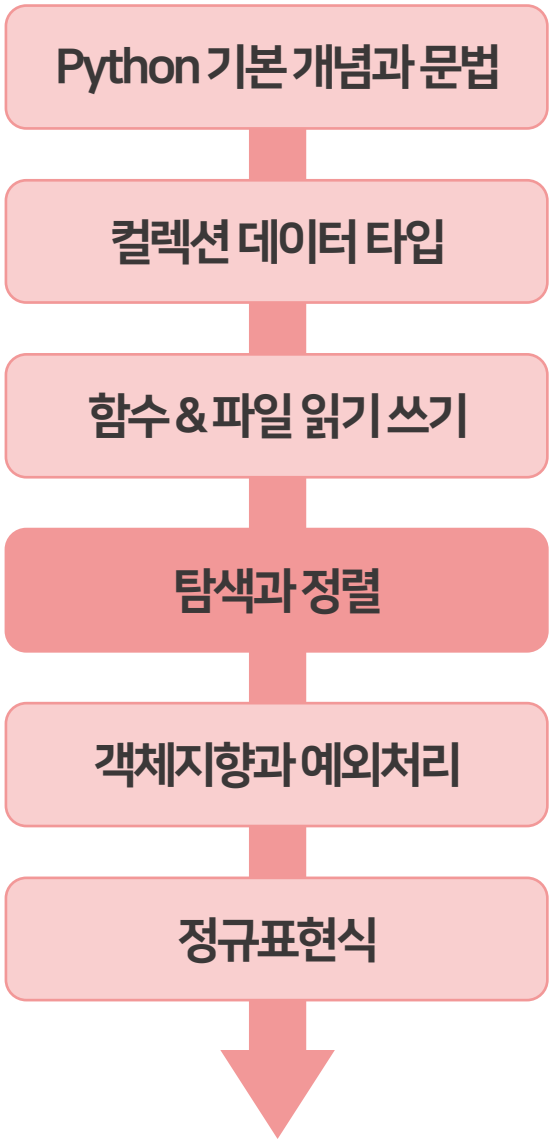

python™



1. 탐색
2. 시간 복잡도와 공간 복잡도
3. 시간 복잡도 표기법 (빅-오 표기법)
4. 순차 탐색
5. 이진 탐색
6. 정렬
7. 선택 정렬
8. 삽입 정렬
9. 병합 정렬

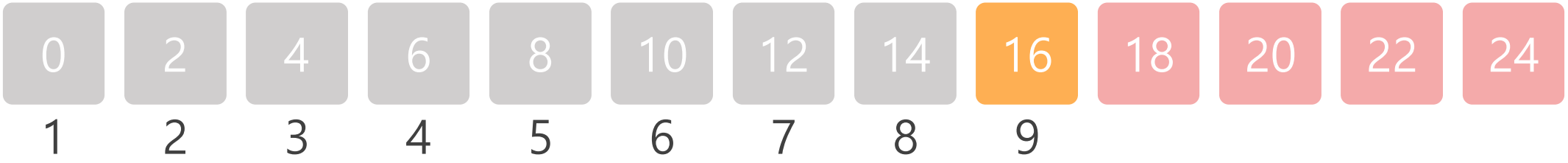
탐색

순차 탐색 (Sequential Search) - 앞에서부터 데이터를 하나씩 확인하는 방법

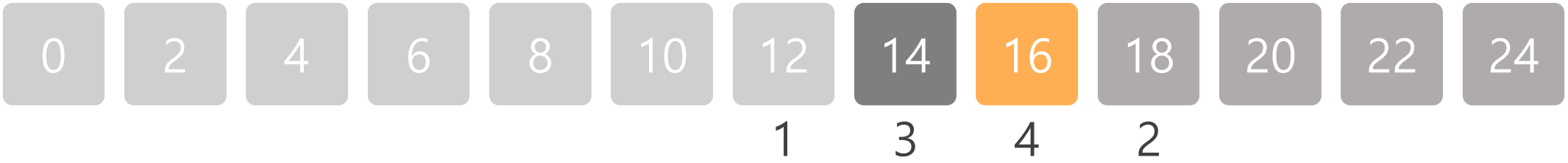
이진 탐색 (Binary Search) - 탐색 범위를 절반씩 좁혀가며 데이터를 탐색하는 방법

탐색

순차 탐색 (Sequential Search) - 앞에서부터 데이터를 하나씩 확인하는 방법



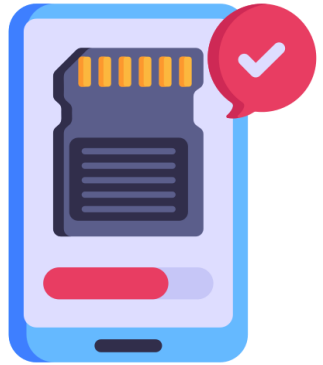
이진 탐색 (Binary Search) - 탐색 범위를 절반씩 좁혀가며 데이터를 탐색하는 방법



복잡도

시간 복잡도

특정한 크기의 입력에 대한
알고리즘의 수행 시간을 분석



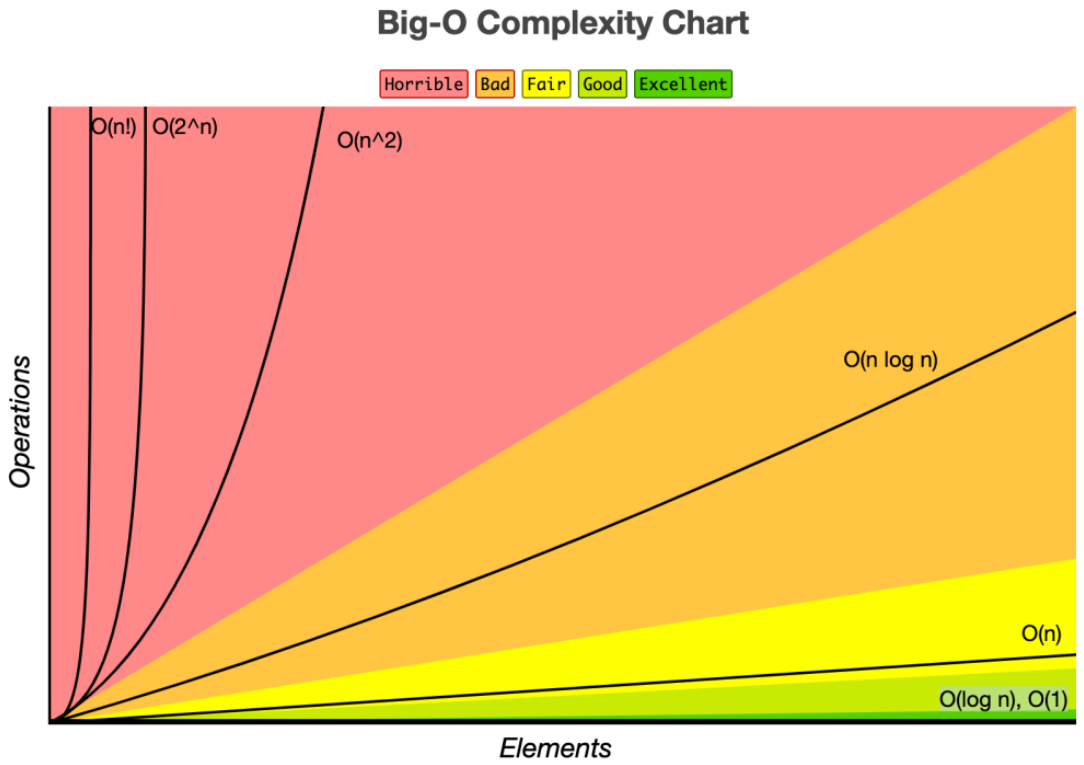
공간 복잡도

특정한 크기의 입력에 대한
알고리즘의 메모리 사용량을 분석

시간 복잡도 표기법 (빅-오 표기법 : big-O notation)

입력값의 변화에 따라 연산을 실행할 때, 연산 횟수에 비해 시간이 얼마만큼 걸리는가?

빅-오 표기법의 종류
$O(1)$
$O(n)$
$O(\log n)$
$O(n^2)$
$O(2^n)$
$O(n!)$



순차 탐색

```
def find_my_number(list, value) :
    length = len(list)
    for i in range(length) :
        if value == list[i] :
            return i
    return -1
```

이진 탐색

```
def find_my_number(list, value):
    start = 0
    end = len(list) - 1
    while start <= end:
        mid = (start + end) // 2
        if value == list[mid]:
            return mid
        elif value > list[mid]:
            start = mid + 1
        else:
            end = mid - 1
    return -1
```

순차 탐색

앞에서 배운 순차 탐색은 가장 앞에 있는 요소를 찾아서 위치를 반환해주었다.

만약 중복된 요소가 있다면, 모든 위치를 리스트로 반환할 수 있도록 프로그램을 작성하시오.

입력	출력
7 1 9 3 3 4 5 3 3	[3, 4, 7]

수 찾기 https://www.acmicpc.net/problem/1920

N개의 정수 A[1], A[2], …, A[N]이 주어져 있을 때,
이 안에 X라는 정수가 존재하는지 알아내는 프로그램을 작성하시오.

첫째 줄에 자연수 N(1 ≤ N ≤ 100,000)이 주어진다.
다음 줄에는 N개의 정수 A[1], A[2], …, A[N]이 주어진다.
다음 줄에는 M(1 ≤ M ≤ 100,000)이 주어진다.
다음 줄에는 M개의 수들이 주어지는데,
이 수들이 A안에 존재하는지 알아내면 된다.

M개의 줄에 답을 출력한다. 존재하면 1을, 존재하지 않으면 0을 출력한다.

입력	출력
5 41523 5 13795	1 1 0 0 1

선택 정렬

목록 안에 저장된 요소들을 특정한 순서대로 재배치하는 알고리즘

선택 정렬(Selection Sort)

선택 정렬

```
def selection_sorted(list) :
    length = len(list)
    for x in range(length - 1) :
        min = x
        for y in range(x + 1, length) :
            if list[min] > list[y] :
                min = y
        list[x], list[min] = list[min], list[x]
    return list

print(selection_sorted([5, 4, 2, 6, 3, 1]))
```

삽입 정렬

목록 안에 저장된 요소들을 특정한 순서대로 재배치하는 알고리즘

삽입 정렬(Insertion Sort)

삽입 정렬

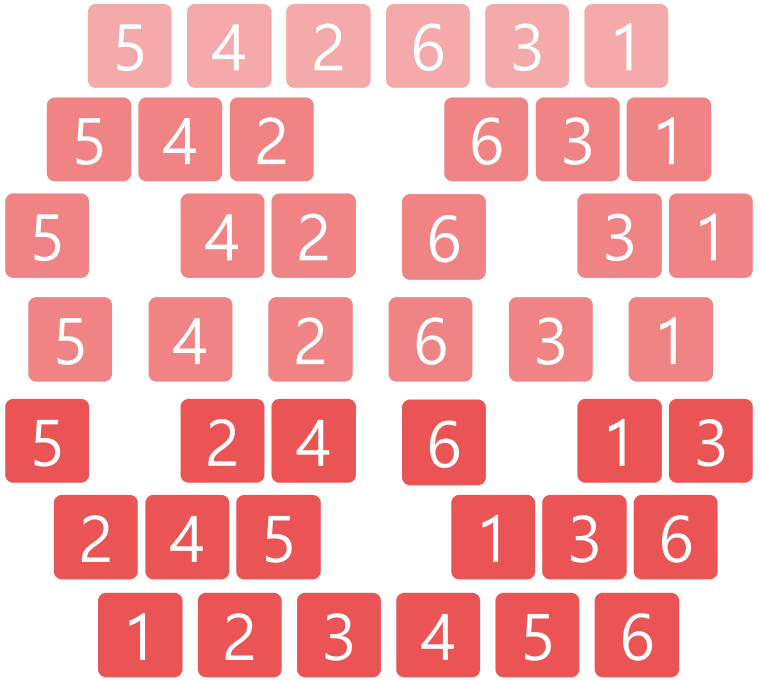
```
def insertion_sorted(list) :
    length = len(list)
    for x in range(1, length) :
        key = list[x]
        y = x - 1
        while y >= 0 and list[y] > key :
            list[y + 1] = list[y]
            y -= 1
        list[y + 1] = key
    return list

print(insertion_sorted([5, 4, 2, 6, 3, 1]))
```

병합 정렬

목록 안에 저장된 요소들을 특정한 순서대로 재배치하는 알고리즘

병합 정렬(Merge Sort)



병합 정렬

```
def merge_sorted(list) :
    length = len(list)
    if length <= 1 :
        return
    mid = length // 2
    group1, group2 = list[:mid], list[mid:]
    merge_sorted(group1)
    merge_sorted(group2)
```

```
    idx, idx1, idx2 = 0, 0, 0
    while idx1 < len(group1) and idx2 < len(group2):
        if group1[idx1] < group2[idx2]:
            list[idx] = group1[idx1]
            idx1 += 1
            idx += 1
        else:
            list[idx] = group2[idx2]
            idx2 += 1
            idx += 1
    while idx1 < len(group1):
        list[idx] = group1[idx1]
        idx1 += 1
        idx += 1
    while idx2 < len(group2):
        list[idx] = group2[idx2]
        idx2 += 1
        idx += 1
    return list
```