파이썬으로 배우는 데이터 구조

한동대학교
전산전자공학부
김영섭 교수

학습 목표

이진 검색(binary search)을 이해하고

시간복잡도를 구할 수 있다

Data Structures in Python
Chapter 5 - 1

- **Binary Search**
- Recursive Binary Search
- Bubble sort
- Selection sort
- Insertion sort

Agenda & Readings

- **Binary Search**
- Recursive Binary Search

Binary search

- Look at the following program that generates a random integer and then gives clues to a user trying to guess the number.

```
import random
RANGE = 128
secret = random.randrange(RANGE)
print('I am thinking of a secret number between 0 and', RANGE - 1)
guess = 0
while guess != secret:
    guess = int(input('What is your guess? '))
    if    guess < secret: print('Too low')
    elif  guess > secret: print('Too high')
    else:                 print('Congrat!')
```

- Guess the value of a secret number that is one of the n integers between 0 and n - 1.
- Each time that you make a guess, you are told whether your guess is equal to the secret number, too high, or too low.

0	1	2	.	.	.	76	77	78	.	.	.	125	126	127

Binary search

- Look at the following program that generates a random integer and then gives clues to a user trying to guess the number.

```
import random
RANGE = 128
secret = random.randrange(RANGE)
print('I am thinking of a secret number between 0 and', RANGE - 1)
guess = 0
while guess != secret:
    guess = int(input('What is your guess? '))
    if    guess < secret: print('Too low')
    elif  guess > secret: print('Too high')
    else:                 print('Congrat!')
```

- **Question 1:** Assume that n is a power of 2.
  - How many times can you guess to get to the answer all the time?
  - Can you express it in terms of n?
- **Question 2:**
  - What would be the value of the RANGE if you can guess 20 times?
  - Can you express it in terms of n?

Exercise: 스무고개 1

- Improve the program a bit:
  - Show the maximum number of guesses that the user can make.
  - Print the message "Nice try. I'm sure you'll do better next time" if the user fails.
- Sample Run:

```
I am thinking of a secret number between 0 and 127
What is your guess(chance:7)?  64
Too low
What is your guess(chance:6)?  96
Too low
What is your guess(chance:5)?  112
Too high
What is your guess(chance:4)?  94
Too low
What is your guess(chance:3)?  98
Too low
What is your guess(chance:2)?  100
Too low
What is your guess(chance:1)?  101
Too low
Nice try. I'm sure you'll do better next time.
```

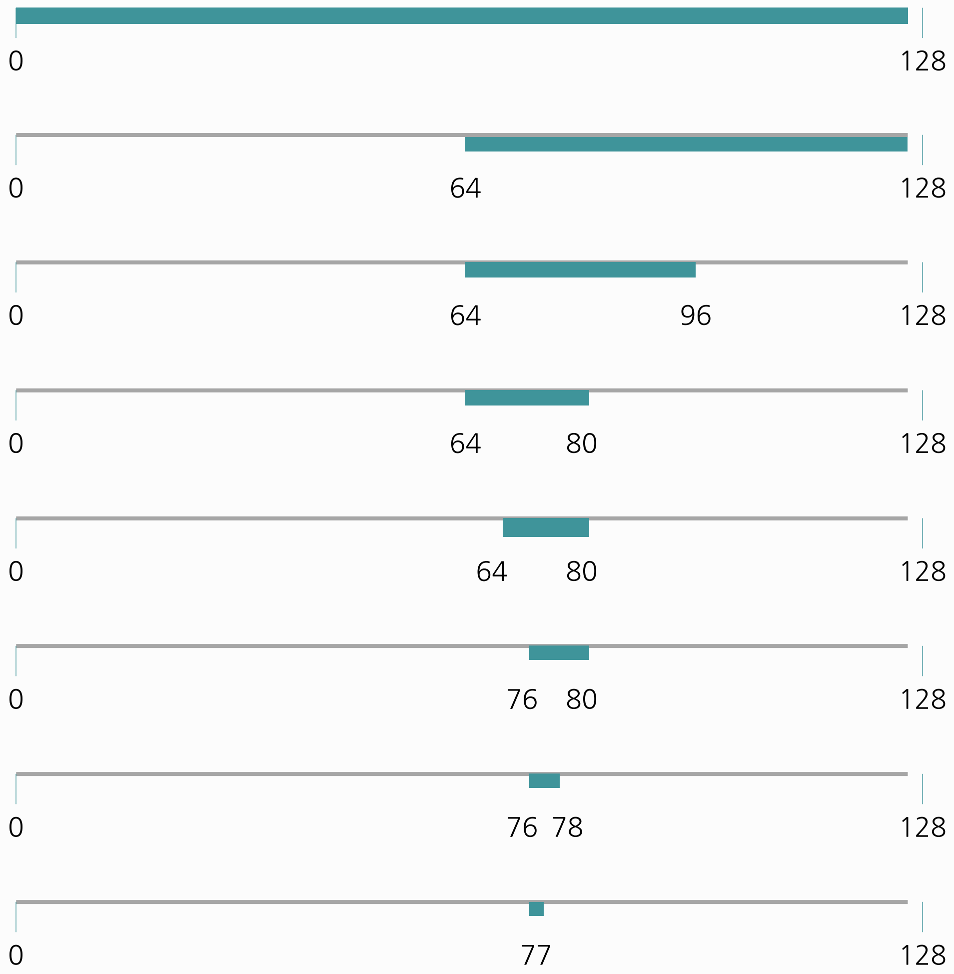
Exercise: 스무고개 2

- This script uses binary search to play the same game, but with the roles **reversed**: you choose the secret number, and the program guesses its value:
  - It asks the user to enter the number of guesses (or questions) k.
  - It displays the RANGE based on k such that the user can think of a number between 0 and 2^k - 1.
  - Then the computer always guesses the answer with k questions.

Exercise: 스무고개 2

- Finding a hidden number with **binary search**: Is the number greater than or equal to m?

search interval



length	question	response
128	>= 64 ?	true
64	>=96 ?	false
32	>= 80 ?	false
16	>= 72 ?	true
8	>= 76 ?	true
4	>= 78 ?	false
2	>= 77 ?	true
1	= 77	

An effective strategy is to guess the number in the middle of the interval m.

Use a half-open interval [lo, hi)
mid = (hi + lo) // 2

Exercise: 스무고개 2

- We use the notation `[lo, hi)` to denote all the integers greater than or equal to lo and less than (but not equal to) hi.
  - `[lo, hi)` is called a half-open interval which contains the left endpoint but not the right one.

- We start with `lo = 0` and `hi = n` and use the following **recursive** strategy.
  - **Base case**: If `hi - lo` equals 1, then the secret number is lo.
  - **Recursive step**: Otherwise, ask whether the secret number is greater than or equal to the number `mid = (hi + lo) // 2`. If so, look for the number in `[mid, hi)`; if not, look for the number in `[lo, mid)`.

Binary search: Analysis of running time

- Let n be the number of possible values. In Exercise 2, we have $\boldsymbol{n} = \boldsymbol{2^k}$, where $\boldsymbol{k} = \log_2 n$. Now, let $T(n)$ be the number of questions. The recursive strategy immediately implies that $T(n)$ must satisfy the following **recurrence relation**:

$$T(n) = T(n/2) + 1$$

with $T(1) = 0$.

- Substituting 2^k for n, we can telescope the recurrence (apply it to itself) to immediately get a closed-form expression:

since $T(2^{k-1}) = T(2^{k-2}) + 1$

$$T(2^k) = T(2^{k-1}) + 1 = T(2^{k-2}) + 2 = \ldots = T(1) + k = k$$

$T(2^{k-1}) = T(2^{k-2}) + 1$

- Substituting back n for 2^k (and $\log_2 n$ for k) gives the result

$$T(n) = \log_2 n$$

- We say Binary Search has the time complexity $\boldsymbol{O}(\log \boldsymbol{n})$. Note: Binary search work even when n is not a power of 2.

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$\log_2 n = \log_2 2^k$$
$$\log_2 n = k$$

Example 1

- If we have an alphabetically sorted list of 100 names, how many records do we need to look at to find a given individual?
  - Since the list is sorted, we can use binary search.
  - Look at the middle element: if it's after than the name we're looking for, search the first half of the list. If it's before the name we're looking for, look at the second half of the list.
  - Each check cuts the size of the list in half. Then, how many times can we do this?

  - **Answer:**

Example 1

- If we have an alphabetically sorted list of 100 names, how many records do we need to look at to find a given individual?
  - Since the list is sorted, we can use binary search.
  - Look at the middle element: if it's after than the name we're looking for, search the first half of the list. If it's before the name we're looking for, look at the second half of the list.
  - Each check cuts the size of the list in half. Then, how many times can we do this?

  - **Answer:**
    If we think backwards, in terms of doubling the list, we'll need n doublings to generate a list of length 2^n. For example, if $2^n = 128$, then $n = log_2 128 = 7$.

Example 2

- Let's suppose that we begin with a value N, divide it by 2, then the result that we divide it by 2, and so on, until reaching 1 or less.
  - N, N/2, N/4, ……, 4, 2, 1

- **Question:** How many times did we divide before reaching 1 or less?
- **Answer:**

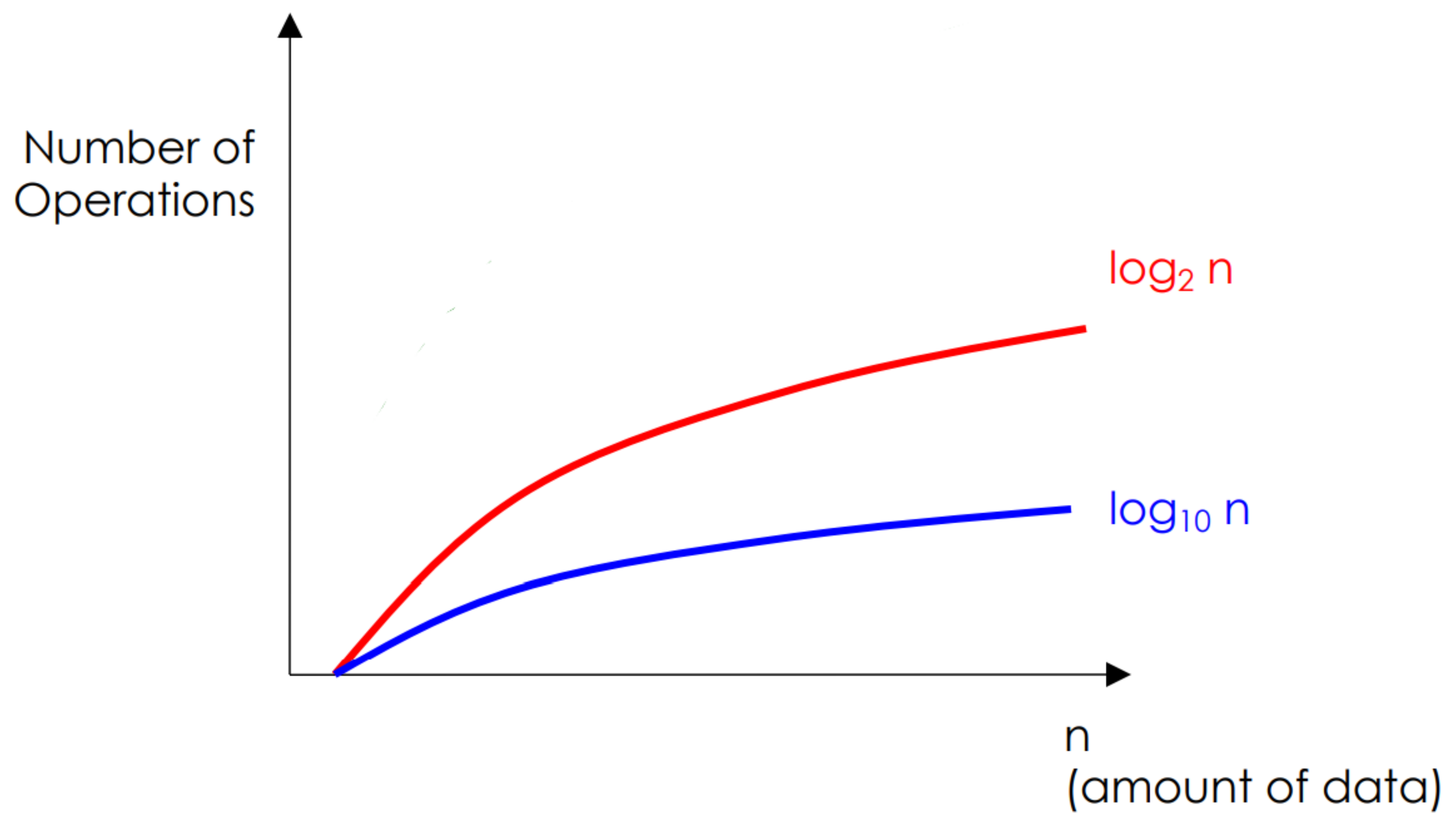
Example 2

- Let's suppose that we begin with a value N, divide it by 2, then the result that we divide it by 2, and so on, until reaching 1 or less.
  - N, N/2, N/4, ……, 4, 2, 1

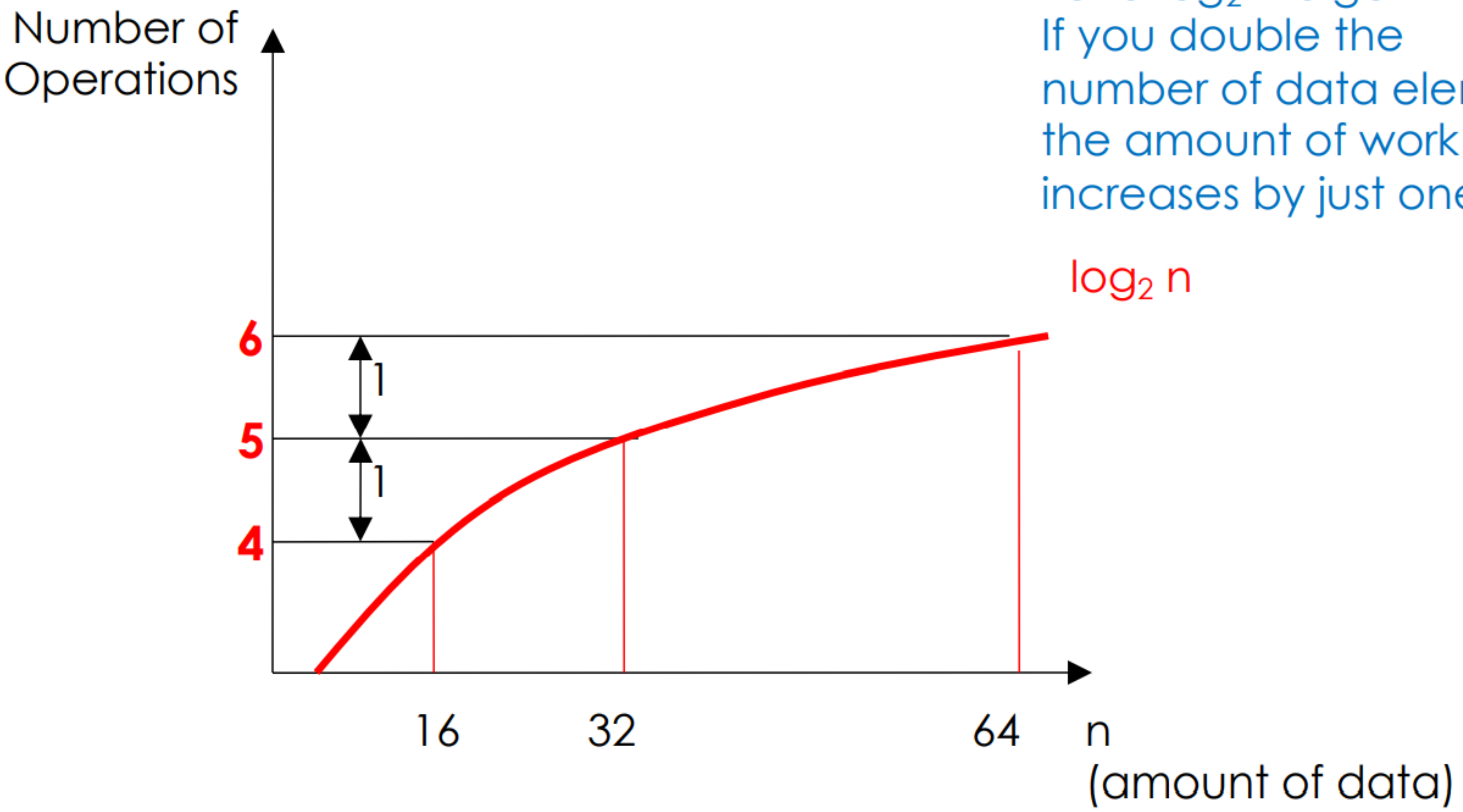
- **Question:** How many times did we divide before reaching 1 or less?
- **Answer:**
  Think of it from the other direction: How many times do I have to multiply by 2 to reach N?
  - 1, 2, 4, …., N/4, N/2, N
    Call this k number of times, then N = 2^$\boldsymbol{k}$, or $\boldsymbol{k}$ = log(N).

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$\log_2 n = \log_2 2^k$$
$$\log_2 n = k$$

Time Complexity $O(\log n)$ "Logarithmic Time"



Time Complexity $O(\log n)$ "Logarithmic Time"



For a $\log_2 n$ algorithm,
If you double the number of data elements the amount of work you do increases by just one unit

Binary Search (Worst Case)

- Finding an element in a list with one million elements requires only 20 guesses (questions or comparison)!
- But the list **must be sorted**.
  - What if we sort the list first using insertion sort?
    - Insertion sort $O(n^2)$ (worst case)
    - Binary search $O(\log n)$ (worst case)
    - Total time complexity $O(n^2) + O(\log n) = O(n^2)$
  - Fortunately, there are faster ways to sort.

Number of Elements	Number of Comparisons
16	4
32	5
64	6
128	7
256	8
1024	10
1,000,000	20

Summary

- Binary search is simple, but powerful!
- Binary search may be implemented using either iteration or recursion.
- Its time complexity is $O(\log n)$.

학습 정리

1) 이진 검색(binary search)은 정렬된 자료를 전제조건으로 한다

2) 정렬된 자료에 대한 이진 검색의 시간복잡도는 O(log n) 이다

파이썬으로 배우는
데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다