Contents

1 Todo

1. Add code and complexity
2. Add brief explanations

2 Contest Setup

2.1 vimrc

```
"General
set number " Show line numbers
set linebreak " Break lines at word (requires Wrap lines)
set showbreak=+++ " Wrap-broken line prefix
set textwidth=150 " Line wrap (number of cols)
set showmatch " Highlight matching brace
set visualbell " Use visual bell (no beeping)

set hlsearch " Highlight all search results
set smartcase " Enable smart-case search
set ignorecase " Always case-insensitive
set incsearch " Searches for strings incrementally

set autoindent " Auto-indent new lines
set shiftwidth=4 " Number of auto-indent spaces
set smartindent " Enable smart-indent
set smarttab " Enable smart-tabs
set softtabstop=4 " Number of spaces per Tab
set cursorline " Show underline
set cursorcolumn
set ruler " Show row and column ruler information

set mouse=a

set undolevels=10000 " Number of undo levels
set backspace=indent,eol,start " Backspace behaviour

set splitbelow "New spilt starts below
set splitright "New split starts on the right
set scrolloff=5 "auto scroll on the bottom 5 lines

filetype on "enable file detection
syntax on "syntax highlight

highlight Comment ctermfg=cyan
set showmode

set encoding=utf-8
set fileencoding=utf-8
scriptencoding=utf-8
```

contest_setup/vimrc

2.2 bashrc

```
alias clang++="clang++-3.6 -Wall -Wextra -O2 -o main
    .o"
alias clang-format="clang-format-3.6 -i -style=LLVM"
alias astyle="astyle --style=linux"
```

contest_setup/bashrc

2.3 c++ template

2.4 Java template

3 Reminder

1. Read the problem statements carefully. Input and output specifications are crucial!
2. Estimate the **time complexity** and **memory complexity** carefully.
3. Time penalty is 20 minutes per WA, **don't rush**!
4. Sample test cases must all be tested and passed before every submission!
5. Test the corner cases, such as 0, 1, -1. Test all edge cases of the input specification.

4 Useful code

4.1 Fast Exponentiation

4.2 GCD

小心負數!

4.3 Extended Euclidean Algorithm

4.4 STL quick reference

4.4.1 Map / Set

4.4.2 String

5 Search

5.1 Binary Search

5.1.1 Find key

5.1.2 Upper / lower Bound

5.2 折半完全列舉

5.3 Two-pointer 爬行法

6 Basic data structure

6.1 1D BIT

```cpp
#include <bits/stdc++.h>

using namespace std;

#define LIMIT 500100
#define LSB(x) ((x) & (-x))

long long int bit[LIMIT];

void add(int i, int val)
{
    while (i <= LIMIT) {
        bit[i] += val;
        i += LSB(i);
    }
}

int sum(int i)
{
    int ans = 0;
    while (i > 0) {
        ans += bit[i];
        i -= LSB(i);
    }

    return ans;
}

int main()
{
    int n;
    while (scanf("%d", &n) == 1 && n) {
        int inp[n + 1], orig[n + 1];
        for (int i = 1; i <= n; i++) {
            scanf("%d", &inp[i]);
            orig[i] = inp[i];
        }

        sort(inp + 1, inp + n + 1);

        /*
        因為數字的範圍很大
        所以我們把答案離散化

        按照排列之後的順序
        我們把數字重新給一個編號
        從1開始編起 (BIT是從1開始編起的)
        */
        map<int, int> m;
        for (int i = 1; i <= n; i++)
            m[inp[i]] = i; // duplicated key?

        memset(bit, 0, sizeof(bit));

        long long int ans = 0; // Oops
        for (int cnt = 0; cnt < n; cnt++) {
            /*
            BIT所記錄的資料是利用重新編號之後的數列來紀錄
            所以像說
            6 2 8 4
            會被重新編號成
            3 1 4 2

            在BIT中, 對於每一個i (就是sum(i) )所要記錄的是目前為止合格的數字有幾個(<= i的數字)

            以剛剛那個數列來說 (6 2 8 4)
            對於第一個數字 3 而言
            cnt = 0, sum(6 對應到 3) = 0
            所以ans += 0 - 0; (代表在3之前沒有大於他的數字)
            之後, 在 3 那一格加 1

            這很重要, 因為這麼做就把bit[i >= 3]的所有數值都都加 1 了!
            (這是BIT的特性, 畢竟BIT是拿來算prefix sum的! )

            所以下一次如果我們問到>=3的數字, 假設是4好了
            我們會知道已經有一個<=4的數字已經出現
            也就可以反推有幾個不合格的數字出現了!
            */
            ans += cnt - sum(m[orig[cnt + 1]]);
            add(m[orig[cnt + 1]], 1);
        }

        printf("%lld\n", ans);
    }

    return 0;
}
```

old/"UVA 10810 BIT.cpp"

6.2 2D BIT

6.3 Union Find

6.4 Segment Tree

Hehe