AtCoder ARC 101 D - Median of Medians (700点)

二分探索 転倒数 BIT データ構造 メディアン k番目を求める AtCoder AtCoder700点 ABC/ARC-D

とは言え、完全に一緒ではなくて後半の議論は少し違うね。

問題へのリンク

問題概要 (ARC 101 D / ABC 107 D)

長さ NN の数列 $a_1, a_2, \ldots, a_N$a1,a2,...,aN が与えられる。この数列の連続する ${}_{N+1}\mathrm{C}_2$ N+1C2 個の区間すべてについてメディアンを求めて並べてできる数列のメディアンを求めよ。

制約

- $1 \leq N \leq 10^5$ 1≤N≤105

考えたこと

一目みて、二分探索、それはそう

というわけで以下の判定問題を解く:

数列の連続する区間のうち、値 xx 以下のものが半分 (奇数長の場合は過半数) 以上を占めるようなものが、区間全体の過半数を占めるか？

上で言及した JOI の問題では、「値 x 以下が k 個以下あるような区間の個数」をしゃくとり法で求めた。今回は、条件が成り立つ区間間に単調性がないので、しゃくとり法が使えない。

なので違う方法を考える。まず数列の各項について、x 以下かどうかだけが重要なので、細かい数値情報は捨て去ってしまって

- x 以上: 1
- x 未満: -1

としてしまってよい。こうすると「値 x 以下が半数以上」という条件は

- 区間の和が 0 以上

と明瞭な言い換えができる。つまり、区間の和が 0 以上となるような区間の個数を数え上げて、それが全区間の過半数を占めるかどうかを判定する問題になる。

区間の和は、累積和 S をとると扱いやすい。そうすると区間 [i, j) の和が 0 以上という条件は

- S[j] - S[i] >= 0　⇔　S[i] <= S[j]

と言い換えられる。これを満たす (i, j) の個数は、転倒数を求めるのとほぼ同じ要領で求めることができる。

```
#include <iostream>
#include <vector>
using namespace std;

template <class Abel> struct BIT {
    const Abel UNITY_SUM = 0;                          // to be set
    vector<Abel> dat;

    /* [1, n] */
    BIT(int n) : dat(n + 1, UNITY_SUM) { }
    void init(int n) { dat.assign(n + 1, UNITY_SUM); }

    /* a is 1-indexed */
    inline void add(int a, Abel x) {
        for (int i = a; i < (int)dat.size(); i += i & -i)

            dat[i] = dat[i] + x;
    }

    /* [1, a], a is 1-indexed */
    inline Abel sum(int a) {
        Abel res = UNITY_SUM;
        for (int i = a; i > 0; i -= i & -i)
            res = res + dat[i];
        return res;
    }
```

```
    }

    /* [a, b), a and b are 1-indexed */
    inline Abel sum(int a, int b) {
        return sum(b - 1) - sum(a - 1);
    }

    /* debug */
    void print() {
        for (int i = 1; i < (int)dat.size(); ++i) cout << sum(i, i + 1) << "
        cout << endl;
    }
};

int main() {
    long long N; cin >> N;
    vector<int> a(N); for (int i = 0; i < N; ++i) cin >> a[i];
    int low = 0, high = 1<<30;
    const int geta = N+1;
    while (high - low > 1) {
        int mid = (low + high) / 2;
        long long num = 0;
        BIT<long long> bit(N*2+10);
        int sum = 0;
        bit.add(sum+geta, 1);
        for (int i = 0; i < N; ++i) {
            int val;
            if (a[i] <= mid) val = 1; else val = -1;
            sum += val;
            num += bit.sum(1, sum+geta);
            bit.add(sum+geta, 1);
        }

        if (num > (N+1)*N/2/2) high = mid;
        else low = mid;
    }
    cout << high << endl;
}
```