**1. Two pointers를 사용했을 때 더 효율적으로 풀 수 있는 문제**

**문제** : 정렬된 정수 배열과 목표 합계 x가 주어졌을 때, 합이 x가 되는 두 수가

배열에 존재하는지 확인하라.

**기존 접근법과 한계**

일반적인 방법은 모든 가능한 두 수의 조합을 탐색하여 합이 x와 같은지 확인하는 것이다.

이 방법의 시간 복잡도는 O(n²)로, 배열의 크기가 커질수록 비효율적이다.

**Two pointers 알고리즘 적용**

1. **초기화** : 배열이 정렬되어 있으므로, 하나의 포인터 left는 배열의 시작 인덱스(0)를,

다른 포인터 `right`는 배열의 끝 인덱스(n-1)를 가리킨다.

2. **반복 과정** :

- left와 right 포인터가 가리키는 두 수의 합 cur\_sum을 계산한다.

- cur\_sum이 x와 같다면, 두 수를 찾은 것이므로 **true**를 반환한다.

- cur\_sum이 x와 작으면, left를 1 증가시켜 합계를 키운다.

- cur\_sum이 x와 크면, right를 1 감소시켜 합계를 줄인다.

3. **종료 조건** : left가 right보다 크거나 같아지면 반복을 종료하고,

합이 x가 되는 두 수가 없으므로 **false**를 반환한다.

**알고리즘의 효율성 분석**

- **시간 복잡도** : O(n)

한 번의 순회로 문제를 해결한다.

- **공간 복잡도** : O(1)

추가적인 공간을 사용하지 않는다.

**2. Sliding window를 사용했을 때 더 효율적으로 풀 수 있는 문제**

**문제** : 문자열이 주어졌을 때, 중복 문자가 없는 가장 긴 부분 문자열의 길이를 구하라.

**기존 접근법과 한계**

모든 가능한 부분 문자열을 생성하고 각 부분 문자열에서 중복 문자가 있는지 검사하면

시간 복잡도가 O(n²)에 달한다. 이는 문자열의 길이가 길어질수록 비효율적이다.

**Sliding window 알고리즘 적용**

1. **초기화** :

- 두 개의 포인터 start와 end를 0으로 설정하여 현재 윈도우의 시작과 끝을 나타낸다.

- 현재 윈도우 내의 문자를 저장할 자료구조(ex: Hash Set)를 초기화한다.

2. **반복 과정** :

<end포인터를 이동시키며 문자열의 문자를 하나씩 확인한다.>

- 새로운 문자가 Hash Set 에 없으면:

- Hash Set 에 문자를 추가한다.

- 윈도우의 길이를 갱신하고 최대 길이를 업데이트한다.

- end를 1 증가시킨다.

- 새로운 문자가 Hash Set 에 이미 있으면:

- Hash Set 에서 start 포인터가 가리키는 문자를 제거한다.

- start를 1 증가시킨다.

3. 종료 조건 : end 포인터가 문자열의 끝에 도달하면 반복을 종료한다.

**알고리즘의 효율성 분석**

- **시간 복잡도** : O(n)

각 문자를 최대 두 번씩만 처리한다.

- **공간 복잡도** : O(min(n, m))

n은 문자열의 길이, m은 가능한 문자 집합의 크기이다.