나무 심기

나무 심기 분류

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	6767	1181	706	15.915%

문제

1번부터 N번까지 번호가 매겨져 있는 N개의 나무가 있다. i번 나무는 좌표 X[i]에 심어질 것이다.

동호는 나무를 1번 나무부터 차례대로 좌표 X[i]에 심으려고 한다. 1번 나무를 심는 비용은 없고, 각각의 나무를 심는데 드는 비용은 현재 심어져있는 모든 나무 까지 거리의 합이다. 만약 3번 나무를 심는다면, 1번 나무와의 거리 + 2번 나무와의 거리가 3번 나무를 심는데 드는 비용이다.

2번 나무부터 N번 나무까지를 심는 비용의 곱을 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 나무의 개수 N (2 ≤ N ≤ 200,000)이 주어진다. 둘째 줄부터 N개의 줄에 1번 나무의 좌표부터 차례대로 주어진다. 각각의 좌표는 200,000보다 작은 자연수 또는 0이다.

출력

문제의 정답을 1,000,000,007로 나눈 나머지를 출력한다.

예제 입력 1 복사

```
5
3
4
5
6
7
```

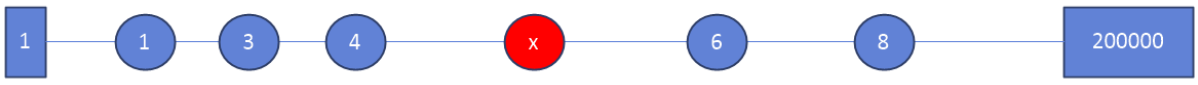
예제 출력 1 복사

```
180
```

접근 방식

1. 각 좌표에 나무를 배치 할 때 마다 이미 심어진 좌표와의 계산을 다 더하여 곱

=> n^2 의 시간이 소모되고 N이 최대 200,000 이기 때문에 시간 초과



2. 현재 심어야 하는 나무의 좌표를 x 라고 하였을 때,

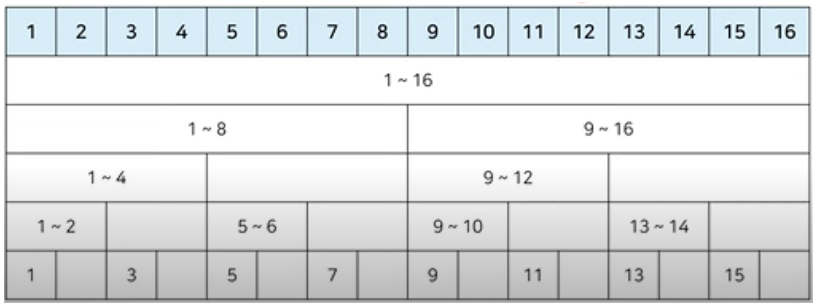
case 1. x 보다 작은 위치에 있는 나무들과의 계산

```
( x - 1 ) + ( x - 3 ) + ( x - 4 ) = 3 * x - ( 1 + 3 + 4 )

                    (작은 나무의 개수의 "합") * (현재 좌표) - (작은 나무의 좌표값의 "합")


case 2. x 보다 큰 위치에 있는 나무들과의 계산

( 6 - x ) + ( 8 - x ) =  ( 6 + 8 ) - 2 * x

                    (큰 나무의 좌표값의 "합" - (큰 나무의 개수의 "합" ) * ( 현재 좌표 )


구해야 할 값

=> 1. 현재 좌표보다 큰 나무의 개수
   2. 현재 좌표보다 작은 나무의 개수

   3. 현재 좌표보다 작은 나무의 좌표들의 "합"
   4. 현재 좌표보다 큰 나무의 좌표들의 "합"
```

구간 합(펜윅 트리)

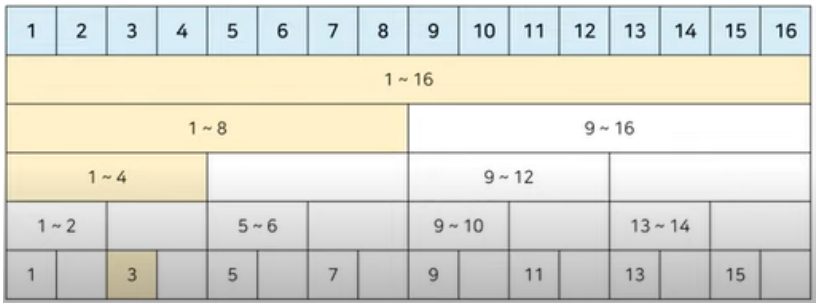
```
2진수 표기 방법

정수 7     0000111

보수를 취한뒤 + 1 을 해준다.

    -7     1111001

임의의 정수 K의 0이 아닌 마지막 비트를 찾기 위해서는

K & -K 비트 연산을 하게 되면 0이 아닌 마지막 비트를 찾을 수 있게 된다.
```

정수 K	2진수 표기	K & -K
0	00000000 00000000 00000000 00000000	0
1	00000000 00000000 00000000 00000001	1
2	00000000 00000000 00000000 00000010	2
3	00000000 00000000 00000000 00000011	1
4	00000000 00000000 00000000 00000100	4
5	00000000 00000000 00000000 00000101	1
6	00000000 00000000 00000000 00000110	2
7	00000000 00000000 00000000 00000111	1
8	00000000 00000000 00000000 00001000	8



해당 인덱스의 값의 `0`이 아닌 마지막 비트는

내가 저장하고 있는 값들의 개수가 됨



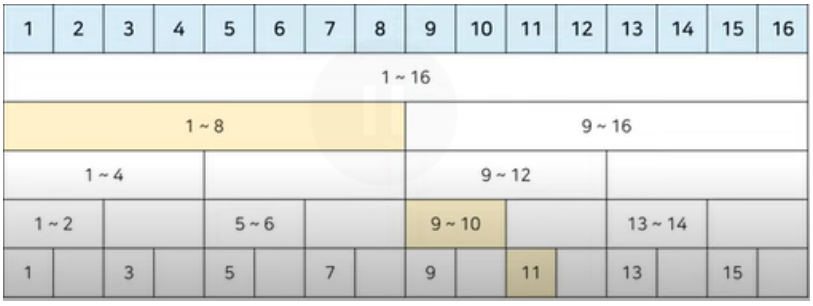
`Update` 에서는 바꾸고자 하는 인덱스 값의 `0`이 아닌 비트만큼 더해주면서

그 구간들의 값을 변경함.

`3` 의 마지막비트 `1 = 1-> 1` 이동

`4` 의 마지막 비트 `1 = 4 -> 4` 이동

`8` 의 마지막 비트 `1 = 8 -> 8` 이동



`Sum` 에서는 끝값의 `0`이 아닌 비트만큼 빼주면서 구간들의 합을 구함

11의 마지막 비트 1 = 1 -> 1 빠짐

10의 마지막 비트 1 = 2 -> 2 빠짐

8의 마지막 비트 1 = 8 -> 8 빠짐

풀이

첫 나무는 비용이 없이 설치된다고 하였으므로,

별다른 과정 없이 cnt, sum 배열에 업데이트를 진행한다.

두 번째 나무 부터는 지금 설치하는 것보다 작은 부분과 큰 부분을 나누어 계산 후, 곱한다.

```
  // (작은 나무의 개수) * (현재 좌표) - (작은 나무의 좌표값의 "합")
  long left = ((sum(tree - 1, cnt) * tree) - sum(tree - 1, sum)) % MOD;

  // (큰 나무의 좌표값의 "합" - (큰 나무의 개수) * ( 현재 좌표 )
  long right = (sum(MAX - 1, sum) - sum(tree, sum)
                  - ((sum(MAX - 1, cnt) - sum(tree, cnt)) * tree)) % MOD;


answer =  (answer * (left + right)) % MOD;
```

코드

```
package Algo_Study_BOJ;

import java.io.*;

public class Main_1280_나무심기 {
  static int N;
  static long[] sum, cnt;
  static final int MAX = 200002;
  static final int MOD = 1000000007;

  static void Update(int idx, int val, long[] arr) {

    for(int i = idx; i <= MAX; i += (i & -i)) {
      arr[i] += val;
    }
  }

  static long sum(int idx, long[] arr) {

    long result = 0;

    for(int i = idx; i > 0; i -= (i & - i)) {
      result += arr[i];
    }

    return result;
  }
  public static void main(String[] args) throws Exception {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
    N = Integer.parseInt(br.readLine());

    sum = new long[MAX + 1];
    cnt = new long[MAX + 1];

    int tree;
    long answer = 1;

    for(int i = 0; i < N; i++) {
      tree = Integer.parseInt(br.readLine());
      tree++;

      if(i != 0) {
        // (작은 나무의 개수) * (현재 좌표) - (작은 나무의 좌표값의 "합")
        long left = ((sum(tree - 1, cnt) * tree) - sum(tree - 1, sum)) % MOD;

        // (큰 나무의 좌표값의 "합" - (큰 나무의 개수) * ( 현재 좌표 )
        long right = (sum(MAX - 1, sum) - sum(tree, sum) - ((sum(MAX - 1, cnt) - sum(tree, cnt)) * tree)) % MOD;

        answer =  (answer * (left + right)) % MOD;
      }

      Update(tree, 1, cnt);
      Update(tree, tree, sum);
    }
    System.out.println(answer);
  }
}
```