MergeCount
Professor Wladimir

Problema Balé

Referência: https://br.spoj.com/problems/BALE11/
Dado um vetor H, queremos contar o número de pares de índices (i, j) tais que

$$i < j \quad \text{e} \quad H[i] > H[j].$$

Algoritmo Ingênuo

O algoritmo mais simples para resolver este problema percorre todos os pares possíveis de índices:

```
int count(int H[], int N){
    long long int resp = 0;
    for(int i = 0; i < N; i++){
        for(int j = i+1; j < N; j++){
            if(H[i] > H[j]) resp++;
        }
    }
    return resp;
}
```

No SPOJ, o problema possui as seguintes restrições:

- Tempo limite: 0,100 s
- Tamanho do código-fonte: 50.000 B
- Limite de memória: 1536 MB

Considerando que um computador típico consegue realizar aproximadamente 10^8 operações por segundo, e que a complexidade do algoritmo é $O(N^2)$, para uma entrada de tamanho $N = 10^5$, o algoritmo realizaria cerca de 10^{10} operações. Dessa forma, o tempo estimado de execução seria da ordem de 100 segundos, muito acima do limite permitido.

Solução Eficiente com Divisão e Conquista

Para resolver o problema de forma eficiente, utilizamos a técnica de **Divisão e Conquista**. A ideia consiste em dividir o vetor em duas metades — *esquerda* (esq) e *direita* (dir) — e contar separadamente:

- O número de pares (i, j) que estão inteiramente em *esq*.
- O número de pares (i, j) que estão inteiramente em *dir*.
- O número de pares (i, j) em que i pertence a *esq* e j pertence a *dir*.

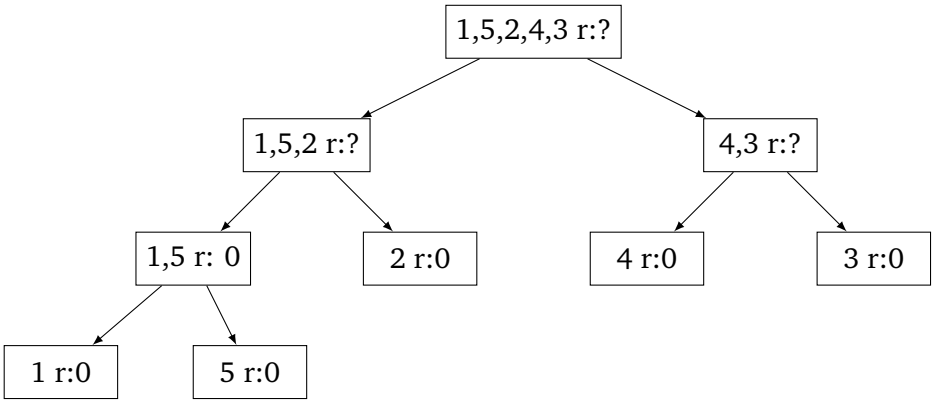

Figura 1: Árvore de subproblemas

Na Figura 1, os casos de vetores de tamanho 1 são triviais: H[0..0], H[1..1], H[4..4]. Para resolver instâncias maiores, como H[0..2] = {1,5,2}, é interessante adotar uma estratégia baseada em **manter cada parte do vetor ordenada de maneira decrescente**, o que facilita a contagem dos pares desejados.

Durante o processo de ordenação, podemos **contar simultaneamente os pares que satisfazem a propriedade de interesse**. Em particular, ao contar pares formados por um índice na primeira parte e um índice na segunda parte, precisamos determinar, para cada elemento da primeira parte, quantos elementos da segunda parte são menores que ele.

Uma abordagem eficiente é começar pelo **maior elemento da primeira parte**. Sabemos, desde o início, que qualquer elemento da segunda parte maior que ele não contribuirá para a contagem, permitindo que esses elementos sejam desconsiderados. Em seguida, o processo continua de forma iterativa, comparando os próximos elementos da primeira parte com os elementos restantes da segunda parte, mantendo a contagem acumulada.

Dessa maneira, conseguimos resolver o problema de forma mais eficiente do que verificando todos os pares individualmente.

A complexidade de tempo desse algoritmo fica $O(n \log n)$

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

typedef long long int ll;
ll  mergesort(int H[], int inicio, int fim){
    if(inicio == fim)
        return 0;
    else{
        int meio = (inicio+fim)/2;

        ll c1 = mergesort(H, inicio, meio);
        //printf("count(%d,%d) = %d\n", inicio, meio, c1);
        ll c2 = mergesort(H, meio+1, fim);
        //printf("count(%d,%d) = %d\n", meio+1, fim, c1);
        int tam = fim-inicio+1;
        int A[tam];

        int i = inicio; // aponta pro comeco do esquerdo de H
        int j = meio+1; // aponta pro comeco do direito de H
        int k = 0;
        ll c3 = 0;
        while( i <= meio || j <= fim){
            if(i > meio){ A[k] = H[j]; k++; j++; }
            else if( j > fim ) { A[k] = H[i]; k++; i++; }
            else if( H[i] > H[j] ){
                 A[k] = H[i]; k++; i++;
                 c3 += fim-j+1;
            }else{
                A[k] = H[j]; k++; j++;
            }

        }
        i = inicio;
        k = 0;
        while( i <= fim){
            H[i] = A[k];
            i++;
            k++;
        }
        //printf("atravesando(%d,%d) = %d\n", inicio, fim, c3);
        return c1+c2+c3;

    }

}

int main(){
    int N;
    scanf("%d", &N);
    int H[N];
    for(int i = 0; i < N; i++)
        scanf("%d", &H[i]);
    ll resp = mergesort(H,0,N-1);
    printf("%lld\n", resp);
}

```

Exercícios

1. **OBI – Balé** *(OBI 2013, Fase 2 - Programação Nível 2)* Contar o número de pares fora de ordem em um vetor.
   - https://olimpiada.ic.unicamp.br/pratique/p2/2013/f2/bale/
   - https://judge.beecrowd.com/pt/problems/view/2400
2. **SPOJ BR – Inversões (INVCNT)** Dado um vetor, contar o número de pares (i, j) tais que $i < j$ e $A[i] > A[j]$. https://www.spoj.com/problems/INVCNT/
3. **Codeforces – Inversions after Shuffle (Round 144C)** Problema que mistura contagem de inversões com manipulação de arranjos. https://codeforces.com/problemset/problem/144/C
4. **UVa 10810 – Ultra-QuickSort** Problema clássico em que é necessário contar o número mínimo de trocas para ordenar o vetor, equivalente ao número de inversões. https://onlinejudge.org/external/108/10810.pdf
5. **Codeforces 1638C – Inversion Graph** https://codeforces.com/problemset/problem/1638/C
6. **Maratona de Programação da SBC 2018 - Problema C - Cortador de Pizza** https://judge.beecrowd.com/pt/problems/view/2878