Lecture #11 | 검색 알고리즘 (search algorithm)
알고리즘 복잡도 (algorithm complexity)

SE213 프로그래밍 (2019)

Written by Mingyu Cho

Edited by Donghoon Shin

3rd assignment (5/20 midnight)

- 과제 3-1: 문자표현
  - character(decoding_table, code) → (str) code에 해당하는 글자 하나를 포함하는 문자열
  - ordinal(decoding_table, char) → (int) Char에 해당하는 정수 코드값
  - decode(decoding_table, codes) → (str) codes에 해당하는 문자열
  - encode(decoding_table, string) → (list) string의 각 문자에 에 해당하는 정수 코드값의 리스트
- 과제 3-2: 동아리
  - read_circles(text_file) → (dict) 동아리 이름을 키로, 학생 리스트를 value로 갖는 사전
  - fine_circles(data, student_name) → (list) 학생이 속한 동아리들로 구성한 리스트를 반환
  - fine_members(data, *circles) → (list) circles에 공동으로 속한 학생들로 구성한 리스트를 반환
- 과제 3-3: 중간고사 성적 정리
  - print_scores(studnets, scores, courses, student_id) → (반환 없음) 화면에 일정 포맷에 맞춰 제목 줄 및 id, 이름, 각 과목 점수, 평균 출력
  - print_top_n(studnets, scores, courses, n) → (반환 없음) 화면에 일정 포맷에 맞춰 성적 상위 n명에 대한 정보 출력
  - print_all(studnets, scores, courses) → (반환 없음) 학번 순으로 정렬 후, 일정 포맷에 맞춰 제목줄 및 id, 이름, 각과목 점수, 평균 출력

지난 시간에 다룬 내용

- 클래스 예
- 재귀호출

오늘 다룰 내용

- 알고리즘이란
- 검색(탐색) 알고리즘
  - 순차 검색 (sequential search)
  - 이진 검색 (binary search)
- 알고리즘 복잡도

알고리즘에 대한 소개

- 알고리즘
  - 특정한 유형의 문제를 해결하기 위한 체계적인 방법
  - 어떠한 형태의 입력에 대해서도 정확한 답을 찾는 방법
  - 예시: 10진수의 사칙연산, 검색, 정렬, …
- 알고리즘 복잡도: 알고리즘이 좋고 나쁨을 판단하는 기준
  - 시간 복잡도: 얼마나 빨리 실행되는지?
  - 공간 복잡도: 얼마나 많은 추가 저장 공간을 필요로 하나?

검색 (searching): 문제의 정의

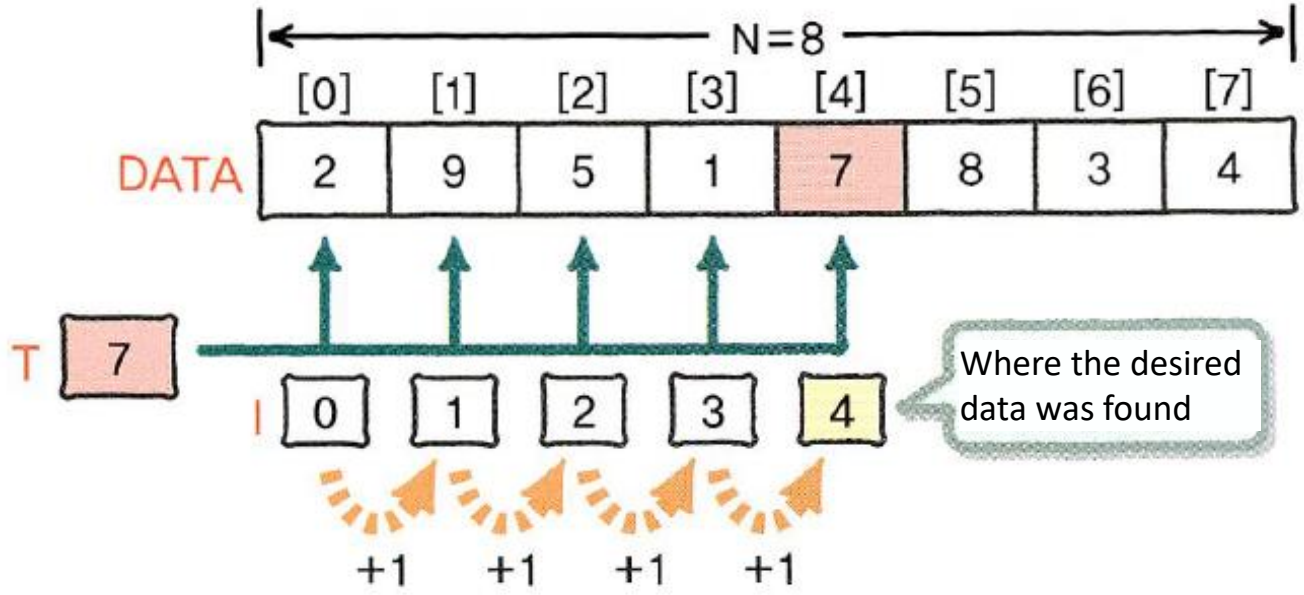
- 리스트에서 찾고자 하는 원소가 어디에 있는지, 즉 인덱스를 찾는 것

'a'	'b'	'a'	'c'	'a'

  - 'a'의 인덱스 0
  - 'c'의 인덱스 3

순차 검색 (sequential search)

- 원하는 값을 찾을 때까지, 리스트의 원소를 하나씩 비교
- 예시: Find item 7 in item list [2, 9, 5, 1, 7, 8, 3, 4]



sequential_search() – with range() and len()

```
def sequential_search(sequence, item):
    for index in range(len(sequence)):
        if sequence[index] == item:
            return index
    return None
```

sequential_search() – with enumerate()

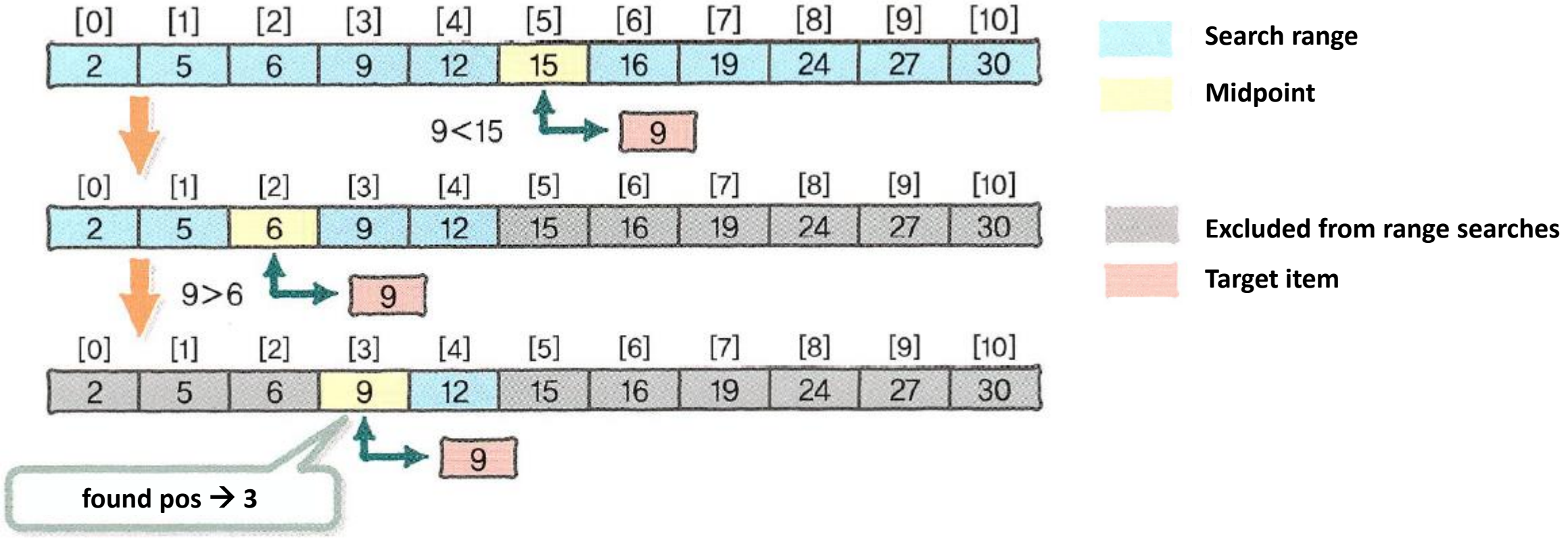
```
def sequential_search(sequence, item):
    for index, value in enumerate(sequence):
        if value == item:
            return index
    return None
```

이진 검색 (binary search)

- **리스트가 정렬되어 있는 경우**, 더 효율적인 검색을 하기 위한 방법
- 리스트가 (오름차순으로) 정렬되어 있을 때, 원하는 원소를 찾기 위해
  - 처음에는 리스트 전체를 검색을 하는 범위로 설정
  - 검색을 하는 범위의 가운데 있는 수가 원하는 원소인지 비교
    - 만약 그렇다면, 검색을 완료!
    - 가운데 있는 수가 원하는 수보다 크면, 처음부터 가운데 하나 전까지를 검색하는 범위로 설정한 후 반복
    - 가운데 있는 수가 원하는 수보다 작다면, 가운데 하나 다음부터 끝까지를 검색하는 범위로 설정한 후 반복

예시: 이진 검색 (binary search)

- Find the item 9 in sorted list [2, 5, 6, 9, 12, 15, 16, 19, 24, 27, 30]



binary_search()

```
def binary_search(sequence, item):
    first = 0
    last = len(sequence) - 1  # last is INCLUSIVE

    while first <= last:
        mid = (first + last) // 2
        if sequence[mid] == item:
            return mid

        if item < sequence[mid]:
            last = mid - 1
        else:
            first = mid + 1

    return None
```

binary_search() – alternative implementation

```
def binary_search(sequence, item):
    first = 0
    last = len(sequence)  # last is EXCLUSIVE

    while first < last:
        mid = (first + last) // 2
        if sequence[mid] == item:
            return mid

        if item < sequence[mid]:
            last = mid
        else:
            first = mid + 1

    return None
```

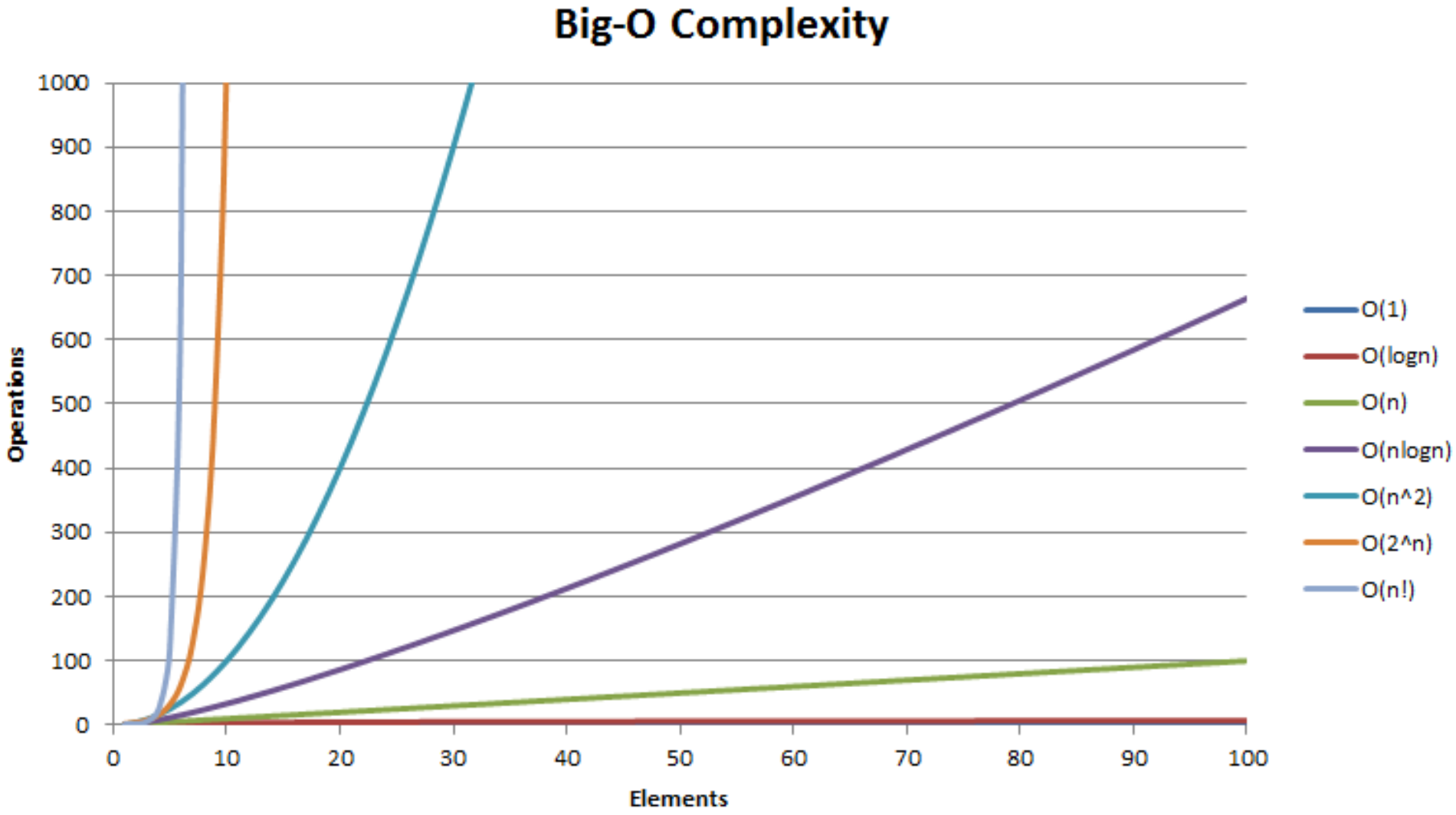
알고리즘 복잡도 (algorithm complexity)

- 알고리즘 복잡도 분석 (algorithm complexity analysis)
  - 특정한 알고리즘이 문제의 크기(n)에 따른 실행시간(시간 복잡도)과 추가적으로 필요한 저장공간(공간 복잡도)을 분석하는 것
  - Big-O notation을 이용하여 나타냄 (주로 고차항만으로 표시)
  - 최선/평균/최악(best/**average**/**worst** case)의 경우 분석
- 시간 복잡도 분석
  - 비교, 대입, 기본연산이 단위 시간(1)의 실행 시간을 가진다고 가정
  - 자료구조(리스트, 튜플, 문자열 등)에 대한 연산은 별도도 어떠한 시간 복잡도를 가지는지 분석해야 함
- 공간 복잡도
  - 기본 자료형(`int`, `float`, 글자)이 단위 저장 공간을 가진다고 가정

알고리즘 복잡도에 따른 비교 (1/2)

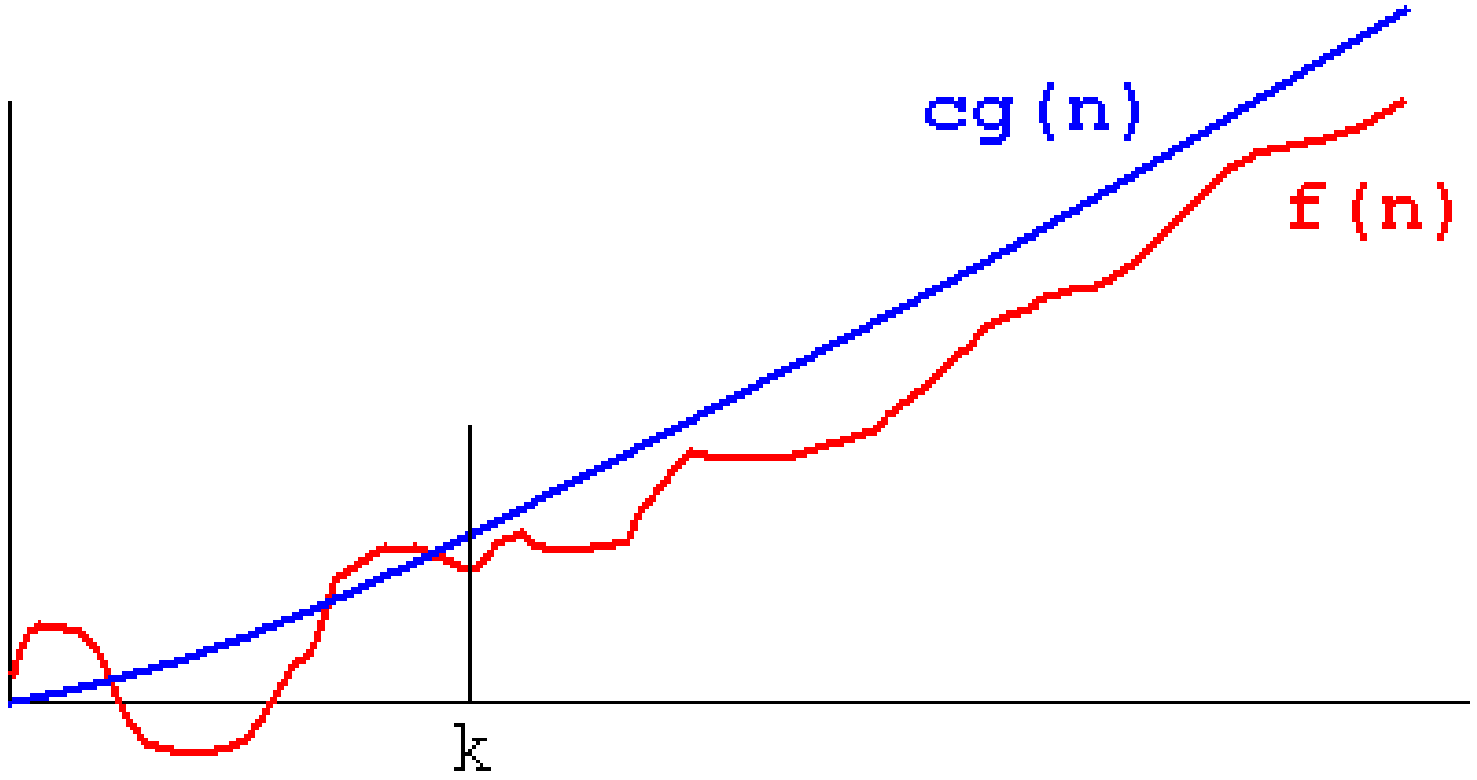
	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
log n	0	1	2	3	4	5
n	1	2	4	8	16	32
n log n	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40320	20.9T	Don't ask!

알고리즘 복잡도에 따른 비교 (2/2)



참고: Big-O notation 정의

- 정의: 모든 $n \geq k$에 대하여, 어떠한 상수 c와 k가 $0 \leq f(n) \leq c \cdot g(n)$을 만족할 때, $f(n)=O(g(n))$으로 표시한다. 이 때 c와 k는 n과 독립적으로 고정된 상수이여야 한다.



검색 알고리즘 시간 복잡도 분석

- 순차 검색: 처음부터 하나씩 원소를 검색
  - Best case: $O(1)$
  - Worst case: $O(n)$
  - Average case: $O\left(\frac{n}{2}\right) = O(n)$

- 이진 검색: 한 번씩 찾는 범위가 반씩 줄어듦
  - Best case: $O(1)$
  - Worst case: $O(\log n)$
  - Average case: $O(\log n)$

읽을 거리

- 검색 알고리즘: https://en.wikipedia.org/wiki/Search_algorithm
- 알고리즘 복잡도 분석: https://en.wikipedia.org/wiki/Analysis_of_algorithms

ANY QUESTIONS?