

분할

티무르와 조수 울룩벡은 *Uzbekistan Got Talent* 쇼에 나가서 보여 줄 마술을 하나 준비하고 있다. 마술의 내용은 다음의 **분할 문제**를 푸는 것이다: 주어진 양의 정수의 집합을 K 개의, 각각이 비어 있지 않은 그룹으로 나누어 각 그룹의 원소 합이 모두 동일하도록 만든다. 자세히 설명하면, 집합의 각 원소는 K 개 중 정확히 하나의 그룹에 배치되어야 한다. 그리고, 각 그룹의 원소 합이 모두 동일한 값이라야 한다. 예를 들어, 주어진 집합이 $[2, 1, 6, 4, 5]$ 이고 $K = 3$ 인 경우, $[2, 4]$, $[1, 5]$, $[6]$ 의 그룹들로 나눌 수 있다. 이 경우 각 그룹의 원소 합은 모두 정확히 6이다.

마술은 아래와 같이 진행된다.

- 울룩벡과 티무르는 각자 서로 다른 방에 들어가고 서로 대화할 수 없다.
- 심판이 울룩벡에게 N 개의 양의 정수를 가진 배열 A 를 준다. 배열 A 의 원소는 1 이상 M 이하의 정수이다. 심판은 그룹의 개수 K 도 울룩벡에게 준다.
- 울룩벡은 A 에 **새로운 원소로 추가**할 최대 $K - 1$ 개의 정수를 생성한다. 생성한 정수들 중에 같은 값이 존재할 수 있다. 생성된 정수들도 1 이상 M 이하라야 한다.
- 심판은 울룩벡이 생성한 원소들을 원 배열의 뒤에 추가한다. 확장된 새 배열은 **단조증가 (비감소) 순서로 정렬되어** 티무르에게 전달된다. 그룹 개수 K 도 티무르에게 전달된다.
- 티무르는 정렬되어 전달된 확장된 배열에서 분할 문제를 풀어야 한다.

당신이 할 일은 티무르와 울룩벡이 사용할 전략을 구상하고 구현하는 것이다. 주어진 조건 하에서 심판이 어떤 배열 A 를 주더라도 문제를 해결할 수 있는 전략이 존재함을 증명할 수 있다.

Implementation Details

아래 두 함수를 구현해야 한다.

울룩벡을 위해 구현해야 하는 함수는 아래와 같다.

```
std::vector<int> add_numbers(std::vector<int> A, int K, int M)
```

- A : 울룩벡에게 주는 길이 N 인 원본 배열.
- K : 목표인 그룹 개수. 울룩벡은 최대 $K - 1$ 개의 값을 추가할 수 있음에 주의하라.
- M : 원본의 원소와 생성될 값에 적용되는 상한.
- 이 함수는 각 테스트 케이스에 대해서 정확히 한번 호출된다.

이 함수는 울룩벡이 생성해서 추가하려는 원소들을 가진 배열 C 를 리턴해야 한다. C 의 길이를 S 라고 하자.

- S 는 최대 $K - 1$ 이라야 한다.
- C 의 각 원소는 1 이상 M 이하라야 한다.

티무르를 위해 구현해야 하는 함수는 아래와 같다.

```
std::vector<int> find_partition(std::vector<int> B, int K)
```

- B : 원본 배열 A 의 원소들과 울루벡이 추가한 원소들을 모두 포함하는 길이 $N + S$ 인 배열. B 의 원소들은 단조증가 하는 순서로 정렬되어 있다.
- K : 목표인 그룹 개수.
- 이 함수는 각 테스트 케이스에 대해서 정확히 한번 호출된다.

이 함수는 B 를 K 개의 그룹으로 분할한 결과인 배열 P 를 리턴해야 한다.

- P 의 길이는 $N + S$ 라야 한다.
- 각 $j(0 \leq j < N + S)$ 에 대해, $P[j]$ 는 $B[j]$ 가 속하는 그룹을 표시한다.
- 그룹들은 0부터 $K - 1$ 까지 번호가 붙어야 한다, 즉, $0 \leq P[j] < K(0 \leq j < N + S)$ 가 성립해야 한다.
- 각 $i(0 \leq i \leq K - 1)$ 에 대해, 최소 하나의 $j(0 \leq j < N + S)$ 가 $P[j] = i$ 를 만족해야 한다.
- K 개 그룹 각각의 합은 동일해야 한다.

실제 채점 과정에서, 위 두 함수를 호출하는 프로그램은 **정확히 두 번** 실행된다.

- 첫번째 실행에서는:
 - `add_numbers`가 정확히 한 번 호출된다.
 - 리턴된 배열이 채점 시스템에 의해 처리되어 B 가 만들어진다.
- 두번째 실행에서는:
 - `find_partition`이 정확히 한 번 호출된다.

Constraints

- $3 \leq N \leq 100\,000$
- $2 \leq K \leq 100\,000$
- $K \leq N$
- $1 \leq M \leq 10^9$
- $1 \leq A[i] \leq M$ ($0 \leq i < N$).

Subtasks

Subtask	Score	Additional Constraints
1	5	$N = 3$
2	4	$M = 1$
3	7	$M \leq 2$
4	10	$N \leq 10$
5	7	$K = 2$
6	12	$K \leq 3$
7	17	$K \leq 10$
8	20	$K \leq 100$
9	18	추가 제한이 없다.

Example

다음 호출을 보자:

```
add_numbers([8, 2, 9, 6, 1, 5, 5], 3, 9)
```

이 예에서 $A = [8, 2, 9, 6, 1, 5, 5]$ 이다. 이 배열을 $K = 3$ 개의 그룹으로 분할하려고 한다. 울룩벡은 1 이상 $M = 9$ 이하의 값들을 추가할 수 있다.

함수는 $[5, 4]$ 를 리턴한다, 즉, 울룩벡은 $S = 2$ 개의 새로운 원소 5와 4를 생성했다. 이 값들은 1 이상 $M = 9$ 이하인 조건을 만족한다.

배열은 확장되고 정렬되어 티무르에게 아래 호출로 전달된다.

```
find_partition([1, 2, 4, 5, 5, 5, 6, 8, 9], 3)
```

이 배열을 $[1, 5, 9]$, $[2, 5, 8]$, $[4, 5, 6]$ 의 3개 그룹으로 나눌 수 있다. 각 그룹의 합은 모두 15로 동일함을 알 수 있다. 이 분할을 보고하기 위해 배열 $[0, 1, 2, 0, 1, 2, 2, 1, 0]$ 를 리턴할 수 있다.

Sample Grader

Input format:

```
N K M
A[0] A[1] ... A[N-1]
```

Output format:

`add_numbers`가 호출된 이후 샘플 그레이더는

- 정렬된 배열 B 를 계산하고
- 배열 C 와 B 를 아래 형식으로 출력하고, 추가로 빈 줄을 하나 출력한다.

```
S
C[0] C[1] ... C[S-1]
B[0] B[1] ... B[N+S-1]
```

`find_partition`이 호출된 후 그레이더는 다음 형식으로 출력한다.

```
L
P[0] P[1] ... P[L-1]
```

L 은 `find_partition`이 리턴한 배열 P 의 길이이다.