树套树（动态区间排名）

题目描述

您需要写一种数据结构，来维护一个有序数列，其中需要提供以下操作：

1. 查询 k 在区间内的排名
2. 查询区间内排名为 k 的值
3. 修改某一位值上的数值
4. 查询 k 在区间内的前驱（前驱定义为严格小于 x，且最大的数，**若不存在输出** `-2147483647`）
5. 查询 k 在区间内的后继（后继定义为严格大于 x，且最小的数，**若不存在输出** `2147483647`）

输入格式

第一行两个数 n, m，表示长度为 n 的有序序列和 m 个操作。

第二行有 n 个数，表示有序序列。

下面有 m 行，opt 表示操作标号。

若 $opt = 1$，则为操作 1，之后有三个数 $l\ r\ k$，表示查询 k 在区间 $[l, r]$ 的排名。

若 $opt = 2$，则为操作 2，之后有三个数 $l\ r\ k$，表示查询区间 $[l, r]$ 内排名为 k 的数。

若 $opt = 3$，则为操作 3，之后有两个数 $pos\ k$，表示将 pos 位置的数修改为 k。

若 $opt = 4$，则为操作 4，之后有三个数 $l\ r\ k$，表示查询区间 $[l, r]$ 内 k 的前驱。

若 $opt = 5$，则为操作 5，之后有三个数 $l\ r\ k$，表示查询区间 $[l, r]$ 内 k 的后继。

> **线段树维护区间，每个线段树区间置一个平衡树，来方便获取排名等信息。**

```cpp
#include<bits/stdc++.h>
using i64 = long long;
#define int i64
const int N = 5e4 + 10, inf = INT_MAX;
// 封装fhq平衡树
namespace FHQ {
    std::mt19937 rng(std::random_device{}());
    struct node {
        int x, rnd, size;
        int lc, rc;
    }t[N << 6];
    int tot = 0;
    class fhq {
    public:
        int root;
        int newNode(int x) {
            t[++ tot] = {x, rng(), 1, 0, 0};
            return tot;
        }
        void pushup(int p) {
            t[p].size = t[t[p].lc].size + t[t[p].rc].size + 1;
        }
        void split(int u, int &x, int &y, int val) {
            if (! u) {
                x = y = 0;
                return ;
```

```
        }
        if (t[u].x <= val) {
            x = u;
            split(t[x].rc, t[x].rc, y, val);
        } else {
            y = u;
            split(t[y].lc, x, t[y].lc, val);
        }
        pushup(u);
    }
    int merge(int x, int y) {
        if (! x || ! y) {
            return x + y;
        }
        if (t[x].rnd < t[y].rnd) {
            t[x].rc = merge(t[x].rc, y);
            pushup(x);
            return x;
        } else {
            t[y].lc = merge(x, t[y].lc);
            pushup(y);
            return y;
        }
    }
    void insert(int x) {
        int l, r;
        split(root, l, r, x);
        root = merge(l, merge(newNode(x), r));
    }
    void del(int x) {
        int l, r, ll, rr;
        split(root, l, r, x);
        split(l, ll, rr, x - 1);
        rr = merge(t[rr].lc, t[rr].rc);
        root = merge(merge(ll, rr), r);
    }
    int kth(int u, int k) {
        int p = t[t[u].lc].size + 1;
        if (k == p) return t[u].x;
        if (k < p) return kth(t[u].lc, k);
        return kth(t[u].rc, k - p);
    }
    int rank(int x) {
        int l, r;
        split(root, l, r, x - 1);
        int temp = t[l].size + 1;
        root = merge(l, r);
        return temp;
    }
    int getkth(int k) {
        return kth(root, k);
    }
    int pre(int x) {
        int l, r;
        split(root, l, r, x - 1);
        int temp = kth(l, t[l].size);
        root = merge(l, r);
        return temp;
    }
    int nxt(int x) {
```

```
            int l, r;
            split(root, l, r, x);
            int temp = kth(r, 1);
            root = merge(l, r);
            return temp;
        }
    };
}

struct node {
    int l, r;
    int max, min;
}t[N << 2];
FHQ::fhq ft[N << 2];
int a[N], n, m;
void pushup(int p) {
    t[p].max = std::max(t[p << 1].max, t[p << 1 | 1].max);
    t[p].min = std::min(t[p << 1].min, t[p << 1 | 1].min);
}
void build(int p, int l, int r) {
    t[p] = {l, r, a[l], a[l]};
    for (int i = l; i <= r; i ++) {
        ft[p].insert(a[i]);
    }
    if (l == r) return ;
    int mid = l + r >> 1;
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    pushup(p);
}
int rank(int p, int l, int r, int x) {
    if (t[p].l >= l && t[p].r <= r) return ft[p].rank(x);
    int mid = t[p].l + t[p].r >> 1, res = 1;
    if (l <= mid) res += rank(p << 1, l, r, x) - 1; // 多少排在前面的
    if (r > mid) res += rank(p << 1 | 1, l, r, x) - 1; // 多少排在前面的
    return res;
}
int kth(int l, int r, int k) {
    int x = 0, y = 1e8 + 10, ans = 0;
    while (x <= y) {
        int mid = x + y >> 1;
        int temp = rank(1, l, r, mid);
        if (temp <= k) {
            ans = mid;
            x = mid + 1;
        } else {
            y = mid - 1;
        }
    }
    return ans;
}
int pre(int p, int l, int r, int x) {
    if (t[p].l >= l && t[p].r <= r) {
        if (t[p].min >= x) return -inf;
        return ft[p].pre(x);
    }
    int mid = t[p].l + t[p].r >> 1, max = - inf;
    if (l <= mid) max = std::max(max, pre(p << 1, l, r, x));
    if (r > mid) max = std::max(max, pre(p << 1 | 1, l, r, x));
    return max;
```

```
}
int nxt(int p, int l, int r, int x) {
    if (t[p].l >= l && t[p].r <= r) {
        if (t[p].max <= x) return inf;
        return ft[p].nxt(x);
    }
    int mid = t[p].l + t[p].r >> 1, min = inf;
    if (l <= mid) min = std::min(min, nxt(p << 1, l, r, x));
    if (r > mid) min = std::min(min, nxt(p << 1 | 1, l, r, x));
    return min;
}
void change(int p, int pos, int x) {
    ft[p].del(a[pos]);
    ft[p].insert(x);
    if (t[p].l == t[p].r) {
        t[p].max = t[p].min = x;
        return ;
    }
    int mid = t[p].l + t[p].r >> 1;
    if (pos <= mid) change(p << 1, pos, x);
    if (pos > mid) change(p << 1 | 1, pos, x);
    pushup(p);
}
void solve() {
    std::cin >> n >> m;
    for (int i = 1; i <= n; i ++) {
        std::cin >> a[i];
    }
    build(1, 1, n);
    for (int i = 0; i < m; i ++) {
        int op, l, r, k;
        std::cin >> op >> l >> r;
        if (op == 3) {
            change(1, l, r);
            a[l] = r;
            continue;
        }
        std::cin >> k;
        if (op == 1) std::cout << rank(1, l, r, k) << '\n';
        if (op == 2) std::cout << kth(l, r, k) << '\n';
        if (op == 4) std::cout << pre(1, l, r, k) << '\n';
        if (op == 5) std::cout << nxt(1, l, r, k) << '\n';

    }
}
signed main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);

    int t = 1;
    //std::cin >> t;
    while (t --) {
        solve();
    }
    return 0;
}
```