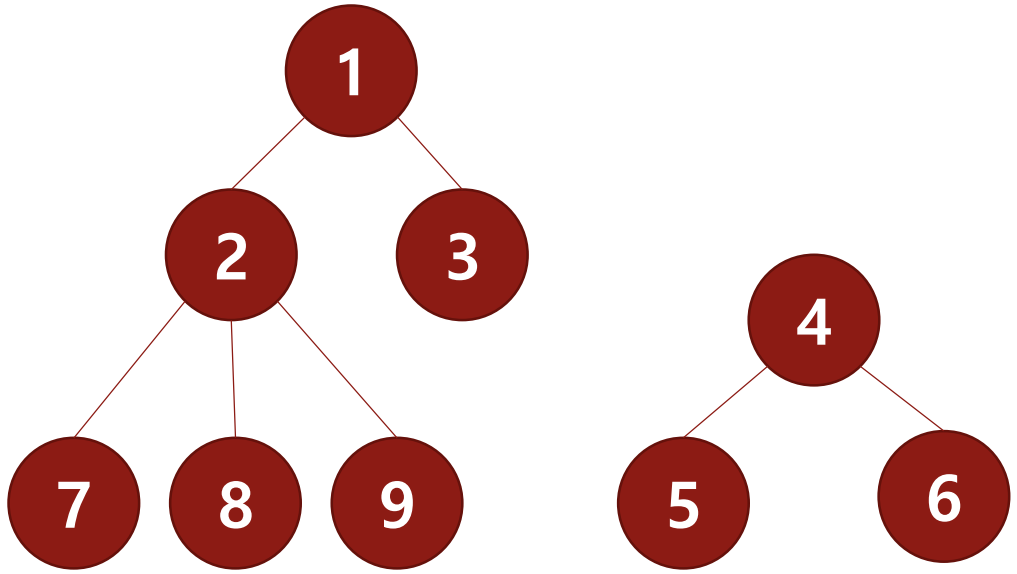
08. Tree

Div. 3 알고리즘 스터디 / 임지환

#2644 촌수 계산

- 입력 예시 :

```
9
7 3
7
1 2
1 3
2 7
2 8
2 9
4 5
4 6
```



#2644 촌수계산

- 부모는 반드시 한명일 수밖에 없다.

- 아래에서 위로 가는 경로는 하나일 수밖에 없고,
- 마찬가지로 위에서 아래로 가는 경로 또한 하나일 수밖에 없다.

#2644 촌수계산

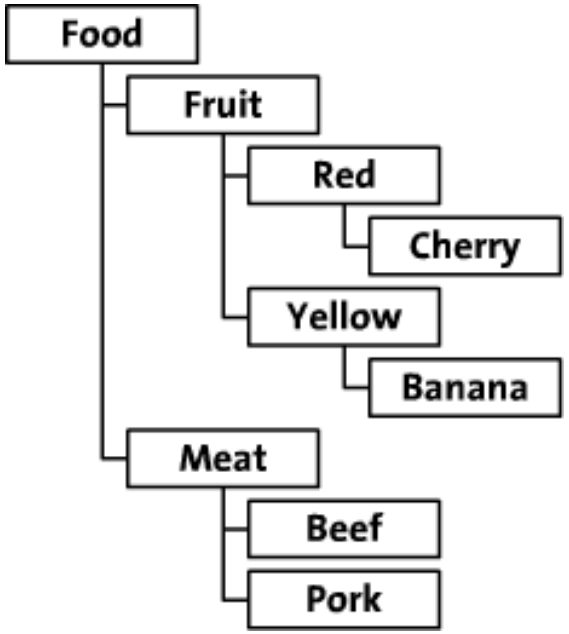
- 부모는 반드시 한명일 수밖에 없다.

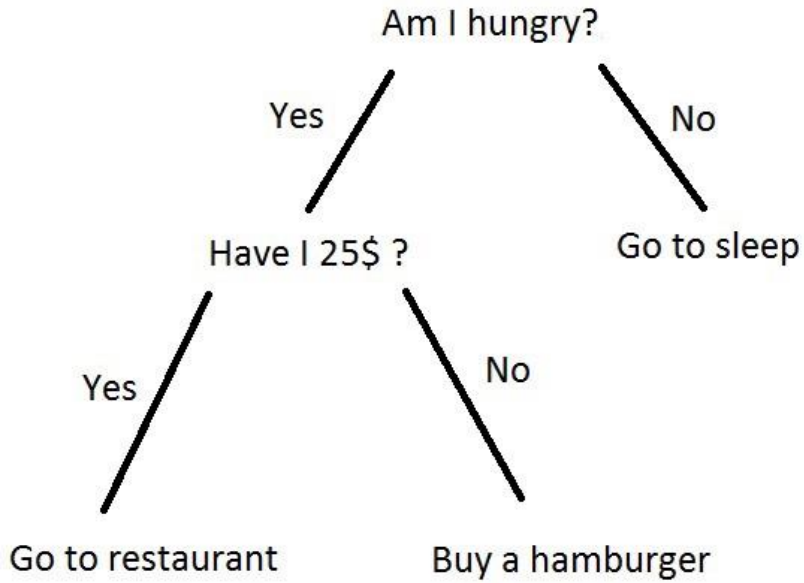
  ⇒트리

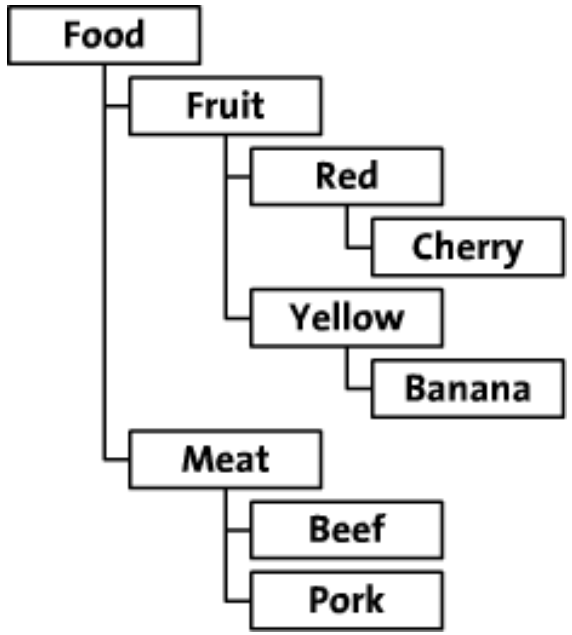
- 아래에서 위로 가는 경로는 하나일 수밖에 없고,
- 마찬가지로 위에서 아래로 가는 경로 또한 하나일 수밖에 없다.

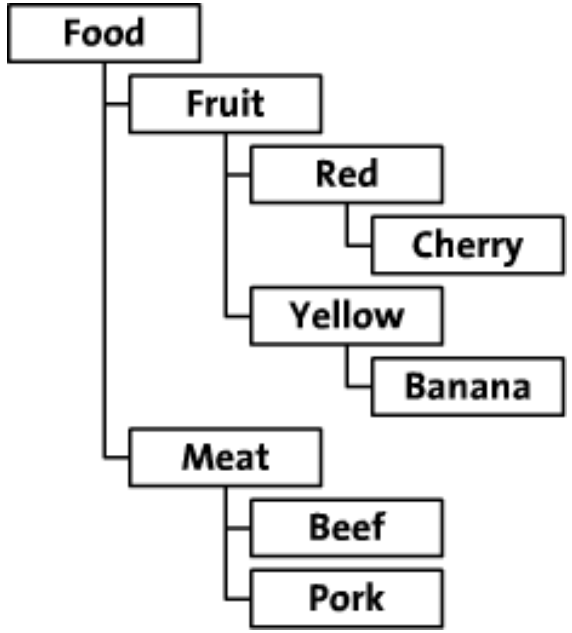
트리?

- 족보같이 계층 구조를 이루는 자료를 표현할 때 용이

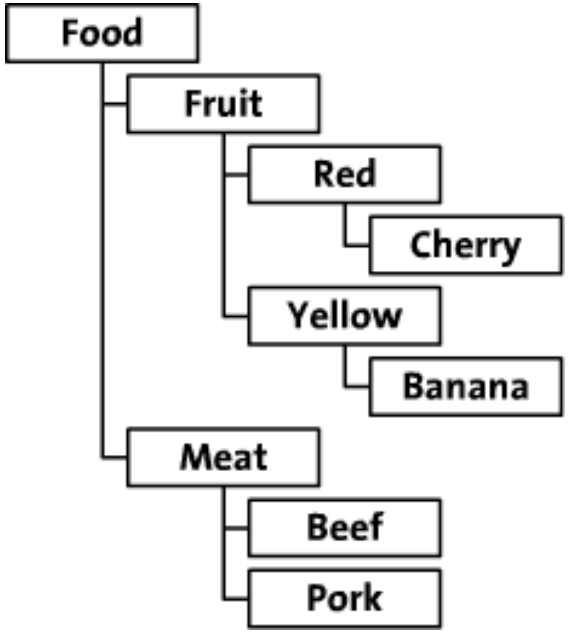




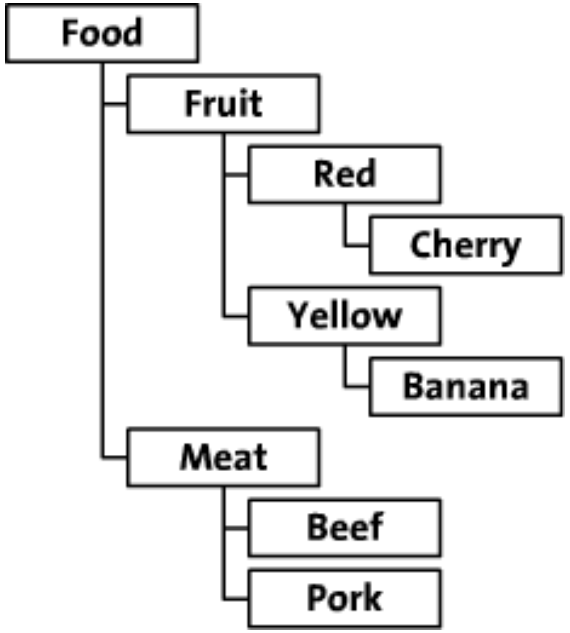

Food
Fruit
Red
Cherry
Yellow
Banana
Meat
Beef
Pork



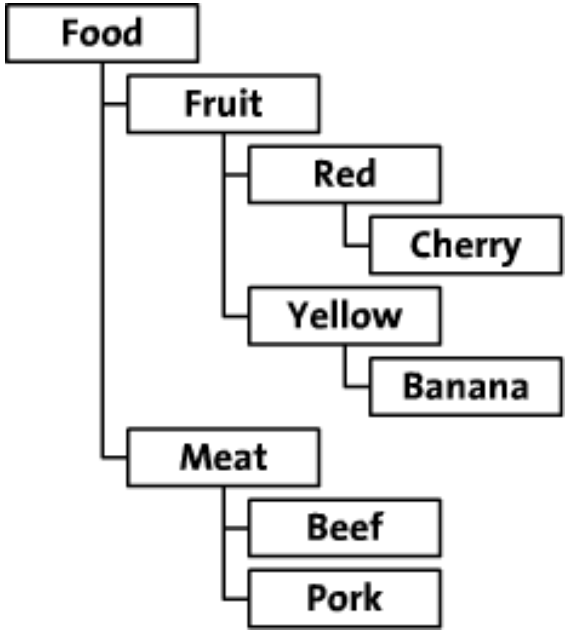
- 상위-하위 개념



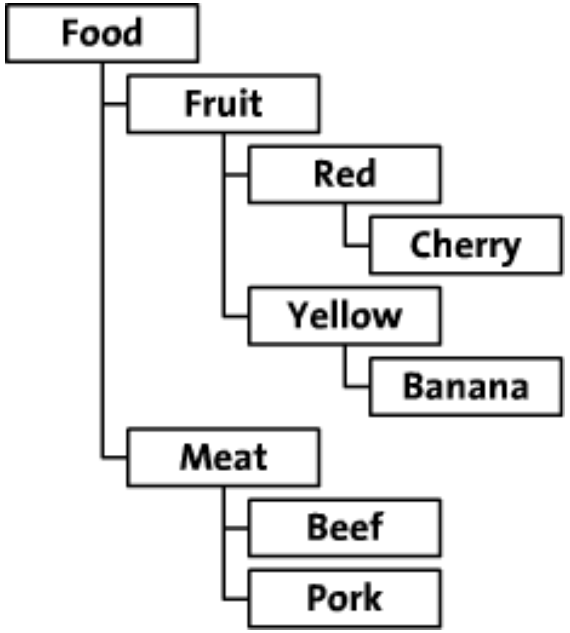
- 상위-하위 개념
  - **부모** : 연결 관계가 있는 노드 중 상위 노드
  - **자식** : 연결 관계가 있는 노드 중 하위 노드



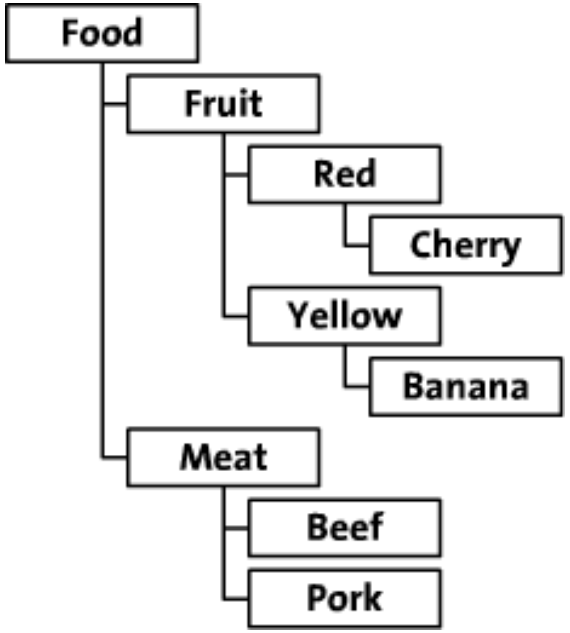
- 상위-하위 개념
  - **부모** : 연결 관계가 있는 노드 중 상위 노드
  - **자식** : 연결 관계가 있는 노드 중 하위 노드
- Fruit & Meat



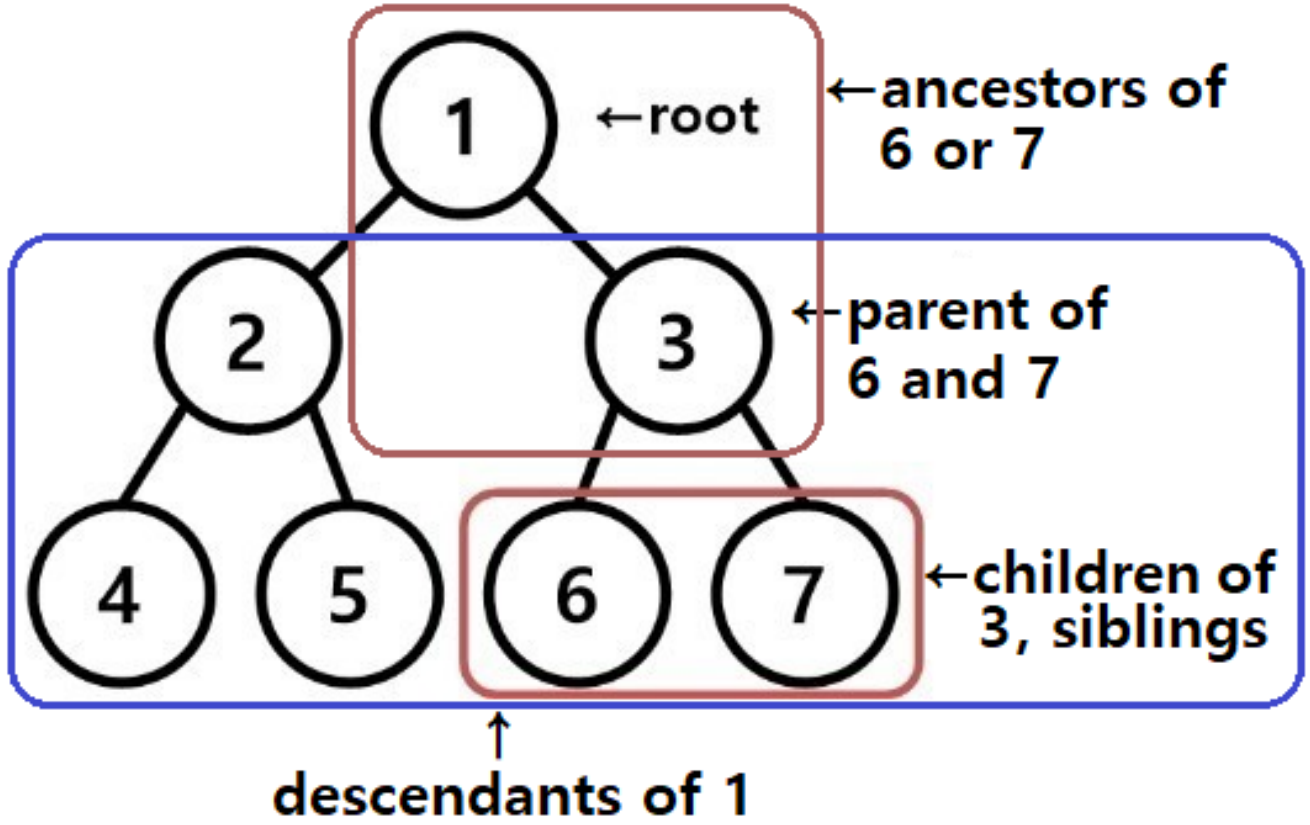
- 상위-하위 개념
  - **부모** : 연결 관계가 있는 노드 중 상위 노드
  - **자식** : 연결 관계가 있는 노드 중 하위 노드
- Fruit & Meat : 상위노드가 Food로 동일



- 상위-하위 개념
  - **부모** : 연결 관계가 있는 노드 중 상위 노드
  - **자식** : 연결 관계가 있는 노드 중 하위 노드
- Fruit & Meat : 상위노드가 Food로 동일 → **형제**



- 상위-하위 개념
  - **부모** : 연결 관계가 있는 노드 중 상위 노드
  - **자식** : 연결 관계가 있는 노드 중 하위 노드
- Fruit & Meat : 상위노드가 Food로 동일 → **형제**
- **선조** : 부모 노드와 그 위에 있는 노드들
- **자손** : 자식노드와 그의 자식들
- **루트** : 모든 노드들을 자손으로 갖는 노드
- **잎** : 자식이 하나도 없는 노드



- 차수(degree) : number of subtrees of the node

트리의 표현

- 요구조건
  - 노드의 데이터
  - 노드 간 연결

트리의 표현

- 요구조건
  - 노드의 데이터
  - 노드 간 연결

```
struct TreeNode {
    element_type data;
    TreeNode* parent;
    vector<TreeNode*> children;
};
```

트리의 표현

- 요구조건
  - 노드의 데이터
  - 노드 간 연결

```
struct TreeNode {
    element_type data;
    TreeNode* parent;
    vector<TreeNode*> children;
};

typedef struct treenode* nptr;
typedef struct treenode {
    element_type data;
    nptr parent;
    nptr children[MAX];
};
```

트리의 표현

- 요구조건
  - 노드의 데이터
  - 노드 간 연결

```cpp
#include <vector>
#include <iostream>
using namespace std;

vector<int> adj[MAX];

int main() {
    int par, ch;
    scanf("%d%d", &par, &ch);
    adj[par].push_back(ch);
}
```

트리의 표현

- 요구조건
  - 노드의 데이터
  - 노드 간 연결

- 용도에 따라

  구현 방식이 달라진다!

```cpp
#include <vector>
#include <iostream>
using namespace std;

vector<int> adj[MAX];

int main() {
    int par, ch;
    scanf("%d%d", &par, &ch);
    adj[par].push_back(ch);
}
```

트리의 순회

- 트리에 포함되어 있는 자료를 전부 방문

ex) 첫번째 방식으로 구현된 트리인 경우

```
struct TreeNode {
    element_type data;
    TreeNode* parent;
    vector<TreeNode*> children;
};
```

트리의 순회

- 어떤 트리노드는 해당 노드의 부모의 자식이었을테고,
- 또 어떤 노드는 위에서 말한 어떤 트리노드의 자식 중 하나이면서 subtree의 루트 역할을 할 것이고,

.....

트리의 순회

- 어떤 트리노드는 해당 노드의 부모의 자식이었을테고,
- 또 어떤 노드는 위에서 말한 어떤 트리노드의 자식 중 하나이면서 subtree의 루트 역할을 할 것이고,

.....

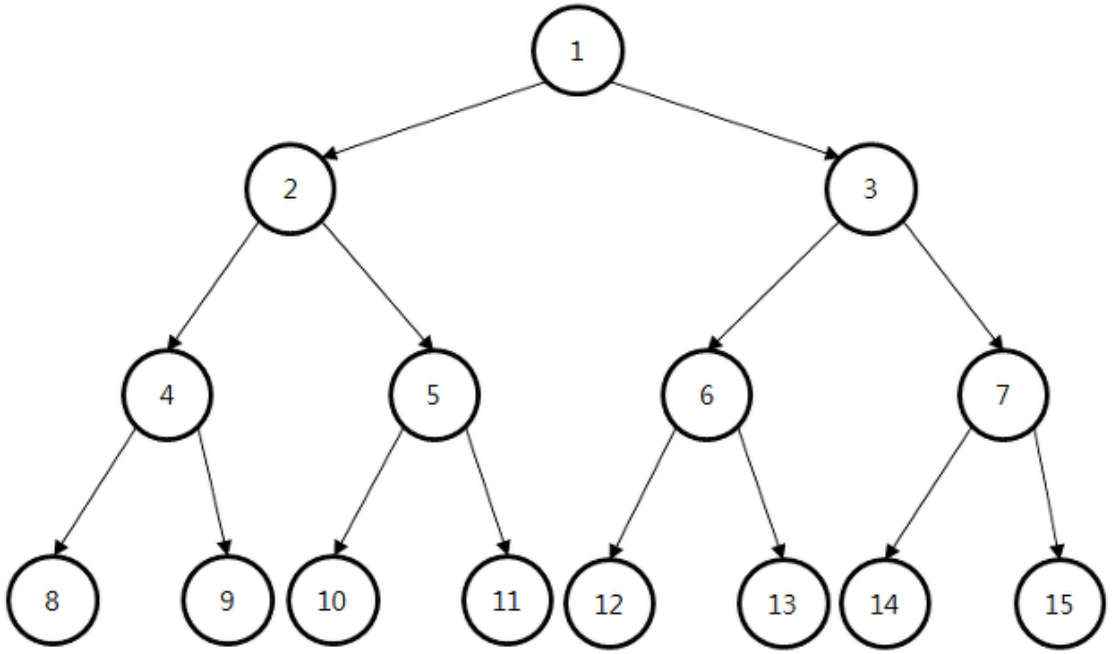
재귀적인 형태로 될 것이다!

트리의 순회

```
struct TreeNode {
    element_type data;
    TreeNode* parent;
    vector<TreeNode*> children;
};

void traverse(TreeNode* root) {
    cout << root->data << '\n';
    for (int i = 0; i < root->children.size(); i++)
        traverse(root->children[i]);
}
```

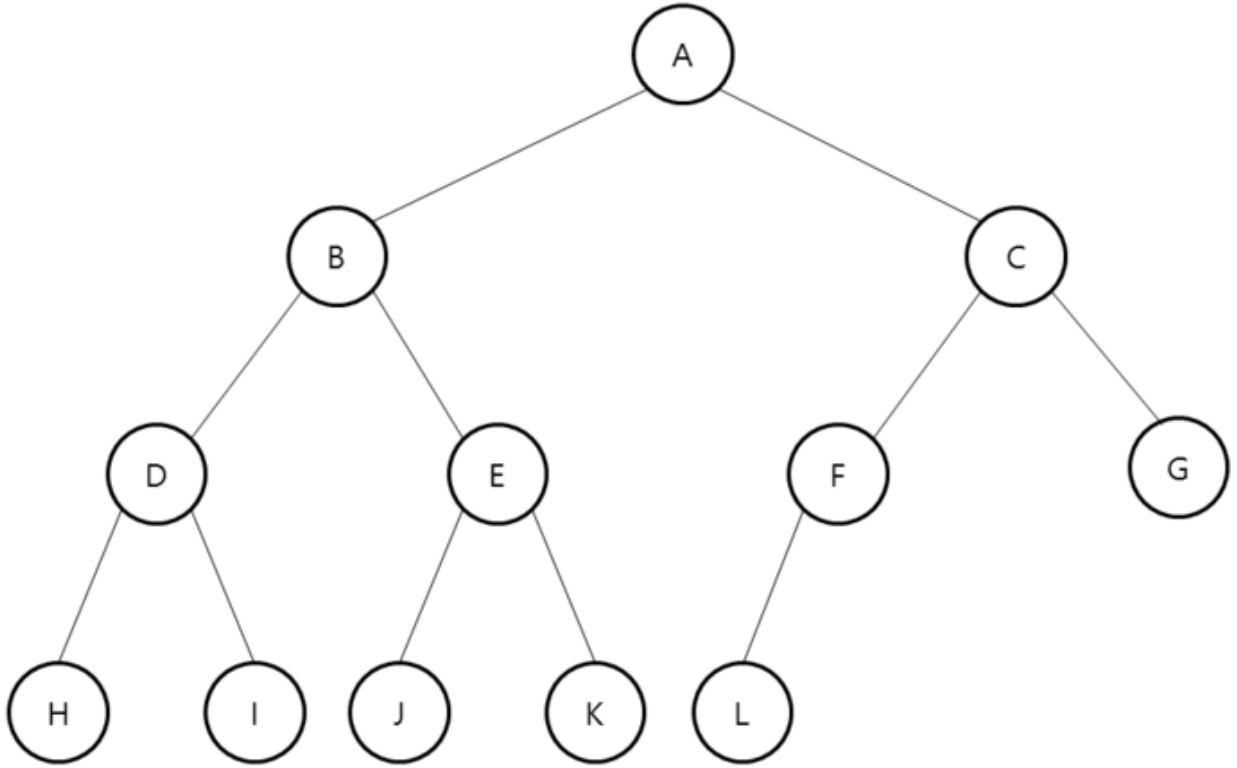
이진 트리(Binary Tree)

- 최대 차수가 2인 트리
- 2개 밖에 없어서 Left, Right로 구분

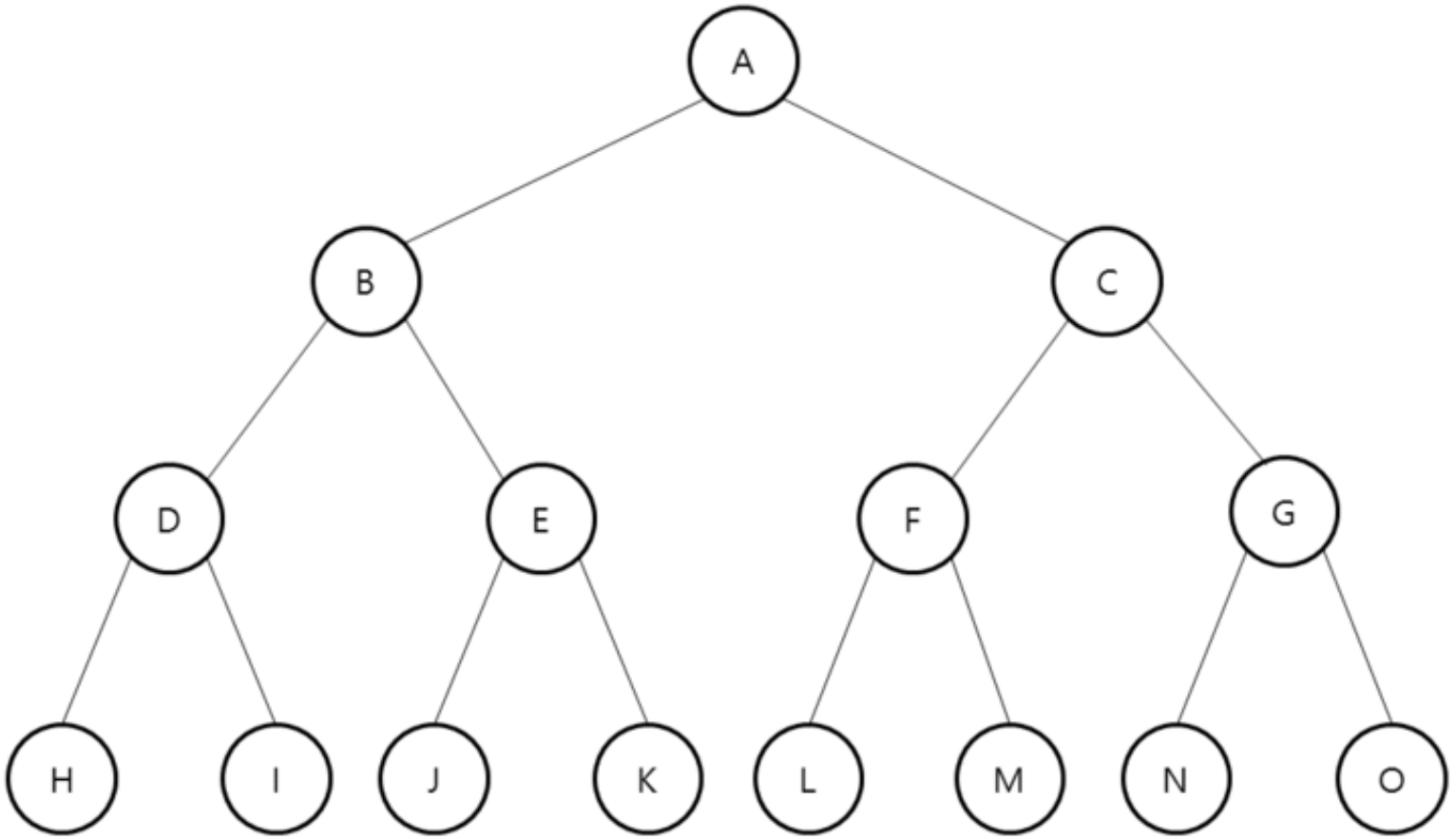


완전 이진트리(Complete Binary Tree)

- 트리의 노드를 왼쪽에서 오른쪽으로, 레벨에 따라 차례로 채운 트리



포화 이진 트리(Full Binary Tree)



이진 트리 표현

- 아까보다 더 쉬울 것 같다.

이진 트리 표현

- 아까보다 더 쉬울 것 같다.

```
struct TreeNode {
    element_type data;
    TreeNode *Left, *Right;
};
```

- 정말 그렇다.

이진 트리 표현

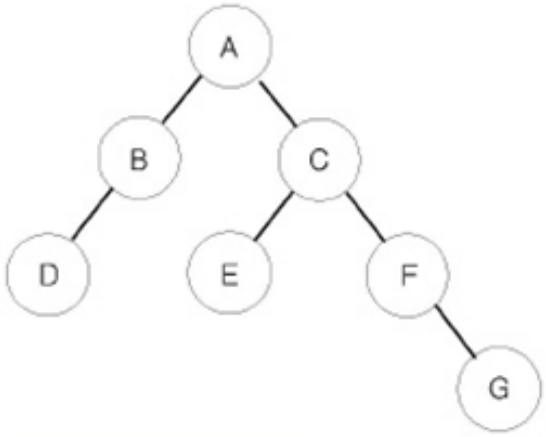
- 전체 노드 개수를 알고 있고, 각각에 대해 번호로 표현할 수 있다면

```
int tree[MAX][2];
```

이걸로도 충분하지 않을까?

#1991 트리 순회

이진 트리를 입력받아 전위 순회(preorder traversal), 중위 순회(inorder traversal), 후위 순회(postorder traversal)한 결과를 출력하는 프로그램을 작성하시오.



예를 들어 위와 같은 이진 트리가 입력되면,

- 전위 순회한 결과 : ABDCEFG // (루트) (왼쪽 자식) (오른쪽 자식)
- 중위 순회한 결과 : DBAECFG // (왼쪽 자식) (루트) (오른쪽 자식)
- 후위 순회한 결과 : DBEGFCA // (왼쪽 자식) (오른쪽 자식) (루트)

가 된다.

이진 트리 순회

```
void preorder(TreeNode* root) {
    if (!root) return;
    cout << root->data << '\n';
    preorder(root->Left);
    preorder(root->Right);
}
void inorder(TreeNode* root) {
    if (!root) return;
    inorder(root->Left);
    cout << root->data << '\n';
    inorder(root->Right);
}
```

```
void postorder(TreeNode* root) {
    if (!root) return;
    postorder(root->Left);
    postorder(root->Right);
    cout << root->data << '\n';
}
```

http://boj.kr/928af7d385f64ad08bc49c4e6e3911db

우선순위 큐(Priority Queue)

- 일단 큐는 맞다.
- 그런데 우선순위가 있다. 즉, 원소에 대한 처리 시 높은 우선순위인 원소가 먼저 처리(=삭제 or 추출)

생각해봅시다

- 우선순위를 결정하는 요소를 데이터의 크기라 하고
- 크기가 큰 값이 우선순위가 높다고 할 때

"우선순위 큐"에 차례로 1, 2, 3, 4, 5라는 값이 들어온다면?

Example

- 우선순위가 큰 원소가 큐의 맨 앞에 위치

1

Example

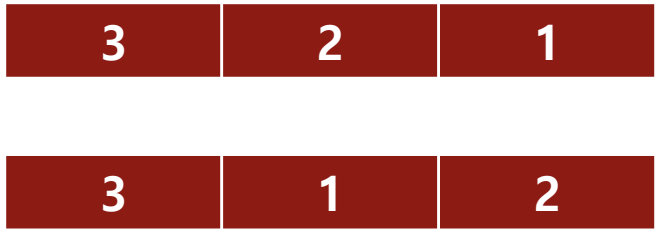
- 우선순위가 큰 원소가 큐의 맨 앞에 위치

Example

- 우선순위가 큰 원소가 큐의 맨 앞에 위치

Example

- 우선순위가 큰 원소가 큐의 맨 앞에 위치

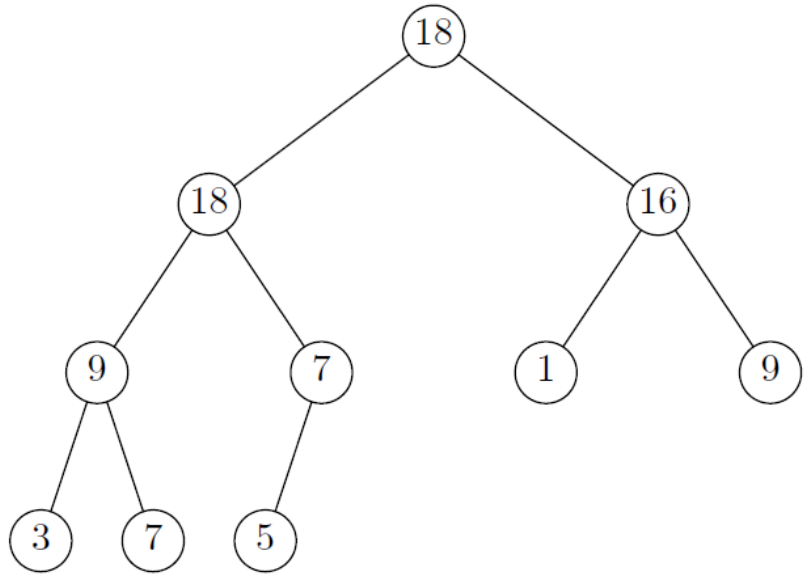


????

Partially Ordered Tree

- Partially Ordered Tree property
  - subtree의 루트에 있는 원소가 항상 해당 subtree 중 가장 큰 원소를 만족
- POT property를 만족하고

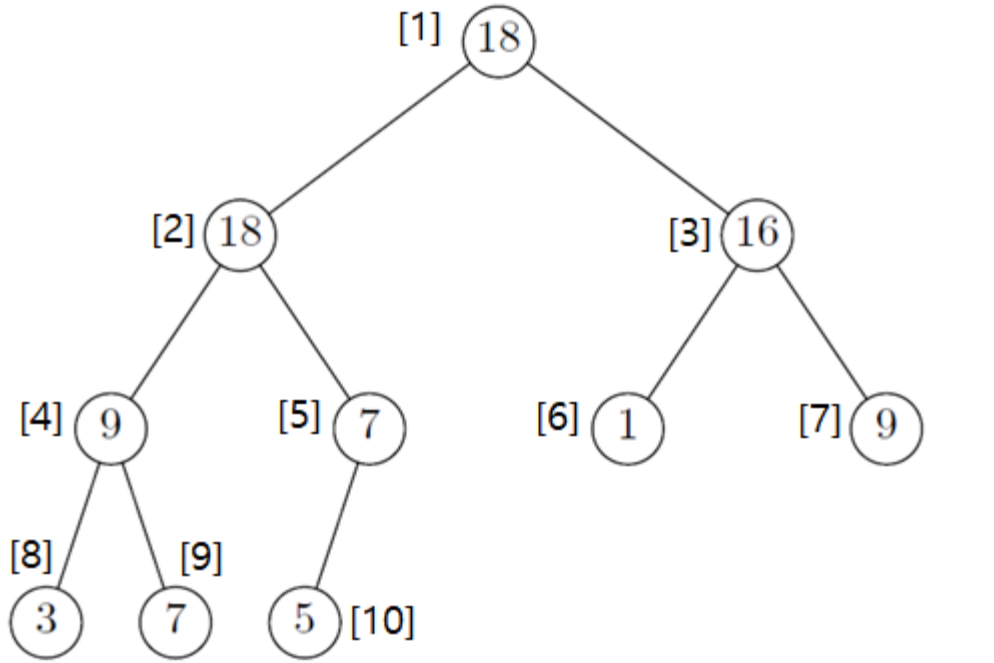
  Complete Binary Tree인 경우



Partially ordered tree with 10 nodes.

Implementing Balanced POT

1	2	3	4	5	6	7	8	9	10
18	18	16	9	7	1	9	3	7	5



참고 자료 : https://drive.google.com/file/d/1lz9HkrfY3un_EPzXfdTljCC9Y81EOo-G/view?usp=sharing

Heap

- 특정 우선순위에 대해, 우선순위가 가장 높은 값을 찾아내는 연산을 빠르게 하기 위해 고안된 자료구조
- tree-based structure

  가장 큰 값을 우선으로 하는 것을 Max heap

  가장 작은 값을 우선으로 하는 것을 Min heap

예시 : Max heap 만들기

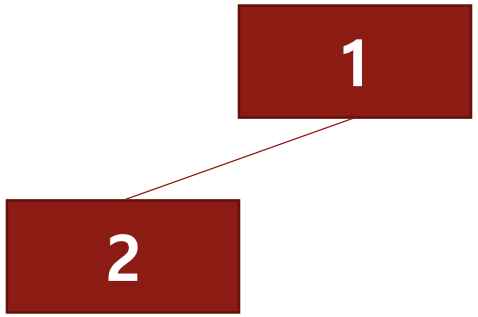
- 1, 2, 3, 4, 5가 차례로 들어올 때

예시 : Max heap 만들기

- **1**, 2, 3, 4, 5가 차례로 들어올 때

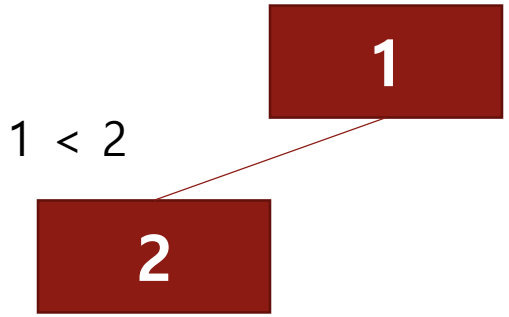
예시 : Max heap 만들기

- 1, **2**, 3, 4, 5가 차례로 들어올 때



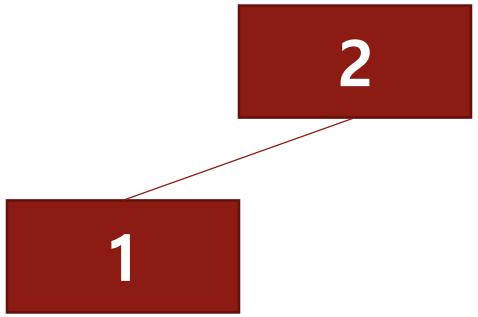
예시 : Max heap 만들기

- 1, **2**, 3, 4, 5가 차례로 들어올 때



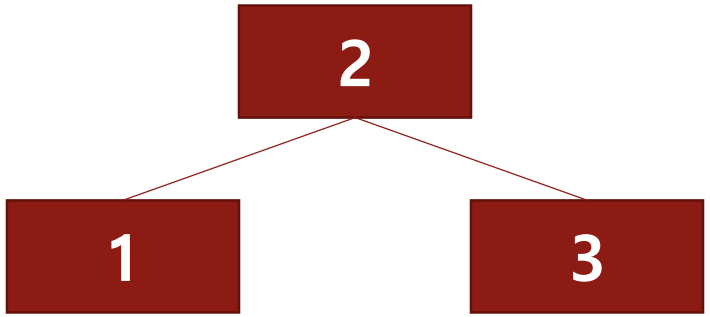
예시 : Max heap 만들기

- 1, **2**, 3, 4, 5가 차례로 들어올 때



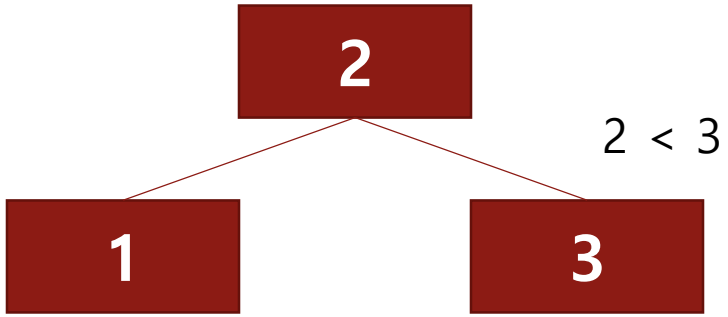
예시 : Max heap 만들기

- 1, 2, **3**, 4, 5가 차례로 들어올 때



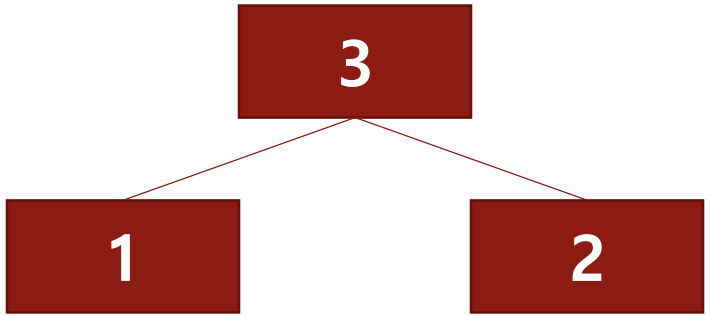
예시 : Max heap 만들기

- 1, 2, **3**, 4, 5가 차례로 들어올 때



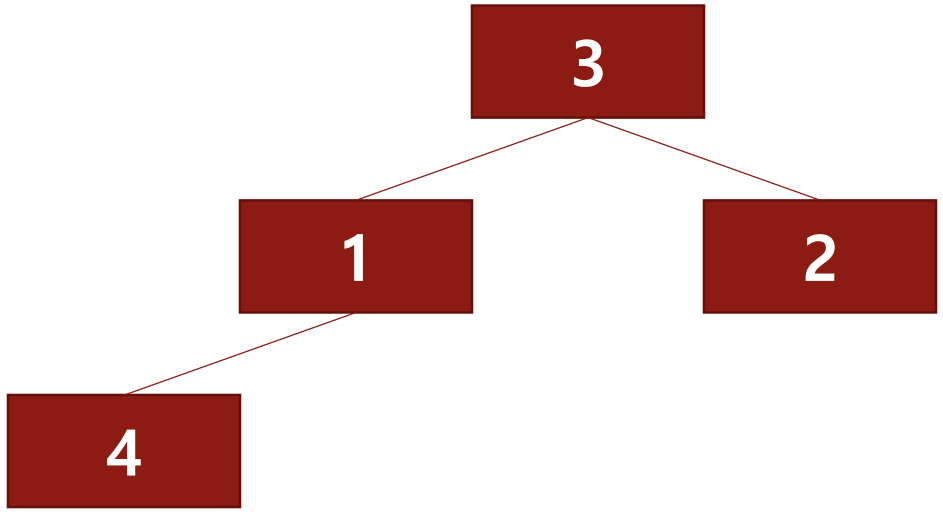
예시 : Max heap 만들기

- 1, 2, **3**, 4, 5가 차례로 들어올 때



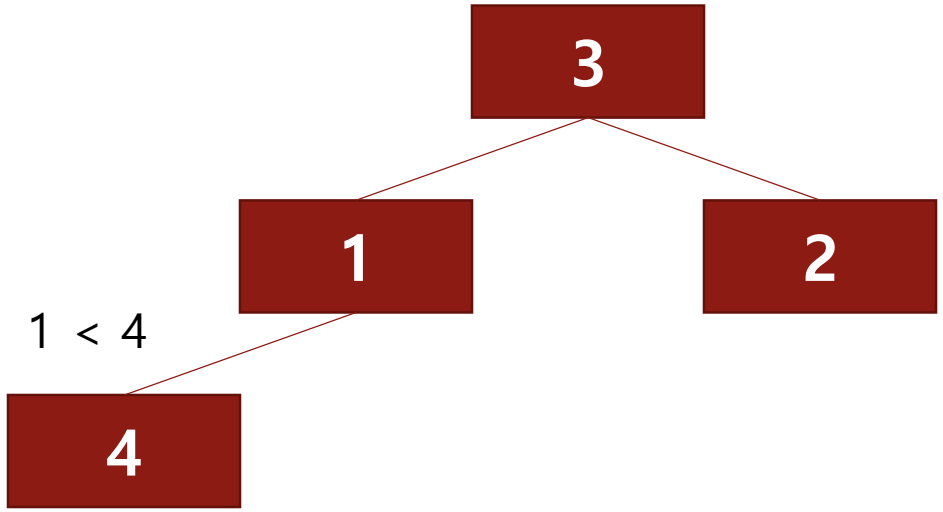
예시 : Max heap 만들기

- 1, 2, 3, **4**, 5가 차례로 들어올 때



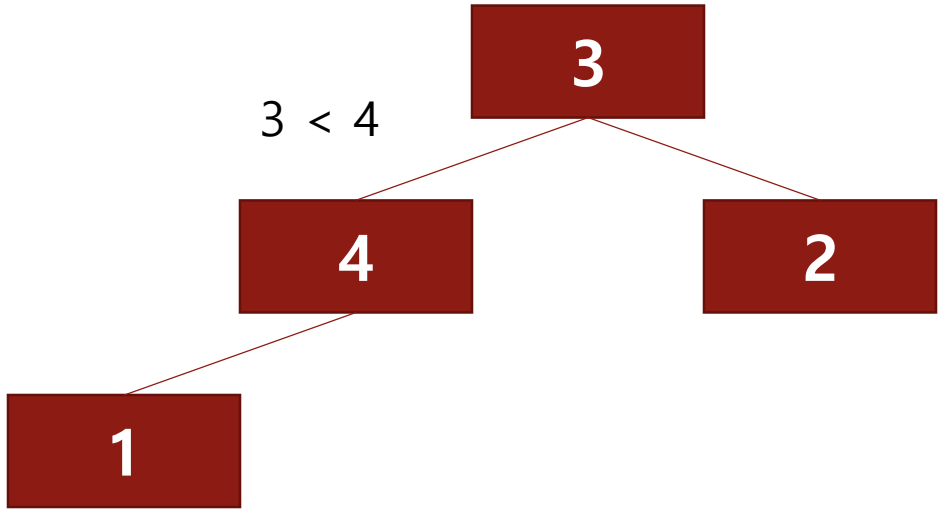
예시 : Max heap 만들기

- 1, 2, 3, **4**, 5가 차례로 들어올 때



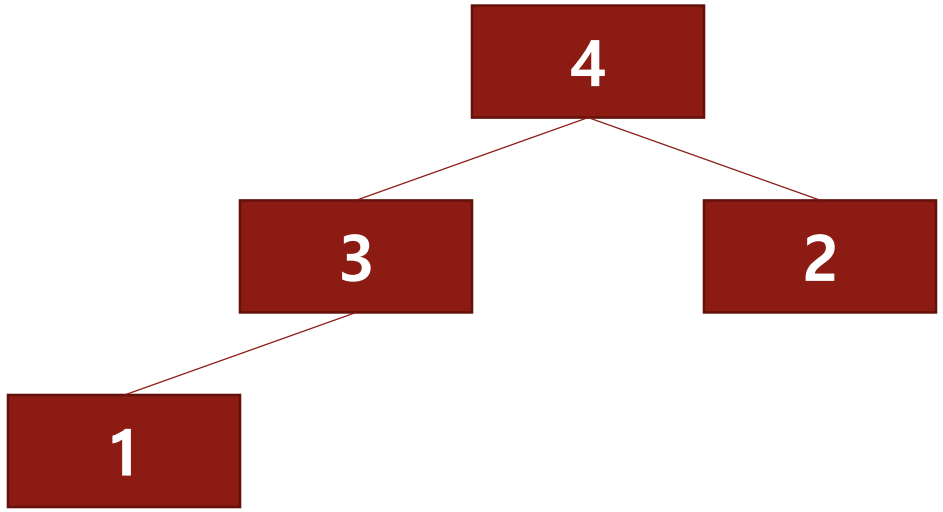
예시 : Max heap 만들기

- 1, 2, 3, **4**, 5가 차례로 들어올 때



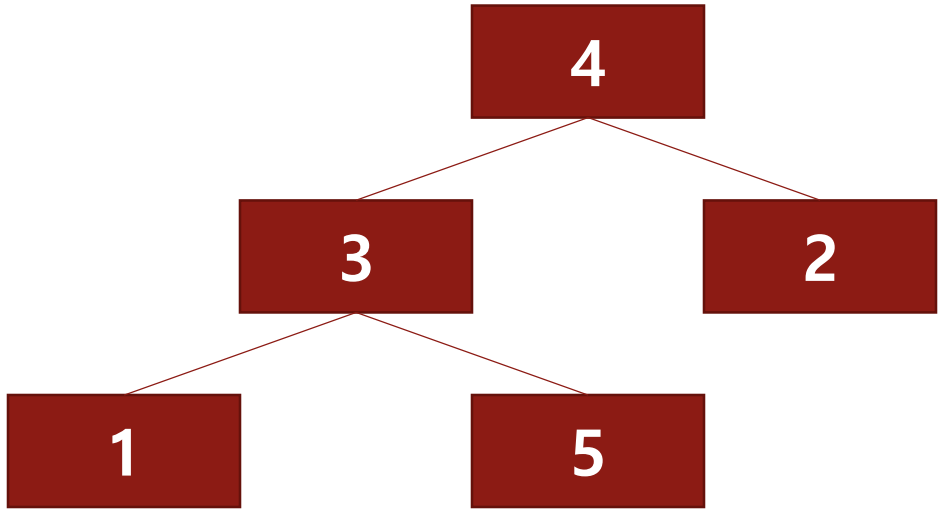
예시 : Max heap 만들기

- 1, 2, 3, **4**, 5가 차례로 들어올 때



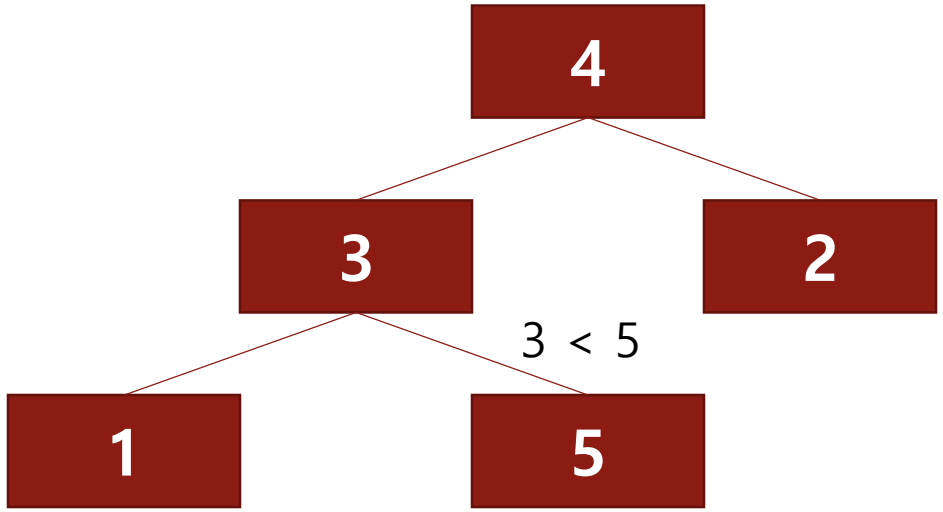
예시 : Max heap 만들기

- 1, 2, 3, 4, **5**가 차례로 들어올 때



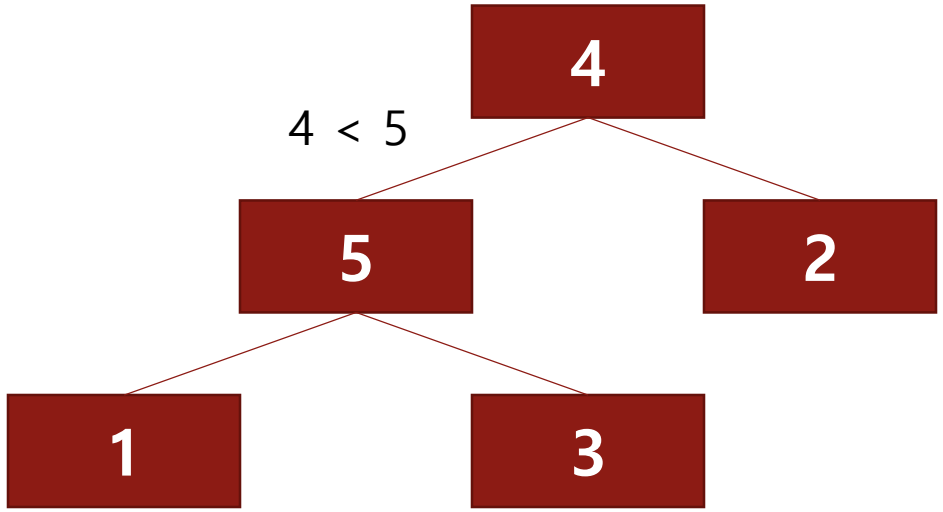
예시 : Max heap 만들기

- 1, 2, 3, 4, **5**가 차례로 들어올 때



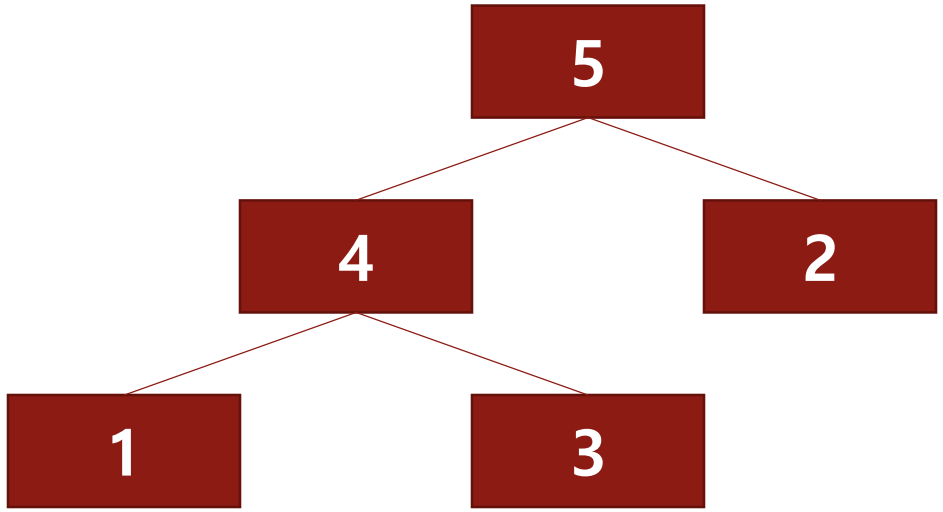
예시 : Max heap 만들기

- 1, 2, 3, 4, **5**가 차례로 들어올 때

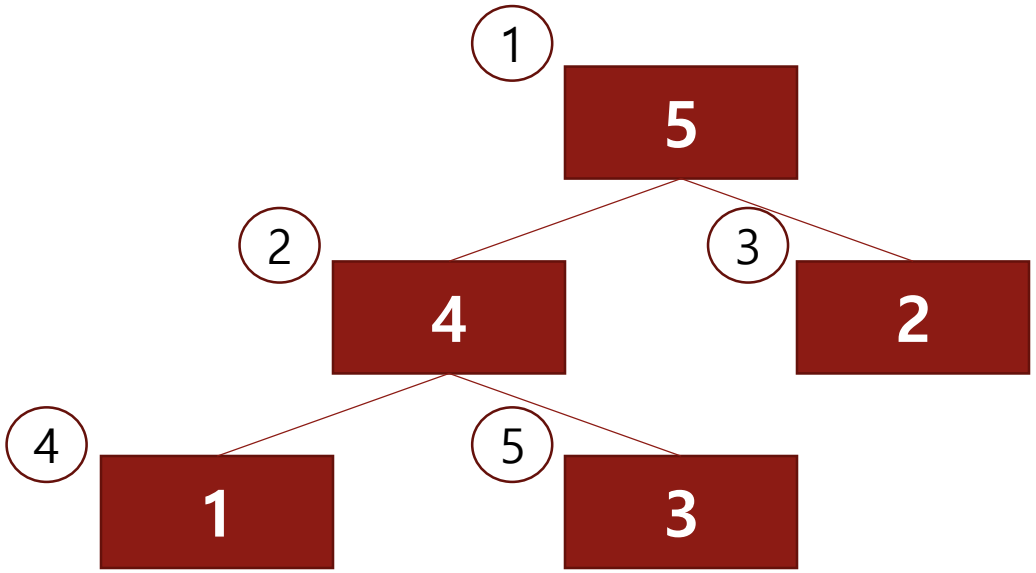


예시 : Max heap 만들기

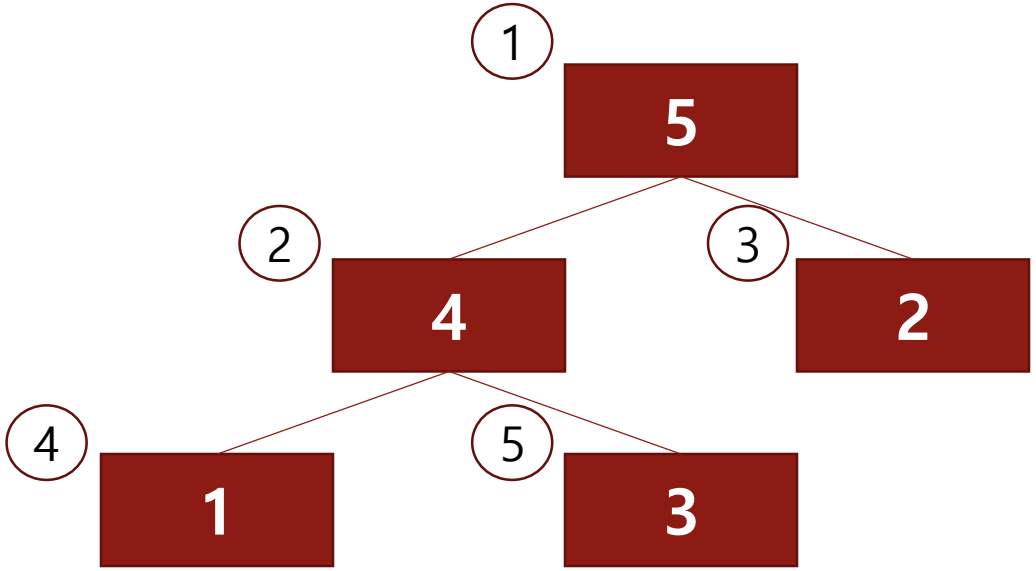
- 1, 2, 3, 4, **5**가 차례로 들어올 때



Swap을 어떻게 하지?

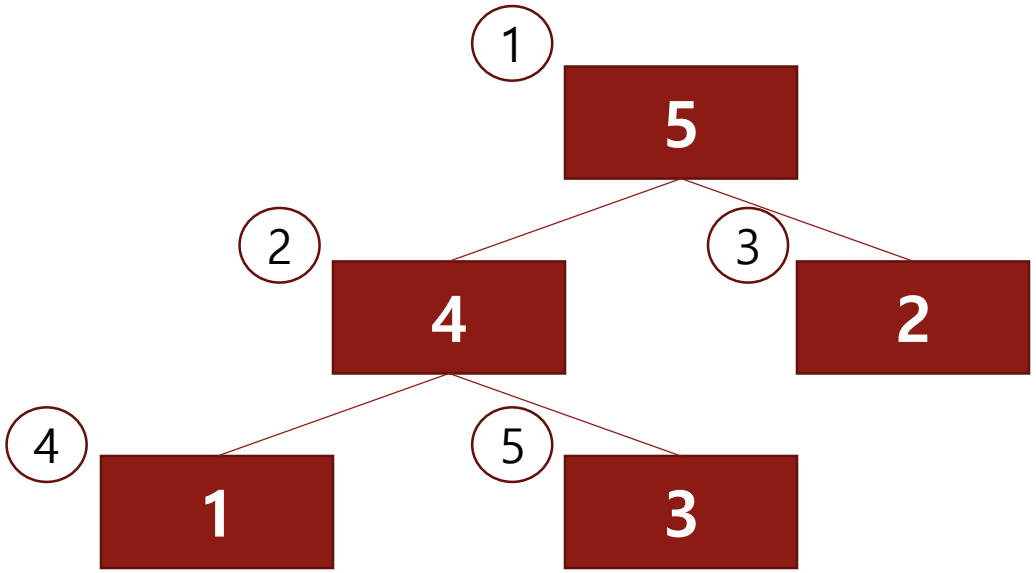


Swap을 어떻게 하지?



Swap을 어떻게 하지?

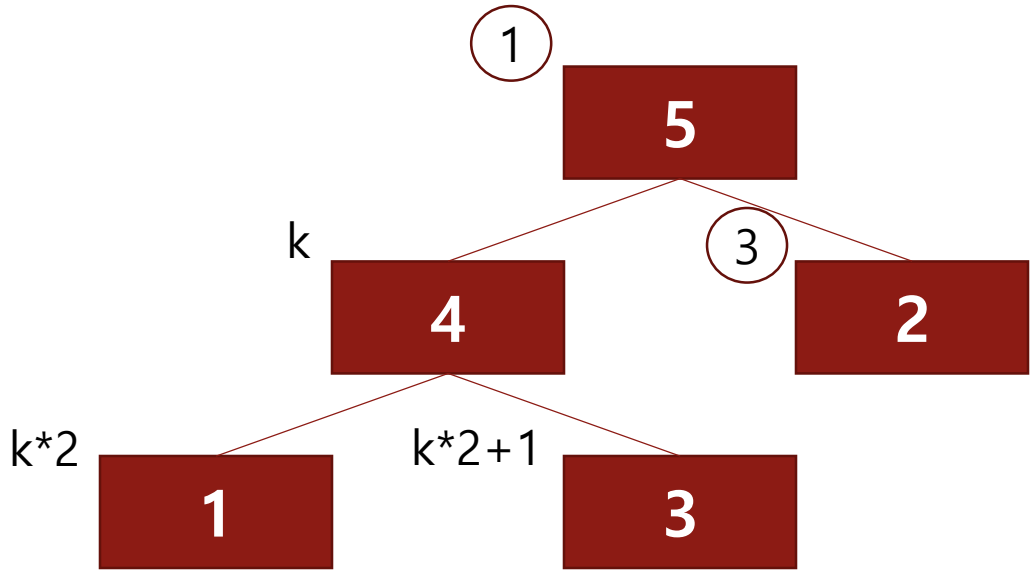
Let k = 2,



Array

Swap을 어떻게 하지?

Let k = 2,



Heap에서 자식노드로의 접근

- 왼쪽 자식으로의 접근 : 현재 노드번호 × 2
- 오른쪽 자식으로의 접근 : 현재 노드번호 × 2 + 1

- 반대로 자식에서 부모로 접근한다면?

- To express Insertion

```
void swap(int* arr, int i, int j) {
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
}

void BubbleUp(int* arr, int i) {
    if(i > 1 && arr[i] > arr[i/2]) {
        swap(arr, i, i/2);
        BubbleUp(arr, i/2);
    }
}
```

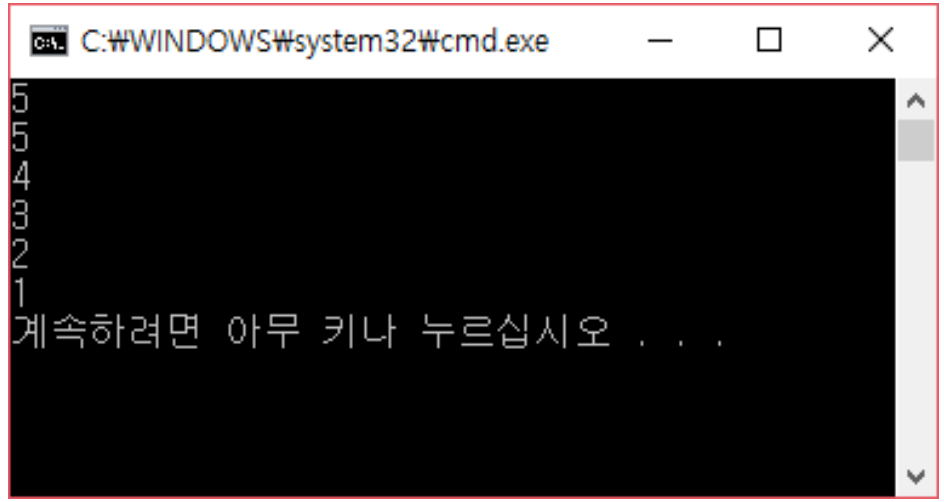
```
void insert(int* arr, int data, int idx) {
    arr[idx] = data;
    BubbleUp(arr, idx);
}
```

- To express Deletion

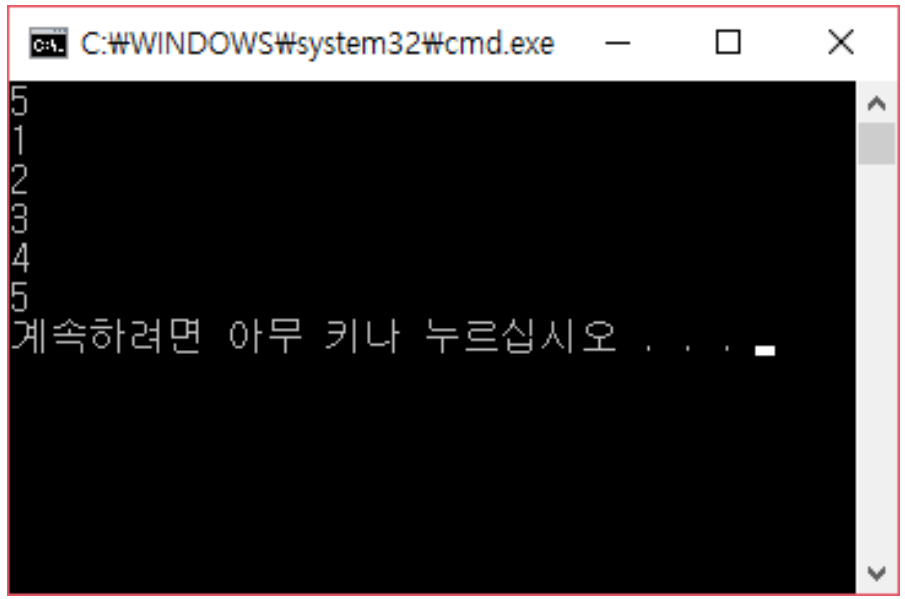
```
void swap(int* arr, int i, int j) {
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
}
void BubbleDown(int* arr, int i, int lastindex) {
    int child = 2 * i;
    if(child < lastindex && arr[child + 1] arr[child])
        child++;
    if(child <= lastindex && arr[i] < arr[child]) {
        swap(arr, i, child);
        BubbleDown(arr, child, lastindex);
    }
}
void delete(int* arr, int *lastindex) {
    swap(arr, 1, *lastindex);
    (*lastindex)--;
    BubbleDown(arr, 1, *lastindex);
}
```

STL을 쓴다면?

```cpp
#include <queue>
#include <iostream>
using namespace std;

priority_queue<int> pq;

int main() {
    pq.push(1);
    pq.push(2);
    pq.push(3);
    pq.push(4);
    pq.push(5);
    cout << pq.size() << '\n';
    while (!pq.empty()) {
        cout << pq.top() << '\n';
        pq.pop();
    }
}
```



```cpp
#include <queue>
#include <iostream>
#include <functional>
#include <vector>
using namespace std;

priority_queue<int, vector<int>, greater<int>> pq;

int main() {
    pq.push(1);
    pq.push(2);
    pq.push(3);
    pq.push(4);
    pq.push(5);
    cout << pq.size() << '\n';
    while (!pq.empty()) {
        cout << pq.top() << '\n';
        pq.pop();
    }
}
```



#1715 카드 정렬하기

정렬된 두 묶음의 숫자 카드가 있다고 하자. 각 묶음의 카드의 수를 A, B라 하면 보통 두 묶음을 합쳐서 하나로 만드는 데에는 A+B 번의 비교를 해야 한다. 이를테면, 20장의 숫자 카드 묶음과 30장의 숫자 카드 묶음을 합치려면 50번의 비교가 필요하다.

매우 많은 숫자 카드 묶음이 책상 위에 놓여 있다. 이들을 두 묶음씩 골라 서로 합쳐나간다면, 고르는 순서에 따라서 비교 횟수가 매우 달라진다. 예를 들어 10장, 20장, 40장의 묶음이 있다면 10장과 20장을 합친 뒤, 합친 30장 묶음과 40장을 합친다면 (10+20)+(30+40) = 100번의 비교가 필요하다. 그러나 10장과 40장을 합친 뒤, 합친 50장 묶음과 20장을 합친다면 (10+40)+(50+20) = 120 번의 비교가 필요하므로 덜 효율적인 방법이다.

N개의 숫자 카드 묶음의 각각의 크기가 주어질 때, 최소한 몇 번의 비교가 필요한지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N이 주어진다. (1≤N≤100,000) 이어서 N개의 줄에 걸쳐 숫자 카드 묶음의 각각의 크기가 주어진다.

http://boj.kr/de46b820a584483dae8544ffa3cb1ed8

Problem Set

- 11279 최대 힙
- 1927 최소 힙
- 11286 절댓값 힙
- 1715 카드 정렬하기
- 2075 N번째 큰 수*
- 2014 소수의 곱*
- 15761 Lemonade Line*