Desbalance

Primero podemos observar lo siguiente:

S_i = Subarreglo i-esimo (en un orden arbitrario)

$S = [S_1, \dots, S_{\frac{n(n+1)}{2}}]$

La respuesta esperada es la siguiente:

$$(Max(S_1) - Min(S_1)) + \dots + (Max(S_{\frac{n(n+1)}{2}}) - Min(S_{\frac{n(n+1)}{2}}))$$

Podemos reescribirlo de la siguiente forma:

$$\underbrace{(Max(S_1) + \dots + Max(S_{\frac{n(n+1)}{2}}))}_{\text{Suma de maximos}} - \underbrace{(Min(S_1) + \dots + Min(S_{\frac{n(n+1)}{2}}))}_{\text{Suma de minimos}}$$

Podemos separar lo anterior en dos problemas diferentes: Suma de maximos y suma de minimos.

Podriamos intentar resolver estos problemas con un enfoque D&C.

Nota: Estos problemas se pueden resolver muy facil usando una EDD llamada Segment Tree, por si quieren investigar.

Resolveremos el problema de la suma de los maximos.

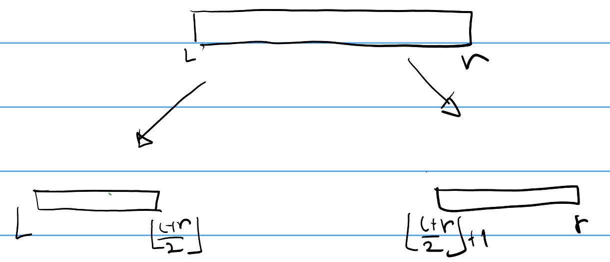
Vamos a definir la siguiente función:

SumaMaximo(A, L, R) → la idea es que responda el problema desde L hasta R.

$$\text{SumaMaximo}(A, L, R) \begin{cases} \to \text{SumaMaximo}(A, L, \lfloor\frac{L+r}{2}\rfloor) \\ \to \text{SumaMaxima}(A, \lfloor\frac{L+r}{2}\rfloor + 1, r) \end{cases}$$

Esta sería la lógica DyC.

Suma Maxima(A,l,r)



podemos observar que si asuminos de que al llamar Suma Maxima De Los Mitades funciona

Solo nos falta calcular las respuestas de los subarreglos que pasan por ambos segmentos.

Para calcularlo podemos hacer lo siguiente:

PreMax: Arreglo que guarda el maximo de cada prefijo de la mitad derecha.
SufMax: Arreglo que guarda el maximo de cada sufijo de la mitad derecha.
Calcular estos arreglos toma O(N).

Ej: Sufijos Prefijos

7	4	1	2	3	3	3	1	5	7
7	4	3	3	3	3	3	3	5	7

Maximo sufijos Maximo prefijos

Esto nos sirve par calcular la suma de los arreglos que pasan por ambas mitades.

Podemos recorrer cada sufijo de la mitad izquierda y podemos tomar el maximo (el cual ya calculamos antes) para verificar en la otra mitad hasta que punto el maximo del subarreglo se encuentra en la mitad izquierda.

7	4	1	2	3		3	3	1	5	7
7	4	3	3	3		3	3	3	5	7
				x			y			

El maximo subarreglo en el sufijo actual donde el maximo esta

Los segmentos que comienzan en el sufijo el maximo esta a la Derecha.

En cada iteracion vamos moviendo el puntero de la Mitad izquierda.

2° iteracion

7	4	1	2	3		3	3	1	5	7
7	4	3	3	3		3	3	3	5	7
			x					y		

3° iteracion

7	4	1	2	3		3	3	1	5	7
7	4	3	3	3		3	3	3	5	7
		x						y		

3° iteracion

7	4	1	2	3		3	3	1	5	7
7	4	3	3	3		3	3	3	5	7
	x							y		

4° iteracion

7	4	1	2	3		3	3	1	5	7
7	4	3	3	3		3	3	3	5	7
x										y

Podemos ver que como el segmento verde es donde el maximo esta a la izquierda en cada iteracion podemos sumar a la respuesta $(y - mid) \cdot SufMax[x]$

y ademas

Podemos observar que ademas podemos añadir:

$$(L - y) \cdot \underbrace{(PreMax[y+1] + \cdots + PreMax[r])}_{\text{Podemos responder esto en } O(1) \text{ si lo precalculamos}}$$

Para buscar de forma eficiente la pos "y" podemos hacerlo usando un puntero que parta en el inicio y lo vamos moviendo siempre y cuando la nueva pos sea ≤.

Siguiendo el procedimiento anterior se calcula la suma de todos los subarreglos que pasan por ambas mitades.

SumaMinima() → Podemos implementarlo de la misma forma pero lo hacemos con los minimos

Codigo:

```cpp
#include <iostream>
#include <climits>
using namespace std;

int nums[1000000];
int aux[1000000];

int solveMin(int l, int r) {
    if (l >= r) return 0;

    int mid = (l + r) / 2;
    int minVal = INT_MAX;
    int acum = 0;

    for (int i = mid + 1; i <= r; ++i) {
        minVal = min(minVal, nums[i]);
        aux[i] = minVal;
        acum += minVal;
    }

    minVal = INT_MAX;
    int curr = mid + 1;
    int preans = 0;

    for (int i = mid; i >= l; --i) {
        minVal = min(minVal, nums[i]);
        while (curr <= r && aux[curr] >= minVal) {
            acum -= aux[curr];
            curr++;
        }
        preans += acum + (curr - mid - 1) * minVal;
    }

    return preans + solveMin(l, mid) + solveMin(mid + 1,
r);

int solveMax(int l, int r) {
    if (l >= r) return 0;

    int mid = (l + r) / 2;
    int maxVal = 0;
    int acum = 0;

    for (int i = mid + 1; i <= r; ++i) {
        maxVal = max(maxVal, nums[i]);
        aux[i] = maxVal;
        acum += maxVal;
    }

    maxVal = 0;
    int curr = mid + 1;
    int preans = 0;

    for (int i = mid; i >= l; --i) {
        maxVal = max(maxVal, nums[i]);
        while (curr <= r && aux[curr] <= maxVal) {
            acum -= aux[curr];
            curr++;
        }
        preans += acum + (curr - mid - 1) * maxVal;
    }

    return preans + solveMax(l, mid) + solveMax(mid + 1,
r);
```