문제 1. 정렬감이 없다

입력 파일:	sort.in
출력 파일:	sort.out
시간 제한:	2 seconds
메모리 제한:	256 megabytes

농장 밖에서의 취직을 생각하던 베시는 최근 온라인 코딩 웹사이트에서 다양한 알고리즘을 배우기 시작했다. 가장 좋아하는 알고리즘은 "버블 정렬"과 "퀵 정렬"이다. 하지만 베시는 이 둘을 쉽게 헷갈려서, 둘을 섞은 이상한 알고리즘을 만들어냈다.

$A[\cdots i]$의 최댓값이 $A[i+1\cdots]$의 최솟값보다 크지 않을 경우에, i번째와 $i+1$번째 원소 사이의 위치를 "분할점" 이라고 한다. 베시는 퀵 정렬이 배열을 재배열해서 분할점이 있도록 만들고, $A[\cdots i]$와 $A[i+1\cdots]$ 사이를 재귀적으로 정렬한다는 것을 기억한다. 배열에서 모든 분할점을 찾는 것이 선형시간에 계산할 수 있지만, 퀵 정렬이 분할점을 만들기 위해서 어떻게 배열을 재배열하는지는 잊어버렸다! 이 일에 버블 정렬을 사용하는 알고리즘계의 최악의 실수라고 할 수 있는 결정을 하고 말았다.

다음은 베시가 배열 A를 정렬하는 방법이다.

일단 처음에 버블 정렬의 한 단계를 진행하는 간단한 함수를 작성했다. (역자 주: length(A)는, 배열 A의 길이를 의미한다.)

```
bubble_sort_pass (A) {
    for i = 0 .. length(A)-2
        if A[i] > A[i+1], A[i]와 A[i+1]의 값을 서로 바꾼다.
}
```

그리고 베시의 퀵(인것 같은) 정렬은, 다음과 같이 작성되어 있다.

```
quickish_sort (A) {
    if length(A) = 1, return
    do { // 주 반복문
        work_counter = work_counter + length(A)
        bubble_sort_pass(A)
    } while(A에 분할점이 존재하지 않을 때 까지)
    A를 모든 분할점을 기준으로 나눈다. 각 조각에 대해 quickish_sort를 호출한다.
}
```

베시는 자신의 코드가 얼마나 빠르게 동작할지 궁금하다. 문제를 단순화 하기 위해서, 그는 주 반복문의 한 단계가 선형 시간 안에 동작하므로, 전역변수 work_counter를 반복문 안에서 증가시켜서, 알고리즘이 동작하는데 걸리는 시간을 계산한다.

주어진 배열에 대해서, quickish_sort를 한 이후에 work_counter의 값을 구하여라.

입력 형식

첫째 줄은 N ($1 \leq N \leq 100,000$)이 주어진다. 다음 N개의 줄은 0이상 10^9이하의 정수 $A[0], \cdots, A[N-1]$이 주어진다. 각 원소가 서로 다르다는 사실이 보장되어 있지 않다.

출력 형식

work_counter의 값을 출력하여라.

제한

sort.in	sort.out
7 20 2 3 4 9 8 7	12

참고 사항

이 예제에서, 우리는 배열 20 2 3 4 9 8 7로 시작한다. 버블정렬의 한 단계 이후에 (work_counter에 7을 더한다.), 우리는 2 | 3 | 4 | 9 8 7 | 20으로 배열이 분리 되었다는 것을 알 수 있다. (|은 분할점을 의미한다.) 이제 문제는 2, 3, 4, 20을 정렬하는 것과 (0만큼의 단위 시간이 든다.), 9 8 7을 정렬하는 것이다. 9 8 7 부분문제에 대해서, 주 반복문을 한 번 실행하면 (work_counter에 3을 더한다.) (8 7 | 9) 가 되고, 마지막으로 8 7에 대해서 주 반복문을 한 번 실행하면 (work_counter에 2를 더한다.) 배열이 정렬되었다는 것을 알 수 있다.

문제 2. 열차 추적

시간 제한: 2 seconds
메모리 제한: 256 megabytes

오전마다 특급열차는 농장을 출발하여 대도시로 향하고, 오후에는 반대방향으로 시골로 되돌아온다. 오늘, 베시는 시간을 내서 오전과 오후에 열차를 보려고 한다.

소 베시는 열차가 0번부터 $N-1$번까지의 번호가 붙은 N개의 ($1 \leq N \leq 10^6$) 칸으로 되어있는 것을 안다. i번째 칸에는 식별번호 c_i ($0 \leq c_i \leq 10^9$)가 붙어있다. 모든 숫자는 오전과 오후 모두에 볼 수 있고, 각 칸마다 베시는 수를 알 수 있는 두 번의 기회가 있다. 이는, 오전에 열차가 지나갈 때, c_0, c_1, $\cdots$, c_{N-1}을 차례로 볼 수 있고, 오후에 다시 c_0, c_1, $\cdots$, c_{N-1}을 차례로 볼 수 있다.

베시는 정수 K ($1 \leq K \leq N$)을 골라서, 연속된 K개 칸의 식별번호의 최솟값들을 모두 알고 싶어 한다. 그는 계산을 할 수 있는 공책이 있지만, 이 공책은 작고 손글씨 (발굽글씨?) 는 크다. 예를 들어, $N+1-K$개의 최솟값을 모두 쓸 수 있는 공간이 없을 수도 있다. 비밀스러운 이유로 인해, 베시는 수를 계산 한 후에 하늘에 계산한 결과를 말하기 때문에, 문제는 아니다.

열차가 곧 도착한다! 베시가 열차가 두 번 지나는 동안 제한된 공책을 사용하여 $N+1-K$개의 최솟값을 찾는것을 도와주어라. 공책은 5500개의 구역으로 나뉘어 있고, 0번부터 5499번 까지의 번호가 붙어있으며, 각 구역은 -2^{31}이상 $2^{31}-1$이하의 정수를 담을 수 있다. 처음에 각 구역은 정수 0이 적혀있다.

이 문제는 interactive 문제이다. 하지만 표준 입출력 혹은 파일 입출력을 사용하지 않는 문제이다. 당신은 베시가 제한된 공책 공간을 활용하기 위한 다음 함수를 구현해야 한다.

```
void helpBessie(int ID);
```

오전과 오후에 기차가 지나갈 때, 함수가 호출 될 것이고 인자는 각 칸에 적힌 식별번호이다.

`helpBessie`함수는 다음 함수를 호출할 수 있다.

- `int get(int index)`: 베시의 공책에 해당하는 `index`번 칸에 적힌 숫자를 가져온다.
- `int set(int index, int value)`: 베시의 공책에 해당하는 `index`번 칸에 `value`를 적는다.
- `void shoutMinimum(int output)`: 베시에게 하늘에 결과 `output`을 말하라고 한다.
- `int getTrainLength()`: 칸의 갯수 N을 반환한다.
- `int getWindowLength()`: 연속으로 봐야 할 칸의 갯수 K을 반환한다.
- `int getCurrentCarIndex()`: 현재 보고 있는 열차가 몇번 칸인지를 반환한다.
- `int getCurrentPassIndex()`: 보고 있는 열차가 오전이면 0, 오후면 1을 반환한다.

코드를 작성하는 것을 돕기 위해서, C/C++과 Java로 작성된 기본 템플릿을 제공했다. Python과 Pascal 제출은 이 문제에 대해서 허용되지 않는다.

연속된 K개의 최솟값은 순서대로 출력되어야 한다. (즉 0, 1, $\cdots$, $K-1$번 칸의 식별번호의 최솟값 다음으로 1, 2, $\cdots$, K번 칸의 식별번호의 최솟값이 출력되어야 하고, ... 순서로 출력되어야 한다.)

하지만 이 제한과 별개로 함수가 호출의 어떤 때에 결과를 출력해도 상관 없다. 예를 들어, 함수는 몇 호출에는 출력이 없을 수도, 몇 호출에는 여러개의 출력을 할 수도 있다.

베시는 매우 훌륭한 초단기기억력을 가지고 있어서, `helpBessie`함수 안의 메모리 사용은 256MB 메모리 제한 이외에는 없다. 하지만, 열차의 칸 사이에는 공책에 적히지 않은 그 어떠한 것도 기억할 수 없다. 그래서, 함수 호출 사이에는 `set`과 `get`을 제외하고는 어떠한 상태도 남기면 안된다.

이는:

상수가 아닌 어떠한 전역 혹은 스태틱 변수도 선언해서는 안된다. 이럴 경우 이 문제에 대한 제출은 무효처리 될 것이다. 코치는 이 문제의 정신을 잘 따르는지 손으로 각 제출을 확인 할 것이다. 입출력은 필요로하지 않으므로, 이 문제에서 입출력을 하는것도 허용되지 않는다.

`set`함수와 `get`함수의 호출 제한은 각 테스트 케이스 마다 $25 \cdot 10^6$으로 제한된다.

예제

test	answer
10 3 5 7 9 2 0 1 7 4 3 6	5 2 0 0 0 1 3 3

문제 3. 붕괴

입력 파일: disrupt.in
출력 파일: disrupt.out
시간 제한: 2 seconds
메모리 제한: 256 megabytes

농부 존은 잘 연결된 농장을 운영하는데에 자부심을 느끼고 있다. 농장은 N개의 ($2 \leq N \leq 50,000$) 목장으로 되어있고, $N-1$개의 도로가 목장 쌍을 연결하고 있고, 각 도로의 길이는 단위길이로 모두 같다. 농부 존은 적당한 도로들을 사용하면 어떤 목장에서도 다른 모든 목장으로 오갈수 있다는 것을 깨달았다.

농부 존의 농장이 연결되어 있어도, 어떠한 도로 하나가 막히면, 농장이 두 집합으로 분리되어, 하나의 집합 안에서는 서로 오갈 수 있지만, 두 집합 안에서는 서로 오갈 수 없을수도 있음을 걱정하고 있다. 그래서 농부 존은 M개의 양방향 예비도로 ($1 \leq M \leq 50,000$)을 새로 추가했고, 각 도로의 길이가 최대 10^9 단위길이이다. 소들은 원래 도로가 막히지 않았다면, 원래 도로로 다닌다.

만약 원래 도로중 하나가 막힌 경우, 농장은 두 부분으로 분리되기 때문에, 농부 존은 예비도로 중 하나를 골라서 두 부분 사이를 다시 오갈 수 있게 만든다.

각 원래 도로에 대해서, 농부 존이 가장 짧은 예비도로를 선택하는데 도움을 주자.

입력 형식

첫째 줄에는 N과 M이 주어진다. 다음 $N-1$개의 줄은 원래 도로를 의미하는 $p \neq q$를 만족하는 두 정수 p, q가 주어지고, p번 농장과 q번 농장을 의미한다는 것이다. (농장은 1 이상 N이하의 번호로 표시된다.) 다음 M개의 줄은 예비 도로를 세 정수 p, q, r로 표현하고, r은 예비 도로의 길이를 의미한다. 어떤 쌍의 농장에 대해서도, 최대 한 개의 도로가 연결되어 있다.

출력 형식

각 $N-1$개의 원래 도로에 대해서, 입력에서 주어진 순서대로 도로를 막았을 경우 대체도로로 가능한 길이의 최솟값을 구하여라. 적당한 대체도로가 존재하지 않는다면, -1을 출력하여라.

예제

disrupt.in	disrupt.out
6 3 1 2 1 3 4 1 4 5 6 5 2 3 7 3 6 8 6 4 5	7 7 8 5 5

참고 사항

첫째 비료 덩어리는 위치 1에서 위치 12까지 운반되어야 한다. 고무총을 쓰지 않으면, 11 단위시간이 걸릴것이다. 하지만, 첫번째 고무총을 사용하면 위치 0까지 트랙터로 운반하는 데에 1 단위시간, 하늘을 날아 위치 10으로 비료 덩어리를 옮기는 데에 1 단위시간, 그리고 위치 12로 트랙터로 운반하는 데에 2 단위시간이 든다. 둘째 비료 덩어리는 고무총을 사용하지 않을 때가 제일 빠르게, 셋째 비료 덩어리는 두번째 고무총을 사용했을 때가 제일 빠르게 운반할 수 있다.