<7> 이진탐색

1. 순차탐색

▶ 리스트 안에 있는 특정한 데이터를 찾기 위해 앞에서부터 데이터를 하나씩 차례대로 확인하는 방법

**7-1.py 순차 탐색 소스코드**

```
# 순차 탐색 소스코드 구현
def sequential_search(n, target, array):
    # 각 원소를 하나씩 확인하며
    for i in range(n):
        # 현재의 원소가 찾고자 하는 원소와 동일한 경우
        if array[i] == target:
            return i + 1 # 현재의 위치 반환(인덱스는 0부터 시작하므로 1 더하기)

print("생성할 원소 개수를 입력한 다음 한 칸 띄고 찾을 문자열을 입력하세요.")
input_data = input().split()
n = int(input_data[0]) # 원소의 개수
target = input_data[1] # 찾고자 하는 문자열

print("앞서 적은 원소 개수만큼 문자열을 입력하세요. 구분은 띄어쓰기 한 칸으로 합니다.")
array = input().split()

# 순차 탐색 수행 결과 출력
print(sequential_search(n, target, array))
```

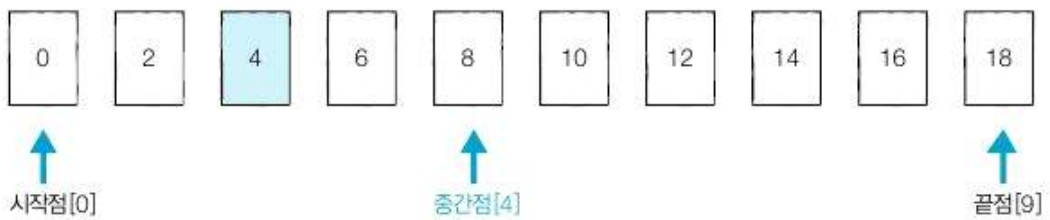
데이터의 정렬 여부와 상관없이 처음부터 원하는 값이 나올 때까지 탐색

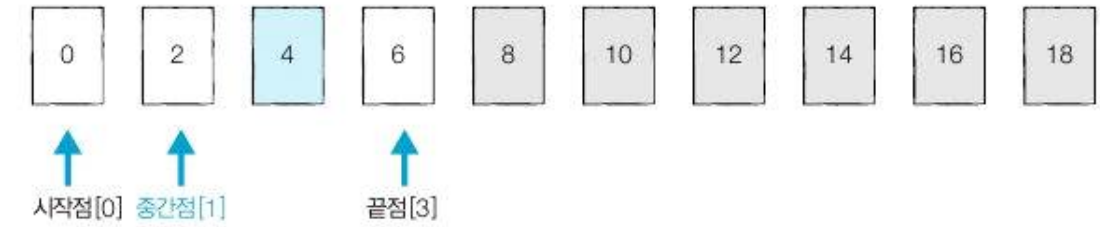
순차 탐색의 시간복잡도 : N개일 때 최대 N번의 비교 연산이 필요하기에 O(N)

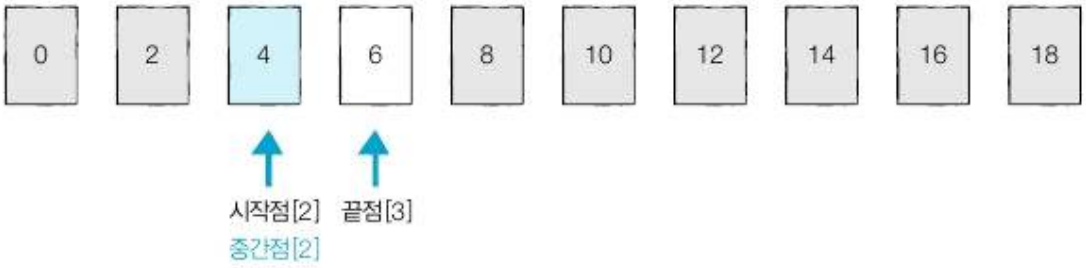
2. 이진탐색

▶ 탐색 범위를 반으로 좁혀가며 빠르게 탐색하는 알고리즘
▶ 정렬된 데이터를 빠르게 탐색할 수 있음

위치를 나타내는 변수 3개 > 시작점, 끝점, 중간점
찾으려는 데이터와 중간점 위치에 있는 데이터를 반복적으로 비교하여 원하는 데이터를 탐색한다






이진 탐색의 시간복잡도 : 한 번 확인할 때마다 확인하는 원소의 개수가 절반씩 줄어들기 때문에 O(logN)

$$\left(\frac{1}{2}\right)^{K} N \approx 1$$

$$2^{K} \approx N$$

$$K \approx \log_2 N$$

이진탐색의 구현 방법은 여러 가지

▶ 재귀함수, 반복문 등

### 7-2.py 재귀 함수로 구현한 이진 탐색 소스코드

```
# 이진 탐색 소스코드 구현(재귀 함수)
def binary_search(array, target, start, end):
    if start > end:
        return None
    mid = (start + end) // 2
    # 찾은 경우 중간점 인덱스 반환


    if array[mid] == target:
        return mid
    # 중간점의 값보다 찾고자 하는 값이 작은 경우 왼쪽 확인
    elif array[mid] > target:
        return binary_search(array, target, start, mid - 1)
    # 중간점의 값보다 찾고자 하는 값이 큰 경우 오른쪽 확인
    else:
        return binary_search(array, target, mid + 1, end)

# n(원소의 개수)과 target(찾고자 하는 문자열)을 입력받기
n, target = list(map(int, input().split()))
# 전체 원소 입력받기
array = list(map(int, input().split()))

# 이진 탐색 수행 결과 출력
result = binary_search(array, target, 0, n - 1)
if result == None:
    print("원소가 존재하지 않습니다.")
else:
    print(result + 1)
```

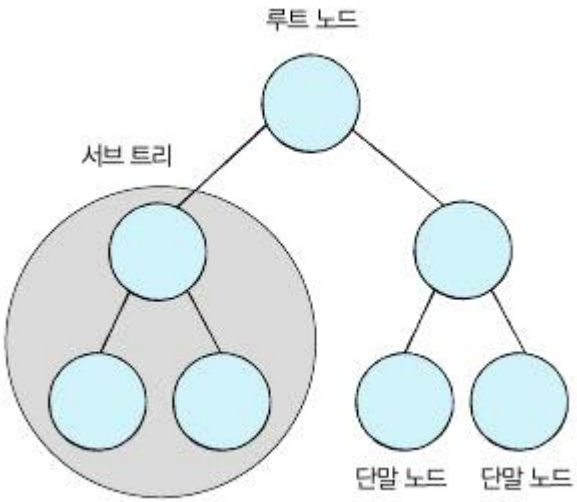
**7-3.py 반복문으로 구현한 이진 탐색 소스코드**

```
# 이진 탐색 소스코드 구현(반복문)
def binary_search(array, target, start, end):
    while start <= end:
        mid = (start + end) // 2
        # 찾은 경우 중간점 인덱스 반환
        if array[mid] == target:
            return mid
        # 중간점의 값보다 찾고자 하는 값이 작은 경우 왼쪽 확인
        elif array[mid] > target:
            end = mid - 1
        # 중간점의 값보다 찾고자 하는 값이 큰 경우 오른쪽 확인
        else:
            start = mid + 1
    return None

# n(원소의 개수)과 target(찾고자 하는 문자열)을 입력받기
n, target = list(map(int, input().split()))
# 전체 원소 입력받기
array = list(map(int, input().split()))

# 이진 탐색 수행 결과 출력
result = binary_search(array, target, 0, n - 1)
if result == None:
    print("원소가 존재하지 않습니다.")
else:
    print(result + 1)
```

3. 트리 자료구조

이진탐색 전제조건
▶ 데이터 정렬

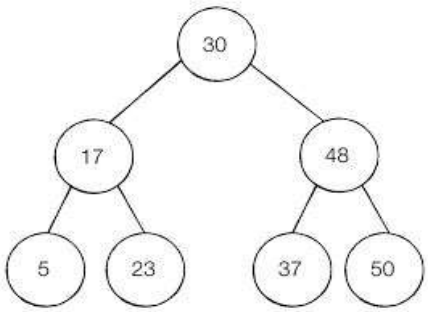
데이터베이스
▶ 내부적으로 대용량 데이터 처리에 적합한 트리 자료구조 이용
▶ 데이터가 항상 정렬되어 있다
▶ 이진 탐색과 조금 다르지만, 이진 탐색과 유사한 방법




- 트리는 부모 노드와 자식 노드의 관계로 표현된다.
- 트리의 최상단 노드를 루트 노드라고 한다.
- 트리의 최하단 노드를 단말 노드라고 한다.
- 트리에서 일부를 떼어내도 트리 구조이며 이를 서브 트리라 한다.
- 트리는 파일 시스템과 같이 계층적이고 정렬된 데이터를 다루기에 적합하다.

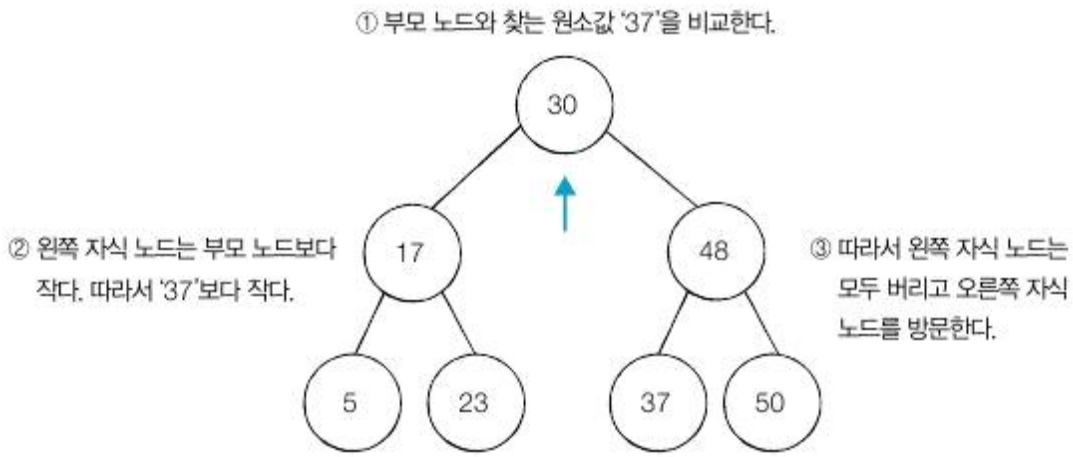
4. 이진 탐색 트리

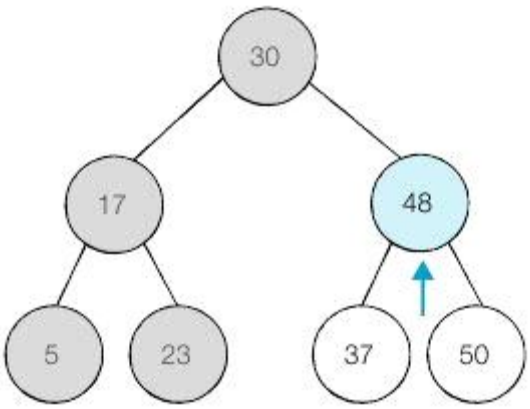
▶ 이진 탐색이 동작할 수 있도록 고안된, 효율적인 탐색이 가능한 자료구조

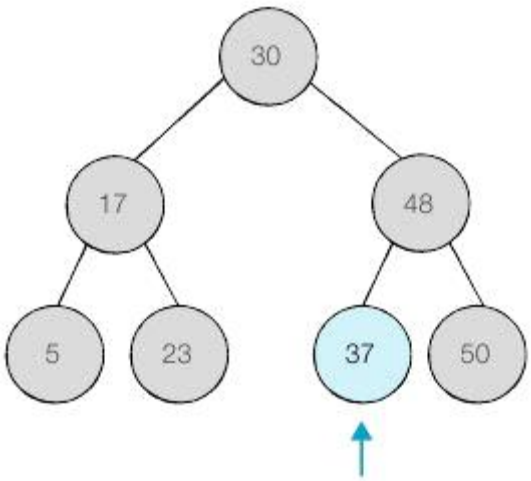



- 부모 노드보다 왼쪽 자식 노드가 작다.
- 부모 노드보다 오른쪽 자식 노드가 크다.

즉, 왼쪽 자식노드 < 부모 노드 < 오른쪽 자식 노드



5. 빠르게 입력받기

이진 탐색 문제
▶ 입력 데이터가 많거나, 탐색 범위가 매우 넓은 편

**7-4.py 한 줄 입력받아 출력하는 소스코드**

```
import sys
# 하나의 문자열 데이터 입력받기
input_data = sys.stdin.readline().rstrip()

# 입력받은 문자열 그대로 출력
print(input_data)
```

```
Hello, Coding Test! ↵
Hello, Coding Test!
```

rstrip()
▶ 입력의 맨 마지막 공백 문자를 제거해준다.

sys 라이브러리를 사용할 때는 한 줄 입력받고 나서 rstrip() 함수를 꼭 호출해야 한다. 소스코드에 readline()으로 입력하면 입력 후 엔터Enter가 줄 바꿈 기호로 입력되는데, 이 공백 문자를 제거하려면 rstrip() 함수를 사용해야 한다