Dibujo en blanco y negro

Descripción generada automáticamente con confianza baja**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**--------------------ANÁLISIS DE ALGORITMOS------------------**

**ACTIVIDAD**

Diseño de soluciones DyV

**PROFESOR:**

Franco Martínez Edgardo Adrián

**ALUMNO:**

Meza Vargas Brandon David – 2020630288

**GRUPO:**

3CM13

Joven con camiseta negra

Descripción generada automáticamente

**índice**

[Problema: Divide and Conquer 1 3](#_Toc87375799)

[Redacción 3](#_Toc87375800)

[Captura de aceptación por juez 3](#_Toc87375801)

[Explicación Algoritmo 4](#_Toc87375802)

[Análisis de complejidad en cota O() 5](#_Toc87375803)

[Código de solución completo 6](#_Toc87375804)

[Problema: INVCNT – Inversión Count 7](#_Toc87375805)

[Redacción 7](#_Toc87375806)

[Input 7](#_Toc87375807)

[Output 7](#_Toc87375808)

[Captura de aceptación por juez 7](#_Toc87375809)

[Explicación Algoritmo 7](#_Toc87375810)

[Análisis de complejidad en cota O() 9](#_Toc87375811)

[Código de solución completo 10](#_Toc87375812)

[Problema: Cumulo 12](#_Toc87375813)

[Redacción 12](#_Toc87375814)

[Captura de aceptación por juez 12](#_Toc87375815)

[Explicación Algoritmo 12](#_Toc87375816)

[Análisis de complejidad en cota O() 13](#_Toc87375817)

[Código de solución completo 14](#_Toc87375818)

[Problema: TRIPINV – Mega Inversions 15](#_Toc87375819)

[Redacción 15](#_Toc87375820)

[Captura de aceptación por juez 15](#_Toc87375821)

[Explicación Algoritmo 15](#_Toc87375822)

[Análisis de complejidad en cota O() 16](#_Toc87375823)

Antes de todo, hay que aclarar que en algunos problemas se usaron tipos de datos long, long long int, ya que se requieren para valores grandes y el juez lo acepte al 100%.

# **Problema: Divide and Conquer 1**

## **Redacción**

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**.

## **Captura de aceptación por juez**



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## **Explicación Algoritmo**

Texto

Descripción generada automáticamenteA continuación, se muestra el algoritmo empleado para resolver este problema.

Primeramente, establecemos la mitad del arreglo, para de eta forma partir el arreglo en dos y empezar a buscar las sumas de sub-arreglos por las mitades, las mismas que serán partidas hasta que no se pueda más.

Posteriormente, se hace la llamada recursiva para encontrar la suma por la parte de la derecha, que va de la mitad del arreglo hasta el primer elemento, y la suma de la parte izquierda va de un elemento después de la mitad hasta el último elemento del arreglo.

Ahora bien, expliquemos lo que sucede en la siguiente parte, solo se explica una vez ya que es lo mismo para derecha e izquierda.

Texto

Descripción generada automáticamente

Primeramente se iguala al lado izquierda el elemento que se encuentra a la mitad del arreglo, en el caso de la parte derecha se asigna el valor mitad+1. Después se hace el ciclo desde la mitad al inicio del arreglo, recorriendo así la parte izquierda. Posteriormente, se van acumulando las sumas de los elementos, si esta suma es mayor al elemento posicionado en la mitad del sub arreglo se establece como la suma mayor de este lado.

Lo mismo sucede con el lado derecho cambiando los límites.

Finalmente se suman las sumas de ambas mitades, siendo así la suma máxima el mayor elemento de la suma central, la suma de la derecha o la de la izquierda.

## **Análisis de complejidad en cota O()**

Para este problema tenemos que se parte en dos el problema cada vez que se manda a llamar a la función, esto lo hace dos veces, por la parte derecha y la izquierda, de esta manera tenemos.

T(n) = 2T(n/2)+n -🡪 n es la complejidad de la mezcla que se hace abajo.

Haciendo uso del teorema maestro tenemos:

f(n) = O(nlog22) 🡺 n = n

Por lo tanto la complejidad en cota O es:

**O(log(n)n)**

## Texto Descripción generada automáticamente**Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: INVCNT – Inversión Count**

## **Redacción**

Let A[0...n - 1] be an array of n distinct positive integers. If i < j and A[i] > A[j] then the pair (i, j) is called an inversion of A. Given n and an array A your task is to find the number of inversions of A.

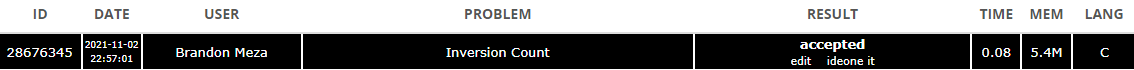
### **Input**

The first line contains t, the number of testcases followed by a blank space. Each of the t tests start with a number n (n <= 200000). Then n + 1 lines follow. In the ith line a number A[i - 1] is given (A[i - 1] <= 10^7). The (n + 1)th line is a blank space.

### **Output**

For every test output one line giving the number of inversions of A.

## **Captura de aceptación por juez**



## **Explicación Algoritmo**

A continuación se muestra el algoritmo empleado para este problema.

Texto

Descripción generada automáticamente

La función anterior, se realizará el conteo de inversiones siempre y cuando el elemento inicial sea menor al final, en otro caso el arreglo es de un elemento y no hay inversiones.

Se hacen llamadas recursivas para encontrar las inversiones por el lado de la derecha y por el lado izquierdo, posteriormente se hace la mezcla de estas.

Podemos observar que es un ordenamiento por mezcla, solo que en la función merge, que es nuestra conteoDivididas es donde contabilizaremos las inversiones.

Texto

Descripción generada automáticamente

Básicamente, en esta función se cuentan las inversiones al momento de que dos mitades del arreglo son mezcladas, se hace creando los índices del inicio que marcan los límites que tendrán al recorrerse. El índice i empieza desde la derecha y el j desde la mitad representando la izquierda.

Al momento en que el elemento i del arreglo es mayor al elemento j significa que las partes izquierdas y derechas del sub arreglo ya están ordenadas, estas serán q-i en total.

Finalmente se hace la mezcla de los elementos en el arreglo original.

## **Análisis de complejidad en cota O()**

Para este algoritmo se parte en dos el arreglo y así hasta el caso base, se hace dos veces, una para el lado izquierdo y otra para el derecho, de esta forma tenemos:

T(n) = 2T(n/2) + n -🡪 donde n es el costo de la función que hace la mezcla de las inversiones

Haciendo uso del teorema maestro tenemos:

f(n) = O(nlog22) 🡺 n = n

Por lo tanto la complejidad en cota O es:

**O(log(n)n)**

## Texto Descripción generada automáticamente**Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: Cumulo**

## **Redacción**

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?

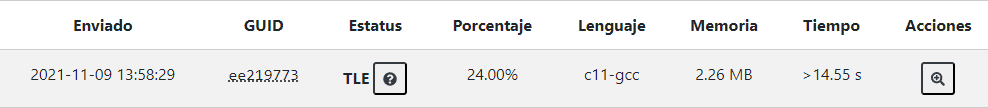
Entrada

La primera línea tendrá un entero 2<=n<=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<=X,Y<=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)

## **Captura de aceptación por juez**



## **Explicación Algoritmo**

A continuación se presenta el algoritmo empleado para esta solución

Texto

Descripción generada automáticamente

Podemos ver que al inicio hacemos la partición del problema en dos, haciendo la parte derecha y la parte izquierda para calcular las respectivas distancias en esos lados a partir de una llamada recursiva asignando los rangos correspondientes.

Posteriormente se hace el calculo de las distancias entre los puntos que hemos ingresado en los arreglos recorriendo los puntos desde el inicio a la mitad y los puntos con los que se calculara la distancia desde el final hasta la mitad+1. Una vez calculada la distancia se pregunta si la distancia resultante es mejor a la mínima (inicialmente 0) o si la mínima sigue siendo 0, si se cumple cualquier condición la nueva distancia mínima será la calculada previamente.

Posteriormente se hace la comparación entre las 3 distancias obtenidas, la del lado derecho, izquierdo y la mínima obtenida que sería la menor entre todos los puntos.

## **Análisis de complejidad en cota O()**

El algoritmo anterior primero se divide en dos partiendo el arreglo a la mitad recursivamente hasta el caso base, de esta forma tenemos:

T(n) = 2T(n/2) + n -🡪 donde n es el costo de recorrer el arreglo para encontrar la distancia mínima

Haciendo uso del teorema maestro tenemos:

f(n) = O(nlog22) 🡺 n = n

Por lo tanto la complejidad en cota O es:

**O(nlog(n))**

## Texto Descripción generada automáticamente**Código de solución completo**

Texto

Descripción generada automáticamente

# **Problema: TRIPINV – Mega Inversions**

## **Redacción**

The n^2 upper bound for any sorting algorithm is easy to obtain: just take two elements that are misplaced with respect to each other and swap them. Conrad conceived an algorithm that proceeds by taking not two, but three misplaced elements. That is, take three elements ai > aj > ak with i < j < k and place them in order ak; aj ; ai. Now if for the original algorithm the steps are bounded by the maximum number of inversions n(n-1)/2, Conrad is at his wits' end as to the upper bound for such triples in a given sequence. He asks you to write a program that counts the number of such triples.

Input

The first line of the input is the length of the sequence, 1 <= n <= 10^5.  
The next line contains the integer sequence a1; a2..an.  
You can assume that all ai belongs [1; n].

Output

Output the number of inverted triples.

## **Captura de aceptación por juez**



## **Explicación Algoritmo**

En el algoritmo se hace uso de un árbol de Fenwick, una estructura de datos que nos permite hacer sumas y actualizaciones de manera muy eficiente, pues cada operación vale O(lonN), pues usa operaciones con bits.

Texto

Descripción generada automáticamente

Primeramente, en la función de obtener suma se encarga de sumar los elementos del árbol de acuerdo al índice en el que estemos, posteriormente se va moviendo entre nodos para seguir con la suma hasta el índice indicado.

La función de actualizar toma un elemento del árbol y se va actualizado de acuerdo a un valor establecido.

## **Análisis de complejidad en cota O()**

Las operaciones del árbol son O(log(n)), pero al estar recorriendo todo el arreglo de tamaño n, tenemos una complejidad final de:

**O(nlog(n))**

**Código de solución completo**

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente