# 투 포인터

## 개념

- 배열이나 **리스트**에서 두 개의 포인터를 이용해 효율적으로 문제를 해결하는 기법.
- 일반적으로 **정렬된 배열**에서 사용되며, 특정 조건을 만족하는 값을 찾거나 범위를 계산할 때 유용.

## 시간 복잡도

- 매 루프마다 항상 두 포인터 중 하나는 1씩 증가함
- 각 포인터를 start, end라고 했을 때 start와 end는 최대 N까지 증가할 수 있고, 항상 start <= end
- start, end가 증가하는 과정은 최대 N번만 반복된다.
- O(N^2) 가 걸리는 문제를 **O(N)**에 해결할 수 있음

## Tip

- 두 개의 포인터는 위치가 정해져있지 않음.
  - 배열의 처음과 끝에 각각 두고 시작할 수도 있고, 모두 처음에 두고 시작할 수도 있음.
- 내가 구하고자 하는 값, 최종 조건이 무엇인지 까먹지 말자. (구간 합, 조건 만족 요소, 교집합 등)

## 사용 예시

1. **정렬된 배열에서 특정 합을 찾기**
2. **두 배열의 교집합 구하기**
3. **중복 제거 및 특정 조건 만족 요소 탐색**

### 1. 특정 합 찾기

```
def two_sum(nums, target):
    nums.sort()  # 배열 정렬
    left, right = 0, len(nums) - 1
    pairs = []

    while left < right:
        curr_sum = nums[left] + nums[right]
```

```
        if curr_sum == target:
            pairs.append((nums[left], nums[right]))  # 쌍 저장
            left += 1
            right -= 1
        elif curr_sum < target:
            left += 1  # 합이 작으면 왼쪽 포인터 이동
        else:
            right -= 1  # 합이 크면 오른쪽 포인터 이동

    return pairs if pairs else None  # 결과 반환

# 사용 예시
arr = [1, 3, 2, 5, 7, 2, 6]
target = 8
print(two_sum(arr, target))  # 출력: [(1, 7), (2, 6)]
```

## 2. 두 배열의 교집합

```
def intersection(arr1, arr2):
    arr1.sort()
    arr2.sort()
    p1, p2 = 0, 0
    result = []

    while p1 < len(arr1) and p2 < len(arr2):
        if arr1[p1] == arr2[p2]:
            result.append(arr1[p1])
            p1 += 1
            p2 += 1
        elif arr1[p1] < arr2[p2]:
            p1 += 1
        else:
            p2 += 1
    return result

# 사용 예시
arr1 = [1, 2, 2, 3]
arr2 = [2, 2, 4]
print(intersection(arr1, arr2))  # 출력: [2, 2]
```

## 대표문제

- 수들의 합 5 https://www.acmicpc.net/problem/2018

```
n = int(input())
cnt, sum = 0, 0
start, end = 0, 0

while end <= n:
    if sum < n:
        end += 1
        sum += end
    elif sum > n:
        sum -= start
        start += 1
    else:
        cnt += 1
        end += 1
        sum += end
print(cnt)
```

- 주몽 https://www.acmicpc.net/problem/1940