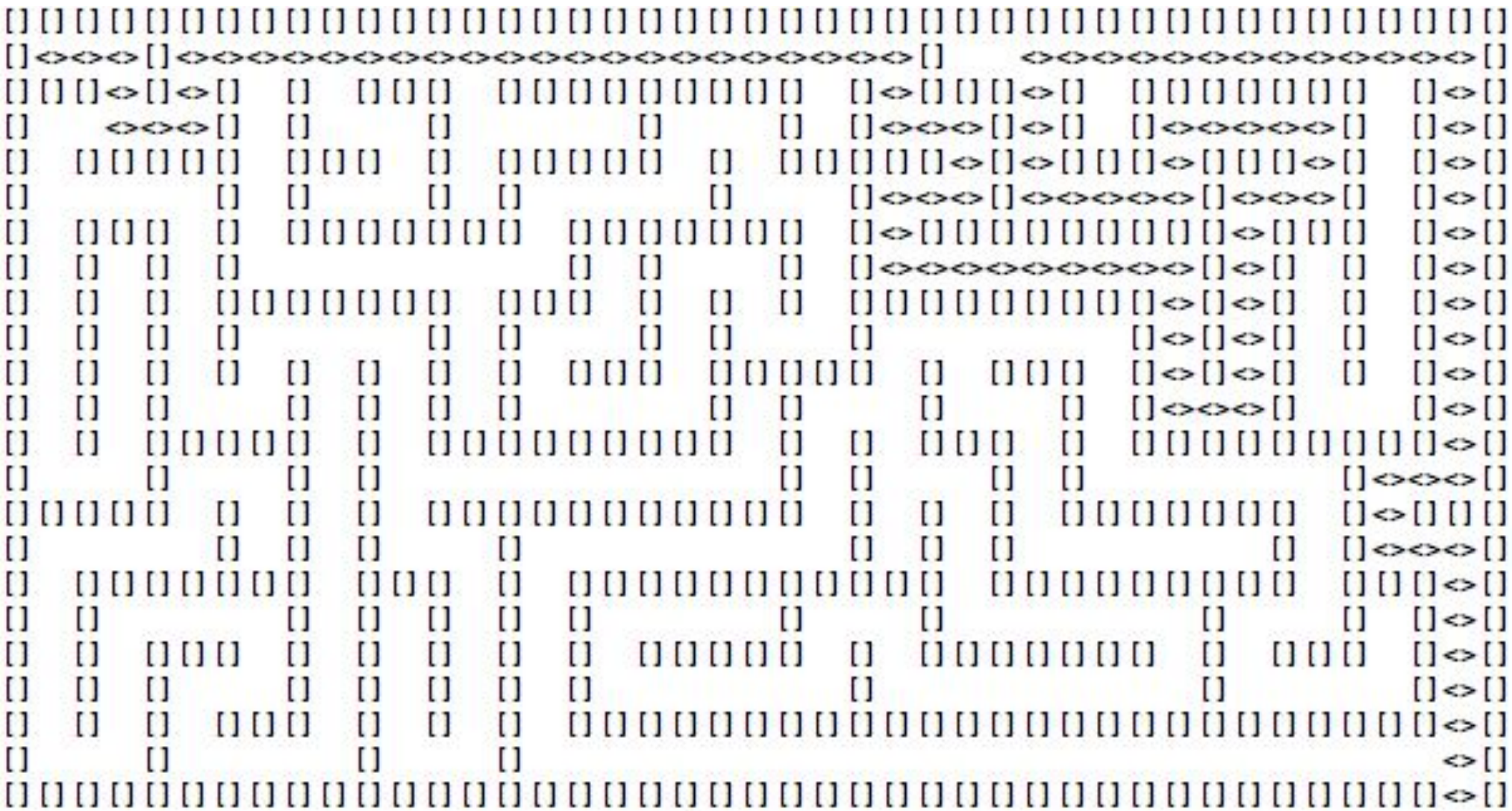
- O(1) : 입력 자료의 수에 관계없이 일정한 실행 시간을 갖는 알고리즘 (상수형)
- O(log N) : 입력 자료의 크기가 N일경우 $\log_2 N$ 번만큼의 수행시간을 가진다. (로그형)
- O(N) : 입력 자료의 크기가 N일경우 N 번만큼의 수행시간을 가진다. (선형)
- O(N log N) : 입력 자료의 크기가 N일경우 $N*(\log_2 N)$ 번만큼의 수행시간을 가진다. (선형로그형)
- $O(N^2)$: 입력 자료의 크기가 N일경우 N^2 번만큼의 수행시간을 가진다. (2차형)
- $O(N^3)$: 입력 자료의 크기가 N일경우 N^3 번만큼의 수행시간을 가진다. (3차형)
- $O(2^n)$: 입력 자료의 크기가 N일경우 2^N 번만큼의 수행시간을 가진다. (지수형)
- O(n!) : 입력 자료의 크기가 N일경우 n*(n-1)*(n-2)... * 1(n!) 번만큼의 수행시간을 가진다.(팩토리얼형)

Trying

- 구현 목적
  - 구현과 이해
- 구현 순서
  - 입력 : 미로 블럭
  - 출력 : 미로 찾기
- 실행결과



최종결과물




수고하셨습니다.