# 树状数组/归并排序应用：计算数组的小和

## 简介

面试文远知行，被问到了这道题，牛客程序员代码面试指南：计算数组的小和

题目描述如下：

> *数组小和的定义如下：*
>
> *例如，数组* `s = [1, 3, 5, 2, 4, 6]`
>
> *在* `s[0]` *的左边小于或等于* `s[0]` *的数的和为* $0$
>
> *在* `s[1]` *的左边小于或等于* `s[1]` *的数的和为* $1$
>
> *在* `s[2]` *的左边小于或等于* `s[2]` *的数的和为* $1+3=4$
>
> *在* `s[3]` *的左边小于或等于* `s[3]` *的数的和为* $1$
>
> *在* `s[4]` *的左边小于或等于* `s[4]` *的数的和为* $1+3+2=6$
>
> *在* `s[5]` *的左边小于或等于* `s[5]` *的数的和为* $1+3+5+2+4=15$
>
> *所以 s 的小和为* $0+1+4+1+6+15=27$
>
> *给定一个数组 s，实现函数返回 s 的小和*
>
> *要求时间复杂度为* $O(n\log n)$，*空间复杂度为* $O(n)$
>
> ```
> 输入描述
> 第一行有一个整数N。表示数组长度
> 接下来一行N个整数表示数组内的数
> 输出
> 一个整数代表答案
> ```

## 思路

### 树状数组

一个思路是，可以计算每个位置的数字在最终的结果中使用的次数。这个次数是当前位置的数字右边大于等于他的元素的个数。

那么可以逆序遍历数组，通过树状数组统计当前元素右侧小于他的元素个数就可以。

流程

1. 排序+离散化 $n \log n$, 初始化树状数组
2. 逆序遍历。在元素加入树状数组之前，统计树状数组中比当前元素小的数字的个数，进而推理当前元素被计算的次数

```python
class FenwickTree:
    def __init__(self, n):
        self.bit = [0] * (n + 1)
        self.size = n + 1

    def _lowbit(self, n):
        return n & (-n)

    def add(self, num):
        while num < self.size:
            self.bit[num] += 1
            num += self._lowbit(num)

    def query(self, num):
        ret = 0
        num -= 1
        while num > 0:
            ret += self.bit[num]
            num -= self._lowbit(num)
        return ret


def solve(nums):
    sorted_nums = sorted(nums)
    mapping = {}
    cnt = 1
    for n in sorted_nums:
        if n not in mapping:
            mapping[n] = cnt
            cnt += 1
    ret = 0
```

```
    bit = FenwickTree(cnt)
    for idx in range(len(nums) - 1, -1, -1):
        right_cnt = len(nums) - 1 - idx
        cur_num = nums[idx]
        map_ret = mapping[cur_num]
        smaller_cnt = bit.query(map_ret)
        ret_cnt = right_cnt - smaller_cnt
        bit.add(map_ret)
        ret += cur_num * ret_cnt
    return ret


if __name__ == '__main__':
    n = int(input())
    nums = list(map(int, input().split()))
    print(solve(nums))
```

## 归并排序

归并排序将两个数组并为一个数组的过程中，可以直接统计小和

```
def merge_sort(nums, lo, hi):
    if lo == hi:
        return 0
    mid = (lo + hi) // 2
    left = merge_sort(nums, lo, mid)
    right = merge_sort(nums, mid + 1, hi)
    return left + right + merge(nums, lo, mid, hi)


def merge(nums, lo, mid, hi):
    ret = 0
    aux = [0] * (hi - lo + 1)
    p, l, r = 0, lo, mid + 1
    while l <= mid and r <= hi:
        if nums[l] <= nums[r]:
            ret += nums[l] * (hi - r + 1)
            aux[p] = nums[l]
            l += 1
        else:
            aux[p] = nums[r]
            r += 1
        p += 1
```

```
    while l <= mid:
        aux[p] = nums[l]
        p += 1
        l += 1
    while r <= hi:
        aux[p] = nums[r]
        p += 1
        r += 1
    nums[lo: hi + 1] = aux[:]
    return ret


if __name__ == '__main__':
    n = int(input())
    nums = list(map(int, input().split()))
    print(merge_sort(nums, 0, len(nums) - 1))
```