



INSTITUTO POLITÉCNICO  
NACIONAL

ESCUELA SUPERIOR DE  
CÓMPUTO



ANALISIS DE ALGORITMOS

PROFESOR TITULAR: FRANCO MARTINEZ  
EDGARDO ADRIAN

2° PARCIAL

EJERCICIOS #10:  
DISEÑO DE SOLUCIONES DIVIDE Y VENCERAS

LEMUS RUIZ MARIANA ELIZABETH  
2020630211

GRUPO: 3CM12



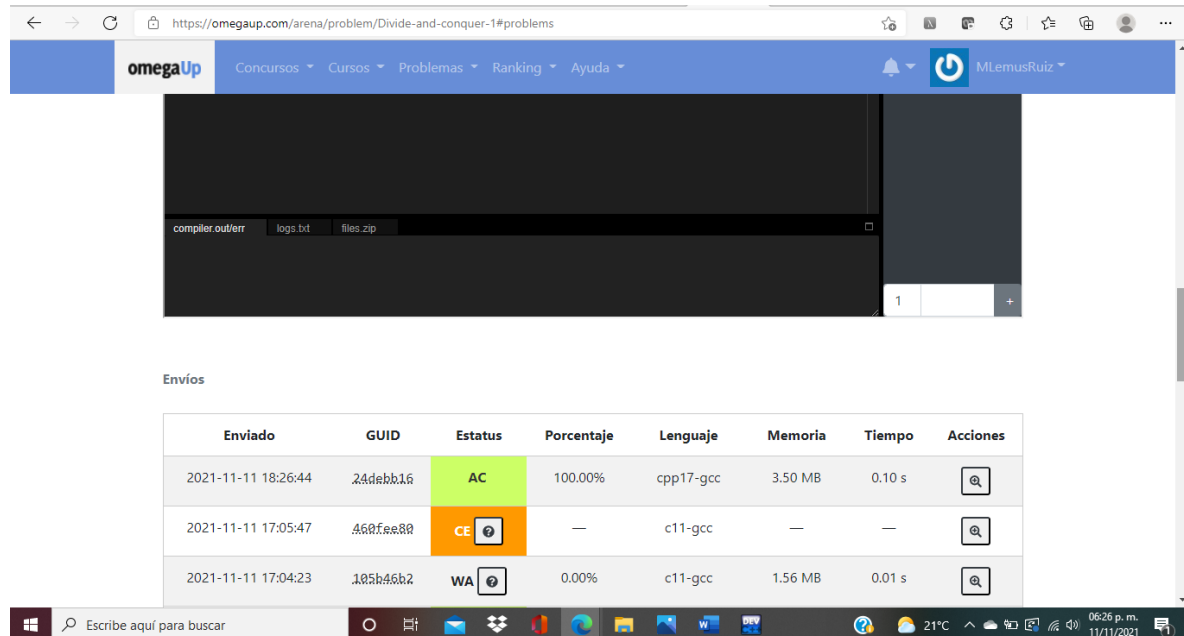
## EJERCICIOS 10: DISEÑO DE SOLUCIONES DIVIDE Y VENCERAS

### INSTRUCCIONES:

De los siguientes 10 problemas que se plantean resolver al menos 4 problemas para completar el ejercicio.

### EJERCICIO 01: DIVIDE AND CONQUER 1

Captura de aceptación



Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-11 18:26:44	2Adebb16	AC	100.00%	cpp17-gcc	3.50 MB	0.10 s	[icon]
2021-11-11 17:05:47	468f8e88	CE	—	c11-gcc	—	—	[icon]
2021-11-11 17:04:23	185b46b2	WA	0.00%	c11-gcc	1.56 MB	0.01 s	[icon]

### Explicación de solución y Análisis de Complejidad

#### Descripción

Edgardo se puso un poco intenso este semestre y puso a trabajar a sus alumnos con problemas de mayor dificultad.

La tarea es simple, dado un arreglo de números enteros debes imprimir cual es la suma máxima en cualquier subarreglo contiguo.

Por ejemplo si el arreglo dado es  $\{-2, -5, 6, -2, -3, 1, 5, -6\}$ , entonces la suma máxima en un subarreglo contiguo es 7.

#### Entrada

La primera línea contendrá un número  $N$ .

En la siguiente línea  $N$  enteros representando el arreglo  $A$ .

## Salida

La suma máxima en cualquier subarreglo contiguo.

8	7
-2 -5 6 -2 -3 1 5 -6	

## Código:

```
#include<algorithm>
#include<iostream>
#include<climits>

typedef long long int largo;
using namespace std;

largo maximoSuma(largo numeros[],
largo izquierda, largo centro,
largo derecha);
largo DivideVencerás(largo
numeros[], largo izquierda, largo
derecha);

int main(void){
    largo elemArreglo=0;
    largo resultado=0;

    cin>>elemArreglo;
    largo numeros[elemArreglo];

    for(largo i=0; i<elemArreglo;
i++)
        cin>>numeros[i];

    resultado=DivideVencerás(numeros
,0, elemArreglo);

    cout<<resultado<<endl;

    return 0;
}

largo DivideVencerás(largo
numeros[], largo izquierda, largo
derecha){
    if(izquierda==derecha)
        return numeros[izquierda];

    largo izqF, derF, mitad,
medio;

    mitad=(izquierda+derecha)/2;
```

Operación Básica: comparaciones  
entre elementos max

$$T(n) = \begin{cases} 0, & \text{si } n=1 \\ 2T\left(\frac{n}{2}\right) + n, & \text{si } n > 1 \end{cases}$$
$$T(n) \in O(n \log(n))$$

Usando el enfoque Divide y Vencerás,  
podemos encontrar la suma máxima  
de submatrices en tiempo  $O(n \log n)$ . El  
siguiente es el algoritmo Divide y  
Vencerás.

Divida la matriz dada en dos mitades  
Devuelve el máximo de los tres  
siguientes

Suma máxima de submatrices en la  
mitad izquierda (Realizar una llamada  
recursiva)

Suma máxima de submatrices en la  
mitad derecha (Realizar una llamada  
recursiva)

Suma máxima de submárray tal que la  
submatriz cruza el punto medio

```
    izqF=DivideVenceras (numeros,
    izqF, mitad);
    derF=DivideVenceras (numeros,
    mitad+1, derF);
    medio=maximoSuma (numeros,
    izquierda, mitad, derecha);

    if(izqF>derF){
        if(izqF>medio)
            return izqF;
        else
            return medio;
    }else{
        if(medio>derF)
            return medio;
        else
            return derF;
    }
}

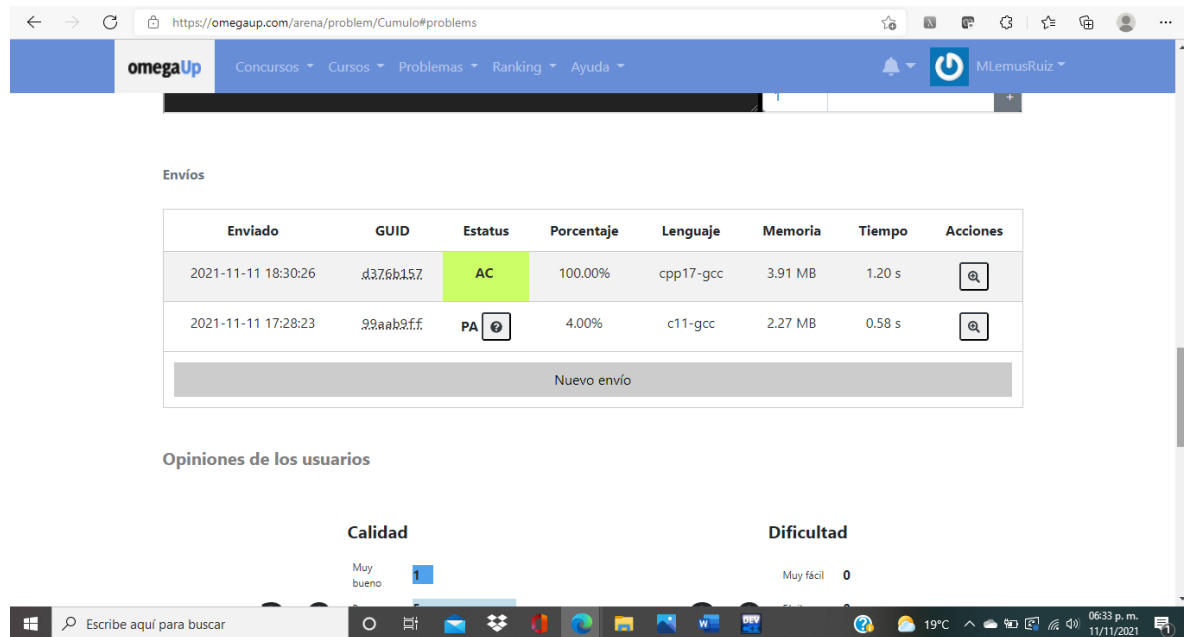
largo maximoSuma(largo numeros[],
largo izquierda, largo centro,
largo derecha){
    largo izqF=LLONG_MIN,
    derF=LLONG_MIN, suma;
    largo i;

    for(i=centro, suma=0; i>=izqF;
i--){
        suma=suma+numeros[i];
        if(suma>izqF)
            izqF=suma;
    }

    for(i=centro+1, suma=0;
i<=derecha; i++){
        suma=suma+numeros[i];
        if(suma>derF)
            derF=suma;
    }
    return derF+izqF;
}
```

## EJERCICIO 02: CUMULO

### Captura de aceptación



## Explicación de solución y Análisis de Complejidad

### Descripción

Te encuentras con un mapa del cúmulo de estrellas R136. En el mapa, cada estrella aparece como un punto ubicado en un plano cartesiano. Te asalta de pronto una pregunta, ¿cuál será la distancia mínima entre dos estrellas en el mapa?

### Entrada

La primera línea tendrá un entero  $2 \leq n \leq 50000$  que indica la cantidad de estrellas en el mapa. Las siguientes  $n$  líneas tendrán las coordenadas de las estrellas, dadas por dos reales  $X$  y  $Y$ . En todos los casos  $0 \leq X, Y \leq 40000$ .

### Salida

La distancia mínima entre dos estrellas, expresada con un número real con tres cifras después del punto decimal. (La distancia se calcula como la raíz cuadrada de la suma de los cuadrados de las diferencias en  $X$  y  $Y$ ).

### Código:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

using namespace std;
```

$$T(n) = \begin{cases} 0, & \text{si } n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & \text{si } n > 1 \end{cases}$$
$$T(n) \in O(n \log(n))$$

```
vector <pair<double, double> > numeros;  
int fin)  
{  
    if(fin - ini == 1)  
        return  
    sqrt(pow(fabs(numeros[fin].first -  
numeros[ini].first), 2) +  
  
        pow(fabs(numeros[fin].second -  
numeros[ini].second), 2));  
    if(fin - ini == 2)  
    {  
        double izq, der, enmedio,  
menor;  
        izq =  
sqrt(pow(fabs(numeros[ini + 1].first -  
numeros[ini].first), 2) +  
        pow(fabs(numeros[ini +  
1].second - numeros[ini].second), 2));  
        der =  
sqrt(pow(fabs(numeros[fin].first -  
numeros[ini + 1].first), 2) +  
  
        pow(fabs(numeros[fin].second -  
numeros[ini + 1].second), 2));  
        enmedio =  
sqrt(pow(fabs(numeros[fin].first -  
numeros[ini].first), 2) +  
  
        pow(fabs(numeros[fin].second -  
numeros[ini].second), 2));  
        menor = min(izq, der);  
        menor = min(enmedio, menor);  
        return menor;  
    }  
    int mid = (fin + ini) / 2;  
    double der, izq, medio, mini;  
    der = cumulo(mid, fin);  
    izq = cumulo(ini, mid);  
    medio = cumulo(mid - 1, mid + 1);  
    mini = min(izq, der);  
    mini = min(mini, medio);  
    return mini;  
}  
void merge(int ini, int mid, int fin,  
int XorY)  
{  
    int i, j, k;  
    int tam1 = mid - ini + 1;  
    int tam2 = fin - mid;  
  
    vector <pair<double, double> >  
izq(tam1);  
    vector <pair<double, double> >  
der(tam2);  
    for(i = 0 ; i < tam1 ; i++)  
    {
```

La función "cúmulo" es donde el problema se divide en subproblemas y los resultados se combinan para llegar a una solución general. Se divide en la coordenada de resolución del problema desde el arreglo hasta que se alcanza la situación básica. Una vez que se encuentra la situación básica, se devolverá la solución y se guardará el valor más bajo cuando aumente el árbol de recursividad.

Merge recibe la matriz dividida para poder unirla, que depende de la variable XorY, que es una bandera para saber si la matriz debe ordenarse como referencia para el eje Y o el eje X. Este función es una implementación combinada del algoritmo de clasificación Merge Sort. La función recibe el límite utilizado

```
        izq[i].first = numeros[ini +
i].first;
        izq[i].second = numeros[ini
+ i].second;
    }
    for(j = 0 ; j < tam2 ; j++)
    {

        der[j].first = numeros[mid
+ 1 + j].first;
        der[j].second = numeros[mid
+ 1 + j].second;
    }
    i = 0;
    j = 0;
    k = ini;
    if(XorY)
    {
        while(i < tam1 && j < tam2)
        {
            if(izq[i].second <
der[j].second)
            {
                numeros[k].first
= izq[i].first;

                numeros[k].second = izq[i].second;
                i++;
            }
            else
            {
                numeros[k].first
= der[j].first;

                numeros[k].second = der[j].second;
                j++;
            }
            k++;
        }
    }
    else
    {
        while(i < tam1 && j < tam2)
        {
            if(izq[i].first <
der[j].first)
            {
                numeros[k].first
= izq[i].first;

                numeros[k].second = izq[i].second;
                i++;
            }
            else
            {
                numeros[k].first
= der[j].first;
```

en esta iteración como parámetro y también se define mediante la variable XorY, que se utiliza para identificar si desea ser relativo al eje Y o el eje X

```
        numeros[k].second = der[j].second;
        j++;
    }
    k++;
}
}
while(i < tam1)
{
    numeros[k].first =
    izq[i].first;
    numeros[k].second =
    izq[i].second;
    i++;
    k++;
}
while(j < tam2)
{
    numeros[k].first =
    der[j].first;
    numeros[k].second =
    der[j].second;
    k++;
    j++;
}
}
void mergeSort(int ini, int fin, int
XorY)
{
    if(ini < fin)
    {
        int mid = (fin + ini) / 2;
        mergeSort(ini, mid, XorY);
        mergeSort(mid + 1, fin,
XorY);
        merge(ini, mid, fin, XorY);
    }
}
int main(void)
{
    ios::sync_with_stdio(false);
    cout << fixed << setprecision(3);
    double cordX, cordY;
    int nums;
    cin >> nums;
    numeros.clear();
    for(int i = 0 ; i < nums ; i++)
    {
        cin >> cordX >> cordY;

        numeros.push_back(make_pair(cordX,
cordY));
    }
    if(nums == 2)
    {
```

El main es principalmente para recibir coordenadas Al enviar una disposición ordenada en relación con el eje Y y el eje X, aquí se recibe el valor devuelto por la función de cúmulo para devolver la solución al problema.



```
        double res =  
sqrt(pow(fabs(numeros[1].first -  
numeros[0].first), 2) +  
  
        pow(fabs(numeros[1].second -  
numeros[0].second), 2));  
        cout << res << "\n";  
    }  
    else  
    {  
        mergeSort(0, numeros.size()  
- 1, 0);  
        double ejeX = cumulo(0,  
numeros.size() - 1);  
        mergeSort(0, numeros.size()  
- 1, 1);  
        double ejeY = cumulo(0,  
numeros.size() - 1);  
        cout << min(ejeX, ejeY) <<  
"\n"  
    }  
    return 0;  
}
```

## EJERCICIO 03: AMIGOS Y REGALOS

### Captura de aceptación

omegaUp

Concursos Cursos Problemas Ranking Ayuda

M. Lemus Ruiz

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-11 18:36:53	218c9363	AC	100.00%	cpp17-gcc	3.50 MB	0.03 s	

Nuevo envío

#### Opiniones de los usuarios

##### Calidad

3.3

16 votos en total

Muy bueno	5
Bueno	10
Regular	0
Malo	1
Muy malo	0

##### Dificultad

2.6

18 votos en total

Muy fácil	0
Fácil	1
Medio	6
Difícil	5
Muy difícil	6

## Explicación de solución y Análisis de Complejidad

### Descripción

Tienes dos amigos. A ambos quieres regalarles varios números enteros como obsequio. A tu primer amigo quieres regalarle  $C_1$  enteros y a tu segundo amigo quieres regalarle  $C_2$  enteros. No satisfecho con eso, también quieres que todos los regalos sean únicos, lo cual implica que no podrás regalar el mismo entero a ambos de tus amigos.

Además de eso, a tu primer amigo no le gustan los enteros que son divisibles por el número primo  $X$ . A tu segundo amigo no le gustan los enteros que son divisibles por el número primo  $Y$ . Por supuesto, tu no le regalaras a tus amigos números que no les gusten.

Tu objetivo es encontrar el mínimo número  $V$ , de tal modo que puedas dar los regalos a tus amigos utilizando únicamente enteros del conjunto  $1, 2, 3, \dots, V$ . Por supuesto, tú podrías decidir no regalar algunos enteros de ese conjunto.

Un número entero positivo mayor a 1 es llamado primo si no tiene divisores enteros positivos además del 1 y el mismo.

### Entrada

Una línea que contiene cuatro enteros positivos  $C_1, C_2, X, Y$ . Se garantiza que  $X$  y  $Y$  son números primos.

## Salida

Una línea. Un entero que representa la respuesta al problema.

## Código:

```
#include <iostream>
#include <cmath>
using namespace std;
long long int C1, C2, P1, P2;

int func(long long int ResBin)
{
    long long int prim, seg,
    resta, suma, LCM = P1 * P2,
    numLCM, R1, R2;
    numLCM = floor(ResBin / LCM);

    prim = (ResBin / P1) - numLCM;
    seg = (ResBin / P2) - numLCM;
    resta = (ResBin - (prim + seg)
- numLCM);

    R1 = max(C2 - prim, (long long
int)0);
    R2 = max(C1 - seg, (long long
int)0);
    suma = R1 + R2;
    if(resta > suma)
        return 1;
    else if(resta < suma)
        return -1;
    else
        return 0;
}

int main(void)
{
    ios::sync_with_stdio(false);
    cin >> C1 >> C2 >> P1 >> P2;
    long long int ini, fin, mid,
res;
    ini = 1;
    fin = 3 * (C1 + C2);
    while(ini <= fin)
    {
        mid = (ini + fin) >> 1;
        res = func(mid);
        if(res == 0)
        {
            if(mid % (P1 * P2) ==
0)
                mid--;
```

Operación Básica: comparaciones  
entre elementos max

$$T(n) = \begin{cases} 0, & \text{si } n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & \text{si } n > 1 \end{cases}$$
$$T(n) \in O(n \log(n))$$

```
        cout << mid << "\n";  
        return 0;  
    }  
    else if(res == 1)  
        fin = mid - 1;  
    else  
        ini = mid + 1;  
    }  
    return 0;  
}
```