# Archivo:Logo Instituto Politécnico Nacional.png - Wikipedia, la enciclopedia libreArchivo:EscudoESCOM.png - Wikipedia, la enciclopedia libreINSTITUTO POLITÉCNICO NACIONAL

# ESCUELA SUPERIOR DE CÓMPUTO

Vargas Hernández Carlo Ariel

Análisis de algoritmos

3CM15

Ejercicios 10



Contenido

[**Problema 1: Divide and conquer 1** 1](#_Toc103637190)

[Descripción: 1](#_Toc103637191)

[Explicación: 1](#_Toc103637192)

[Complejidad: 2](#_Toc103637193)

[Código: 3](#_Toc103637194)

[Captura: 5](#_Toc103637195)

[**Problema 07: INVCNT - Inversion Count** 5](#_Toc103637196)

[Descripción 5](#_Toc103637197)

[Explicación 5](#_Toc103637198)

[Complejidad 6](#_Toc103637199)

[Código 6](#_Toc103637200)

[Captura 8](#_Toc103637201)

[**Problema 06: Con pocos ceros consecutivos** 9](#_Toc103637202)

[Descripción 9](#_Toc103637203)

[Explicación 9](#_Toc103637204)

[Complejidad 10](#_Toc103637205)

[Código 10](#_Toc103637206)

[Captura 11](#_Toc103637207)

[**Problema 02: Amigos y Regalos** 11](#_Toc103637208)

[Descripción 11](#_Toc103637209)

[Explicación 12](#_Toc103637210)

[**Referencias:** 12](#_Toc103637211)

# **Problema 1: Divide and conquer 1**

## 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 A 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.

Salida:

La suma máxima en cualquier subarreglo contiguo. [1]

## Explicación:

El método de divide y vencerás que nos dice que debemos fraccionar el problema hasta un caso que se resuelva fácil, en este problema podemos ver que si tenemos un arreglo de tamaño 1 la solución al problema sería sencilla porque la única respuesta posible es el mismo elemento.

La forma en que llegaremos a un único elemento será usando una función recursiva que divida en dos el arreglo y se llame a si misma por el lado izquierdo y derecho hasta llegar al elemento individual, que es el caso base:

long long int SubArregloSumMax(…){

  ……

  if(inicio==final){

    return A[inicio];

  }

  mitad=(inicio+final)/2;

  izq=SubArregloSumMax( A,inicio,mitad);

  der=SubArregloSumMax(A,mitad+1,final);

……

}

Sin embargo al dividir nos encontramos con el problema de que podría haber un subarreglo en medio de las dos mitades que sea el de mayor suma, por lo que para resolver este problema se debe hacer una suma de la mitad derecha e izquierda, adicional a esto cabe la posibilidad de que la mayor suma sea la que junte estas dos sumas anteriores. En código se vería algo así:

……

  suma=0;

  sumaizq=A[mitad];

  for(int i=mitad;i>=inicio;i--){

    suma+=A[i];

    if(suma>sumaizq){

      sumaizq=suma;

    }

  }

  suma=0;

  sumader=A[mitad+1];

  for(int i=mitad+1;i<=final;i++){

    suma+=A[i];

    if(suma>sumader){

      sumader=suma;

    }

  }

  sumacentro=sumaizq+sumader;

  suma=NumMax(sumaizq,sumader,sumacentro);

……

Al final se escogería la suma más grande de entre la derecha, izquierda o centro. Cabe mencionar que los ciclos de suma del centro de donde se dividió el arreglo hacia la derecha o izquierda dependiendo de la mitad, para que al calcular la suma centro sea de elementos contiguos. Finalmente se compara el valor de todos los valores obtenidos tanto de izquierda y derecha y la suma de centro y se retorna el mayor.

……

sumacentro=sumaizq+sumader;

  suma=NumMax(sumaizq,sumader,sumacentro);

  return NumMax(izq,der,suma);

}

## Complejidad:

Al usar divide y vencerás, el problema se separa en dos mitades. Con lo que la complejidad temporal empieza con n/2, sin embargo a esto hay que añadirle el ciclo que junta las mitades n. Hasta que llegue a un caso base donde el tamaño de n sea 1, en ese caso solo debe retornar ese elemento que es constante por lo cual tomaremos como 0.

T(n)=0 si n=1 y T(n)=2T(n/2)+n.

Por teorema maestro obtenemos que esta recursión pertenece a O(nlog(n)).

## Código:

Función main, y cabezera.

#include <bits/stdc++.h>

using namespace std;

long long int NumMax(long long int a, long long int b) { return (a > b) ? a : b; }

long long int NumMax(long long int a, long long int b, long long int c) { return max(max(a, b), c); }

long long int SubArregloSumMax(long long int A[],long long int inicio,long long int final){

………

}

int main(){

  long long int a,n;

    cin>>n;

    long long int A[n];

    for(int i=0;i<n;i++){

        cin>>a;

        A[i]=a;

    }

    cout<<SubArregloSumMax( A, 0,n-1);

}

Función para buscar el sub arreglo.

long long int SubArregloSumMax(long long int A[],long long int inicio,long long int final){

  long long int mitad,izq,der,suma,sumaizq,sumader,sumacentro;

  if(inicio==final){

    return A[inicio];

  }

  mitad=(inicio+final)/2;

  izq=SubArregloSumMax( A,inicio,mitad);

  der=SubArregloSumMax(A,mitad+1,final);

  suma=0;

  sumaizq=A[mitad];

  for(int i=mitad;i>=inicio;i--){

    suma+=A[i];

    if(suma>sumaizq){

      sumaizq=suma;

    }

  }

  suma=0;

  sumader=A[mitad+1];

  for(int i=mitad+1;i<=final;i++){

    suma+=A[i];

    if(suma>sumader){

      sumader=suma;

    }

  }

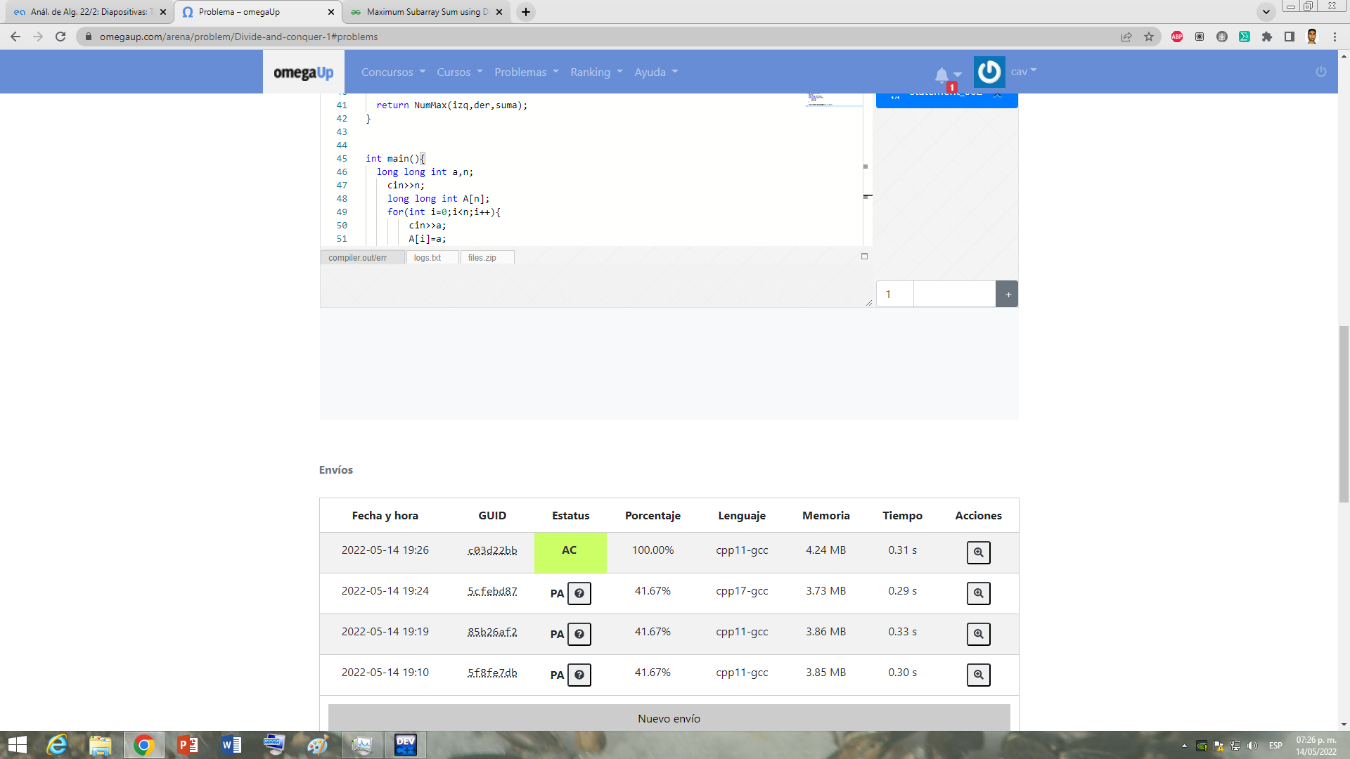
  sumacentro=sumaizq+sumader;

  suma=NumMax(sumaizq,sumader,sumacentro);

  return NumMax(izq,der,suma);

}

## Captura:



# **Problema 07: INVCNT - Inversion Count**

## Descripción

Sea A un arreglo de N números positivos diferentes. Si i<j y A[i] > A[j], entonces el par (i, j) es una inversión. Dada una n y un A encuentra el número de inversiones.

Entrada:

La primer línea contiene t, el número de casos de prueba, seguido por un espacio en blanco, después un n (n<=200000). Las siguientes líneas son los elementos del arreglo.

Salida:

Para cada caso imprime en una línea el número de inversiones de ese caso. [2]

## Explicación

La forma de resolución de este problema es similar a la del algoritmo merge sort, que por sí solo es un divide y vencerás. El método de ordenamiento por mezcla divide el arreglo a la mitad y lo sigue dividiendo hasta llegar al elemento unitario.

Una vez llegado al elemento unitario conquistar el problema es fácil, ya que el elemento es el más grande. Al regresar se compara los elementos de las mitades de izquierda y derecha, al hacer el cambio (si se hace uno) se cuenta una inversión, tanto de la izquierda como de la derecha.

Similar al problema 1, se deben contabilizar las inversiones que se hagan en el medio al juntar dos mitades (al mezclar). En este último caso el número de inversiones es igual al número de elementos que sean mayores al elemento que se debe cambiar.

Esto se puede apreciar en el ciclo while de la función merge del programa, donde mid –i son los elementos que se cambian:

    while ((i <= mid - 1) && (j <= der)) {

        if (arr[i] <= arr[j]) {

            temp[k++] = arr[i++];

        }

        else {

            temp[k++] = arr[j++];

            invcont=invcont+(mid - i);

        }

    }

## Complejidad

Debido a que se usa el algoritmo de ordenamiento por mezcla con una pequeña modificación para resolver el problema. Dado que la modificación es un contador, y esto tiene un costo constante es despreciable para el cálculo de la complejidad, calculo que ya hemos hecho antes para este algoritmo y que es: O(nlog(n)).

## Código

Función main y cabezera.

#include <bits/stdc++.h>

using namespace std;

typedef long long int lli;

lli \_mergeSort(lli arr[], lli temp[], lli izq, lli der);

lli merge(lli arr[], lli temp[], lli izq, lli mid, lli der);

………

int main(){

  lli t, n;

  cin >> t;

  while(t--){

    cin >> n;

    lli arr[n];

    for(lli i = 0; i < n; ++i){

      cin >> arr[i];

    }

    cout <<mergeSort(arr, n)<<"\n";

  }

    return 0;

}

Función de mezcla (donde se cuentan inversiones al mezclar):

/\* Aqui se mezcla\*/

lli merge(lli arr[], lli temp[], lli izq, lli mid, lli der){

    lli i, j, k;

    lli invcont = 0;

    i = izq;

    j = mid;

    k = izq;

    while ((i <= mid - 1) && (j <= der)) {

        if (arr[i] <= arr[j]) {

            temp[k++] = arr[i++];

        }

        else {

            temp[k++] = arr[j++];

            invcont=invcont+(mid - i);

        }

    }

    while (i <= mid - 1){

      temp[k++] = arr[i++];

  }

    while (j <= der){

        temp[k++] = arr[j++];

  }

    /\*Pasar al array original\*/

    for (i = izq; i <= der; i++){

      arr[i] = temp[i];

  }

    return invcont;

}

Función de mezcla (donde se cuentan las inversiones en las mitades y la suma con las de mezcla):

/\*Mediante el orden de merge se obtienen las inversiones\*/

lli \_mergeSort(lli arr[], lli temp[], lli izq, lli der){

    lli mid, invcont = 0;

    if (der > izq) {

        mid = (der + izq) / 2;

        /\*Las inversiones totales seran las sumas de las de la derecha y las de la derecha más las que salgan al mezclar\*/

        invcont+=\_mergeSort(arr, temp, izq, mid);

        invcont+=\_mergeSort(arr, temp, mid + 1, der);

        invcont+=merge(arr, temp, izq, mid + 1, der);

    }

    return invcont;

}

Función de mergeSort que llama a la función anterior, le pasa los parámetros para que funcione:

lli mergeSort(lli arr[], lli n){

    lli temp[n];

    return \_mergeSort(arr, temp, 0, n-1);

}

## Captura



# **Problema 06: Con pocos ceros consecutivos**

## Descripción

Escribe un programa que calcule el número cadenas binarias distintas de longitud que no tienen más de dos ceros consecutivos.

Entrada:

Un entero N. Puedes suponer que 1<=N<=33.

Salida:

Un entero que sea el número cadenas binarias distintas de longitud N que no tienen más de dos ceros consecutivos. [3]

## Explicación

Para este ejercicio (que me tomo unos cuantos días resolver) tuve que ver primero las cadenas binarias que se forman con diferentes N, ejemplo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 |
| 1 | 0 | 00 | 000 | 0000 |
| 2 | 1 | 01 | 001 | 0001 |
| 3 |  | 10 | 010 | 0010 |
| 4 |  | 11 | 011 | 0011 |
| 5 |  |  | 100 | 0100 |
| 6 |  |  | 101 | 0101 |
| 7 |  |  | 110 | 0110 |
| 8 |  |  | 111 | 0111 |
| 9 |  |  |  | 1000 |
| 10 |  |  |  | 1001 |
| 11 |  |  |  | 1010 |
| 12 |  |  |  | 1011 |
| 13 |  |  |  | 1100 |
| 14 |  |  |  | 1101 |
| 15 |  |  |  | 1110 |
| 16 |  |  |  | 1111 |
| No. de cadenas | **2** | **4** | **7** | **13** |

Esta tabla la hice hasta una N=6, para N=5 y N=6 el número de cadenas que cumplen 24 y 44 respectivamente. Al principio no veía donde estaba el dividir pero luego vi que los números en rojo se repetían de la columna de la izquierda, se repetían una vez con un cero hasta la izquierda y otra vez hasta la derecha. El número total de cadenas cumplan o no con la condición está dado por 2N, y en la columna de una N dada se repetían las cadenas de la columna N-1 y así sucesivamente hasta llegar a N=1.

Después me di cuenta que 2N-1=2N/2 con lo que ya había encontrado la parte de la división para el DyV, en este caso sobre 2 como sospechaba pero no encontraba el cómo.

Ahora solo tenía que encontrar la forma de conquistar el problema, después de uno o dos días observe que para una N=1 no puede haber un caso donde haya más de dos ceros seguidos, por el tamaño de la cadena binaria, la primer N para la cual puede salir un caso que no cumpla esta condición es 3, con lo cual ya en este punto pensaba que estos 3 casos eran los casos base, sumando estos 3 casos se obtiene el número de cadenas que cumple la condición para N=4. Para N=5 el resultado es 24, intente sumar todos los valores hasta el N=5 pero me daba 26, me di cuenta que no hay que sumar todos, solo los 3 anteriores que a su vez sumaran los 3 anteriores así hasta llegar a los 3 casos base, con lo que técnicamente si estoy sumando los anteriores a ese número y estoy yendo de lo general a lo particular como se esperaría de un DyV.

## Complejidad

La complejidad del algoritmo recursivo sería:

T(n)=T(n-1)+T(n-2)+T(n-3), T(1)=2, T(2)=4, T(3)=7

T(n)= T(1)+T(2)+T(3)+…+T(n-3)+ T(n-2)+ T(n-1)

T(n)=T((n-1)(n-1+1)/2)=T((n-1)n/2) que sería O(n2)

## Código

#include<bits/stdc++.h>

using namespace std;

int MenosCeros(int n){

  if(n==1)  {

    return 2;

  }else if(n==2){

    return 4;

  } else if(n==3){

    return 7;

  }

  return MenosCeros(n-1)+MenosCeros(n-2)+MenosCeros(n-3);

}

int main ()

{

  int n;

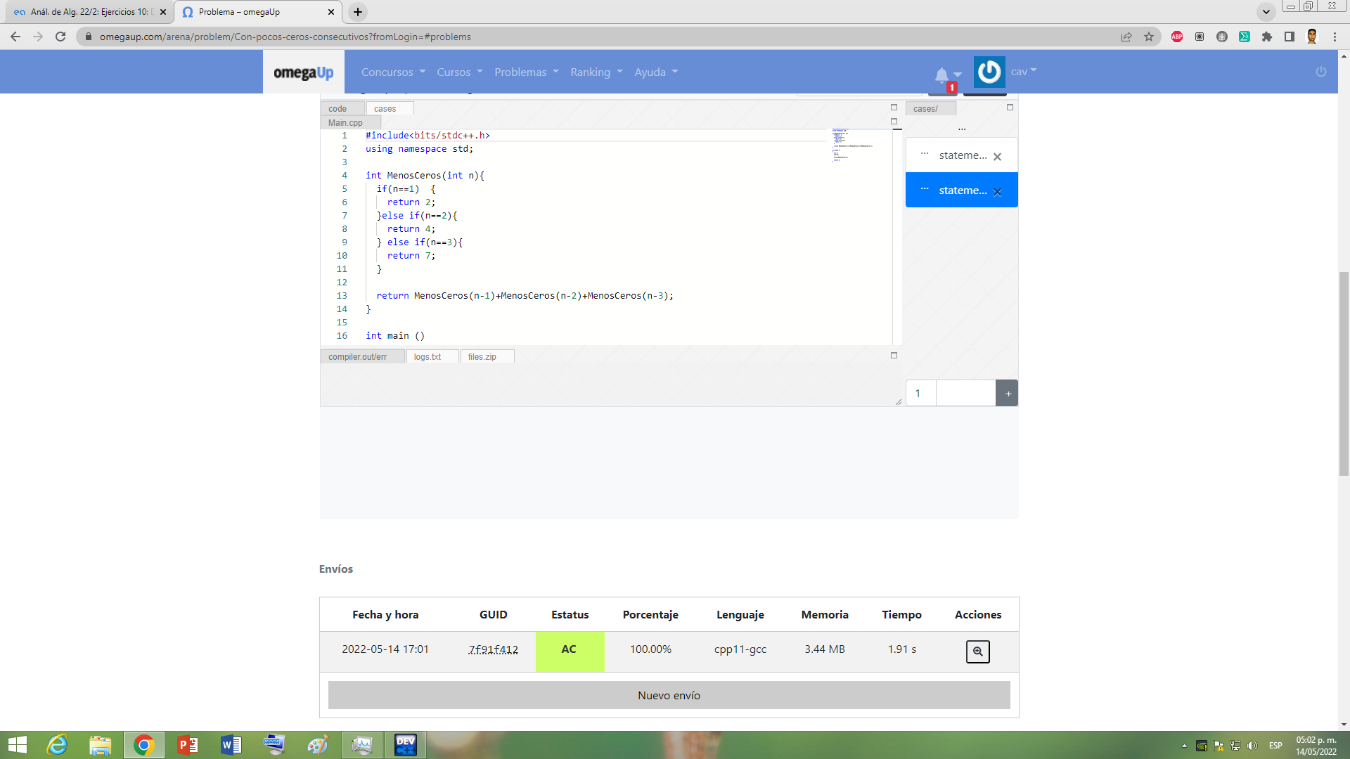
  cin>>n;

  cout<<MenosCeros(n);

  return 0;

}

## Captura



# **Problema 02: Amigos y Regalos**

## Descripción

Tienes dos amigos. A ambos quieres regalarles varios números enteros como obsequio. A tu primer amigo quieres regalarle C1 enteros y a tu segundo amigo quieres regalarle C2 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, tú no les 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, …, 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 C1, C2, X, Y. Se garantiza que X y Y son números primos.

Salida

Una línea. Un entero que representa la respuesta al problema. [4]

## Explicación

Intente resolverlo pero no lo entendí, investigue la solución entre los del club de algoritmia y vi que era más teoría de números que DyV, igual no entendí su código y por intentar resolver este y pensar que el de mega inversiones sería facial no resolví ni uno ni otro.

# **Referencias:**

[1] F. Filiberto. (2017, Mar 6). Divide and conquer 1. <https://omegaup.com/arena/problem/Divide-and-conquer-1>

[2] Paranoid Android. (2010, Jun 3). INVCNT - Inversion Count. <https://www.spoj.com/problems/INVCNT/>

[3] UAM Azcapotzalco. (2018, Feb 6). Con pocos ceros consecutivos. <https://omegaup.com/arena/problem/Con-pocos-ceros-consecutivos#problems>

[4] J. Luis. (2018, Abr 14). Amigos y Regalos. <https://omegaup.com/arena/problem/Amigos-y-Regalos/#problems>