LCP 24.数字游戏

题目大意

给定一个数组 $nums$，对于这个数组的第 0 到 i 位，你需要对每一位数字进行一些操作，从而使这个前缀数组满足一个特定的“性质”。

你可以做的操作为，把任意一位的数字加一或减一；数组需要满足的“性质”为，$\forall j \in (0,i]$，$nums[j] = nums[j-1] + 1$，即成一个顺子，记 ans_i 为你对前缀数组 $nums[0]$ 到 $nums[i]$ 进行操作，使其能够满足“性质”的最小操作次数，输出 ans 数组。

题目分析

我们假设经过操作，$nums[0]$ 最终变成了 x，于是 $nums[i]$ 最终会变成 $x+i$，才能满足“性质”。我们要求的是 $\sum_{j=0}^{i}|nums[i]-(x+i)|$ 的最小值。为了简便，我们令 $nums\prime[i] := nums[i]-i$，转化为求 $\min_x\{\sum_{j=0}^{i}|nums[i]-(x+i)|\}$的值。

根据绝对值函数的性质，对于有序数列 $\{x_1, x_2, \dots x_{2n}\}$，当且仅当 $x = x_n$或x_{n+1} 时，$\sum_{i=1}^{2n}|x-x_i|$ 取得最小值 $\left(\sum_{i=1}^{n}x_i - \sum_{i=n+1}^{2n}x_i\right)$。对应地，对于有序数列 $\{x_1, x_2, \dots x_{2n+1}\}$，当且仅当 $x = x_{n+1}$ 时，$\sum_{i=1}^{2n+1}|x-x_i|$ 取得最小值 $\left(\sum_{i=1}^{n}x_i - \sum_{i=n+2}^{2n+1}x_i\right)$。

因此，我们需要维护“相对大的那些数”的和与“相对小的那些数”的和的信息，自然地，我们选用一个大顶堆和一个小顶堆来维护这项信息，因为每次加入一个新的数，要么把“相对小的那些数”中最大的那个交换出来，要么把“相对大的那些数”中最小的那个交换出来，并且我们需要时刻保持大顶堆与小顶堆中元素个数时刻相同。每次推入新数字，时间复杂度为 $O(\log n)$，总时间复杂度为 $O(n\log n)$。说了这么多，还是看看具体的代码实现吧：

```
priority_queue<int, vector<int>, greater<int>> big;//min top heap
priority_queue<int, vector<int>, less<int>> small;//max top heap
long long bigsum = 0, smallsum = 0;
int center1, center2;
vector<int> ret;
bool try_push_big(int& num) {
    if (num > big.top()) {
        int temp = big.top();
        bigsum -= temp;
        bigsum += num;
        big.pop();
        big.push(num);
```

```
        num = temp;
        return true;
    }
    return false;
}
bool try_push_small(int& num) {
    if (num < small.top()) {
        int temp = small.top();
        smallsum -= temp;
        smallsum += num;
        small.pop();
        small.push(num);
        num = temp;
        return true;
    }
    return false;
}
vector<int> numsGame(vector<int>& nums) {
    for (int i = 0;i < nums.size();i++) nums[i] -= i;
    if (nums.size() == 1) return { 0 };
    if (nums.size() == 2) return { 0, abs(nums[0] - nums[1]) };
    if (nums[0] < nums[1]) swap(nums[0], nums[1]);
    big.push(nums[0]), small.push(nums[1]);
    bigsum = nums[0], smallsum = nums[1];
    ret.push_back(0), ret.push_back(abs(nums[0] - nums[1]));
    for (int i = 2;i < nums.size();i++) {
        if (i % 2 == 0) {
            center1 = nums[i];
            try_push_big(center1), try_push_small(center1);
        }
        else {
            center2 = nums[i];
            try_push_big(center2), try_push_small(center2);
            if (center1 > center2) swap(center1, center2);
            small.push(center1), big.push(center2);
            smallsum += center1, bigsum += center2;
        }
        ret.push_back((bigsum - smallsum) % 1000000007);
    }
    return ret;
}
```