Sort and Search

Sort

- 정렬(sort) :입력된 데이터를 특정한 기준에 따라 오름차순(ascending order)이나 내림차순(descending order)으로 정렬하는 알고리즘

- 선택 정렬(Selection Sort)
  - 가장 간단한 정렬 알고리즘 중 하나
  - 주어진 배열에서 최소값을 찾아서 맨 앞의 요소와 교체하는 과정을 반복
- 삽입 정렬(Insertion Sort)
  - 배열의 두 번째 요소부터 시작하여 그 앞의 요소들과 비교하여 삽입할 위치를 지정한 후, 그 위치에 삽입하는 정렬 알고리즘
  - 선택 정렬보다 빠른 실행 시간을 가지지만, 데이터의 크기가 클 경우 효율성이 떨어짐
- 퀵 정렬(Quick Sort)
  - 분할 정복(divide and conquer) 알고리즘의 하나
  - pivot(기준값)을 기준으로 큰 값과 작은 값으로 분할한 후, 각각에 대해서 재귀적으로 정렬을 수행하는 방식
  - 대부분의 경우 가장 빠른 실행 시간을 가지는 알고리즘 중 하나

Sort

- 병합 정렬(Merge Sort)
  - 분할 정복 알고리즘의 하나
  - 주어진 배열을 두 부분으로 나누어 각각을 정렬한 후, 다시 병합하는 방식
  - 퀵 정렬과 마찬가지로 대부분의 경우 가장 빠른 실행 시간을 가지는 알고리즘 중 하나
- 팀 정렬(TimSort)
  - Timsort는 Python의 정렬 알고리즘 중 하나
  - 안정적이고 원소 수가 적을 때는 삽입 정렬, 일부분은 삽입정렬, 일부분은 병합정렬을 사용
  - 이미 정렬된 배열이나 거의 정렬된 배열을 빠르게 처리
  - 작은 부분 배열에 대해 삽입 정렬을 적용하고, 이 부분 배열을 병합

Sort

- 버블 정렬(Bubble Sort)
  - 인접한 두 원소를 비교하여 정렬하는 알고리즘 중 하나
  - 처음부터 끝까지 인접한 두 원소를 비교하며, 큰 값이 오른쪽으로 이동하고 작은 값이 왼쪽으로 이동
  - 이렇게 맨 뒤까지 간 후, 다시 처음부터 끝까지 비교하여 정렬 작업을 반복
  - 대부분의 경우에는 다른 알고리즘보다 비효율적이며, 큰 데이터를 정렬할 때에는 성능이 매우 떨어지는 알고리즘
  - 구현이 쉬우므로 간단한 정렬 문제나 알고리즘 학습을 위한 예제로 많이 사용

Example(Bubble Sort)

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

arr = [5, 3, 8, 4, 9, 1, 6, 2, 7]
print("Before sorting:", arr)
arr = bubble_sort(arr)
print("After sorting:", arr)
```

```
Before sorting: [5, 3, 8, 4, 9, 1, 6, 2, 7]
After sorting: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Example(Bubble Sort)

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n-i-1):
            print("i, j : ", i, j)
            print(arr)
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                print(arr)
    return arr

arr = [4, 3, 2, 1]
print("Before sorting:", arr)
arr = bubble_sort(arr)
print("After sorting:", arr)
```

```
Before sorting: [4, 3, 2, 1]
i, j :  0 0
[4, 3, 2, 1]
[3, 4, 2, 1]
i, j :  0 1
[3, 4, 2, 1]
[3, 2, 4, 1]
i, j :  0 2
[3, 2, 4, 1]
[3, 2, 1, 4]
i, j :  1 0
[3, 2, 1, 4]
[2, 3, 1, 4]
i, j :  1 1
[2, 3, 1, 4]
[2, 1, 3, 4]
i, j :  2 0
[2, 1, 3, 4]
[1, 2, 3, 4]
After sorting: [1, 2, 3, 4]
```

Sort 내장 함수

- 파이썬의 리스트를 정렬하기 위해서는 built-in 함수인 sort() 혹은 sorted() 사용 가능

- sort() : 리스트 내부에서 정렬을 하며, 반환값은 None
- sorted() : 리스트를 정렬한 결과를 반환하며, 원본 리스트는 변경되지 않음

Example(sort(), sorted())

```
# sort() 함수 사용 예시
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort()
print(numbers)  # [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

# sorted() 함수 사용 예시
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # [1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
print(numbers)  # [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
```

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
```

Example(sort(), sorted())

```
# sort() 함수 사용 예시 (내림차순)
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
numbers.sort(reverse=True)
print(numbers)  # [9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]

# sorted() 함수 사용 예시 (내림차순)
numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
sorted_numbers = sorted(numbers, reverse=True)
print(sorted_numbers)  # [9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
print(numbers)
```

```
[9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
[9, 6, 5, 5, 5, 4, 3, 3, 2, 1, 1]
[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]
```

Search

- 검색(search) 알고리즘 : 데이터에서 원하는 데이터를 검색하는 방법

- 순차 검색(sequential search)
  - 배열이나 리스트를 앞에서부터 순서대로 하나씩 비교하면서 원하는 값을 찾는 알고리즘
  - 정렬되지 않은 배열에서도 사용할 수 있지만, 배열의 크기가 큰 경우 비효율적
- 이진 검색(Binary Search)
  - 정렬된 데이터로 검색
  - 이진 검색은 배열의 중간 값을 검사하여 찾는 값이 중간 값보다 큰지 작은지를 판단
  - 찾는 값이 중간 값보다 크면, 배열의 오른쪽 절반에서 다시 검색

Example(sequential search)

```
def sequential_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

arr = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
result = sequential_search(arr, 12)
print(result) # 찾는 숫자의 index 출력
result = sequential_search(arr, 13)
print(result) # 찾는 숫자가 없는 경우 -1 출력
```

```
5
-1
```

Example(binary search)

```
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

arr = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
result = binary_search(arr, 12)
print(result) # 찾는 숫자의 인덱스 출력
result = binary_search(arr, 13)
print(result)
```

```
5
-1
```

Example(index())

```
my_list = [1, 3, 5, 7, 9]

# 값이 5인 요소의 인덱스를 반환
index = my_list.index(5)
print(index) # 2

# 값이 3인 요소가 있는지 검사
if 3 in my_list:
    print("3이 my_list에 있습니다.")

# 값이 10인 요소가 있는지 검사
if 10 not in my_list:
    print("10은 my_list에 없습니다.")
```

```
2
3이 my_list에 있습니다.
10은 my_list에 없습니다.
```