

## 노드 번호를 배열의 인덱스로 사용하기

부모노드의 인덱스가 2라면 왼쪽 자식노드는 4 오른쪽 자식 노드는 5

부모노드 인덱스 : i

왼쪽 자식 노드:  $2*i$

오른쪽 자식 노드:  $(2*i) + 1$

<참고>

$\text{int}(-3/2) = -1$ (버리는 것)

$(-3)/2 = -2$  (floor 연산과 같은 값, 이 숫자보다 작은 수 중에 가장 큰 정수)

연결리스트(대충 개요만 보고 가기)

: 주소정보를 담고간다는 특징

정점의 개수 = 간선 개수 + 1

## 트리의 저장 방법

- 부모 노드를 인덱스로 자식 번호를 저장  
 $\text{ch1}[\text{부모}] == 0$ 이면  $\text{ch1}[\text{부모}] = \text{자식(왼쪽)}$   
else인 경우는  $\text{ch2}[\text{부모}] = \text{자식(오른쪽)}$
- 자식 노드를 인덱스로 부모 번호를 저장  
루트는 부모가 없는 노드(부모가 0인 노드)

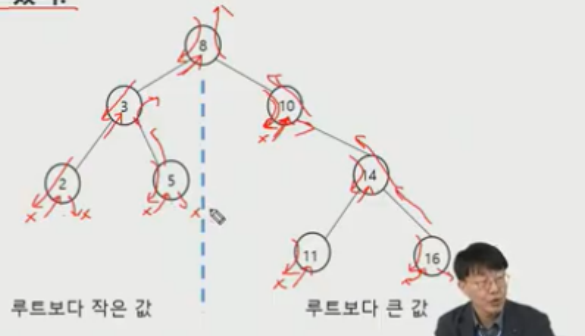
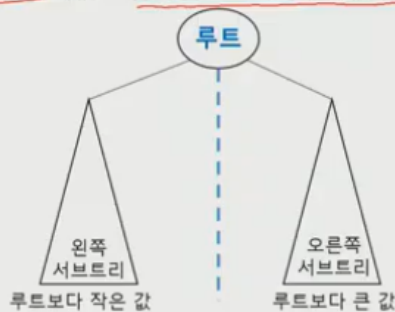
## 이진 탐색 트리

중위 순회 시 오름차순 정렬값

- ❖ 탐색작업을 효율적으로 하기 위한 자료구조
- ❖ 모든 원소는 서로 다른 유일한 키를 갖는다.
- ❖  $key(\text{왼쪽 서브트리}) < key(\text{루트 노드}) < key(\text{오른쪽 서브트리})$
- ❖ 왼쪽 서브트리와 오른쪽 서브트리도 이진 탐색 트리다.
- ❖ 중위 순회하면 오름차순으로 정렬된 값을 얻을 수 있다.

L V R

2 3 5 8 10 11 14 16



균형이 잘 잡힌 이진탐색트리는 탐색시간을 많이 줄일수 있지만  
편향되었다면 선형과 결국 차이가 없어진다  
그래도 새로운 원소의 삽입할 때 삽입 시간을 줄일 수 있다

## ❖ 평균의 경우

- 이진 트리가 균형적으로 생성되어 있는 경우
- $O(\log n)$

## ❖ 최악의 경우

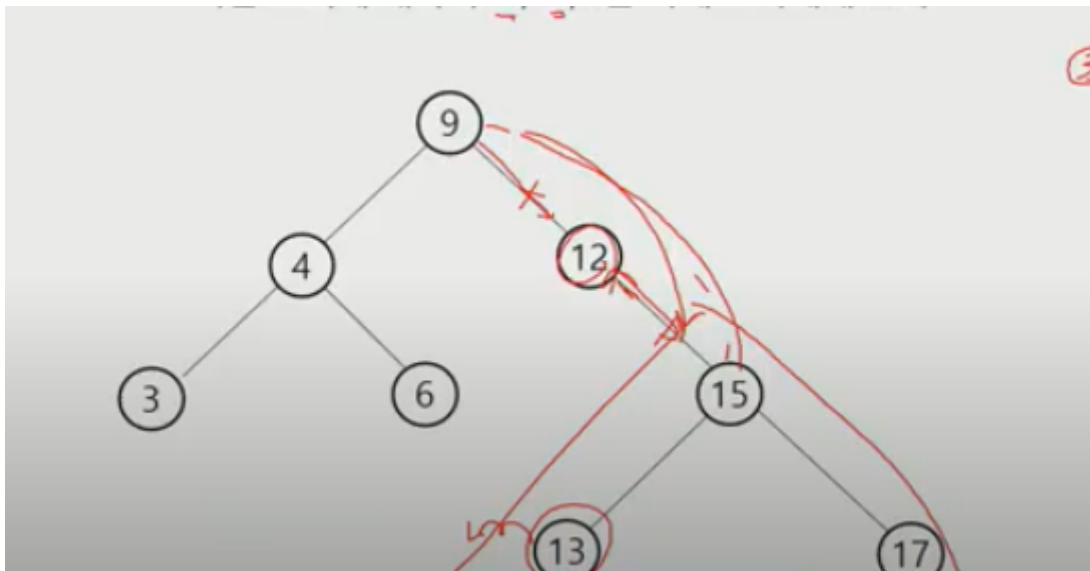
- 한쪽으로 치우친 경사 이진트리의 경우
- $O(n)$
- 순차탐색과 시간복잡도가 같다.

- 레드블랙트리

## 삭제 연산

자식이 없으면 찾아서 그냥 삭제

자식이 1개 있으면 연결을 끊은 후 그 다음 자식을 연결(12를 지우고 9와 15를 연결)



루트를 삭제할 때

루트에 올 정점을 먼저 정하는데 새로운 루트는 루트 오른쪽에 위치한 정점 중(루트보다 큰 친구들 중) 왼쪽 자식이 없는 최초의 친구

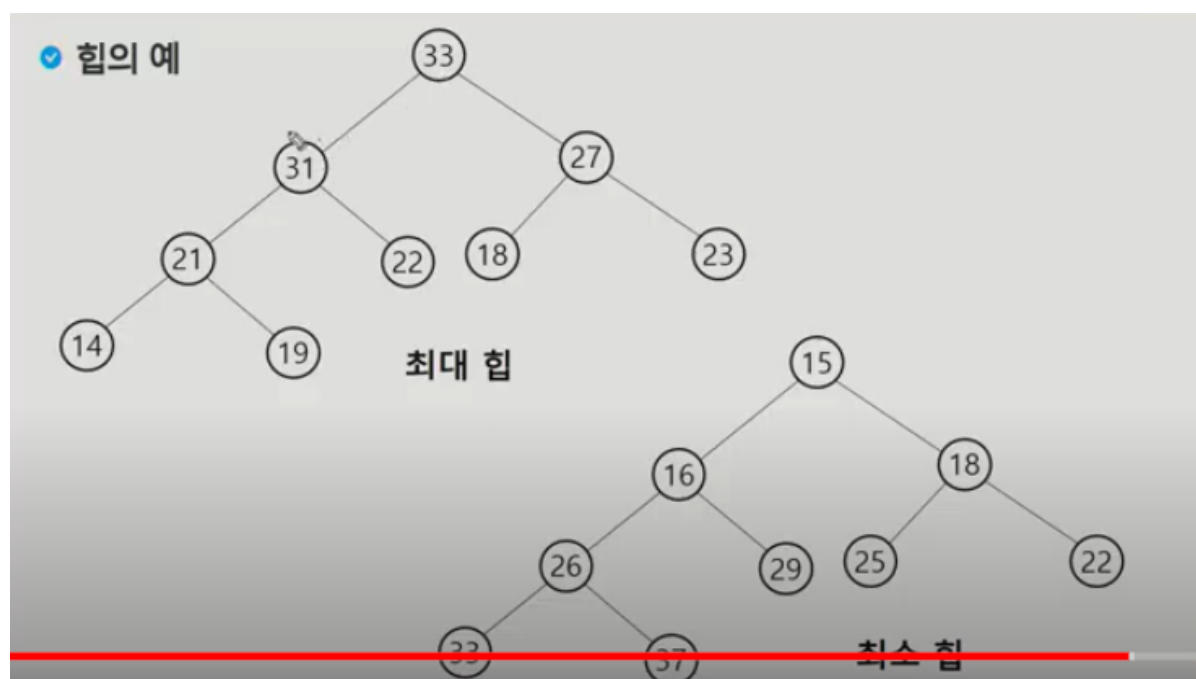
....개요라도 대충 알고 가자..는 마음으로!

### 참고: 힙

완전이진트리로 구현된 자료구조

가장 큰 노드나 가장 작은 노드 찾기에 용이함

힙의 키를 우선순위로 활용하면 우선순위 큐 구현 가능



최대힙: 부모가 제일 큼

최소힙: 부모가 제일 작음

힙연산 삽입은

1. last node의 인덱스 + 1 자리에 넣어준다
2. 부모값과 비교해서 기준에 맞는지 확인하고 맞지 않으면 부모와 위치 교환

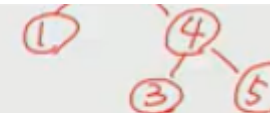
순회 코드 구현

입력받기

```
'''
~~~~~
5 4      V E
2 1 2 4 4 3 4 5  P C P C ...
~~~~~
'''

# 1번부터 V번까지 노드, E개의 간선
V, E = map(int, input().split())
edge = list(map(int, input().split()))

left = [0]*(V+1) # 부모를 인덱스로 왼쪽 자식번호 저장
right = [0]*(V+1) # 부모를 인덱스로 오른쪽 자식번호 저장
```



```
pa = [0]*(V+1) # 자식을 인덱스로 부모번호 저장

for i in range(E):
    n1, n2 = edge[i*2], edge[i*2+1] # n1 부모, n2 자식노드
    if left[n1] == 0: # 왼쪽자식이 없으면
        left[n1] = n2 # 부모를 인덱스로 자식번호 저장
    else: # 왼쪽자식이 있으면
        right[n1] = n2 # 부모를 인덱스로 자식번호 저장
```