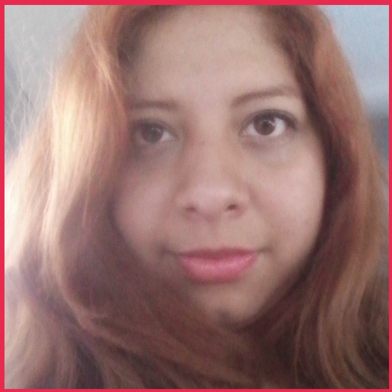
ESCOM-IPN

# Ejercicio 5
# Divide y Vencerás

ANÁLISIS DE ALGORITMOS

Laura Andrea Morales López

Mayo 2018



# Índice

# 1. Algoritmo 1: Divide and Conquer 1

| | | | |
|---|---|---|---|
| Points | 16.54 | Memory limit | 32MB |
| Time limit (case) | 1s | Time limit (total) | 60s |

## 1.1. 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.

## 1.2. Entrada

La primera línea contendrá un numero N.

En la siguiente línea N enteros representando el arreglo A.

## 1.3. Salida

La suma máxima en cualquier subarreglo contiguo.

## 1.4. Limites

$$1 <= n <= 10^5$$

$$|A_i| <= 10^9$$

## 1.5. Solución A: Fuerza Bruta

Evalúa los $O(n^2)$ subarreglos, siendo n la cantidad de elementos del arreglo A. Al invertir tiempo O(1) en cada uno de ellos, la complejidad temporal resulta ser $O(n^2)$.

## 1.6. Código

```
lli MaximaSumaFuerzaB(const vector<lli> &A)
{
    lli Maximo = A[0], Suma;
    lli n = A.size();

    for(lli i = 0; i < n; ++i)
    {
        Suma = lli();
        for(lli j = i; j < n; ++j)
        {
            Suma += A[j];
            if(Suma > Maximo)
                Maximo = Suma;
        }
    }

    return Maximo;
}
```

## 1.7. Solución B: Divide y Venceras

Divide el problema básicamente en 3 partes:

- Si la solución se encuentra del lado izquierdo del arreglo.

  Para este caso usaremos recursividad para obtener solo el arreglo máximo del lado izquierdo.

- Si la solución se encuentra del lado derecho del arreglo.

  Para este caso usaremos recursividad para obtener solo el arreglo máximo del lado derecho.

- Si se encuentra en la mitad entre el arreglo izquierdo y el arreglo derecho.

  Se realiza el calculo de suma máxima de la mitad hacia los extremos cada uno por separado y se suman.

El caso base es cuando ya no puedas partirte en más mitades.

Una vez evaluados estos casos obtenemos el máximo de los 3 y ese sera el resultado del arreglo.

## 1.8. Código

```
#include <algorithm>
#include <iostream>
#include <vector>
typedef long long int lli;
using namespace std;

// Función auxiliar de Maxima Suma: calcula la máxima suma de un subarreglo
// comenzando en la mitad de arreglo A y extendiéndose hacia el extremo.

lli MaximoSecuencial(const vector<lli> &A, lli inicio, lli j){
    lli Suma = A[inicio];                                    //El maximo es el inicio
    del subarreglo
    lli Maximo = Suma;
```

```

    if(inicio<j){                                              //Lado derecho del arreglo

        for(lli k = inicio+1; k < j ; k++){
            Suma += A[k];                                      //Suma el siguiente digito.
            if(Suma > Maximo)                                  //Si es mayor que el maximo que tenemos guardamos este
                Maximo = Suma;
        }
    }
    else{                                                      //Lado izquierdo del arreglo

        for(lli k = inicio-1; k > j ; k--){
            Suma += A[k];                                      //Suma el siguiente digito
            if(Suma > Maximo)                                  //Si la suma es mayor al maximo guardamos este valor
                Maximo = Suma;
        }

    }


    return Maximo;                                             //Regresa la suma maxima del segmento
}

//Parte el problema, obtiene la maxima suma del lado izquierdo, del lado derecho, asi como la suma de las sumas maximas
//de los subarreglos de izquierda y derecha de la mitad hacia los extremos.
//Obtiene el maximo de los 3
lli MaximaSuma(const vector<lli> &A, lli inicio=0, lli j=-1)
{
    if(j == -1)
        j = A.size();

    if(j - inicio == 1)
        return A[inicio];

    lli m = inicio + (j-inicio)/2;

    lli SumaIzquierda = MaximaSuma(A, inicio, m);
    lli SumaDerecha = MaximaSuma(A, m, j);
    lli t_izq = MaximoSecuencial(A, m-1, inicio-1);
    lli t_der = MaximoSecuencial(A, m, j);

    return max({SumaIzquierda, SumaDerecha, t_izq + t_der});           //Obtenemos el máximo en un array
}
int main()
{
    lli n=0;
    cin>>n;
    vector<lli> A;
    lli auxiliar;
    for(long long int i=0; i<n;i++)
    {
            cin>>auxiliar;    // Se lee el numero
            A.push_back(auxiliar);
            cin.ignore(1); // Se descarta el separador

    }



    printf("%lld\n",MaximaSuma(A) );

    return 0;
}
```

## 1.9. Análisis

El caso base es de:

$$T(1) = 1$$

Con la ecuación;

$$T(n) = 2T(\frac{n}{2}) + n + 1$$

Entonces usamos el teorema maestro para resolver la cota.

Con este teorema decimos que $a = 2 \ \ b = 2 \ \ f(n) = n$

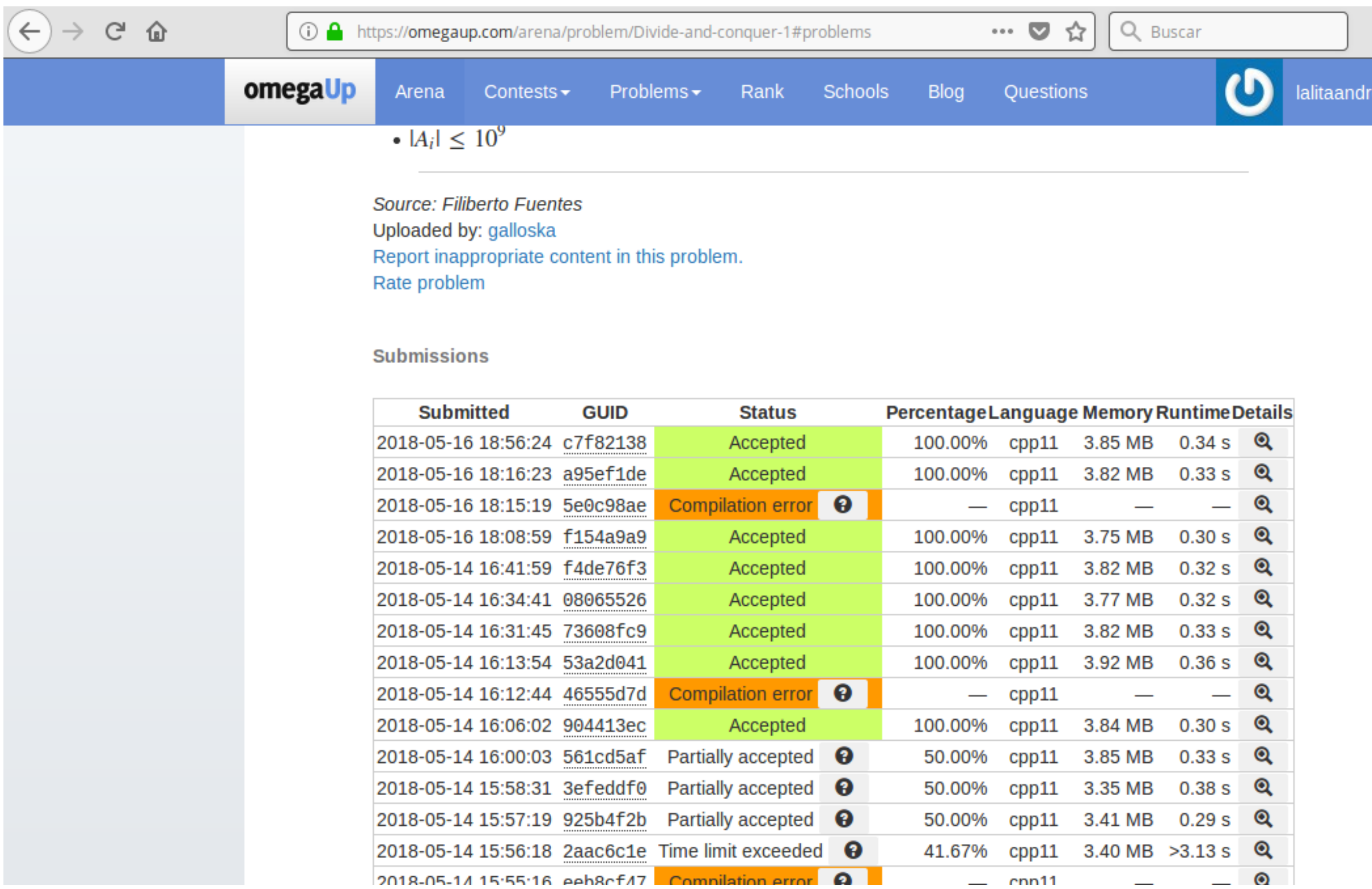
Entonces:

$n^{log_2 2}=(n^1) = n$

Y tenemos la cota:

$$T(n) = \theta(nlogn)$$

## 1.10. Resultados



https://omegaup.com/arena/problem/Divide-and-conquer-1#problems

omegaUp Arena Contests Problems Rank Schools Blog Questions lalitaandre

• $|A_i| \leq 10^9$

*Source: Filiberto Fuentes*

Uploaded by: galloska

Report inappropriate content in this problem.

Rate problem

**Submissions**

| Submitted | GUID | Status | Percentage | Language | Memory | Runtime | Details |
|---|---|---|---|---|---|---|---|
| 2018-05-16 18:56:24 | c7f82138 | Accepted | 100.00% | cpp11 | 3.85 MB | 0.34 s | |
| 2018-05-16 18:16:23 | a95ef1de | Accepted | 100.00% | cpp11 | 3.82 MB | 0.33 s | |
| 2018-05-16 18:15:19 | 5e0c98ae | Compilation error | — | cpp11 | — | — | |
| 2018-05-16 18:08:59 | f154a9a9 | Accepted | 100.00% | cpp11 | 3.75 MB | 0.30 s | |
| 2018-05-14 16:41:59 | f4de76f3 | Accepted | 100.00% | cpp11 | 3.82 MB | 0.32 s | |
| 2018-05-14 16:34:41 | 08065526 | Accepted | 100.00% | cpp11 | 3.77 MB | 0.32 s | |
| 2018-05-14 16:31:45 | 73608fc9 | Accepted | 100.00% | cpp11 | 3.82 MB | 0.33 s | |
| 2018-05-14 16:13:54 | 53a2d041 | Accepted | 100.00% | cpp11 | 3.92 MB | 0.36 s | |
| 2018-05-14 16:12:44 | 46555d7d | Compilation error | — | cpp11 | — | — | |
| 2018-05-14 16:06:02 | 904413ec | Accepted | 100.00% | cpp11 | 3.84 MB | 0.30 s | |
| 2018-05-14 16:00:03 | 561cd5af | Partially accepted | 50.00% | cpp11 | 3.85 MB | 0.33 s | |
| 2018-05-14 15:58:31 | 3efeddf0 | Partially accepted | 50.00% | cpp11 | 3.35 MB | 0.38 s | |
| 2018-05-14 15:57:19 | 925b4f2b | Partially accepted | 50.00% | cpp11 | 3.41 MB | 0.29 s | |
| 2018-05-14 15:56:18 | 2aac6c1e | Time limit exceeded | 41.67% | cpp11 | 3.40 MB | >3.13 s | |
| 2018-05-14 15:55:16 | eeb8cf47 | Compilation error | — | cpp11 | — | — | |

# 2. Algoritmo 2: Inversiones

## 2.1. Descripción

Deje que A [0 ... n - 1] sea una matriz de n enteros positivos distintos. Si i <j y A [i]>A [j], entonces el par (i, j) se llama inversión de A. Dado ny una matriz A, su tarea es encontrar el número de inversiones de A.

## 2.2. Entrada

La primera línea contiene t, el número de test seguido de un espacio en blanco. Cada una de las pruebas t comienza con un número n ($n <= 200000$). Entonces le sigen n + 1 líneas. En la línea i, se da un número $A[i-1](A[i-1] <= 10^7)$. La $(n+1)$-ésima línea es un espacio en blanco.

## 2.3. Salida

Para cada salida de prueba, una línea indica el número de inversiones de A.

## 2.4. Solución A: Fuerza Bruta

Evalúa los $O(n^2)$ inversiones, siendo n la cantidad de elementos del arreglo A. Al invertir tiempo O(1) en cada uno de ellos, la complejidad temporal resulta ser $O(n^2)$.

## 2.5. Código

```
lli getInvCount(lli Numeros[], lli n)
{
  lli NumInversiones = 0;
  for (lli i = 0; i < n - 1; i++)                                        //Para
     Cada elemento del arreglo
    for (lli j = i+1; j < n; j++)
     //Checar con cada elemento del array
      if (Numeros[i] > Numeros[j])                                       //Ver
     si la posicion es correcta.
        NumInversiones++;

  return NumInversiones;                                                 //Dime
     cuantos son
}
```

## 2.6. Solución B: Divide y Venceras

Ordena el arreglo por merge sort y cuenta las inversiones realizadas

Si sabemos el numero de inversiones de la derecha el de la izquierda y le sumamos el número de inversiones de la combinación tendermos el numero de inversiones del array.

En cualquier paso de Merge(), si Arreglo[i] es mayor que Arreglo [j], entonces hay (Centro - i) inversiones porque los subarreglos izquierdo y derecho se ordenan, entonces todos los elementos restantes en subcampo izquierdo serán mayores que Arreglo [j]

## 2.7. Código

```
#include <algorithm>
#include <iostream>
#include <vector>
#include <string.h>
typedef long long int lli;

using namespace std;
//************************   PROTOTIPOS    ******************************
lli  MergeSortInversiones(lli Numeros[], lli ArrayTemporal[], lli Izquierda, lli Derecha);
lli Merge(lli Numeros[], lli ArrayTemporal[], lli Izquierda, lli Centro, lli Derecha);
//**********************************************************************

//==========================================================================
//=====================          FUNCIONES         ==========================
//==========================================================================

//***********************    MERGE SORT    *******************************
/* Crea un array temporal y llama a merge sort*/
lli MergeSort(lli Numeros[], lli n)
{
    lli *ArrayTemporal = (lli *)malloc(sizeof(lli)*n);
    return MergeSortInversiones(Numeros, ArrayTemporal, 0, n - 1);
}
//Realiza el merge sort y regresa el número de inversiones
lli MergeSortInversiones(lli Numeros[], lli ArrayTemporal[], lli Izquierda, lli Derecha)
{
  lli Centro, NumInversiones = 0;
  if (Derecha > Izquierda)
  {

    Centro = (Derecha + Izquierda)/2;                                   //Divide el arreglo en 2 partes

    NumInversiones  = MergeSortInversiones(Numeros, ArrayTemporal, Izquierda, Centro);
    NumInversiones += MergeSortInversiones(Numeros, ArrayTemporal, Centro+1, Derecha);


    NumInversiones += Merge(Numeros, ArrayTemporal, Izquierda, Centro+1, Derecha);      //Combina las 2 partes
  }
  return NumInversiones;
}

///Combina 2 arreglos ordenados y regresa el número de inversiones de los arreglos
lli Merge(lli Numeros[], lli ArrayTemporal[], lli Izquierda, lli Centro, lli Derecha)
{
  lli i, j, k;
  lli NumInversiones = 0;

  i = Izquierda;                                                        //Indice del subarray izquierdo
  j = Centro;                                                           //Indice del subarray derecho
  k = Izquierda;                                                        //Indice del subarray combinado resultante
  while ((i <= Centro - 1) && (j <= Derecha))
  {
    if (Numeros[i] <= Numeros[j]){
      ArrayTemporal[k++] = Numeros[i++];
    }
    else{
      ArrayTemporal[k++] = Numeros[j++];
      NumInversiones += (Centro - i);                                   //Si no esta ordenado hay Centro-i inversiones
    }
  }


  while (i <= Centro - 1)                                               //Copia los elementos restantes del subarray Izquierdo al array temporal
```

```
    ArrayTemporal[k++] = Numeros[i++];

  while (j <= Derecha)                                             //Copia
     los elementos restantes del subarray Derecho al array temporal
    ArrayTemporal[k++] = Numeros[j++];

  for (i=Izquierda; i <= Derecha; i++)                             //Copia
     los elementos mezclados al array original
    Numeros[i] = ArrayTemporal[i];

  return NumInversiones;
}

//=======================================================================
//===================           MAIN           ==========================
//=======================================================================

int main(){
  lli t=0;                                                     //Obtenemos
     el número de casos
  cin>>t;                                                      //Obtenemos
     el número de casos
  char resultado[100000]="";                                   //Donde
     guardaremos la respuesta

  for (lli i = 0; i < t; ++i){
    lli n=0;
    cin>>n;
    lli Numeros[n];
        for(lli i=0; i<n;i++)
        {
                cin>>Numeros[i];                               // Se lee el
     numero
        }
        sprintf(resultado+strlen(resultado),"%lld\n",MergeSort(Numeros, n));
        n=0;
  }
  printf("%s\n",resultado );
  return 0;
}
```

## 2.8. Análisis

El caso base es de:

$$T(1) = 1$$

Con la ecuación;

$$T(n) = 2T(\frac{n}{2}) + n$$

Entonces usamos el teorema maestro para resolver la cota.

Con este teorema decimos que $a = 2$ $b = 2$ $f(n) = n$

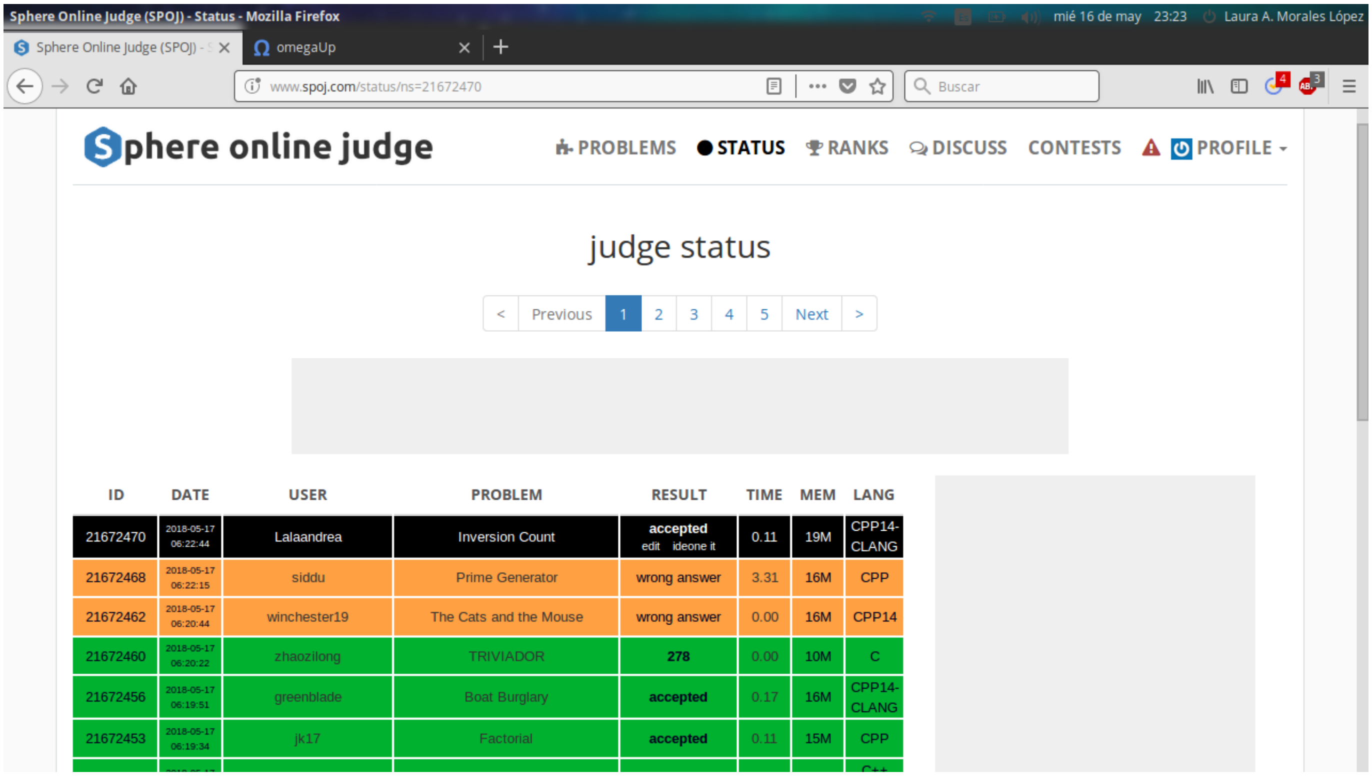
Entonces:

$n^{log_2 2} = (n^1) = n$

Y tenemos la cota:

$$T(n) = \theta(nlogn)$$

## 2.9. Resultados




# 3. Algoritmo 3: Cumulo

## 3.1. Descripción

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

## 3.2. Entrada

La primera línea tendrá un entero 2<=n<=500002<=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<=400000<=X,Y<=40000.

## 3.3. 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)

## 3.4. Solución A: Fuerza Bruta

Evalúa los $O(n^2)$ Distancias, siendo n la cantidad de elementos del arreglo A. Al invertir tiempo O(1) en cada uno de ellos, la complejidad temporal resulta ser $O(n^2)$.

## 3.5. Código

```
void Buscar(p[],n){

   MinimaDistancia = 1e10;
   for (int i = 1; i < n-1; ++i){
      for (int j = i+1; j < n; ++j){
         if (Distancia(P[i],P[j])<MinimaDistancia){
            MinimaDistancia=Distancia(P[i],P[j]);
         }
      }

   }
}
```

## 3.6. Solución B: Divide y Venceras

Ordenamos los puntos según la coordenada x. Ahora que se tiene el conjunto ordenado, se puede trazar una línea vertical, que divida al conjunto de puntos en dos: $P_1$ y $P_2$. Ahora, o el par más cercano está en $P_1$, o está en $P_2$, o un punto está en $P_1$ y el otro en $P_2$. Si los dos estuvieran en $P_1$ o en $P_2$, se hallaría recursivamente, subdividiendo más el problema, por lo que ahora el problema se reduce al tercer caso, un punto en cada zona.

Llamemos $d_1$, $d_2$ y $d_3$ a las mínimas distancias en el primer caso, en el segundo, y en el tercero, respectivamente, y MinimaDistancia al menor de $d_1$ y $d_2$. Para resolver el tercer caso, sólo hace falta mirar los puntos cuya coordenada x esté entre *centro*-MinimaDistancia y *centro*+MinimaDistancia.

## 3.7. Código

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <algorithm>
#include <iostream>
#include <string.h>
typedef long long int lli;
```

```
using namespace std;
struct Punto{
        // Estructura de un Punto.
    double x;
    double y;
 };
/***********************************       PROTOTIPOS       *****************************/
void Busca(struct Punto *,int);
int Ordenax(const void *,const void *);
double Distancia(struct Punto,struct Punto);
/*****************************************************************************/
double MinimaDistancia=1e10;
        // Mínima distancia.


int main()
{
    int a,n;
        //Algunas variables
    double x,y;
    cin>>n;
        //Obten n
    struct Punto p[n];
        //Declara el arreglo de puntos

    for(int i=0;i<n;i++)
        // Obten los datos
    {
       scanf(" %lf %lf",&x,&y);
       p[i].x=x;
       p[i].y=y;
     }

    Busca(p,n);
        //Busca la minima distancia

    printf("%.3lf\n",MinimaDistancia);

    return(0);
 }

void Busca(struct Punto *p,int n)
{
    double d;
    int i,j,a,b;

    if(n<=1)
        // Si no hay pares de puntos:
       return;
        // salir.

    qsort(p,n,sizeof(struct Punto),Ordenax);
        // 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 izquiero 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=Distancia(p[a],p[b]))<MinimaDistancia)
        //Si encuentras una distancia mínima actualizala
          {
             MinimaDistancia=d;
           }
 }


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

double Distancia(struct Punto a,struct Punto b){
         // Función que calcula la distancia entre dos puntos.
    return(sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y)));
```

```
}
```

## 3.8. Análisis

El caso base es de;

$$T(1) = 1$$

Como se mandan 2 busquedas cada una con la mitad del algoritmo y despues se analiza por l derecha y por la izquierda con la siguiente ecuación podemos describir el algoritmo;

$$T(n) = 2T(\frac{n}{2}) + 2n$$

Entonces usamos el teorema maestro para resolver la cota.

Con este teorema decimos que $a = 2$ $b = 2$ $f(2n) = n$

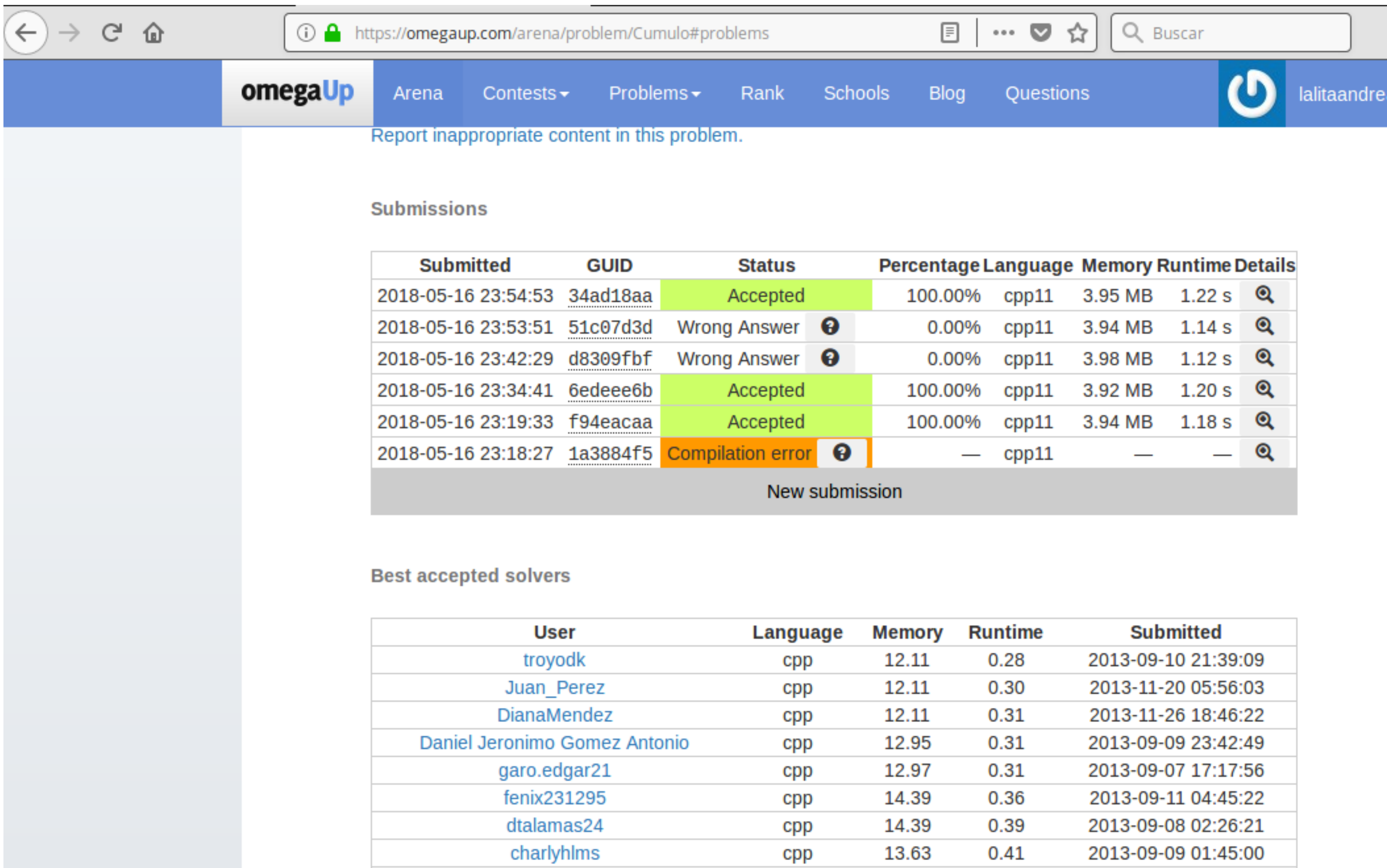
Entonces:

$n^{log_2 2} = (2n^1) = n$

Y tenemos la cota:

$$T(n) = \theta(nlogn)$$

## 3.9. Resultados




| Submitted | GUID | Status | Percentage | Language | Memory | Runtime | Details |
|---|---|---|---|---|---|---|---|
| 2018-05-16 23:54:53 | 34ad18aa | Accepted | 100.00% | cpp11 | 3.95 MB | 1.22 s | |
| 2018-05-16 23:53:51 | 51c07d3d | Wrong Answer | 0.00% | cpp11 | 3.94 MB | 1.14 s | |
| 2018-05-16 23:42:29 | d8309fbf | Wrong Answer | 0.00% | cpp11 | 3.98 MB | 1.12 s | |
| 2018-05-16 23:34:41 | 6edeee6b | Accepted | 100.00% | cpp11 | 3.92 MB | 1.20 s | |
| 2018-05-16 23:19:33 | f94eacaa | Accepted | 100.00% | cpp11 | 3.94 MB | 1.18 s | |
| 2018-05-16 23:18:27 | 1a3884f5 | Compilation error | — | cpp11 | — | — | |

New submission

Best accepted solvers

| User | Language | Memory | Runtime | Submitted |
|---|---|---|---|---|
| troyodk | cpp | 12.11 | 0.28 | 2013-09-10 21:39:09 |
| Juan_Perez | cpp | 12.11 | 0.30 | 2013-11-20 05:56:03 |
| DianaMendez | cpp | 12.11 | 0.31 | 2013-11-26 18:46:22 |
| Daniel Jeronimo Gomez Antonio | cpp | 12.95 | 0.31 | 2013-09-09 23:42:49 |
| garo.edgar21 | cpp | 12.97 | 0.31 | 2013-09-07 17:17:56 |
| fenix231295 | cpp | 14.39 | 0.36 | 2013-09-11 04:45:22 |
| dtalamas24 | cpp | 14.39 | 0.39 | 2013-09-08 02:26:21 |
| charlyhlms | cpp | 13.63 | 0.41 | 2013-09-09 01:45:00 |