



Instituto Politécnico Nacional

Escuela Superior de Computo



Ejercicio 10

"Análisis de Algoritmos mediante la técnica Divide&Conquer"

Mora Ayala José Antonio

Análisis de Algoritmos



CONTENIDO

Instituto Politécnico Nacional.....	1
Escuela Superior de Computo.....	1
Ejercicio 10	1
Algoritmo numero 1.....	3
Divide and conquer 1.....	3
Planteamiento del problema:.....	3
Algoritmo numero 2.....	8
Cúmulo	8
Algoritmo numero 3.....	12
Inversion Count.....	12
Algoritmo numero 4.....	15
Amigos y Regalos.....	15
Análisis	17
ANEXOS.....	19
Código Divide y Venceras.....	19
Cumulo	20
Amigos y Regalos.....	23
Inversion Count.....	25



ALGORITMO NUMERO 1

DIVIDE AND CONQUER 1

PLANTEAMIENTO DEL PROBLEMA:

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.

Debemos tener en consideración las entradas y salidas esperadas y especificadas dentro de la página web:

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.

El funcionamiento del algoritmo de este código como tal no resulta muy complejo de comprender siempre y cuando se tengas conocimientos acerca de la recursividad y se tenga la suficiente noción de como es que funciona esta,

Como bien sabemos y tenemos conocimiento existen distintas formas de poder dar solución a este problema, una de las formas mas comunes sería hacerlo por una forma "greedy" o bien conocida como fuerza bruta mediante la implementación de dos ciclos 2 anidados, los cuales irían iterando sobre todo el arreglo y mediante el uso de condicionales ir modificando la suma máxima que vayamos adquiriendo, así como de la comparación con respecto a los diferentes elementos dentro del arreglo de números involucrados

En este caso se presenta una forma de resolverlo mediante el método de divide y vencerás, el cual consiste principalmente en la segmentación de un problema en otros mas pequeños mediante el empleo de la recursividad, donde tenemos un caso base que será en el cual nos indicara el final de esta recursividad, para que no se este realizando de una forma infinita.



Por lo que en este algoritmo realizamos la recursividad separando nuestro arreglo en 2 partes básicamente, primero se esta buscando del lado izquierdo del arreglo así como del lado derecho, es importante cuidar la forma en que se manejan los limites que se estan parametrizando en las llamadas a las funciones, pues podría existir un choque o una integración de un elemento que no debería ser considerado de alguno de los lados.

Cuando se encuentra en la mitad entre el arreglo izquierdo y el arreglo derecho es cuando se procede a realizar el calculo de la suma máxima de la mitad hacia cada uno de los extremos.

Nuestro caso base como ya mencionamos consiste en aquella situación en la cual el arreglo ya no puede ser partido en más.

```
Tareas - dv1.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int sumaMaximaSubDyV(int a[], int inicio, int final);
5  int max(int n1, int n2, int n3);
6  int main(int argc, char const *argv[])
7  {
8      int n, x, a;
9      scanf("%d", &n);
10     int numeros[n];
11     for (int j = 0; j < n; j++)
12     {
13         scanf("%d", &x);
14         numeros[j] = x;
15     }
16
17     a = sumaMaximaSubDyV(numeros, 0, n-1);
18     printf("%d",a);
19     return 0;
20 }
```

Como podemos observar se presenta el código del lado izquierdo para mas comodidad de redacción junto a seguimiento del código.

Observamos las cabeceras de las funciones que estamos próximos a implementar dentro del algoritmo

Tal como solicita ele ejercicio se necesitan de 3 variables para poder obtener los datos pertinentes y después se realiza un for para ir asignando dichos valores dentro del arreglo, la cantidad de valores está determinada por n

Posteriormente realizamos la llamada a la función mandando los parámetros pertinentes, los cuales son el arreglo, el inicio y el final

La parte recursiva se implementa dentro de la función de sumaMaximaSubDyV



Necesitamos de distintas variables con el propósito de poder ir almacenando los distintos valores que vamos obteniendo a lo largo de la ejecución de este algoritmo, comenzamos definiendo un caso base, en el cual terminará nuestra recursión y nos devolverá el primer elemento del arreglo, caso que sucede cuando el arreglo ya no puede seguir siendo sometido a particiones lógicas pertinentes

Realizamos la partición de nuestro arreglo y lo mandamos como parámetro nuevamente a la función recursiva, tal como se menciona anteriormente es importante tener cuidado con la forma en que se mandan estos parámetros, pues debemos considerar de manera correcta los límites de inicio y de final que tendrá en cada caso.

Al momento de que vamos regresando de la recursión (cuando ya hemos llegado a un caso base), *izq* y *der* ya tendrán un valor que puede ser sometido a evaluación, por lo cual la suma inicialmente será de 0, *sumaIzq* procede a ser igual a el elemento del arreglo en la posición media y realizamos el ciclo *for* en el cual vamos realizando la suma, en dado caso de que esta suma que estamos obteniendo sea mayor a la *sumaIzq*, *sumaIzq* va a adquirir este valor (realizamos un cambio), en caso contrario continua, misma situación se da con el lado derecho, por último realizamos la suma central, sumando los valores de la derecha y de la izquierda, y retornaremos aquel valor que sea mayor.

```
Tareas - dv1.c

1  int sumaMaximaSubDyV(int a[],int inicio , int final){
2      int mitad, izq, der,suma,i,sumaizq,sumader,sumacentro;
3      if (inicio == final)
4      {
5          return a[inicio];
6      }
7      mitad = (inicio + (final-1)) / 2;
8      izq = (sumaMaximaSubDyV(a, inicio, mitad));
9      der = (sumaMaximaSubDyV(a, mitad + 1, final));
10
11     suma = 0;
12     sumaizq = a[mitad];
13     for( i = mitad; i >= inicio; i--)
14     {
15         suma += a[i];
16         if (suma>sumaizq)
17             sumaizq = suma;
18     }
19     suma = 0;
20     sumader = a[mitad+1];
21     for( i = mitad+1; i <= final; i++)
22     {
23         suma += a[i];
24         if (suma>sumader)
25             sumader = suma;
26     }
27     sumacentro = sumaizq + sumader;
28     return max(izq,der,sumacentro);
29 }
```



```

Tareas - dv1.c
1  int max(int n1, int n2, int n3) {
2      if ( n1 >= n2 && n1 >= n3 )
3          return n1;
4      else
5          if (n2 > n3)
6              return n2;
7      else
8          return n3;
9  }

```

La función de max, simplemente recibe los 3 números y realiza operaciones lógicas junto a condiciones, para evaluar los valores, dependiendo cual sea mayor será el que sea retornado.

Nuestro caso base está determinado por $T(1) = 1$ con la ecuación de $T(n) = 2T\left(\frac{n}{2}\right) + n$

Representándolo de la forma en que lo hemos venido manejando nuestras ecuaciones quedarían de la siguiente manera:

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & n > 1 \end{cases}$$

Dado que estamos partiendo el tamaño del problema inicial en dos subproblemas, uno ira por la izquierda y otro por la derecha en búsqueda de la suma correspondiente, por lo tanto el costo de cada uno de estos será $T\left(\frac{n}{2}\right)$ y por último tenemos que hacer la consideración de que nos encontramos también con la situación en que ambos resultados deberán ser combinados, por lo cual tenemos una complejidad de rango n


Entonces podemos realizar uso del teorema maestro para poder tener una cota de este algoritmo teniendo en consideración los siguientes datos que ya hemos obtenido:

$$a = 2, b = 2, f(n) = n$$


$$n^{\log_2 2} = n^1 = n$$

Por lo cual tenemos la siguiente cota: $T(n) = \theta(n \log n)$

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-09 17:30:14	986d61f9	AC	100.00%	c11-gcc	4.08 MB	0.31 s	




 antonioayalam2001 ▾

1

+

Envíos

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-09 17:30:17	986d61f9	AC	100.00%	c11-gcc	4.08 MB	0.31 s	



ALGORITMO NUMERO 2

CÚMULO

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)

Entrada	Salida	Descripción
3 0.0 0.0 2.0 0.0 3.0 3.0	2.000	Las estrellas más cercanas son las primeras dos.
5 10.8 142.5 20.2 14.05 112.4 0.2 24.2 17.05 0.0 512.3	5.000	Las estrellas más cercanas son la segunda y la cuarta.

Posterior a revisar el planteamiento del problema en cuestión es momento de proceder a la realización e implementación del código que proporcionara solución a la misma, teniendo en consideración las condiciones iniciales. Nuevamente se presenta la imagen del código en uno de los lados para poder ir dando el seguimiento pertinente conforme la redacción de este documento



Mora Ayala José Antonio
Análisis de Algoritmos

Dada la naturaleza del problema planteado, comenzamos estableciendo una de las restricciones mediante una variable global, la cual esta definida con uno de los limites, posteriormente procedemos a la declaración de una estructura que lleva como nombre Point

(haciendo referencia a que es uno de los puntos), el cual posee las coordenadas que le serán introducidas, las cuales se almacenan en dos variables de tipo flotante (pues como observamos la forma en que se puede meter una coordenada también puede llevar un punto decimal), para hacer mas facil la creación y referencia de las estructuras nos auxiliamos del *typedef*,

En nuestro main agregamos otra de las condiciones donde **n** debe ser al menor mayor o igual que 2, pues tenemos que tener dos estrellas que comparar, en caso de que no se cumple simplemente terminaremos la ejecución del programa, en caso de que haya mas estrellas procedemos a la asignación de valores en cada punto

Tendremos una n cantidad de puntos, por lo cual es necesario realizar un ciclo para la asignación de cada valor en los distintos puntos (estrellas) creadas por el usuario.

Ahora procedemos a realizar la búsqueda de la mínima distancia, mandando el arreglo de puntos y la cantidad de los mismos

```
Tareas - Cumulo.c
1  #include <stdio.h>
2  #include <float.h>
3  #include <stdlib.h>
4  #include <math.h>
5  double MinimaDistancia = 1e10; // Mínima distancia.
6
7  // A structure to represent a Point in 2D plane
8  typedef struct Point
9  {
10     float x, y;
11 } Point;
```

```
Tareas - Cumulo.c
1  int main()
2  {
3     int n;
4     double x, y;
5     scanf("%d", &n);
6     Point P[n];
7     int aux;
8     if (n >= 2 && n <= 50000)
9     {
10         for (int j = 0; j < n; j++)
11         {
12             scanf("%lf %lf", &x, &y);
13             P[j].x = x;
14             P[j].y = y;
15         }
16         Busca(P, n); //Busca la minima distancia
17         printf("%.3lf\n", MinimaDistancia);
18     }
19     else{
20         return 0;
21     }
22
23     return 0;
24 }
```



```

Tareas - Cumulo.c
1 void Busca(struct Point *p, int n)
2 {
3     double d;
4     int i, j, a, b;
5     if (n <= 1) // Si no hay pares de puntos:
6         return; // salir.
7

```

```

Tareas - Cumulo.c
1     qsort(p, n, sizeof(struct Point), compareX); // Ordenar los puntos por la co
ordenada x.
2
3     Busca(p, n / 2); // Mirar en el subconjunto de la izquierda.
4
5     Busca(p + n / 2, (n + 1) / 2); // Mirar en el subconjunto de la derecha.
6
7     //Busca del lado derecho e izquierdo los puntos mas cercanos
8     for (i = n / 2; (i > 0) && (p[n / 2].x - p[i].x < MinimaDistancia); i--)
9         ; // Hallar los límites del conjunto central.
10
11     for (j = n / 2; j < n - 1 && p[j].x - p[n / 2].x < MinimaDistancia; j++)
12         ;
13
14     // Búsqueda en el conjunto central.
15     for (a = i; a <= j; a++)
16         for (b = a + 1; b <= j; b++)
17             if ((d = dist(p[a], p[b])) < MinimaDistancia) //Si encuentras una dista
ncia mínima actualízala
18                 {
19                     MinimaDistancia = d;
20                 }
21 }
22

```

```

Tareas - Cumulo.c
1 // La función de comparación tiene tres valores de retorno, 1, 0 y -1. Pero d
ebe dividirse en tres tipos: <0, > 0, 0. El valor de retorno de la función es de
tipo int, y ambos parámetros son const void * (const es de solo lectura, es
decir, no se puede modificar. Void puede ser de cualquier tipo, y a y b se n
ombran arbitrariamente). El nombre de la función cmp se puede personaliza
r con cualquier nombre, pero el nombre pasado en qsort () debe ser el que
desea definir.
2 int compareX(const void *a, const void *b)
3 {
4     return (((*(Point *)a).x < (*(Point *)b).x) ? -1 : 1); //Manera en que se comp
aran los puntos x
5 }
6
7 float dist(Point p1, Point p2)
8 {
9     return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
10 }
11

```

Presentamos la función que realiza la búsqueda de la mínima distancia, la cual usaremos para realizar el divide & Conquer pertinente, vemos como aquí sucede el **caso base**, en caso de que no haya mas pares de puntos con los cuales realizar la comparación (cuando el problema ya no pueda ser dividido en mas partes)

En caso contrario haremos uso de la función del quickSort que viene en la librería de stdio.h, la cual como se puede ver en la declaración, esta función no tiene valor de retorno y necesita cuatro parámetros. El cuarto parámetro es el puntero de función

1. base: un puntero al primer elemento de la matriz que se va a ordenar.

2. nitems: el número de elementos en la matriz a los que apunta la base.

3. size - el tamaño de cada elemento en la matriz, en bytes.

4. compareX - una función utilizada para comparar dos elementos.

La cual tiene como principal propósito realizar el ordenamiento de los elementos con respecto a la coordenada x, de igual forma podemos ver como estamos realizando la recursividad para poder analizar los elementos de lado derecho y de lado izquierdo lo cual nos proporciona un $T\left(\frac{n}{2}\right)$ cuando se realiza la llamada de la función nuevamente, pues se esta realizando la partición, posteriormente la búsqueda realizada en la parte central al momento que va



volviendo la función, que es el momento en el cual se actualiza

Debemos tener en consideración la complejidad temporal que necesita el algoritmo de ordenamiento utilizado para la parte de los puntos de la estructura, suponiendo que tengamos uno de los casos medios constantes y no nos enfrentemos a el peor caso que proporción una complejidad temporal de n^2 pero quedémonos con la consideración de que obtengamos un t de $n \log(n)$ dado esto tendremos la siguiente formula: [1] [2]

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) + O(n) + O(n \log(n))$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n \log(n))$$

$$T(n) = T(n * \log(n) * \log(n))$$

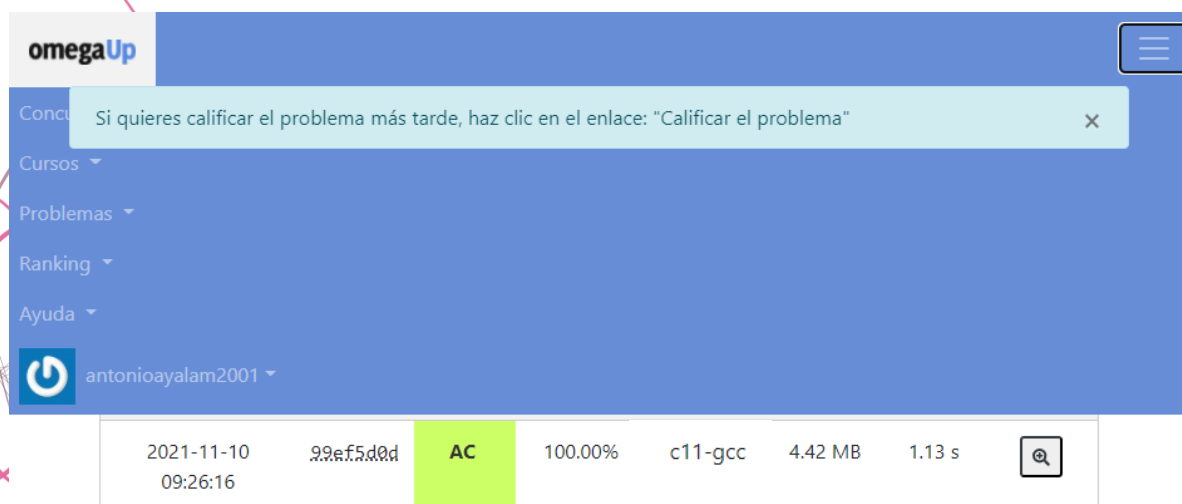
$$T(n) = O(n(\log(n)^2))$$

Realizando una recapitulación de lo que cada elemento significa tenemos lo siguiente:

- $T\left(\frac{n}{2}\right)$ *Busqueda por la derecha*
- $T\left(\frac{n}{2}\right)$ *Busqueda por la izquierda*
- $n \log(n)$ *Por parte del metodo de ordenamiento necesario*
- $O(n)$ *Por la busqueda de la distancia entre los puntos*
- $O(n)$ *Por la busqueda de la distancia entre los puntos*

Para poder llegar a dicho resultado se siguió el siguiente procedimiento:

$$n = 2^m \rightarrow \log_2 n = m \rightarrow \log_2 \left(\frac{n}{2}\right) = m - 1$$



ALGORITMO NUMERO 3

INVERSION COUNT

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 ($n \leq 200000$). Then $n + 1$ lines follow. In the i th line a number $A[i - 1]$ is given ($A[i - 1] \leq 10^7$). The $(n + 1)$ th line is a blank space.

Output

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

```
Tareas - Inversion.c
1  int mergeSort(int inicio, int fin){
2      int mid = (inicio + fin) / 2;
3      int cnt = 0;
4      if (inicio < fin)
5      {
6          cnt += mergeSort(inicio, mid);
7          cnt += mergeSort(mid + 1, fin);
8          cnt += merge(inicio, mid+1, fin);
9      }
10     return cnt;
11 }
12
13 int main(){
14     // t : Numero de pruebas que se van a realizar
15     // n : Variable para la insercion de numeros
16     // i : Variable que maneja el ciclo dentro del while
17     int i, n, t;
18     scanf("%d", &t);
19     while (t--){
20         scanf("%d", &n);
21         for (i = 1; i <= n; i++){
22             scanf("%d", &a[i]);
23         }
24         printf("%lld\n", mergeSort(0, n));
25     }
26     return 0;
27 }
28 }
```

Como Podemos observar el funcionamiento de la cuenta de inversiones que se deben realizar hace uso del funcionamiento del algoritmo que ya hemos analizado en trabajos anteriores y que ya fue implementado en su momento, el cual es el algoritmo de ordenamiento por el método de mergesort, el cual como ya sabemos se encarga de realizar particiones del arreglo original e ir realizando las comparaciones pertinentes y realizando el ordenamiento de los elementos que componen el arreglo, como podemos observar, en esta ocasión lo que nos interesa como tal es saber la cantidad de cambios que deberán ser realizados, por lo que estamos dando un valor de retorno y este es asignado y actualizado conforme se realiza la llamada recursiva

Aquí podemos realizar un pequeño "truco" para poder solucionar este problema y proporcionar la salida deseada



```

1  #include <stdio.h>
2
3  #define sz 500000
4  #define inf 1000000000
5  int a[sz + 2], l[sz + 2];
6
7  int merge(int inicio, int mid, int fin){
8      int cnt = 0;
9      int left = inicio;
10     int index = inicio, center = mid;
11     while ((left <= mid-1) && (center <= fin))
12     {
13         if(a[left] <= a[center])
14             l[index++] = a[left++];
15         else{
16             l[index++] = a[center++];
17             cnt += (mid - left);
18         }
19     }
20     while (left <= mid-1){
21         l[index++] = a[left++];
22     }
23     while (center <= fin){
24         l[index++] = a[center++];
25     }
26     for (int i = inicio; i <= fin; i++){
27         a[i] = l[i];
28     }
29     return cnt;
30 }

```

como ya sabemos merge sort cada sub arreglo que retorna cada recursión estará ordenado por tanto podemos comparar los elementos del subarreglo izquierdo, con los del lado derecho, eso se realiza en esta parte del código:

```

1  while ((left <= mid-1) && (center <= fin))
2  {
3      if(a[left] <= a[center])
4          l[index++] = a[left++];
5      else{
6          l[index++] = a[center++];
7          cnt += (mid - left);
8      }
9  }

```

Dado que cada subarreglo ya se encuentra ordenado de forma descendente, sabemos que si el i-ésimo elemento del arreglo si es mayor que el j-ésimo elemento del sub arreglo derecho entonces los n elementos del sub arreglo izquierdo serán más grandes que el j-ésimo elemento con el que se está haciendo la comparación, por tanto nuestra respuesta sumará la posición inicial del subarreglo derecho, menos la posición en la que se encuentra el subarreglo izquierdo. De

tal forma que cuando se de esta situación realizaremos el incremento de nuestro contador. Como podemos observar nuevamente nos presentamos a un algoritmo que realiza particiones y básicamente se basa en el funcionamiento del algoritmo de ordenación que ya conocemos tenemos entonces:

- $T(n) = T\left(\frac{n}{2}\right)$ Búsqueda por la izquierda
- $T(n) = T\left(\frac{n}{2}\right)$ Búsqueda por la derecha
- $T(n) = n$ Momento en que realiza la union

Por lo que tenemos como resultado lo siguiente

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T\left(\frac{n}{2}\right) + n, & n > 1 \end{cases}$$

$$a = 2, b = 2 \quad f(n) = n$$

$$n^{\log_2 2} = n^1 = n \quad \text{Por lo cual tenemos la siguiente cota: } T(n) = \theta(n \log n)$$



ID	DATE	USER	PROBLEM	RESULT	TIME	MEM	LANG
28711675	2021-11-10 02:27:54	AntonioAyala01	Inversion Count	accepted edit ideone it	0.08	10M	C
28711672	2021-11-10 02:27:11	Byakko	Coconuts	time limit exceeded	-	6.3M	C++ 4.3.2
28711668	2021-11-10 02:25:10	vas14	Sometimes, a penalty is good!	wrong answer	0.01	5.4M	CPP14
28711659	2021-11-10 02:22:57	Lama_Tarek	Number of common divisors	accepted	0.38	5.3M	CPP14
28711655	2021-11-10 02:21:25	vas14	Sometimes, a penalty is good!	wrong answer	0.01	5.4M	CPP14



ALGORITMO NUMERO 4

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, 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 $C1, C2, X, Y$. Se garantiza que X y Y son números primos.

Salida

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



```

1  #include <stdio.h>
2
3  int buscar(int p, int r);
4  int posibilidad(int mitad);
5  int Max(int primero, int segundo);
6
7  // declaración de las variables globales para el manejo de los datos
8  int c1,c2,x,y;
9
10 int main()
11 {
12     // Escaneado de datos por teclado
13     // Numero de regalos posibles para el primer amigo
14     scanf("%d",&c1);
15     // Numero de regalos posibles para el segundo amigo
16     scanf("%d",&c2);
17     // Numero primo que no le gusta el primer amigo
18     scanf("%d",&x);
19     // Numero primo que no le gusta el segundo amigo
20     scanf("%d",&y);
21     printf("%d",buscar(c1+c2,(c1+c2)*2));
22     return 0;
23 }
24

```

Tal como solicita el programa asignamos los valores pertinentes como el requerido para poder proceder a realizar la ejecución del algoritmo, así que procedemos a la llamada de la función que se encargara de realizar la búsqueda, mandamos como parámetros la suma de los arreglos, así como el doble de la cantidad de regalos que queremos dar.

Una vez dentro de la función obtendremos el numero de regalos totales mas la mitad de este total de regalos, de tal forma que obtenemos el valor máximo de números que pueden ser regalados, ahora lo que resta es someter a una evaluación para saber si dicho numero es el mínimo conjunto de valores que podemos tener para dar regalos, es

por eso que procedemos a realizar la ejecución de la función de la posibilidad, para saber si en efecto es o no, inicialmente lo mandamos a llamar mediante el parámetro obtenido inicialmente. Lo siguiente que haremos es obtener un valor n que es el numero total de regalos que quieren nuestros amigos entre la multiplicación de números primos, esto para saber si estamos en el arreglo máximo de números posibles, posteriormente procedemos a obtener el número de números dentro de nuestro conjunto de números que son divisibles tanto por x tanto por y y esto lo obtenemos mediante dividir el numero de conjunto entre el numero primo

```

1  int buscar(int p, int r){
2      // realizamos la búsqueda del numero de regalos totales mas la mitad de dicho regalos totales
3      int mitad=(p+(r-p)/2);
4      // Comprobacion de la Secuencia que llevamos
5      if(posibilidad(mitad)){
6          // Si la secuencia anterior fue correcta pasaremos a realizar una comprobacion de menor tamaño
7          if(posibilidad(mitad-1)) {
8              // Dado que las dos condiciones anteriores han sido cumplidas podemos realizar nuevamente la llamada
9              // al algoritmo para ver si hay otras secuencia que satisfaga
10             return buscar(p,mitad-1);
11         }
12         // En caso de que una secuencia menor no sea cumplida pasamos la secuencia ya obtenida como
13         // el resultado obtenido
14         return mitad;
15     }
16     else{
17         // En caso de que nuestra secuencia menor no haya sido satisfactoria realizamos la comprobacion de una
18         // secuencia mayor
19         return buscar(mitad+1,r);
20     }
21 }

```

que no les gusta a nuestros amigos menos nuestro valor de n , una vez que ya tenemos estos valores ahora procederemos a obtener los que nos permite comprobar si el arreglo es o no valido, en primera instancia este arreglo será valida si es la primera vez pero conforme vamos avanzando estos arreglos ya no lo serán, como estamos comenzando en el caso donde tenemos el mayor numero de




```

1 int posibilidad(int mitad){
2     // Realizamos la obtencion de regalos totales, entre la multiplicacion de los numeros primos
3     int n=mitad/(x*y);
4     // Numero de elementos primos divisibles entre x
5     int primx=(mitad/x)-n;
6     // Numero de elementos primos divisibles entre y
7     int primy=(mitad/y)-n;
8     // Obtenemos el numero mas grande entre la resta de regalos para C2 menos el numero de enteros
9     // divisibles entre y
10    int auxX=Max(c1-primy,0);
11    // Obtenemos el numero mas grande entre la resta de regalos para C1 menos el numero de enteros
12    // divisibles entre x
13    int auxY=Max(c2-primx,0);
14
15    // Retornamos la comprobacion de si el numer total de regalos menos la cantidad de numeros
16    // primos que no se requieren es mayor a la suma de los numeros mas grandes entre
17    // los divisibles y 0
18    return(mitad-primx-primy-n>=auxX+auxY);
19
20    // En caso de que haya una igualdad podemos restar el numero del conjunto, si son diferentes
21    // tendremos que sumar o regresar a un conjunto predecesor
22 }
23
24 int Max(int primero, int segundo){
25     if(primero>segundo) return primero;
26     return segundo;
27 }

```

numero en nuestro conjunto posible, lo que haremos será ir restando en 1 este valor, una vez que este valor restado ya no sea válido, regresaremos de nuevo al numero anterior mayor y mandaremos ese numero como resultado, esta comprobación se basa en si el número de números en nuestro conjunto menos el número de números que nuestros amigos que no quieren, menos el numero n es mayor o igual que el valor de regalos que quiero menos el valor de regalos que el otro no quiere, si esta comprobación es igual quiere decir que el conjunto es válido, si no lo es el conjunto no lo será.

ANALISIS

Consideremos la siguiente función en términos de recursión

$$T(n) = T(n - 1) + 2$$

Donde el caso base como tal no está fijo o explicito, este va variando respecto a la cantidad de números que le demos a cada amigo, pero como tal no es posible realizar una función que de una generalidad para todos los casos a los cuales pueda ser sometido este algoritmo.

Pero dado que se nos proporciona que el numero mínimo de regalos que podemos proporcionar a cada uno de nuestros amigos es de 1 y si debemos tener siempre garantizados 2 numeros primos diferentes, podríamos considerar de base el 2 y el 3

Por lo que nuestro caso quedaría como

$$T(0)=0, T(1)=0, T(2)=4$$

$$T(n) = \begin{cases} 0, & n < 2 \\ T(n - 1) + 2, & n \geq 2 \end{cases}$$

$$b^n = 1^n \quad p(n) = 2 \quad d = 0$$

$$(x - 1)(x - 1)$$

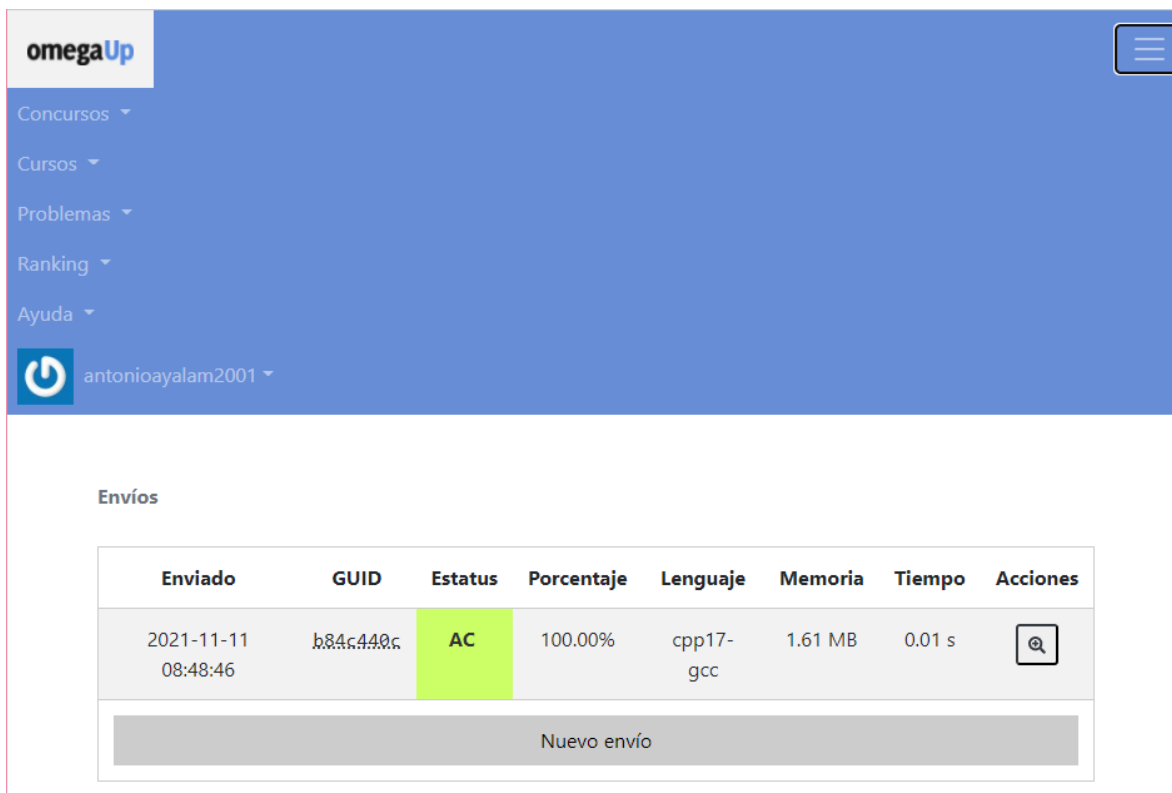
$$T(n) = C_1 + C_2 n$$

$$T(3) = T(2) + 2 = 4$$


$$T(2) = C_1 + C_2(2) = 2$$



$$T(3) = C_1 + C_2(3) = 4$$
$$T(0) = -2 + 2n \rightarrow \in O(n)$$



The screenshot shows the omegaUp website interface. The top navigation bar is blue with the omegaUp logo on the left and a menu icon on the right. Below the navigation bar, there is a sidebar with links to Concursos, Cursos, Problemas, Ranking, and Ayuda. The main content area displays the user's profile (antonioayalam2001) and a table of submissions under the heading "Envíos".

Enviado	GUID	Estatus	Porcentaje	Lenguaje	Memoria	Tiempo	Acciones
2021-11-11 08:48:46	b84c440c- b84c440c	AC	100.00%	cpp17- gcc	1.61 MB	0.01 s	
Nuevo envío							



ANEXOS

CÓDIGO DIVIDE Y VENCERAS

```
#include <stdio.h>
#include <stdlib.h>

int sumaMaximaSubDyV(int a[], int inicio, int final);
int max(int n1, int n2, int n3);
int main(int argc, char const *argv[])
{
    int n, x, a;
    scanf("%d", &n);
    int numeros[n];
    for (int j = 0; j < n; j++)
    {
        scanf("%d", &x);
        numeros[j] = x;
    }

    a = sumaMaximaSubDyV(numeros, 0, n-1);
    printf("%d",a);
    return 0;
}

int sumaMaximaSubDyV(int a[],int inicio , int final){
    int mitad, izq, der,suma,i,sumaizq,sumader,sumacentro;
    if (inicio == final)
    {
        return a[inicio];
    }
    mitad = (inicio + (final-1)) / 2;
    izq = (sumaMaximaSubDyV(a, inicio, mitad));
    der = (sumaMaximaSubDyV(a, mitad + 1, final));

    suma = 0;
    sumaizq = a[mitad];
    for( i = mitad; i >= inicio; i--)
    {
        suma += a[i];
        if (suma>sumaizq)
            sumaizq = suma;
    }
}
```



```
    suma = 0;
    sumader = a[mitad+1];
    for( i = mitad+1; i <= final; i++)
    {
        suma += a[i];
        if (suma>sumader)
            sumader = suma;
    }
    sumacentro = sumaizq + sumader;
    return max(izq,der,sumacentro);
}

int max(int n1, int n2,int n3) {
    if ( n1 >= n2 && n1 >= n3 )
        return n1;
    else
        if (n2 > n3)
            return n2;
        else
            return n3;
}
```

CUMULO

```
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <math.h>
double MinimaDistancia = 1e10; // Mínima distancia.

// A structure to represent a Point in 2D plane
typedef struct Point
{
    float x, y;
} Point;

// La función de comparación tiene tres valores de retorno, 1, 0 y -1. Pero debe
dividirse en tres tipos: <0,> 0, 0. El valor de retorno de la función es de tipo int, y
ambos parámetros son const void * (const es de solo lectura, es decir, no se puede
modificar. Void puede ser de cualquier tipo, y a y b se nombran arbitrariamente). El
```



nombre de la función cmp se puede personalizar con cualquier nombre, pero el nombre pasado en qsort () debe ser el que desea definir.

```
int compareX(const void *a, const void *b)
{
    return (((*(Point *)a).x < (*(Point *)b).x) ? -1 : 1); //Manera en que se comparan los puntos x
}

float dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
}

// A Brute Force method to return the smallest distance between two points
// // in P[] of size n
// float bruteForce(Point P[], int n)
// {
//     float min = FLT_MAX;
//     for (int i = 0; i < n; ++i)
//         for (int j = i + 1; j < n; ++j)
//             if (dist(P[i], P[j]) < min)
//                 min = dist(P[i], P[j]);
//     return min;
// }

void Busca(struct Point *p, int n)
{
    double d;
    int i, j, a, b;
    if (n <= 1) // Si no hay pares de puntos:
        return; // salir.

    // Parámetro
    // Como se puede ver en la declaración, esta función no tiene valor de retorno y necesita cuatro parámetros. El cuarto parámetro es el puntero de función, y el concepto del puntero de función no se explicará aquí. Aquellos que estén interesados deben entenderlo.

    // 1. base: un puntero al primer elemento de la matriz que se va a ordenar.
    // 2. nitems: el número de elementos en la matriz a los que apunta la base.
    // 3. size - el tamaño de cada elemento en la matriz, en bytes.
    // 4. compar - una función utilizada para comparar dos elementos.
```



```
qsort(p, n, sizeof(struct Point), compareX); // Ordenar los puntos por la coordenada
x.

Busca(p, n / 2); // Mirar en el subconjunto de la izquierda.

Busca(p + n / 2, (n + 1) / 2); // Mirar en el subconjunto de la derecha.

//Busca del lado derecho e izquierdo los puntos mas cercanos
for (i = n / 2; (i > 0) && (p[n / 2].x - p[i].x < MinimaDistancia); i--)
    ; // Hallar los límites del conjunto central.

for (j = n / 2; j < n - 1 && p[j].x - p[n / 2].x < MinimaDistancia; j++)
    ;

// Búsqueda en el conjunto central.
for (a = i; a <= j; a++)
    for (b = a + 1; b <= j; b++)
        if ((d = dist(p[a], p[b])) < MinimaDistancia) //Si encuentras una distancia
mínima actualizala
        {
            MinimaDistancia = d;
        }
}

int main()
{
    int n;
    double x, y;
    scanf("%d", &n);
    Point P[n];
    int aux;
    if (n>=2 && n<=50000)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%lf %lf", &x,&y);
            P[j].x = x;
            P[j].y = y;
        }
        Busca(P, n); //Busca la minima distancia
        printf("%.3lf\n", MinimaDistancia);
    }
}
```



```
}else{  
    return 0;  
}  
  
return 0;  
}
```

AMIGOS Y REGALOS

```
#include <stdio.h>  
  
int buscar(int p, int r);  
int posibilidad(int mitad);  
int Max(int primero, int segundo);  
  
// declaración de las variables globales para el manejo de los datos  
int c1,c2,x,y;  
  
int main()  
{  
    // Escaneado de datos por teclado  
    // Numero de regalos posibles para el primer amigo  
    scanf("%d",&c1);  
    // Numero de regalos posibles para el segundo amigo  
    scanf("%d",&c2);  
    // Numero primo que no le gusta el primer amigo  
    scanf("%d",&x);  
    // Numero primo que no le gusta el segundo amigo  
    scanf("%d",&y);  
    printf("%d",buscar(c1+c2,(c1+c2)*2));  
    return 0;  
}  
  
int buscar(int p, int r){  
    // realizamos la busqueda del numero de regalos totales mas la mitad de dicho  
    // regalos totales  
    int mitad=p+(r-p)/2;  
    // Comprobacion de la Secuencia que llevamos  
    if(posibilidad(mitad)){
```



```
// Si la secuencia anterior fue correcta pasaremos a realizar una comprobacion
de menor tamaño
if(posibilidad(mitad-1)) {
    // Dado que las dos condiciones anteriores han sido cumplidas podemos
    realizar nuevamente la llamada
    // al algoritmo para ver si hay otra secuencia que satisfaga
    return buscar(p,mitad-1);
}
// En caso de que una secuencia menor no sea cumplida pasamos la secuencia
ya obtenida como
// el resultado obtenido
return mitad;
}
else{
    // En caso de que nuestra secuencia menor no haya sido satisfactoria
    realizamos la comprobacion de una
    // secuencia mayor
    return buscar(mitad+1,r);
}
}
int posibilidad(int mitad){
    // Realizamos la obtencion de regalos totales, entre la multiplicacion de los
    numeros primos
    int n=mitad/(x*y);
    // Numero de elementos primos divisibles entre x
    int primx=(mitad/x)-n;
    // Numero de elementos primos divisibles entre y
    int primy=(mitad/y)-n;
    // Obtenemos el numero mas grande entre la resta de regalos para C2 menos el
    numero de enteros
    // divisibles entre y
    int auxX=Max(c1-primy,0);
    // Obtenemos el numero mas grande entre la resta de regalos para C1 menos el
    numero de enteros
    // divisibles entre x
    int auxY=Max(c2-primx,0);

    // Retornamos la comprobacion de si el numero total de regalos menos la cantidad de
    numeros
    // primos que no se requieren es mayor a la suma de los numeros mas grandes entre
    // los divisibles y 0
    return(mitad-primx-primy-n>=auxX+yf);

    // En caso de que haya una igualdad podemos restar el numero del conjunto, si son
    diferentes
    // tendremos que sumar o regresar a un conjunto predecesor
}
int Max(int primero, int segundo){
    if(primero>segundo) return primero;
    return segundo;
}
```



INVERSION COUNT

```
#include <stdio.h>

#define sz 500000
#define inf 1000000000

long a[sz + 2], left[sz + 2], right[sz + 2];
long long cnt;

void merge(long inicio, long mid, long fin)
{
    long i, j, k, ind1, ind2;

    for (i = inicio, ind1 = 1; i <= mid; i++)
    {
        left[ind1++] = a[i];
    }
    left[ind1] = inf;

    for (i = mid + 1, ind2 = 1; i <= fin; i++)
    {
        right[ind2++] = a[i];
    }
    right[ind2] = inf;

    i = j = 1;

    for (k = inicio; k <= fin; k++)
    {
        if (left[i] > right[j])
        {
            cnt += ind1 - i;
            a[k] = right[j];
            j++;
        }
        else
        {
            a[k] = left[i];
            i++;
        }
    }
}
```



```
void mergeSort(long inicio, long fin)
{
    if (inicio < fin)
    {
        long mid = (inicio + fin) / 2;
        mergeSort(inicio, mid);
        mergeSort(mid + 1, fin);
        merge(inicio, mid, fin);
    }
}

int main()
{
    // t : Numero de pruebas que se van a realizar
    // n : Variable para la insercion de numeros
    // i : Variable que maneja el ciclo dentro del while
    long i, n, t;

    scanf("%ld", &t);
    while (t--)
    {
        scanf("%ld", &n);
        for (i = 1; i <= n; i++)
        {
            scanf("%ld", &a[i]);
        }
        cnt = 0;
        mergeSort(1, n);
        printf("%lld\n", cnt);
    }

    return 0;
}
```



BIBLIOGRAFÍA

BIBLIOGRAFÍA

- [1] University of Toronto, «University of Toronto,» - - 2020. [En línea]. Available: http://www.cs.toronto.edu/~bor/373s20/docs/SR_Proof_recurssion_closestPoints.pdf. [Último acceso: 11 Noviembre 2021].
- [2] JNTUA College of Engg, «JNTUA College,» - - 2017. [En línea]. Available: <https://www.jntua.ac.in/gate-online-classes/registration/downloads/material/a159237881555.pdf>. [Último acceso: 10 Noviembre 2021].
- [3] GeeksforGeeks, «GeeksforGeeks,» GeeksforGeeks, 7 Julio 2021. [En línea]. Available: <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/>. [Último acceso: 6 Noviembre 2021].
- [4] E. A. F. Martinez, «EaFranco.Eakdemy,» [En línea]. Available: <https://eafranco.eakdemy.com/mod/scorm/player.php>. [Último acceso: 07 Octubre 2021].
- [5] «OmegaUp,» [En línea]. Available: <https://omegaup.com/arena/problem/Amigos-y-Regalos/#problems>. [Último acceso: 07 Noviembre 2021].

