

Instituto Politécnico Nacional

Escuela Superior de Cómputo

* **UNIDAD DE APRENDIZAJE:** ANÁLISIS DE ALGORITMOS
* **PROFESOR:** EDGARDO ADRIÁN FRANCO MARTÍNEZ
* **ALUMNO:**

BARRERA PÉREZ CARLOS TONATIHU



## **Ejercicio 6: Diseño de soluciones divide y vencerás**

## 

* **GRUPO:** 3CM3

# Ejercicio 1. Divide and Conquer 1

Redacción

# **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 A.

# **Salida**

La suma máxima en cualquier subarreglo contiguo.

Análisis

Para encontrar la solución a este problema de debemos dividir el arreglo en pequeños subarreglos, esto se hace separando el arreglo en dos partes de manera recursiva.

Es por esto que tenemos tres posibilidades, que la suma se encuentre en el arreglo izquierdo, que se encuentre en el arreglo derecho o que se encuentre entre estos arreglos. Por lo que se deben de resolver estos tres casos y comparar sus soluciones y determinar cuál es la mayor.

|  |
| --- |
| long int maximo\_sub\_array(int \*arreglo, long int inicio, long int final) {  **if** (inicio == final)  **return** arreglo[inicio];  **else** {  long int mitad = (inicio + final)/2;  long int izq = maximo\_sub\_array(arreglo, inicio, mitad);  long int der = maximo\_sub\_array(arreglo, mitad+1, final);  long int centro = sumar\_mitad(arreglo, inicio, mitad, final);  **return** maximo(izq, der, centro);  }  } |

En el código anterior podemos observar que si nos encontramos con un arreglo de un elemento ese elemento es la suma máxima, si por el contrario tenemos un arreglo de más de un elemento debemos de buscar la suma máxima a su izquierda y derecha de manera recursiva y finalmente encontrar la suma en todo el arreglo partiendo de la mitad de este.

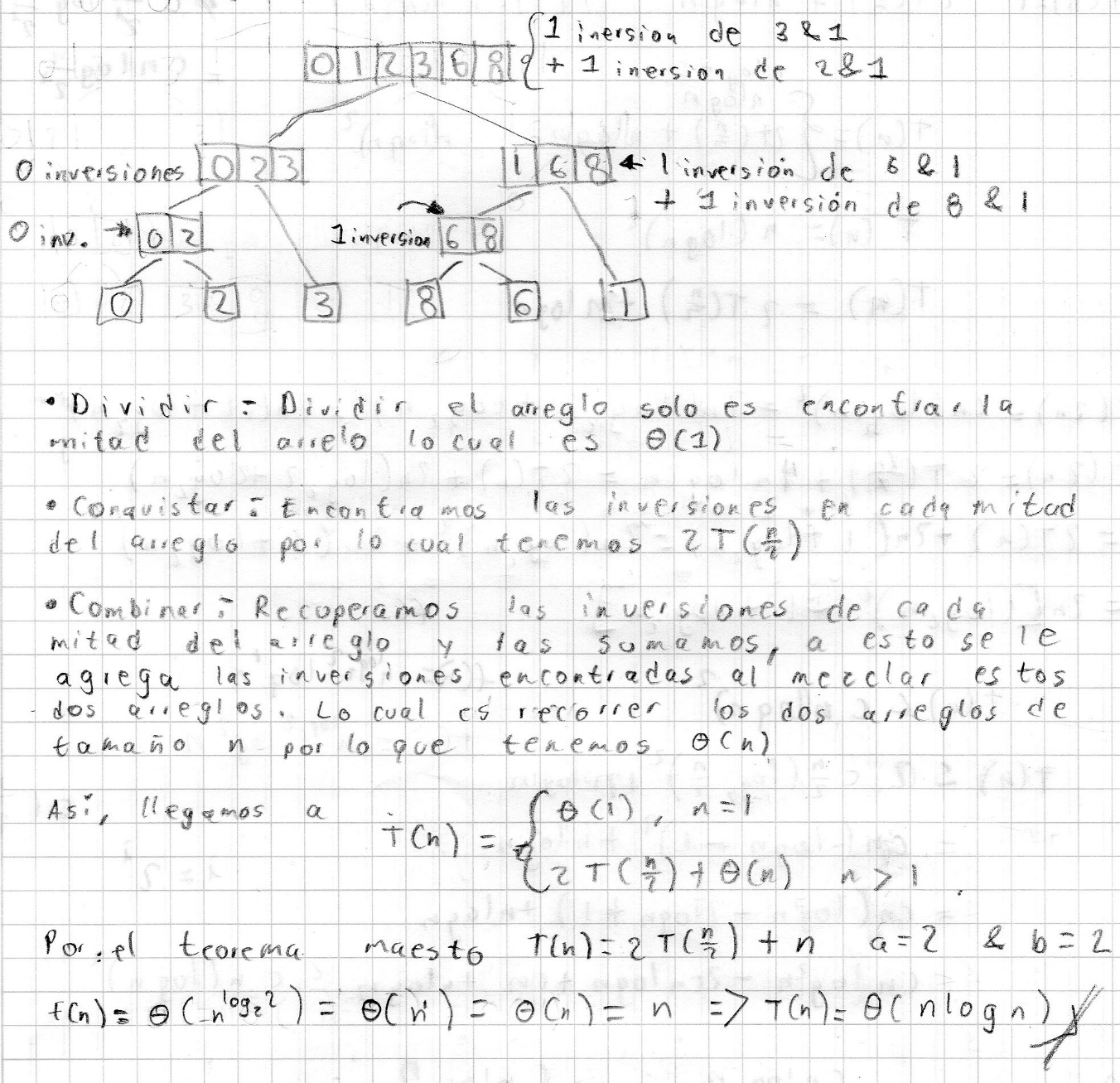
La función que se encarga de encontrar esta suma es la siguiente.

|  |
| --- |
| long int sumar\_mitad(int \*arreglo, long int inicio, long int mitad, long int final) {  long int suma\_izq = arreglo[mitad];  long int suma\_max\_izq = arreglo[mitad];  **for** (long int i = mitad-1; i >= inicio; i--) {  suma\_izq += arreglo[i];  **if** (suma\_izq > suma\_max\_izq)  suma\_max\_izq = suma\_izq;  }  long int suma\_der = arreglo[mitad+1];  long int suma\_max\_der = arreglo[mitad+1];  **for** (long int i = mitad+2; i <= final; i++) {  suma\_der += arreglo[i];  **if** (suma\_der > suma\_max\_der)  suma\_max\_der = suma\_der;  }  **return** suma\_max\_der + suma\_max\_izq;  } |

En esta función comparamos la suma máxima de una mitad con la suma que se obtiene al sumar el siguiente elemento, si la suma continua es mayor que la suma máxima actual la suma continua se convierte en la nueva suma, de otro modo la suma máxima no cambia. Esto se realiza para cada mitad del arreglo y se termina cuando ya se haya recorrido cada mitad.

Al final se retorna la suma de las dos sumas, la cual se compara con las otras dos sumas obtenidas de forma recursiva.

La complejidad de este algoritmo es la siguiente.



Complejidad

Código

Para compilar el código se utiliza el siguiente comando

|  |
| --- |
| $ g++ maximo\_subarray.cpp -o ./maximo\_subarray |

Para ejecutar el archivo resultante utilizamos el siguiente comando

|  |
| --- |
| $ ./maximo\_subarray |

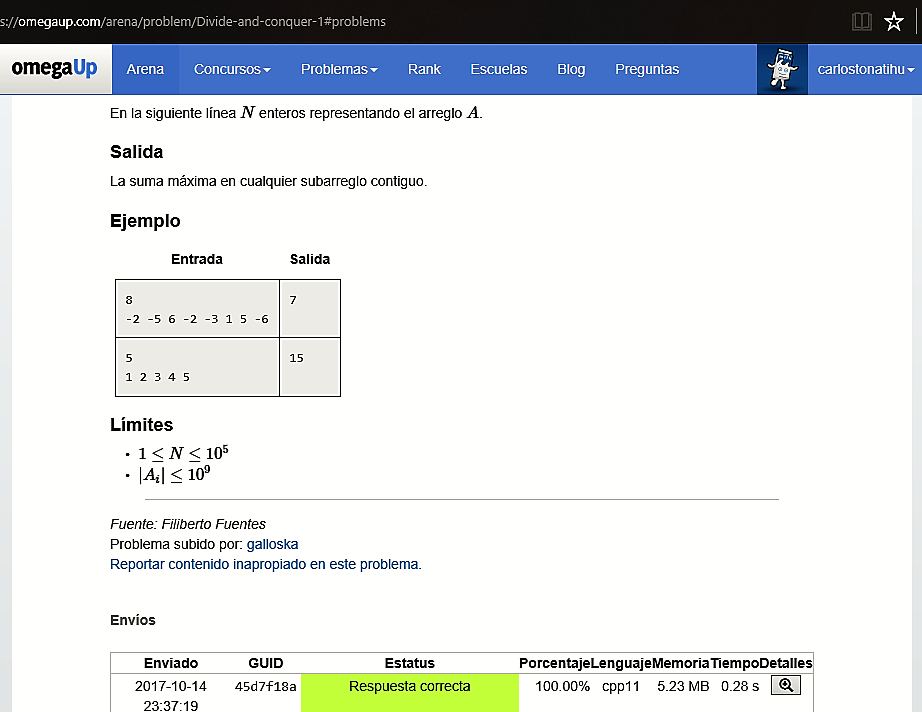
maximo\_subarray.cpp

|  |
| --- |
| #include <iostream>  #include <stdlib.h>  #include <string>  #include <sstream>  **using** **namespace** std;  *// Funciones que se utilizan en este programa*  long int sumar\_mitad(int \*, long int, long int, long int);  long int maximo(long int, long int, long int);  long int maximo\_sub\_array(int \*, long int, long int);  */\*Funcion principal que captura los datos de entrada\*/*  int main(int argc, char **const** \*argv[]) {  int inicio = 0;  int n;  string s;  getline(cin, s);  n = atoi(s.c\_str());  int \*arreglo = (int\*) malloc(**sizeof**(int) \* n);  string str;  getline(cin, str);  long int j = 0;  istringstream is(str);  long int k;  **while**(is >> k) {  arreglo[j++] = k;  }  long int maximo = maximo\_sub\_array(arreglo, inicio, n-1);  cout << maximo;  **return** 0;  }  */\*Encuentra el mayor de tres numeros\*/*  long int maximo(long int a, long int b, long int c) {  **if** (a > b)  **if** (a > c)  **return** a;  **else**  **return** c;  **else** **if** (b > c)  **return** b;  **else**  **return** c;  }  */\*Funcion que busca de manera recursiva la maxima*  *\* suma entre un arreglo de numeros, recibe el inicio*  *\* del arreglo y el final de este, devuelve la suma maxima\*/*  long int maximo\_sub\_array(int \*arreglo, long int inicio, long int final) {  **if** (inicio == final)  **return** arreglo[inicio];  **else** {  long int mitad = (inicio + final)/2;  long int izq = maximo\_sub\_array(arreglo, inicio, mitad);  long int der = maximo\_sub\_array(arreglo, mitad+1, final);  long int centro = sumar\_mitad(arreglo, inicio, mitad, final);  **return** maximo(izq, der, centro);  }  }  */\*Encuentra la suma entre la mitad izquierda y la derecha*  *\* de un arreglo con ayuda de su inicio, mitad y el final\*/*  long int sumar\_mitad(int \*arreglo, long int inicio, long int mitad, long int final) {  long int suma\_izq = arreglo[mitad];  long int suma\_max\_izq = arreglo[mitad];  **for** (long int i = mitad-1; i >= inicio; i--) {  suma\_izq += arreglo[i];  **if** (suma\_izq > suma\_max\_izq)  suma\_max\_izq = suma\_izq;  }  long int suma\_der = arreglo[mitad+1];  long int suma\_max\_der = arreglo[mitad+1];  **for** (long int i = mitad+2; i <= final; i++) {  suma\_der += arreglo[i];  **if** (suma\_der > suma\_max\_der)  suma\_max\_der = suma\_der;  }  **return** suma\_max\_der + suma\_max\_izq;  } |

Pruebas



Captura de pantalla



# Ejercicio 3. Inversion Count

Redacción

Descripción

Sea A [0…n-1] un arreglo de n diferentes enteros positivos. Si i<j y A[i] > A[j] entonces el par (i, j) es una inversión de A. Dados n y un arreglo A tu tarea es encontrar el número de inversiones de A.

Entrada

La primera línea contiene t, el número de casos de prueba seguidos por un espacio en blanco. Cada línea de las t pruebas empiezan con un número n (n <= 200000). Luego siguen n+1 líneas. En la i línea un numero A [i-1] es dado (A[i-1]n<= 10^7). La línea n+1 es una línea en blanco.

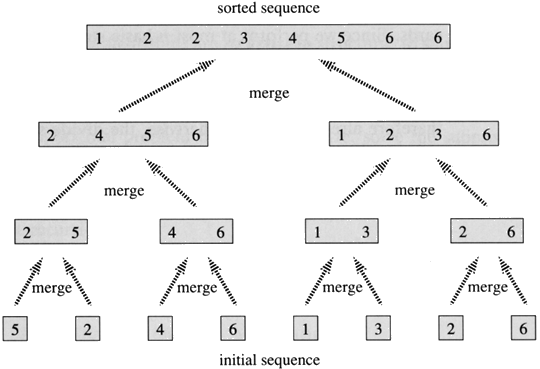
Salida

Para cada caso de prueba la salida es una línea dando el número de inversiones de A.

Análisis

Una inversión ocurre cuando tenemos números mayores a la izquierda de un array con respecto a algún punto del arreglo. Por lo que podemos encontrar inversiones en pequeños sub arreglos y sumar las inversiones de estos subproblemas.

Para hacer esto podemos utilizar el algoritmo merge-sort en el cual se basa en tener dos arreglos ordenados y después mezclarlos en otro arreglo ordenado.



Como se puede observar en la imagen, se construye un nuevo arreglo comparando los elementos que los dos arreglos ya ordenados.

Esto es lo que se utiliza en este algoritmo para contar las inversiones, recorremos dos arreglos ya ordenados de derecha a izquierda (del elemento mayor al menor) y si el número del arreglo izquierdo es mayor que el de la derecha eso quiere decir que es mayor que todos los números que se encuentran en ese arreglo por lo que nuestro total de inversiones se le agrega n, donde n es el total de elementos del arreglo de la derecha.

Después de contar estas inversiones, agregamos el número más grande del arreglo izquierdo a nuestro nuevo arreglo ordenado, obviamente agregamos este número al final del nuevo arreglo.

Si por el contrario el número del arreglo derecho es mayor al número del izquierdo simplemente agregamos el número del arreglo de la derecha al final de nuestro nuevo arreglo ordenado sin contar alguna inversión.

Este proceso se repite hasta que nuestros arreglos a comparar estén vacíos.

Ya que el algoritmo que utilizamos es merge-sort combinado con sumas del total de inversiones tenemos la siguiente complejidad.



Complejidad

Código

Para compilar el código se utiliza el siguiente comando

|  |
| --- |
| $ g++ inversion\_count.cpp -o ./ inversion\_count |

Para ejecutar el archivo resultante utilizamos el siguiente comando

|  |
| --- |
| $ ./ inversion\_count |

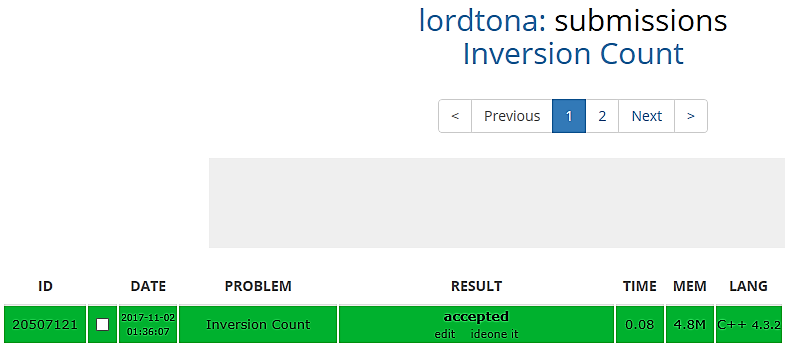
inversion\_count.cpp

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  */\*Definicion de funciones auxiliares\*/*  long long merge\_sort(long \*, long, long);  long long merge(long \*, long, long, long);  */\*Caputra los elementos de cada caso de prueba\*/*  long long obtener\_inversiones() {  long n;  scanf("%ld", &n);  long dato;  long \*arreglo = (long\*)malloc(**sizeof**(long)\*n);  **for** (long i = 0; i < n; i++) {  scanf("%ld", &dato);  arreglo[i] = dato;  }  **return** merge\_sort(arreglo, 0, n-1);  }  */\*Funcion principal para capturar los datos y mostar los resultados\*/*  int main(void){  int pruebas;  int i;  scanf("%d", &pruebas);  char c;  scanf("%c", &c);  long long resultados[pruebas];  **for** (i = 0; i < pruebas; i++) {  resultados[i] = obtener\_inversiones();  scanf("%c", &c);  }  scanf("%c", &c);  **for** (i = 0; i < pruebas; i++)  printf("%lld**\n**", resultados[i]);    **return** 0;  }  */\*Funcion que se encarga de dividir el array de numeros*  *\* y obtener el total de inversiones\*/*  long long merge\_sort(long \*arreglo, long inicio, long fin) {  long long total = 0;  **if** (inicio < fin) {  long mitad = (inicio + fin) / 2;  total += merge\_sort(arreglo, inicio, mitad);  total += merge\_sort(arreglo, mitad+1, fin);  total += merge(arreglo, inicio, mitad, fin);  }  **return** total;  }  */\*Funcion que cuenta el total de inversiones con base al algoritmo de*  *\* ordenamiento merge-sort\*/*  long long merge(long \*arreglo, long inicio, long mitad, long fin) {  long n = mitad - inicio;  long n2 = fin - mitad;  long izq[n];  long der[n2];  **for** (long i=0; i<=n; i++){  izq[i] = arreglo[inicio+i];  }  **for** (long i=0; i<n2; i++){  der[i] = arreglo[mitad+i+1];  }  long i = n;  long j = n2-1;  long k = fin;  long long total = 0;  **while** (i > -1 || j >-1) {  **if** (i > -1 && j >-1) {  **if** (izq[i] < der[j]) {  arreglo[k--] = der[j--];  } **else** {  total += j+1;  arreglo[k--] = izq[i--];  }  } **else** **if** (i > -1) {  arreglo[k--] = izq[i--];  } **else** {  arreglo[k--] = der[j--];  }  }  **return** total;  } |

Pruebas



Captura de pantalla



# Ejercicio 6. Cumulo

Redacción

# **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?

# **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.

# **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)

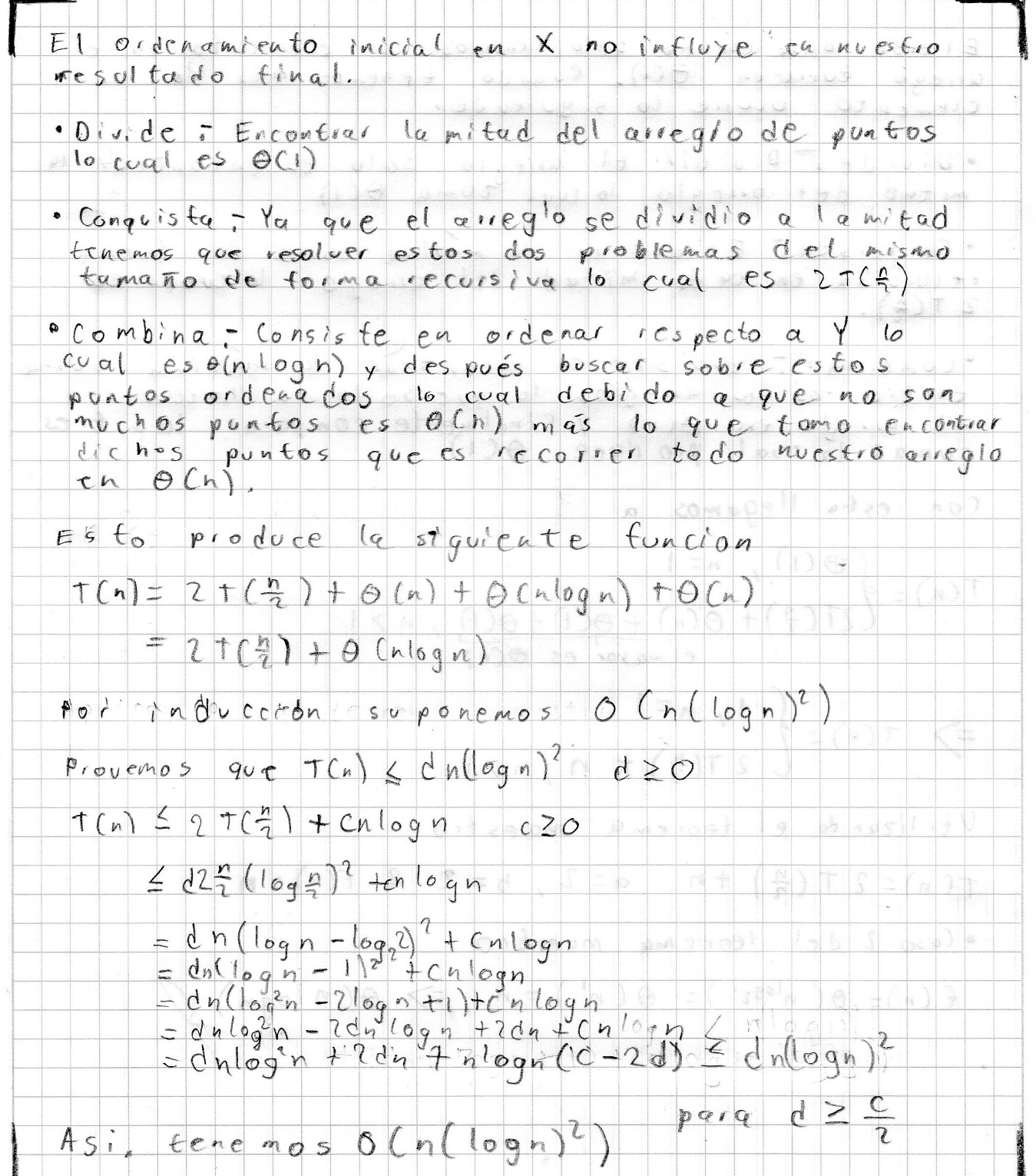
Análisis

Lo primero que se hace en este problema es ordenar todos los puntos respecto a la coordenada X lo cual en el costo de ejecución no influye, para después dividir el problema a la mitad de forma recursiva y encontrar las distancias más pequeñas en cada mitad del arreglo y por consiguiente la más pequeña de estas dos, de manera similar al problema de la máxima suma en un arreglo.

Luego, se tiene que buscar la distancia más corta en los puntos que se encuentran cruzando la mitad del arreglo, los puntos que entran en esta categoría son aquellos que se encuentran a una distancia menor a la menor distancia encontrada en las dos mitades.

Pare realizar esta búsqueda se realiza un ordenamiento respecto al eje y de los puntos que cumplan la condición anterior y buscar de manera directa una distancia menor a la ya encontrada si es que existe.

La complejidad de este algoritmo es la siguiente.



Complejidad

Código

Para compilar el código se utiliza el siguiente comando

|  |
| --- |
| $ g++ puntos.cpp -o ./puntos |

Para ejecutar el archivo resultante utilizamos el siguiente comando

|  |
| --- |
| $ ./puntos |

puntos.cpp

|  |
| --- |
| #include <math.h>  #include <stdio.h>  #include <stdlib.h>  #include <string>  #include <iostream>  #include <sstream>  **using** **namespace** std;  *// Estructura para trabajar las coordenadas de los puntos*  **struct** Punto {  double x;  double y;  };  *// Definicion de nuestras funciones*  double encontrar\_cercano(Punto [], int, int);  double encontrar\_cercano\_mitad(Punto [], int, double);  double min(double, double);  double distancia(Punto, Punto);  int compararX (**const** void \*, **const** void \*);  int compararY (**const** void \*, **const** void \*);  double cercano(Punto [], int, int);  *// Funcion principal que realiza la captura de los valores de entrada y los almacena en*  *// un arreglo para trabajarlos*  int main(int argc, char **const** \*argv[]) {  int n;  string s;  getline(cin, s);  n = atoi(s.c\_str());  Punto puntos[n];  **for** (int i = 0; i < n; i++) {  string str;  getline(cin, str);  stringstream is(str);  double k;  is >> k;  puntos[i].x = k;  is >> k;  puntos[i].y = k;  }  *// ordenar puntos*  qsort(puntos, n, **sizeof**(Punto), compararX);  double punto\_cercano = encontrar\_cercano(puntos, 0, n-1);  printf("%.3f", punto\_cercano);  **return** 0;  }  *// Realiza una busqueda de los puntos mas cercanos sobre un arreglo de maximo tres elementos*  double cercano(Punto P[], int inicio, int fin) {  double min = distancia(P[inicio], P[inicio+1]);  **for** (int i = inicio; i < fin; ++i)  **for** (int j = i+1; j <= fin; ++j){  **if** (distancia(P[i], P[j]) < min){  min = distancia(P[i], P[j]);  }  }  **return** min;  }  *// Realiza la busqueda la distancia mas corta entre dos puntos*  *// de manera recursiva y recibiendo como parametro el arreglo de puntos*  *// ordenados por la coordenada x, el inicio del arreglo y su final*  *// devuelve la distancia mas corta*  double encontrar\_cercano(Punto puntos[], int inicio, int fin) {  **if** ((fin - inicio) <3)  **return** cercano(puntos, inicio, fin);  int mitad = (fin + inicio)/2;  Punto punto\_medio = puntos[mitad];  double cercano\_izq = encontrar\_cercano(puntos, inicio, mitad);  double cercano\_der = encontrar\_cercano(puntos, mitad+1, fin);  double minimo = min(cercano\_izq, cercano\_der);  Punto puntos\_mitad[fin+1];  int j = 0;  **for** (int i = inicio; i < fin; i++)  **if** (abs(puntos[i].x - punto\_medio.x) < minimo) {  puntos\_mitad[j] = puntos[i];  j++;  }  *// Ordenar y*  qsort(puntos\_mitad, j, **sizeof**(Punto), compararY);  minimo = encontrar\_cercano\_mitad(puntos\_mitad, j, minimo);  **return** minimo;  }  */\* Busca la distancia mas corta de los puntos que se encuentren*  *\* a cierta distancia de un punto medio con relacion a los puntos*  *\* ordenados por su coordenada y*  *\*/*  double encontrar\_cercano\_mitad(Punto puntos[], int fin, double min) {  **for** (int i = 0; i < fin; i++)  **for** (int j=i+1; j < fin && (puntos[j].y - puntos[i].y) < min; j++)  **if** (distancia(puntos[i], puntos[j]) < min )  min = distancia(puntos[i], puntos[j]);  **return** min;  }  */\*Calcula la distancia entre dos puntos\*/*  double distancia(Punto a, Punto b) {  **return** sqrt(((a.x - b.x)\*(a.x - b.x)) + ((a.y - b.y)\*(a.y - b.y)));  }  */\*Devuelve el minimo entre dos numeros\*/*  double min(double x, double y) {  **if** (x < y)  **return** x;  **return** y;  }  */\*Funcion que se utiliza como auxiliar para ordenar en x\*/*  int compararX (**const** void \* a, **const** void \* b) {  Punto \*p1 = (Punto \*)a;  Punto \*p2 = (Punto \*)b;  **return** (p1->x - p2->x);  }  */\*Funcion que se utiliza como auxiliar para ordenar en y\*/*  int compararY (**const** void \* a, **const** void \* b) {  Punto \*p1 = (Punto \*)a;  Punto \*p2 = (Punto \*)b;  **return** (p1->y - p2->y);  } |

Pruebas



Captura de pantalla

