CHAPTER.26

이진 검색 알고리즘의 원리와 구현 및 응용

학습 내용

[1] 이진 검색 알고리즘의 원리와 구현

[2] 이진 검색 알고리즘의 응용

학습 목표

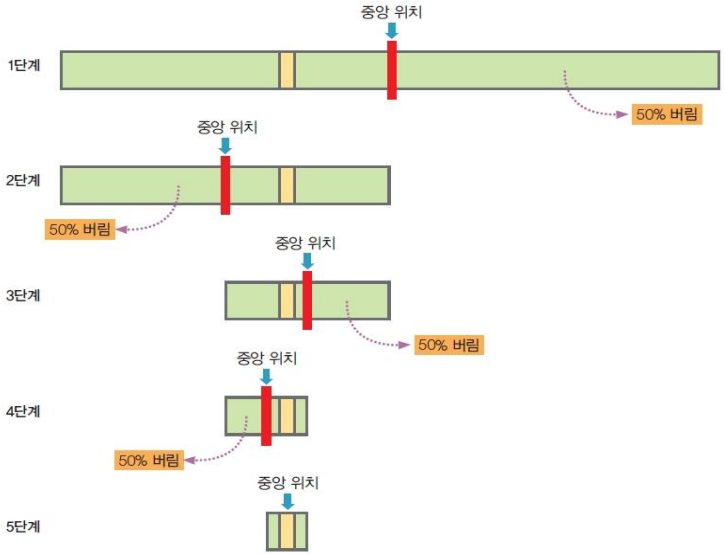
- 검색의 다양한 알고리즘에 대해 설명할 수 있다.
- 검색을 활용한 응용 프로그램을 작성할 수 있다.

01 이진 검색 알고리즘의 원리와 구현

1) 이진 검색
2) 이진 검색 구현

이진 검색의 원리와 시간 복잡도

- 이진 검색은 전체를 반씩 잘라 내서 한쪽을 버리는 방식을 사용
- 이진 탐색의 간단한 그림



이진 검색의 시간 복잡도

- 이진 검색의 데이터 개수를 ½로 줄여 가면서 비교한 시간 복잡도는 $O(log_2 n)$
- O(n)에 비해 엄청나게 효과적인 방법

예 데이터 개수가 100,000,000(1억)이라면

→ $log_2$1억 = 약 27회 정도의 비교로 데이터를 찾을 수 있음

2] 이진 검색 구현

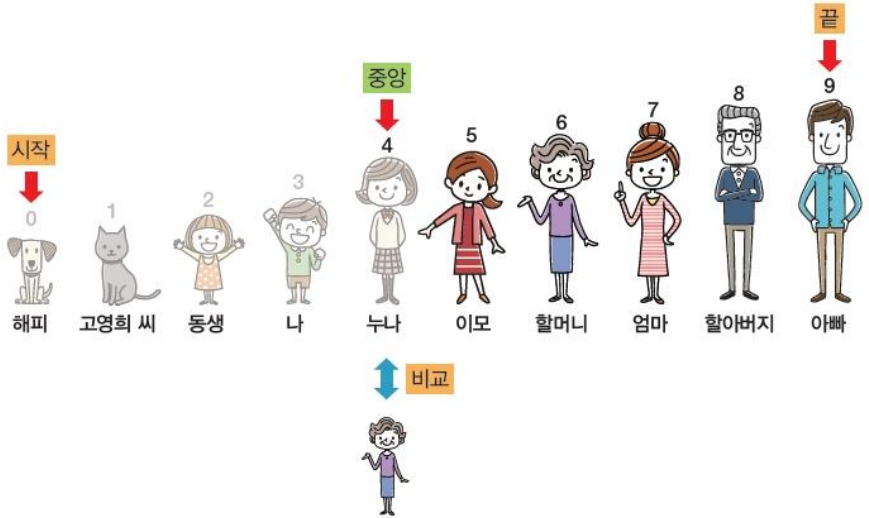
정렬된 가족 데이터에서 '할머니'와 키가 같은 사람을 찾는 과정을 이진 탐색으로 구현하는 예

- 이진 탐색할 정렬된 가족 데이터

2] 이진 검색 구현

1 전체의 첫 데이터를 '시작'으로 지정하고, 마지막 데이터를 '끝'으로 지정

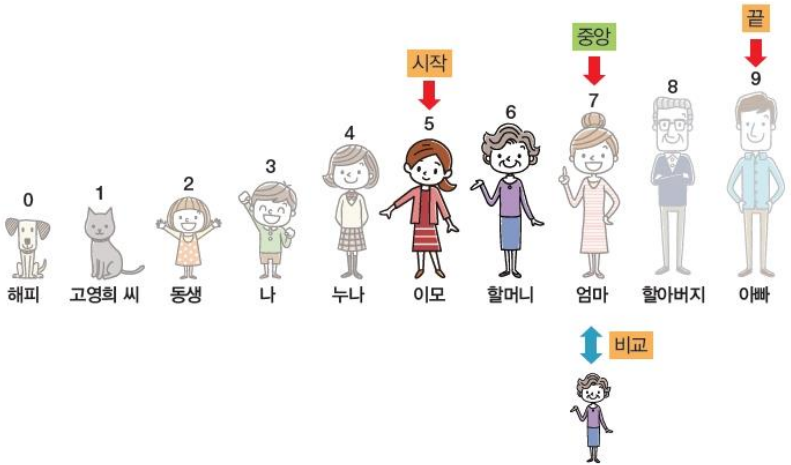
- 시작과 끝의 중앙인 누나를 할머니와 비교



2] 이진 검색 구현

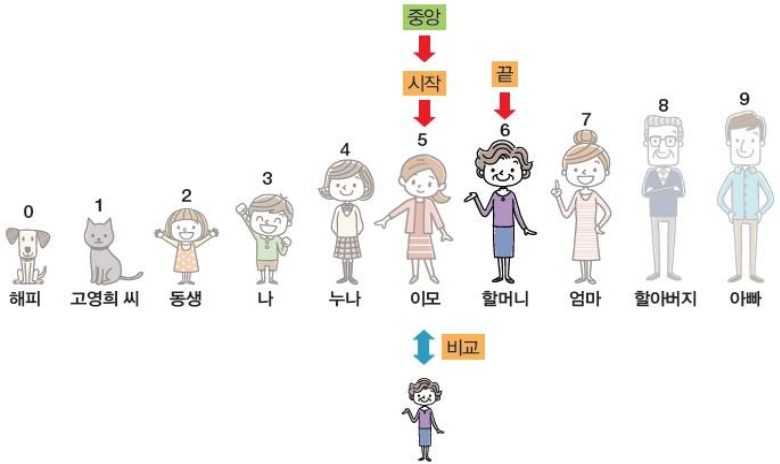
2 끝은 그대로 두고 시작을 중앙(누나)의 바로 오른쪽 이모로 옮김

- 중앙(누나)의 오른쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(엄마)을 할머니와 비교



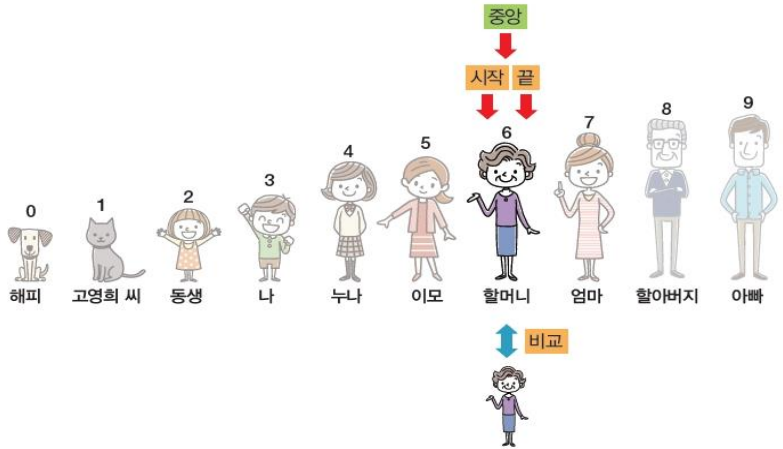
3 시작은 그대로 두고 끝을 중앙(엄마)의 바로 왼쪽인 할머니로 옮김

- 중앙(엄마)의 왼쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(이모)을 할머니와 비교

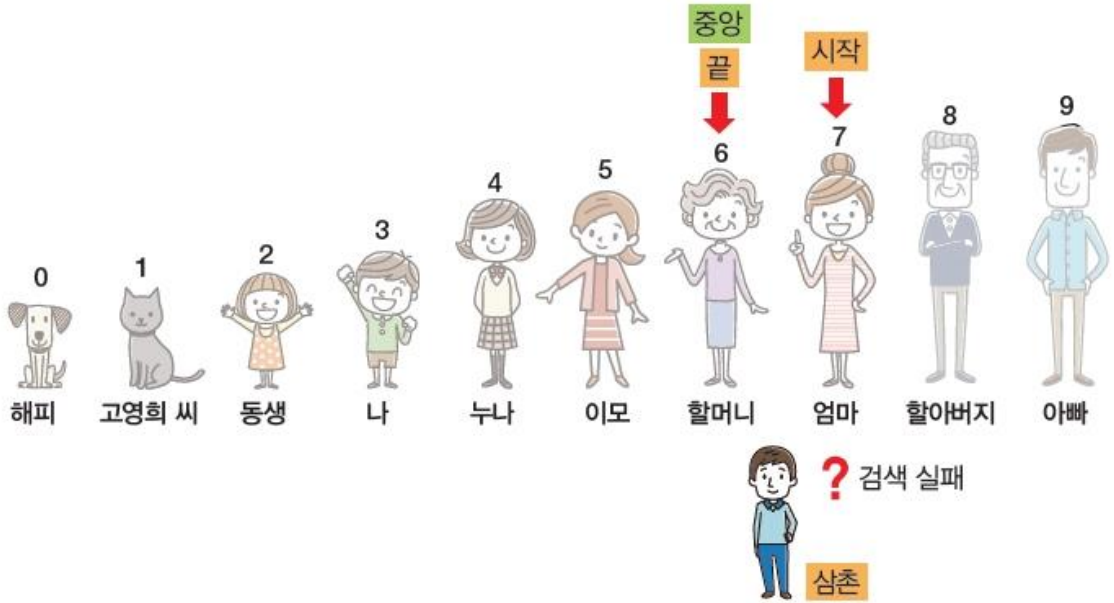


4 끝은 그대로 두고 시작을 중앙(이모)의 바로 오른쪽으로 옮김

- 중앙(이모)의 오른쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(할머니)을 할머니와 비교



이진 탐색에서 검색 실패



2] 이진 검색 구현

분할 정복

- 이진 검색처럼 검색할 범위를 ½씩 반복해서 분할하는 기법

```
시작 = 0
끝 = 배열길이 - 1
while (시작 <= 끝)
    중앙 = (시작+끝) // 2
    if 찾는 값 == 중앙 :
        검색 성공. 중앙 위치 반환
    elif 찾는값 > 중앙 :
        시작 = 중앙 + 1
    else :
        끝 = 중앙 - 1

while 문에서 중앙 위치를 반환하지 못하고, 여기까지 오면 검색 실패
```

정렬된 데이터의 이진 검색

```
1  ## 클래스와 함수 선언 부분 ##
2  def binSearch(ary, fData) :
3      pos = -1
4      start = 0
5      end = len(ary) - 1
6
7      while (start <= end) :
8          mid = (start + end ) // 2
9          if fData == ary[mid] :
10             return mid
11         elif fData > ary[mid] :
12             start = mid + 1
```

정렬된 데이터의 이진 검색

```
13          else :
14              end = mid - 1
15
16      return pos
17
18  ## 전역 변수 선언 부분 ##
19  dataAry = [50, 60, 105, 120, 150, 160, 162, 168, 177, 188]
20  findData = 162    # 할머니 키
21
22  ## 메인 코드 부분 ##
23  print('배열 -->', dataAry)
24  position = binSearch(dataAry, findData)
```

정렬된 데이터의 이진 검색

```
25 if position == -1 :
26     print(findData, '(이)가 없네요.')
27 else :
28     print(findData, '(은)는 ', position, '위치에 있음')
```

실행 결과

```
배열 --> [50, 60, 105, 120, 150, 160, 162, 168, 177, 188]
162 (은)는  6 위치에 있음
```

2] 이진 검색 구현

이진 검색 실습

앞쪽의 소스를 수정해서 랜덤하게 0과 100000 사이의 숫자 100000개를 생성한 후, 0과 100000 사이의 숫자 하나를 랜덤하게 뽑아서 찾도록 코드를 작성하자.
또 몇 번 반복해서 검색에 성공하거나 실패했는지도 확인하자.

실행 결과

```
배열 일부 --> [0, 2, 2, 3, 3] ~~~~ [99995, 99997, 99997, 99999, 100000]
35495 (은)는  35852 위치에 있음
## 17회 검색함

배열 일부 --> [0, 1, 4, 4, 7] ~~~~ [99991, 99992, 99993, 99995, 99998]
36779 (이)가 없네요.
## 16회 검색함
```

이진 검색 실습

02

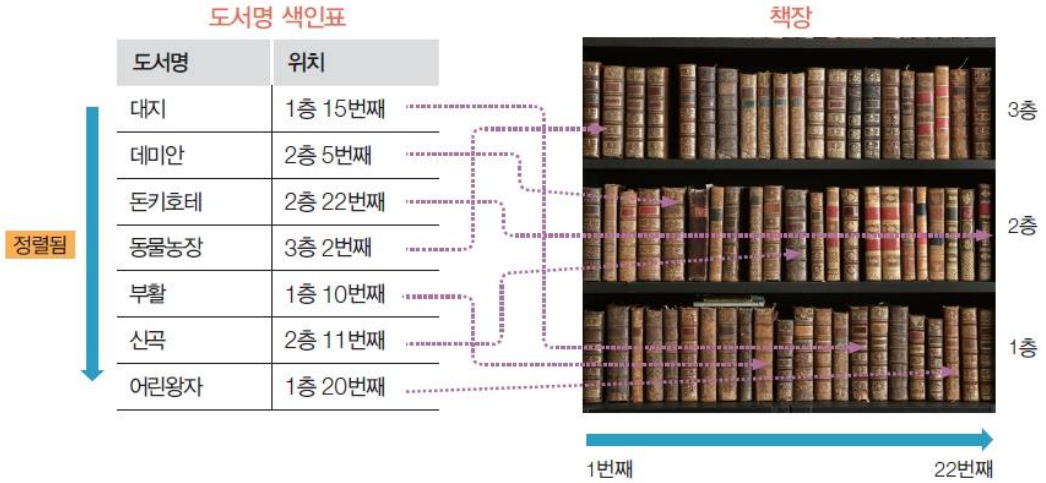
이진 검색 알고리즘의 응용

1) 색인으로 도서관 책 찾기
2) 이진 검색 알고리즘 실습

1] 색인으로 도서관 책 찾기

색인 또는 인덱스(Index)는 순서가 없는 데이터에서 특정 열(Column)만 추출하여 순서대로 정렬한 것

책장의 책을 도서명으로 찾는 경우



1] 색인으로 도서관 책 찾기

색인 또는 인덱스(Index)는 순서가 없는 데이터에서
특정 열(Column)만 추출하여 순서대로 정렬한 것

책장의 책을 도서명으로 찾는 경우

- 순서 없이 꽂혀 있는 책들을 왼쪽 [도서명 색인표]를 만들어 사용하면 필요한 책을 빠르게 찾을 수 있음

1] 색인으로 도서관 책 찾기

도서명과 작가가 있는 책장에서 도서명으로 색인을 만드는 예

작가명 색인표

정렬됨

작가명	위치
단테	2층 11번째
세르반테스	2층 22번째
쌩떽쥐베리	1층 20번째
조지오웰	3층 2번째
톨스토이	1층 10번째
펄벅	1층 15번째
헤르만헤세	2층 5번째

작가명 색인표

순번(책 위치)	도서명	작가명
0	어린왕자	쌩떽쥐베리
1	이방인	까뮈
2	부활	톨스토이
3	신곡	단테
4	돈키호테	세르반테스
5	동물농장	조지오웰
6	데미안	헤르만헤세
7	파우스트	괴테
8	대지	펄벅

색인을 만들 책장 예

1] 색인으로 도서관 책 찾기

1 책장 배열을 정의

```
책장배열 = [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
            ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
            ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
```

1] 색인으로 도서관 책 찾기

2 도서명만 추출하고, 도서명 옆에 책의 순번(=책장 위치)을 기록

- **정렬 전 색인표**

```
정렬전_색인표 = []
순번 = 0
for 책한권 in 책장배열 :
    정렬전_색인표.append((책한권[도서명], 순번))
    순번 += 1
```

도서명	순번
어린왕자	0
이방인	1
부활	2
신곡	3
돈키호테	4
동물농장	5
데미안	6
파우스트	7
대지	8

3 도서명을 기준으로 색인표 배열을 정렬

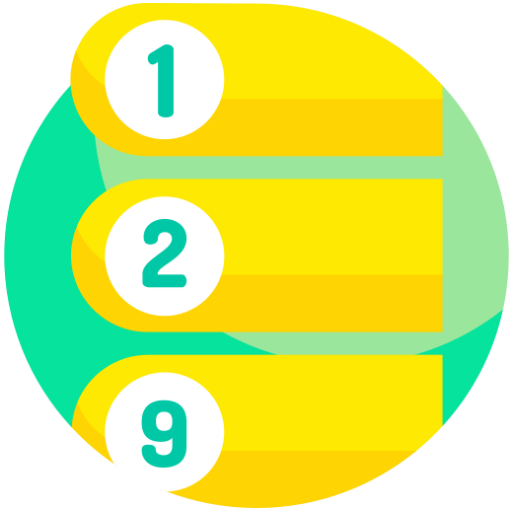
- 파이썬의 sorted( ) 함수를 사용해서 0번째 열인 도서명 열을 기준으로 정렬
- 정렬 후 색인표

```
정렬후_색인표 = sorted(정렬전_색인표, key=0번째 열)
```

도서명	순번
대지	8
데미안	6
돈키호테	4
동물농장	5
부활	2
신곡	3
어린왕자	0
이방인	1
파우스트	7

1] 색인으로 도서관 책 찾기

4 같은 방식으로 ❷ ~ ❸의 과정을 거쳐 [작가명 색인표]를 만들기



1] 색인으로 도서관 책 찾기

정렬된 데이터의 이진 검색

```
1  from  operator import itemgetter
2
3  ## 클래스와 함수 선언 부분 ##
4  def makeIndex(ary, pos) :
5      beforeAry = []
6      index = 0
7      for data in ary :
8          beforeAry.append((data[pos], index))
9          index += 1
10
11     sortedAry = sorted(beforeAry, key=itemgetter(0))
12     return sortedAry
13
14 def bookSearch(ary, fData) :
15     pos = -1
```

정렬된 데이터의 이진 검색

```
16      start = 0
17      end = len(ary) - 1
18
19      while (start <= end) :
20          mid = (start + end ) // 2
21          if fData == ary[mid][0] :
22              return ary[mid][1]
23          elif fData > ary[mid][0] :
24              start = mid + 1
25          else :
26              end = mid - 1
27
28      return pos
29
```

정렬된 데이터의 이진 검색

```
30 ## 전역 변수 선언 부분 ##
31 bookAry = [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
32           ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
33           ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
34 nameIndex = []
35 authIndex = []
36
37 ## 메인 코드 부분 ##
38 print('# 책장의 도서 ==>', bookAry)
39 nameIndex = makeIndex(bookAry, 0)
40 print('# 도서명 색인표 ==>', nameIndex)
41 authIndex = makeIndex(bookAry, 1)
42 print('# 작가명 색인표 ==>', authIndex)
43
```

정렬된 데이터의 이진 검색

```
findName = '신곡'
findPos = bookSearch(nameIndex, findName)
if findPos != -1 :
    print(findName, '의 작가는', bookAry[findPos][1], '입니다.')
else :
    print(findName, ' 책은 없습니다.')

findName = '괴테'
findPos = bookSearch(authIndex, findName)
if findPos != -1 :
    print(findName, '의 도서는', bookAry[findPos][0], '입니다.')
else :
    print(findName, ' 작가는 없습니다.')
```

정렬된 데이터의 이진 검색

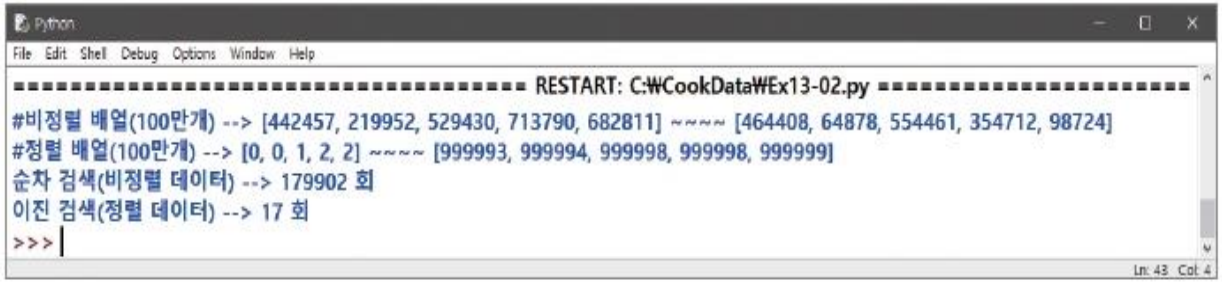
실행 결과

```
# 책장의 도서 ==> [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'], ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'], ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]

# 도서명 색인표 ==> [('대지', 8), ('데미안', 6), ('돈키호테', 4), ('동물농장', 5), ('부활', 2), ('신곡', 3), ('어린왕자', 0), ('이방인', 1), ('파우스트', 7)]

# 작가명 색인표 ==> [('괴테', 7), ('까뮈', 1), ('단테', 3), ('세르반테스', 4), ('쌩떽쥐베리', 0), ('조지오웰', 5), ('톨스토이', 2), ('펄벅', 8), ('헤르만헤세', 6)]

신곡 의 작가는 단테 입니다.
괴테 의 도서는 파우스트 입니다.
```

이진 검색 알고리즘 실습

데이터 100만 개를 랜덤하게 생성해서 정렬되지 않은 배열과 정렬된 배열을 각각 준비한다. 정렬 여부만 다르지 두 배열에는 동일한 값이 들어 있다. 정렬되지 않은 배열에는 순차 검색으로 몇 번 만에 데이터를 찾는지, 정렬된 배열에서는 이진 검색으로 몇 번 만에 데이터를 찾는지 비교한다.

실행 결과



이진 검색 알고리즘 실습

학습평가

Q1 Q2

Q1

이진 검색의 성능에 대한 빅-오 표기법은 다음 중 무엇인가?

1. O(n)
2. O(2n)
3. O($log_2 n$)
4. O(n^2)

학습평가

Q1 Q2

Q1 이진 검색의 성능에 대한 빅-오 표기법은 다음 중 무엇인가?

1. O(n)
2. O(2n)
3. $O(log_2 n)$
4. $O(n^2)$

정 답 3 $O(log_2 n)$

해 설 이진 검색은 $O(log_2 n)$의 성능을 가집니다.

학습평가

Q1 Q2

Q2 정렬된 집합을 이진 검색할 때의 설명으로 맞지 않는 것은?

1. 데이터가 반드시 정렬되어 있어야 사용할 수 있다.
2. 순차 검색에 비해 엄청난 성능으로 데이터를 검색할 수 있다.
3. 시간 복잡도는 O(n) 이다.
4. 데이터 수가 ½씩 급격히 줄어드는 효과를 준다.

학습평가

Q1 Q2

Q2 정렬된 집합을 이진 검색할 때의 설명으로 맞지 않는 것은?

1. 데이터가 반드시 정렬되어 있어야 사용할 수 있다.
2. 순차 검색에 비해 엄청난 성능으로 데이터를 검색할 수 있다.
3. ✓ 시간 복잡도는 O(n) 이다.
4. 데이터 수가 ½씩 급격히 줄어드는 효과를 준다.

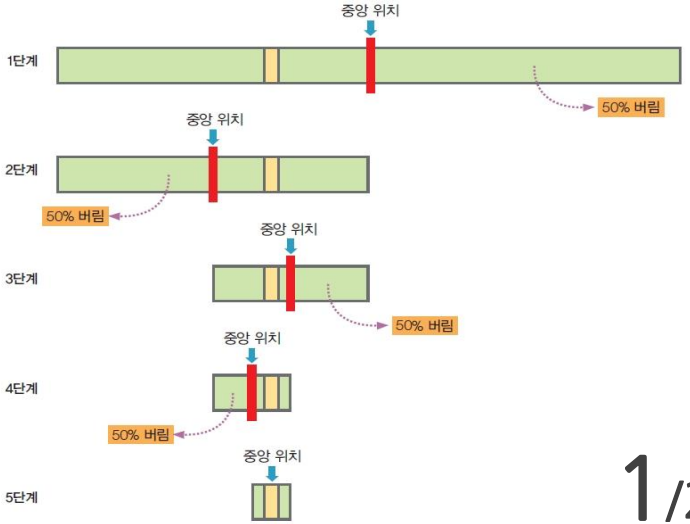
정 답 3 시간 복잡도는 $O(n)$이다.

해 설 이진 검색은 $O(log_2 n)$의 성능을 가집니다.

이진 검색 알고리즘의 원리와 구현

이진 검색의 개념

- 이진 검색은 전체를 반씩 잘라 내서 한쪽을 버리는 방식을 사용
- 이진 탐색의 간단한 그림



이진 검색 알고리즘의 원리와 구현

이진 검색의 시간 복잡도

- 이진 검색의 데이터 개수를 ½로 줄여 가면서 비교한 시간 복잡도는 $O(log_2 n)$
- $O(n)$에 비해 엄청나게 효과적인 방법

예 데이터 개수가 100,000,000(1억)이라면

→ $log_2 1억$ = 약 27회 정도의 비교로 데이터를 찾을 수 있음