**步骤：**

1. **从数组的末尾开始遍历**。这样我们能够确保每次计算时，能够统计出当前数后面比它小的数的出现次数（也就是逆序对）。
2. 对于当前数 arr[i]，我们查询树状数组中，所有比 arr[i] 小的数的出现次数，这些就是比当前数小并且在当前数后面的数，也就是**逆序对**的数量。
3. 然后，更新树状数组，将当前数 arr[i] 加入树状数组中，记录它的出现次数。
4. 最终，累加所有的逆序对数目。

#include <bits/stdc++.h>

using namespace std;

// 树状数组类

class FenwickTree

{

public:

    vector<int> tree;

    int size;

    FenwickTree(int n) : size(n), tree(n + 1, 0) {}

    // 更新树状数组，在位置 index 增加 delta

    void update(int index, int delta)

    {

        while (index <= size)

        {

            tree[index] += delta;

            index += index & -index; // 移动到下一个节点

        }

    }

    // 查询前缀和：查询 [1, index] 的和

    int query(int index)

    {

        int sum = 0;

        while (index > 0)

        {

            sum += tree[index];

            index -= index & -index; // 移动到上一个节点

        }

        return sum;

    }

};

// 使用树状数组计算逆序数

int countInversions(vector<int> &arr)

{

    int n = arr.size();

    // 离散化：将数组的元素映射到 1 到 n 范围内

    vector<int> sortedArr = arr;

    sort(sortedArr.begin(), sortedArr.end());

    for (int &x : arr)

    {

        x = lower\_bound(sortedArr.begin(), sortedArr.end(), x) - sortedArr.begin() + 1;

    }

    FenwickTree fenwickTree(n);

    // 逆序数计算

    int inversionCount = 0;

    for (int i = n - 1; i >= 0; --i)

    {

        // 查询前面有多少个比 arr[i] 小的数

        inversionCount += fenwickTree.query(arr[i] - 1);

        // 更新树状数组，插入 arr[i]

        fenwickTree.update(arr[i], 1);

    }

    return inversionCount;

}

int main()

{

    vector<int> arr = {5, 3, 8, 6, 2};

    cout << "逆序数: " << countInversions(arr) << endl;

    return 0;

}