**고급소프트웨어실습1**

**20191574 김예진**

**1. Sliding Window 실습 3의 효율적인 풀이를 위해서는 어떤 자료구조를 사용해야 되는지를 이유와 함께 제시하시오.**

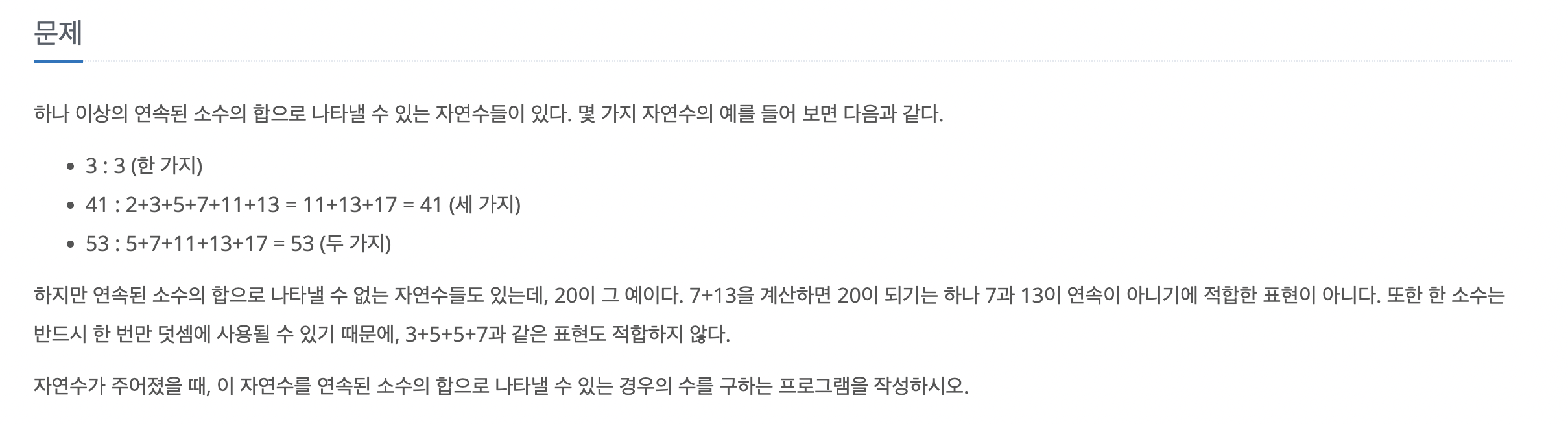
Queue를 사용하면 해당 문제를 보다 효율적으로 해결할 수 있다.

해당 문제는 window를 한칸 씩 이동시키면서 window 내 원소의 평균을 구한다. 결국 가장 오래 전에 window에 들어온 원소를 하나씩 빼고 새로운 원소가 하나씩 window로 추가되는 FIFO 형태의 구조로 알고리즘이 진행된다.

따라서 Queue를 사용하면 문제의 구조에 맞게 효율적으로 알고리즘을 구현할 수 있다.

**2. 앞에서 다루지 않은 문제 중에 Two pointers와 Sliding window를 사용했을 때 더 효율적으로 풀 수 있는 문제를 각각 하나씩 들고, 어떤 면에서 효율적인지 알고리즘을 설명하시오.**

- Two pointers

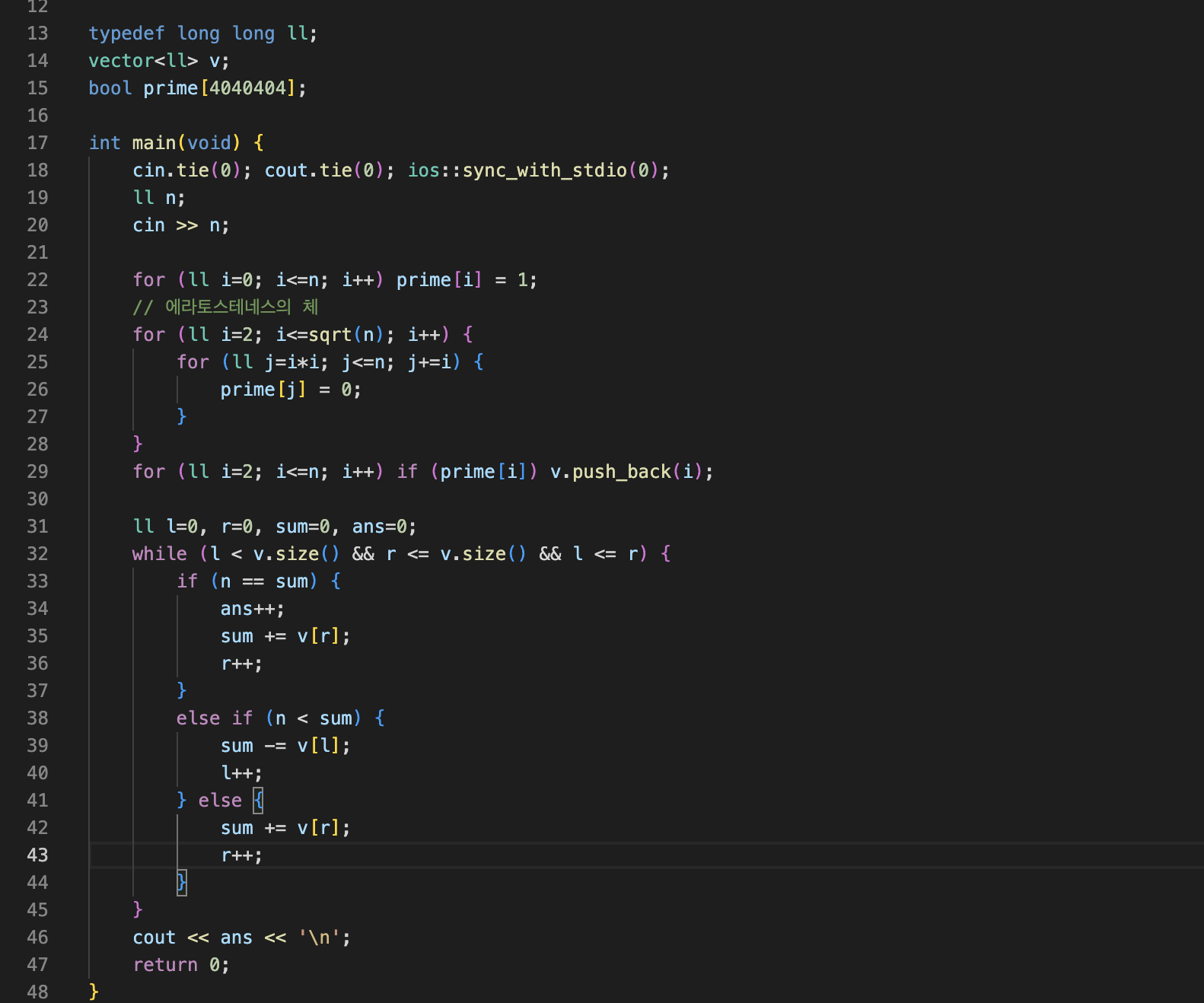


이 문제는 임의의 자연수 N을 연속된 소수들의 합으로 나타낼 수 있는 경우의 수를 구하는 문제이다.

에라토스테네스의 체 알고리즘을 활용하여 vector에 n이하의 소수를 넣어두고, two pointer를 사용하면 효율적으로 문제 해결이 가능하다.

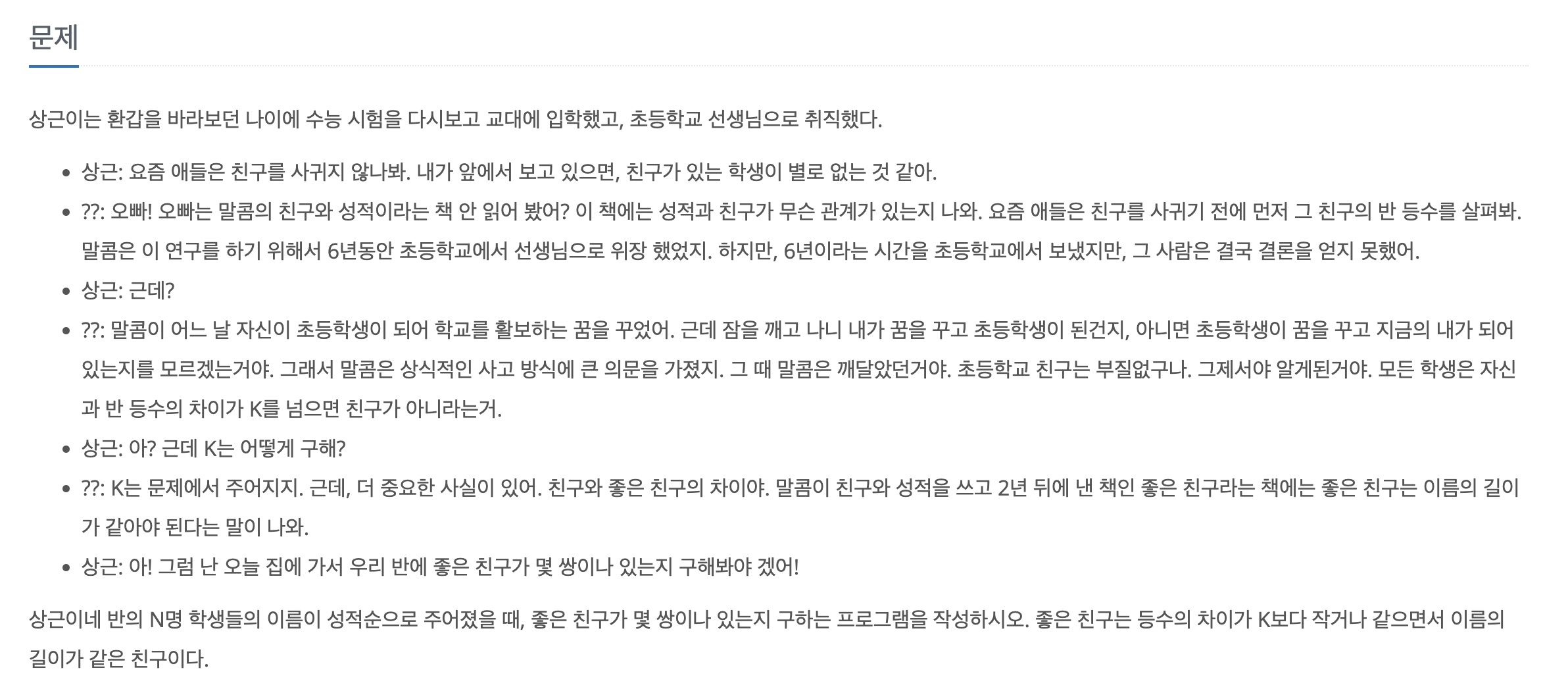
누적 합을 계산해야 하는 문제이기 때문에 pointer는 2개 모두 앞에서부터 시작할 수 있도록 초기화를 한다. 현재 sum이 n보다 크면 left pointer를 증가시켜 sum을 줄여주고, sum이 n보다 작으면 right pointer를 증가시켜 sum을 증가시킨다. sum과 n이 같은 순간이 생기면 counter를 1 증가시켜서 경우의 수롤 counting하고, 모든 경우의 수를 계산해야 하므로 right pointer를 1 증카시켜 다음 경우를 찾는다.

코드는 다음과 같이 작성할 수 있다.



n 이하의 소수가 p개 일 때, two pointer를 사용하지 않는 경우에는 부분 집합의 수가 pC2개 이므로 O(pC2)만큼의 시간 복잡도를 갖게 된다. 위의 코드와 같이 two pointer를 사용하게 되면 l pointer와 r pointer가 p번 움직이는 것이 worst case라고 볼 수 있으므로 시간 복잡도는 O(2p) 정도라고 할 수 있다. 따라서 코드 실행의 시간적인 측면에서 two pointer가 효율적이다.

- Sliding windows

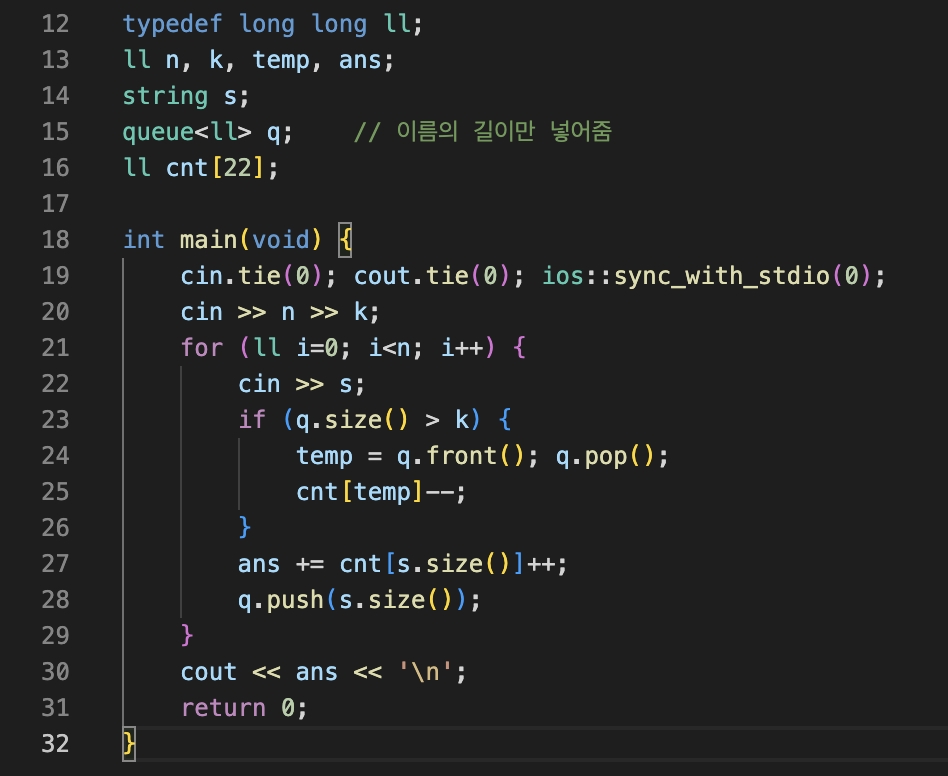


이 문제는 좋은 친구의 쌍을 구하는 문제이다. 좋은 친구는 등수의 차이가 K보다 작거나 같으면서 이름의 길이가 같은 친구를 의미한다.

등수의 차이가 K이하여야 좋은 친구일 수 있으므로 window의 size는 K로 둘 수 있다. 등수의 차이가 K이하이면서 이름의 길이가 같아야 하므로 이름의 길이에 대한 정보만을 queue에 등수 순서대로 넣어두고, window를 움직이면서 해당 window 내에 좋은 친구 관계가 몇 쌍 있는 지를 계산하여 결과 값에 더해준다.

window 내에 좋은 친구 관계의 수는 count array를 사용해서 쉽게 계산할 수 있다. 현재 window 내에 학생들 이름 길이 별 cnt를 저장하는 배열을 만들고, window에서 나가는 학생의 이름 길이는 cnt[x]--, 추가되는 학생의 이름 길이는 cnt[y]++를 해준다. 이 방식으로 별도의 계산 없이 O(1) 만에 window 내의 이름의 길이가 같은 학생의 수를 계산해낼 수 있다.

코드는 다음과 같이 작성할 수 있다.



Sliding window를 사용하지 않았다면 각 2명의 학생에 대해 좋은 관계인지 판단하는 알고리즘이 필요한데 2명의 학생을 선택하는 경우의 수가 nC2이므로 O(nC2)의 시간복잡도를 가진다. 위의 코드와 같이 Sliding window를 사용하면 O(n)만에 모든 좋은 관계를 찾아낼 수 있으므로 시간적인 측면에서 효율적이다.