1247：河中跳房子

题目描述

每年奶牛们都要举办各种特殊版本的跳房子比赛，包括在河里从一个岩石跳到另一个岩石。这项激动人心的活动在一条长长的笔直河道中进行，在起点和离起点L远（1 ≤ L≤ 1,000,000,000）的终点处均有一个岩石。在起点和终点之间，有N（0 ≤ N ≤ 50,000）个岩石，每个岩石与起点的距离分别为Di （0 < Di < L)。

在比赛过程中，奶牛轮流从起点出发，尝试到达终点，每一步只能从一个岩石跳到另一个岩石。当然，实力不济的奶牛是没有办法完成目标的。

农夫约翰为他的奶牛们感到自豪并且年年都观看了这项比赛。但随着时间的推移，看着其他农夫的胆小奶牛们在相距很近的岩石之间缓慢前行，他感到非常厌烦。他计划移走一些岩石，使得从起点到终点的过程中，最短的跳跃距离最长。他可以移走除起点和终点外的至多M （0 ≤ M ≤ N）个岩石。

请帮助约翰确定移走这些岩石后，最长可能的最短跳跃距离是多少？

输入

第一行包含三个整数L，N，M，相邻两个整数之间用单个空格隔开。

接下来N行，每行一个整数，表示每个岩石与起点的距离。岩石按与起点距离从近到远给出，且不会有两个岩石出现在同一个位置。

输出

一个整数，最长可能的最短跳跃距离。

输入样例

```
25 5 2
2
11
14
17
21
```

输出样例

```
4
```

提示

在移除位于2和14的两个岩石之后，最短跳跃距离为4（从17到21或从21到25）。

分析

在做这道题之前，请同学们自行回忆类似的题目。看看能否找到解题思路。对，这道题也是一道非常标准的二分边界搜索应用，很明显，我们搜索的是右边界。

接下来，我们依次来分析它的核心要素：

1、左边界：0。为什么是0？考虑一下极端情况，当只有两块石头存在时，只需跳跃一次就会到达终点，那么最大距离必然是起点和终点的距离，那么最小距离，必然就是0了。

2、右边界：终点石头距离起点的距离

3、搜索逻辑：不断的尝试最长跳跃距离，如果两块石头的距离小于目标距离，则移除一块；否则，则将新的石头作为起点，进行下一次计算。

注意，本题中的**最短的跳跃距离最长**的概念一定是每一步都能达到这个距离，例如这组数据：

```
4 2 1
2
3
```

起始点距离为4，中间共两个石头，需要移走一个。那么正确的结果应该是距离2，而不是3！！因为选择3，会导致最后一步的距离是1，不符合题意。

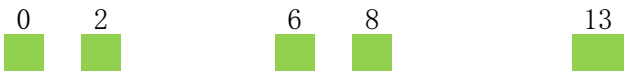
另外一个问题，为什么比较的时候是小于，如果按等于计算，那么原本用于落脚的石头就会被移除，相当于多计算了一块，会导致错误。

最后比较移除的总量，大于目标值则边界右移，小于目标值则边界左移。

当然，等于目标值的时候，也要边界右移。因为我们要求的是尽可能长的跨越路线。

看这个例子：终点距离为13，中间有三块距离分别为2,6,8的石头，现在需要移走两块。最佳方案如下：

初始状态



目标值为6

在上图中，距离为2和8的砖块被移出，因为2与0的间距小于6,8与6的间距也小于6。在上面的方案中，奶牛最短也需要跳6米，最长则为7米(13-6)。

编码

```
#include<bits/stdc++.h>

using namespace std;
long long n, m, a[100010];

int CountDistance(int x) {
    int d = 0, cnt = 0;
    //依次比较每两块石头的间距，别忘了终点
    for (int i = 1; i <= n+1; i++) {
        int gap = a[i] - d;
        //两块石头的间距小于目标值，则必须搬掉一块
        if (gap < x) {
            cnt++;
        }
            //否则将新的石头作为起点
        else {
            d = a[i];
        }
    }
    //搬的石头大于目标数，说明间距太大，应该缩小间距
    return cnt;
}


int Search(int left, int right) {
    while (left <= right) {
        int mid = left + (right - left) / 2;
        int num = CountDistance(mid);
        //数量合适，但是要寻找尽可能长的距离，因此左边界右移
        if (num == m) {
            left = mid + 1;
        }
            //搬的石头大于目标数，说明间距太大，应该缩小间距
        else if (num > m) {
            right = mid - 1;
        }
            //搬的石头小于目标数，说明间距太小，应该增加间距
        else if (num < m) {
            left = mid + 1;
        }
    }
    return right;
}

long long l = 0, r = 0;
long long len;
```

```cpp
int main() {
    cin >> len >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    l = 0;  //左边界需要从0开始
   //别忘了将终点也要存入数组，并将其设为最大值
   r = a[n + 1] = len;
    cout << Search(l, r);
    return 0;
}
```