REPORTE DE PRACTICA

P12

IMPLEMENTACIÓN DEL ALGORITMO SHELL Y SU ANALISIS

Julio Cesar Hernández Reyes

Grupo:

2CV11

Profesor:

Juan Jesús Gutiérrez García

Algoritmos y Estructura de Datos

Noviembre-2020

DESCRIPCION DE LA PRACTICA

El objetivo de esta práctica es que se implemente el algoritmo Shell agregando un contador que se incremente por cada comparación e intercambio de los datos que se están ordenando.

Así mismo se deben generar datos de prueba de tamaños n=10, 20, 30, …, 10,000. Que contemplen el caso: Orden inverso, peor caso: 10,9,8,7,6,5,4,3,2,1

Al finalizar de la implementación del algoritmo shel con todo lo anterior mencionado también se debe construir una lista de puntos: (x,y) donde x es el tamaño del arreglo de entrada y la segunda coordenada es el número de operaciones que realizó para ese tamaño en específico. Con esta lista de puntos usar el comando de GeoGebra para ajustar los datos a un comportamiento polinomial.

VERSION 0.0

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include <stdlib.h>
int main(){
    int arreglo[100] = {0};
    int i;

    /*Inicializar*/
    for(i=0;i<25;i++){
        arreglo[i]=rand();
    }
    for(i=0;i<25;i++){
        printf("%d\t",arreglo[i]);
    }
    return 0;
}
```

-comenzar por generar y escribir los datos a ordenar
-Usar rand() para datos aleatorias

VERSION 0.1

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include <stdlib.h>
#define NUM_MAX 100//Tamaño del arreglo
#define NUM_A_ANA 25//datos a ser analizados
int main(){
    int arreglo[NUM_MAX] = {0};
    int i;

    /*Inicializar*/
    for(i=0;i<NUM_A_ANA;i++){
        arreglo[i]=rand();
    }
    for(i=0;i<NUM_A_ANA;i++){
        printf("%d\t",arreglo[i]);
    }
    return 0;
}
```

-Ponerle nombres a las constantes

VERSION 0.2

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include <stdlib.h>
#include "shell.h"

#define NUM_MAX 100//Tamaño del arreglo
#define NUM_A_ANA 25//datos a ser analizados
int main(){
    int arreglo[NUM_MAX] = {0};
    int i;

    /*Inicializar*/
    for(i=0;i<NUM_A_ANA;i++){
        arreglo[i]=NUM_A_ANA - i;
    }

    if(shell(arreglo,NUM_A_ANA)!=OK){
        printf("Error al ordenar\n");
    }

    for(i=0;i<NUM_A_ANA;i++){
        printf("%d\t",arreglo[i]);
    }

    return 0;
}
```

-Incluir el archivo Shell.h
Que es el que ordena los datos
-Se ejecuto sin tener código de Shell.c

```c
#ifndef _SHELL_H
#define _SHELL_H

#define OK 0 //No hay error

int shell(int* arr, int tam);

#endif
```

Se declaro la función shell

```c
#include<stdio.h>
#include "shell.h"

int shell(int* arr, int tam){
    return OK;
}
```

Defini la función Shell que es la que va a ordenar en el archivo Shell.c

VERSION 0.3

```c
#ifndef _SHELL_H
#define _SHELL_H
/*****************************/
/*          Constantes       */
/*****************************/
/*No hay error*/
#define OK 0
/*Apuntador nulo*/
#define AP_INV 1
/*Tamaño de arreglo invalido*/
#define TA_INV 2
/*Tamaño Maximo del arreglo*/
#define TAM_MAX 10000

/*****************************/
/*          Macros           */
/*****************************/
/*Apuntador valido, No nulo */
#define AP_VAL(a)   ((a)!=NULL)
/*Tamaño de arreglo valido*/
#define TA_VAL(a)   (a)>0&&(a<TAM_MAX)

/*****************************/
/*      Ordena los datos     */
/*************************+**/
/*Recibe:
    arr: Arreglo con los datos
    tam: número de datos a ordenar
Regresa:
    Costante de error*/
int shell(int* arr, int tam);
/*
Intercambia los valores de dos variables
Recibe:
    Apuntador a las variables
    Intercambiar de valor
Regresa:
    Constante de error*/
int swap(int* a, int* b);
#endif
```

En el archivo Shell.h Se agregan constantes para el manejo de errores.

- Constante para el tamaño máximo del arreglo a ordenar.
- Macro para validar un apuntador.
- Macro para validar el tamaño de un arreglo.

Comentarios y prototipo de la función que ordena.

- Comentario y prototipo dela función que intercambia el valor de dos variables

```c
#include<stdio.h>
#include "shell.h"
/*Intercambia los valores de dos variables
Recibe:
    Apuntador a las variables
    Intercambiar de valor
Regresa:
    Constante de error*/
int swap(int *a, int *b){
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return OK;
}
/*Ordena los numeros usando el algoritmo shell
en el peor caso que es de forma descendente/orden inverso
Recibe: arr-->Arreglo con los datos
        tam-->Número de datos a ordenar
Regresa:    Costante de error*/
int shell(int* arr, int tam){
    if(!AP_VAL(arr)){
        return AP_INV;
    }
    if(!TA_VAL(tam)){
        return TA_INV;
    }
    int cont,temp,i;
    for(cont=tam/2;cont!=0;cont/=2)
    {
        for(i=cont;i<tam;i++)
        {
            if(arr[i-cont]<arr[i])
            {
                swap(&arr[i],&arr[i-cont]);
            }
        }
    }
    return OK;
}
```

Agregar comentarios similares a los del archivo cabecera.

- Código de la función que inter cambia dos valores
- Código de la función que ordena, en este caso él ordenamiento shell

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include <stdlib.h>
#include "shell.h"

#define NUM_MAX 100//Tamaño del arreglo
#define NUM_A_ANA 25//datos a ser analizados
int main(){
    int arreglo[NUM_MAX] = {0};
    int i;
    srand(3);
    /*Inicializar*/
    for(i=0;i<NUM_A_ANA;i++){
        arreglo[i]=NUM_A_ANA - i;
    }

    if(shell(arreglo,NUM_A_ANA)!=OK){
        printf("Error al ordenar\n");
    }

    shell(arreglo,NUM_A_ANA);

    for(i=0;i<NUM_A_ANA;i++){
        printf("%d\t",arreglo[i]);
    }
    return 0;
}
```

En el archivo main.c

- Se agrega la llamada al método que ordena.
- Se verifica que no haya error al ejecutarla.
- El resto queda igual.

Se ejecuta

VERSION 1.0

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include "shell.h"

#define NUM_MAX 1000//Tamaño del arreglo
#define NUM_A_ANA 1000//datos a ser analizados

int main(){
    int arreglo[NUM_MAX] = {0};
    int i,j;
    int num;
    /*Inicializar*/
    for(i=0;i<NUM_A_ANA;i++){
        //num=rand()%1000;
        arreglo[i]=NUM_A_ANA - i;
    }
    //Imprimir la secuencia de numeros
    //de forma decreciente que es el peor caso
    int k;
    for(k=0;k<NUM_A_ANA;k++){
        printf("%d\t",arreglo[k]);
    }
    printf("\n\n");
    //Llama a la funcion shell y verifica
    //si todo salio bien
    if(shell(arreglo,NUM_A_ANA)!=OK){
        printf("Error al ordenar\n");
    }
    //Llama a la funcion verifica que checa que
    //todos los numero esten ordenados
    verifica(arreglo,NUM_A_ANA);
    return 0;
}
```

Se arreglo el archivo main.c para que mandara llamar a la función Shell y además que verifique que los datos estes correctamente acomodados. En este caso se uso la inicialización de los valores en el peor caso, en forma decreciente.

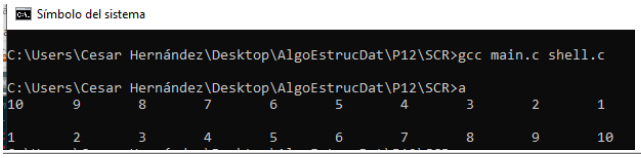
```c
#include<stdio.h>
#include "shell.h"
/*Ordena los numeros usando el algoritmo shell
en el peor caso que es de forma descendente/orden inverso
Recibe: arr-->Arreglo con los datos
        tam-->Número de datos a ordenar
Regresa:    Costante de error*/
int shell(int* arr, int tam){
    if(!AP_VAL(arr)){
        return AP_INV;
    }
    if(!TA_VAL(tam)){
        return TA_INV;
    }

    int h, i, j, temp;
    for (h=tam/2; h>0; h=h/2) {
        for (i=h; i<tam; i++) { //aplica inserción a partir de h
            temp=arr[i];
            for (j=i-h; j>=0 && arr[j]>temp; j-=h) {
                arr[j+h]=arr[j];
            }
            arr[j+h]=temp; //inserta
        }
    }
    int k;
    for(k=0;k<tam;k++){
        printf("%d\t",arr[k]);
    }
    return OK;
}

    /*Verifica que los datos estan ordenados,
    en lugar de desplegar los datos,
    asi evitamos un error al visualizar
    los datos en pantalla*/
int verifica(int* arr, int tam){
    int verificar=0,i;
    for(i=0;i<tam - 1;i++){
        if(arr[i]<=arr[i+1]){
            verificar++;
        }
    }
    if(verificar!=tam - 1){
        printf("Error al ordenar %d\n",verificar);
    }
    return OK;
}
```

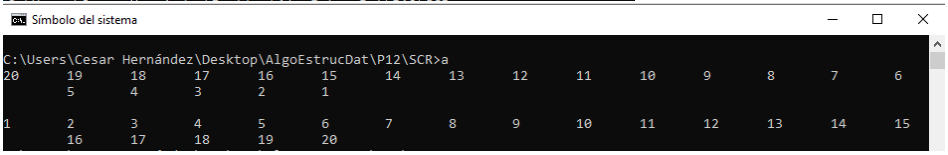
Se arreglo la función Shell para que ordenara de forma correcta pues encontré vario códigos de Shell sort pero este fue el que más me funciono y que además después se le puede agregar fácilmente los contadores de movimientos y comparaciones.

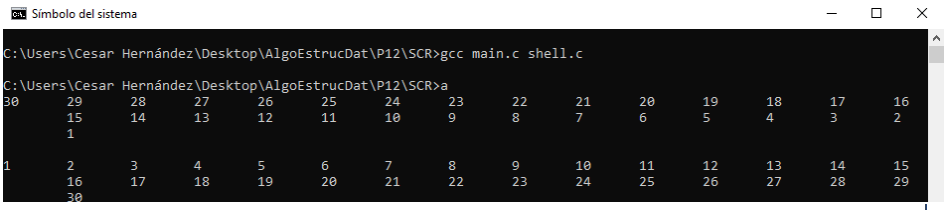
La parte de la verificación del orden de los datos se paso a función

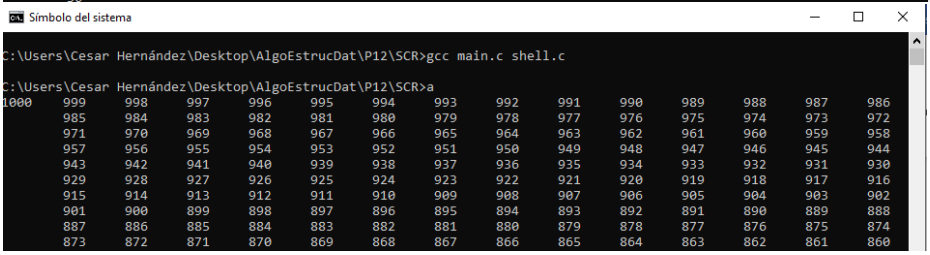
```c
#ifndef _SHELL_H
#define _SHELL_H
/*****************************/
/*          Constantes       */
/*****************************/
/*No hay error*/
#define OK 0
/*Apuntador nulo*/
#define AP_INV 1
/*Tamaño de arreglo invalido*/
#define TA_INV 2
/*Tamaño Maximo del arreglo*/
#define TAM_MAX 10000

/*****************************/
/*          Macros           */
/*****************************/
/*Apuntador valido, No nulo */
#define AP_VAL(a)   ((a)!=NULL)
/*Tamaño de arreglo valido*/
#define TA_VAL(a)   (a)>0&&(a<TAM_MAX)

/*****************************/
/*      Ordena los datos     */
/*************************+**/
/*Recibe:
    arr: Arreglo con los datos
    tam: número de datos a ordenar
Regresa:
    Costante de error*/
int shell(int* arr, int tam);
/*
Intercambia los valores de dos variables
Recibe:
    Apuntador a las variables
    Intercambiar de valor
Regresa:
    Constante de error*/
int swap(int* a, int* b);
/*
Verifica que los datos estan ordenados,
en lugar de desplegar los datos,
asi evitamos un error al visualizar
los datos en pantalla*/
int verifica(int* arr,int tam);
#endif
```

Como se paso la parte de verificar a función se tuvo que declarar en el archivo Shell.h para que se pueda usar en el main.c y en Shell.c

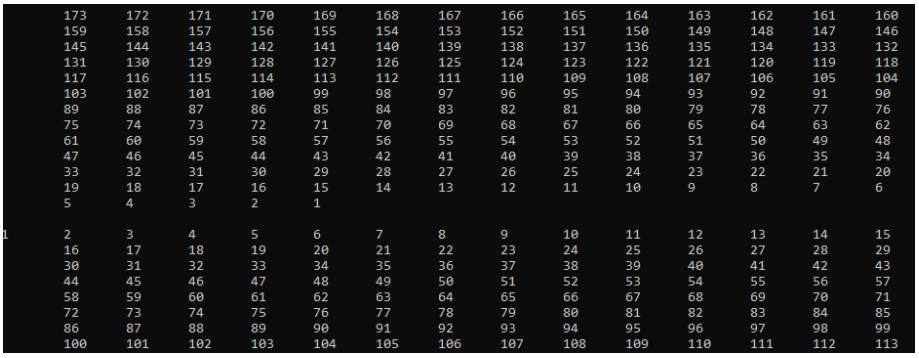




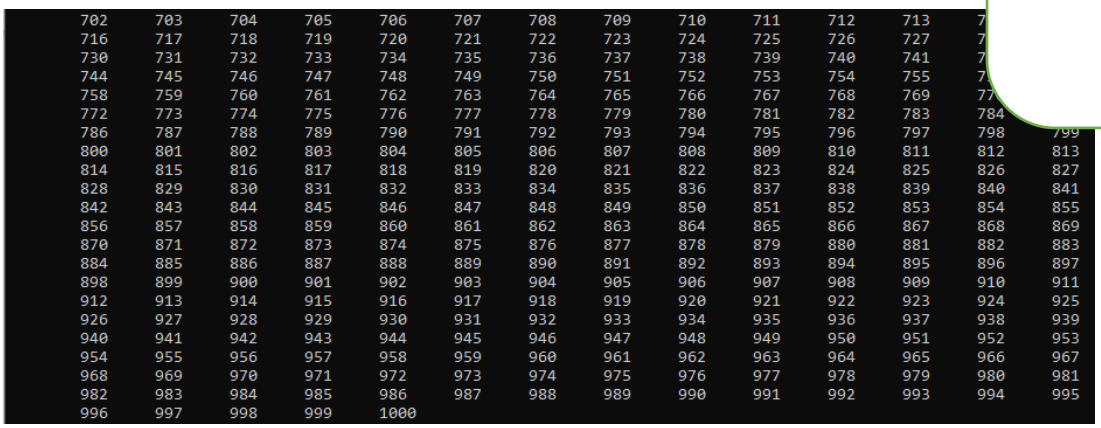




...



...



Se hicieron pruebas para ver y confirmar que mi algoritmo de Shell que implemente ordena de forma correcta.
En una primera parte se verifico con 10, 20, 30, 1000 valores.
Se imprimía en pantalla el orden inicial en el que se empezaba el arreglo y luego se imprimía el arreglo pero ya ordenado

VERSION 1.1

```c
#ifndef _SHELL_H
#define _SHELL_H
/*****************************/
/*          Constantes       */
/*****************************/
/*No hay error*/
#define OK 0
/*Apuntador nulo*/
#define AP_INV 1
/*Tamaño de arreglo invalido*/
#define TA_INV 2
/*Tamaño Maximo del arreglo*/
#define TAM_MAX 10000

/*****************************/
/*          Macros           */
/*****************************/
/*Apuntador valido, No nulo */
#define AP_VAL(a)   ((a)!=NULL)
/*Tamaño de arreglo valido*/
#define TA_VAL(a)   (a)>0&&(a<TAM_MAX)
/*****************************/
/*      Ordena los datos     */
/*************************+**/
/*Recibe:
    arr: Arreglo con los datos
    tam: número de datos a ordenar
Regresa:
    Costante de error*/
int shell(int* camb, int* comp, int* arr, int tam);
/*****************************/
/*      Cambia dos datos     */
/*************************+**/
/*Recibe:
    a*:Primer valor del arreglo
    b*:segundo valor del arreglo
Regresa:
    Costante de error*/
int swap(int* c, int*a, int* b);
//Compara dos datos del arreglo y le suma
//uno al contador de comparaciones
int comp_c(int* c, int a, int b);
//Imprime el arreglo en pantalla
void Imprime_Arr(int* arr, int tam);
#endif
```

Se agregaron los prototipos de las funciones que cuentan los intercambios y las comparaciones.

```c
#include<stdio.h>
#include "shell.h"
//intercambia dos valores del arreglo
int swap(int* camb, int* a, int* b){
    int tmp;
    (*camb)++;
    tmp= *a;
    *a = *b,
    *b = tmp;
    return OK;
}
/*Ordena los numeros usando el algoritmo shell
en el peor caso que es de forma descendente/orden inverso
Recibe: arr-->Arreglo con los datos
        tam-->Número de datos a ordenar
Regresa:    Costante de error*/
int shell(int* camb,int* comp,int* arr, int tam){
    if(!AP_VAL(arr)){
        return AP_INV;
    }
    if(!TA_VAL(tam)){
        return TA_INV;
    }

    int salto,pos,cambios;
    salto = tam/2;
    do{
        do{
            cambios = 0;
            for(pos = salto; pos < tam; pos++){
                if(comp_c(comp,arr[pos - salto],arr[pos])){
                    swap(camb,&arr[pos],&arr[pos - salto]);
                    cambios++;
                }
            }
        }while(cambios != 0);
        salto = salto/2;
    }while(salto != 0);
    return OK;
}
//compara dos valores del arreglo
int comp_c(int* comp, int a, int b){
    (*comp)++;
    return a>b;
}
//Imprime el arreglo por pantalla
void Imprime_Arr(int* arr, int tam){
    int i;
    for(i=0;i<tam;i++){
        printf("%d\t",arr[i]);
    }
}
```

También en shell.c se agregan las definiciones que sólo incrementan el contador que reciben.

- En la función main.c se declara el contador se inicializa y se pasa como parámetro.
- Se imprime el tamaño del arreglo y contador.

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso(Orden Inverso):10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include "shell.h"

#define NUM_MAX 100//Tamaño del arreglo
#define CANT_NUM 100//datos a ser analizados
int main(){
    srand(time(NULL));
    int arreglo[NUM_MAX] = {0};
    int i,j,k,n,num,comparacion,cambio,verificar;
    float promedio;
            /*Inicializar en Orden Inverso*/
            for(i=0;i<CANT_NUM;i++){
                //num=rand()%1000;
                arreglo[i]=CANT_NUM - i;//CANT_NUM - i;
            }
            //Llama a la funcion shell y verifica
            //si todo salio bien
            cambio=0;
            comparacion=0;
            if(shell(&cambio,&comparacion,arreglo,CANT_NUM)!=OK){
                printf("Error al ordenar\n");
            }
            //Impresion de arreglo
            Imprime_Arr(arreglo,CANT_NUM);
            //Verificar que se ordenaron los datos
            verificar=0;
            for(i=0;i<CANT_NUM - 1;i++){
                if(arreglo[i]<=arreglo[i+1]){
                    verificar++;
                }
            }
            if(verificar!=CANT_NUM - 1){
                printf("Error al ordenar %d\n",verificar);
            }

        printf("\n\nCambios:%d\n",cambio);
        printf("Comparaciones:%d\n",comparacion);
    return 0;
}
```

Se regreso la funcion de verificar al main para poder agregar el contador de cambios de forma mas fácil.

VERSION DEL ANALISIS / VERSION FINAL

```c
//Algoritmo 3 --> Shell
//Caso 2-->Peor caso(Orden Inverso):10,9,8,7,6,5,4,3,2,1
#include <stdio.h>
#include "shell.h"

#define NUM_MAX 10010//Tamaño del arreglo
int main(){
    int arreglo[NUM_MAX] = {0};
    int i,j,k,n,num,comparacion,cambio,verificar,promedio;
    for(n=10;n<NUM_MAX;n+=10)
    {

        /*Inicializar en Orden Inverso*/
        for(i=0;i<n;i++){
            //num=rand()%1000;
            arreglo[i]=n - i;//CANT_NUM - i;
        }
        //Llama a la funcion shell y verifica
        //si todo salio bien
        cambio=0;
        comparacion=0;
        if(shell(&cambio,&comparacion,arreglo,n)!=OK){
            printf("Error al ordenar\n");
        }
        //Impresion de arreglo
        //Imprime_Arr(arreglo,n);
        //Verificar que se ordenaron los datos
        verificar=0;
        for(i=0;i<n - 1;i++){
            if(arreglo[i]<=arreglo[i+1]){
                verificar++;
            }
        }
        if(verificar!=n - 1){
            printf("Error al ordenar %d\n",verificar);
        }

        //printf("\n\nCambios:%d\n",cambio);
        //printf("Comparaciones:%d\n",comparacion);
        promedio= cambio + comparacion;
        printf("%d\t%d\n",n,promedio);
    }
    return 0;
}
```

Se implementaron los contadores de cambios y comparaciones y se saco el promedio que en si solo fue la suma de estos contadores.

```c
#include<stdio.h>
#include "shell.h"
//intercambia dos valores del arreglo
int swap(int* camb, int* a, int* b){
    int tmp;
    (*camb)++;
    tmp= *a;
    *a = *b,
    *b = tmp;
    return OK;
}
/*Ordena los numeros usando el algoritmo shell
en el peor caso que es de forma descendente/orden inverso
Recibe: arr-->Arreglo con los datos
        tam-->Número de datos a ordenar
Regresa:    Costante de error*/
int shell(int* camb,int* comp,int* arr, int tam){
    if(!AP_VAL(arr)){
        return AP_INV;
    }
    if(!TA_VAL(tam)){
        return TA_INV;
    }

    int salto,pos,cambios;
    salto = tam/2;
    do{
        do{
            cambios = 0;
            for(pos = salto; pos < tam; pos++){
                if(comp_c(comp,arr[pos - salto],arr[pos])){
                    swap(camb,&arr[pos],&arr[pos - salto]);
                    cambios++;
                }
            }
        }while(cambios != 0);
        salto = salto/2;
    }while(salto != 0);
    return OK;
}
//compara dos valores del arreglo
int comp_c(int* comp, int a, int b){
    (*comp)++;
    return a>b;
}
//Imprime el arreglo por pantalla
void Imprime_Arr(int* arr, int tam){
    int i;
    for(i=0;i<tam;i++){
        printf("%d\t",arr[i]);
    }
}
```

Se implementaron los contadores de cambios y comparaciones en el código Shell sort

Se agregó otra función que cuenta las comparaciones entre valores.

```c
#ifndef _SHELL_H
#define _SHELL_H
/*****************************/
/*          Constantes       */
/*****************************/
/*No hay error*/
#define OK 0
/*Apuntador nulo*/
#define AP_INV 1
/*Tamaño de arreglo invalido*/
#define TA_INV 2
/*Tamaño Maximo del arreglo*/
#define TAM_MAX 10000

/*****************************/
/*          Macros           */
/*****************************/
/*Apuntador valido, No nulo */
#define AP_VAL(a)   ((a)!=NULL)
/*Tamaño de arreglo valido*/
#define TA_VAL(a)   (a)>0&&(a<TAM_MAX)
/*****************************/
/*      Ordena los datos     */
/*****************************+**/
/*Recibe:
    arr: Arreglo con los datos
    tam: número de datos a ordenar
Regresa:
    Costante de error*/
int shell(int* camb, int* comp, int* arr, int tam);
/*****************************/
/*      Cambia dos datos     */
/*****************************+**/
/*Recibe:
    a*:Primer valor del arreglo
    b*:segundo valor del arreglo
Regresa:
    Costante de error*/
int swap(int* c, int*a, int* b);
//Compara dos datos del arreglo y le suma
//uno al contador de comparaciones
int comp_c(int* c, int a, int b);
//Imprime el arreglo en pantalla
void Imprime_Arr(int* arr, int tam);
#endif
```

Se agregaron los prototipos de las funciones que faltaban.

Símbolo del sistema

```
C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P12\SCR>gcc main.c shell.c

C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P12\SCR>a > salida.txt

C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P12\SCR>
```

```
  1    10   56
  2    20   159
  3    30   309
  4    40   415
  5    50   549
  6    60   768
  7    70   960
  8    80   1027
  9    90   1212
 10    100  1354
 11    110  1590
 12    120  1836
 13    130  1862
 14    140  2281
 15    150  2343
 16    160  2451
 17    170  2716
 18    180  2889
 19    190  3210
 20    200  3214
 21    210  3688
 22    220  3753
 23    230  3963
 24    240  4272
 25    250  4617
 26    260  4401
 27    270  4797
 28    280  5273
 29    290  5365
 30    300  5468
 31    310  6061
 32    320  5699
 33    330  6104
 34    340  6317
 35    350  6788
 36    360  6693
 37    370  7218
 38    380  7417
 39    390  7887
 40    400  7434
 41    410  7909
 42    420  8471
 43    430  9017
 44    440  8629
 45    450  8960
 46    460  9126
```

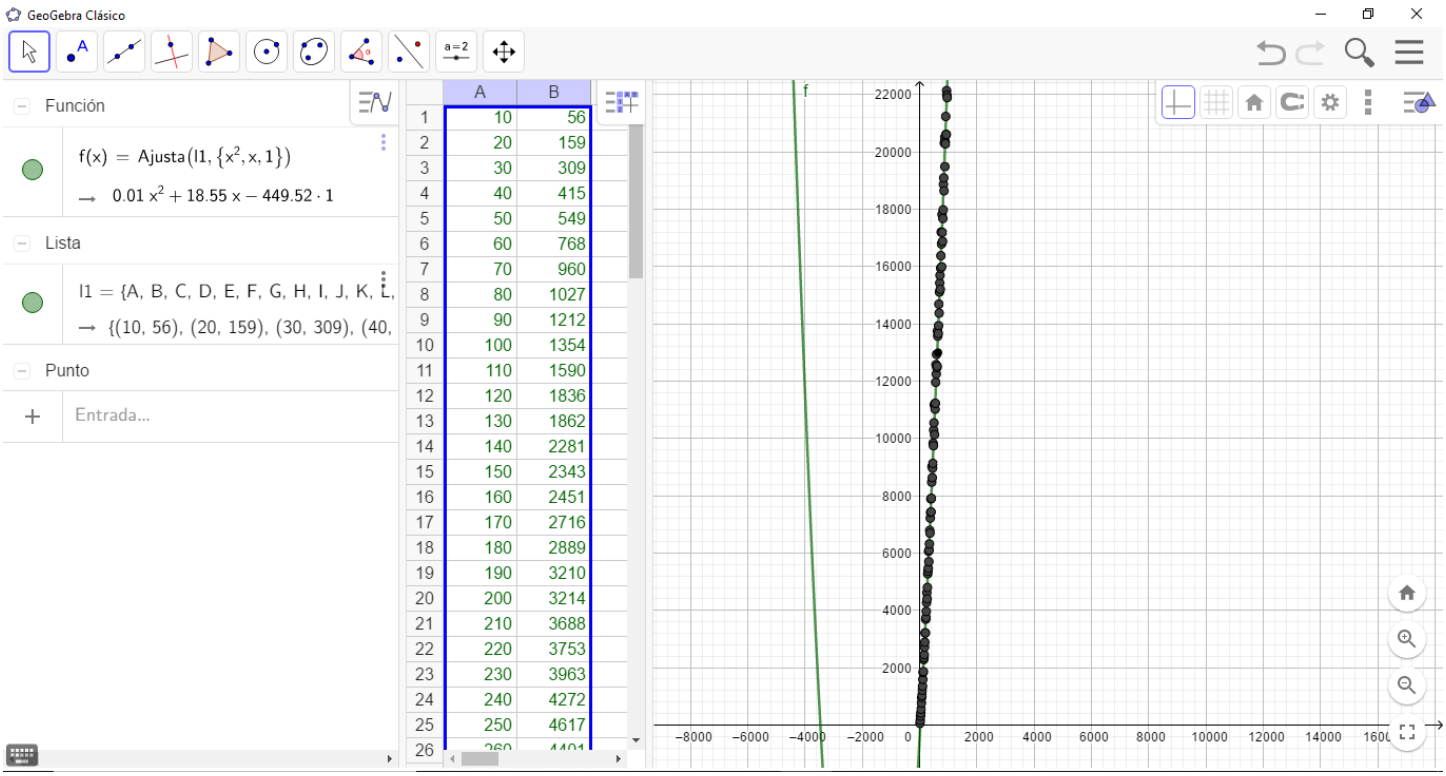
```
 956    9560    303446
 957    9570    301908
 958    9580    317062
 959    9590    311814
 960    9600    295968
 961    9610    303443
 962    9620    304926
 963    9630    313047
 964    9640    306102
 965    9650    317929
 966    9660    315456
 967    9670    318763
 968    9680    307574
 969    9690    315119
 970    9700    320837
 971    9710    329033
 972    9720    318054
 973    9730    296896
 974    9740    313744
 975    9750    306937
 976    9760    300125
 977    9770    307720
 978    9780    309223
 979    9790    317474
 980    9800    316057
 981    9810    309078
 982    9820    325556
 983    9830    324846
 984    9840    312122
 985    9850    319787
 986    9860    317165
 987    9870    325481
 988    9880    313318
 989    9890    311720
 990    9900    328162
 991    9910    321936
 992    9920    320308
 993    9930    328033
 994    9940    329566
 995    9950    337957
 996    9960    330782
 997    9970    323903
 998    9980    340446
 999    9990    315354
1000    10000   322249
```

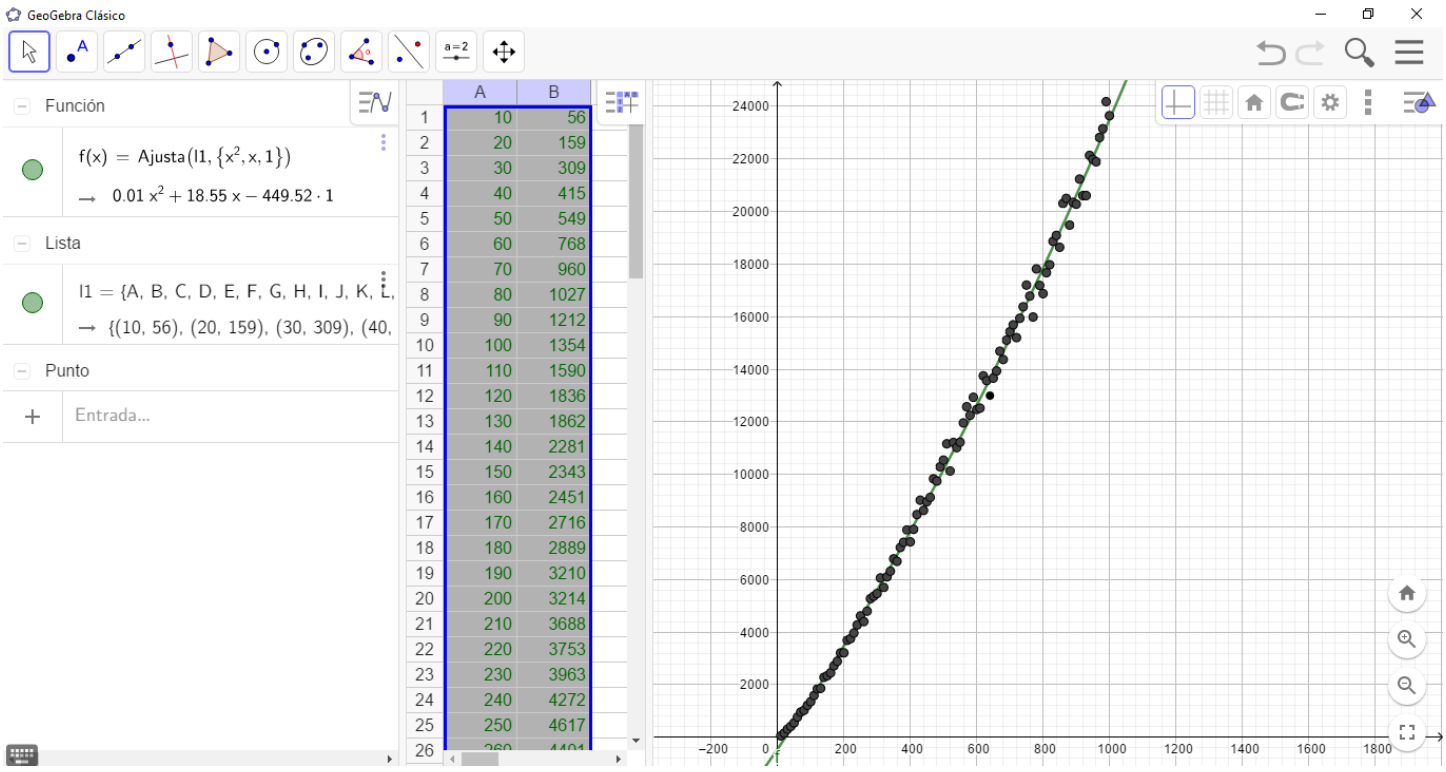
Para correr este código solo se espera como impresión la cantidad de valores y la suma de los cambios y comparaciones que se realizaron.
Se redirigió la impresión del código a un archivo salida.txt

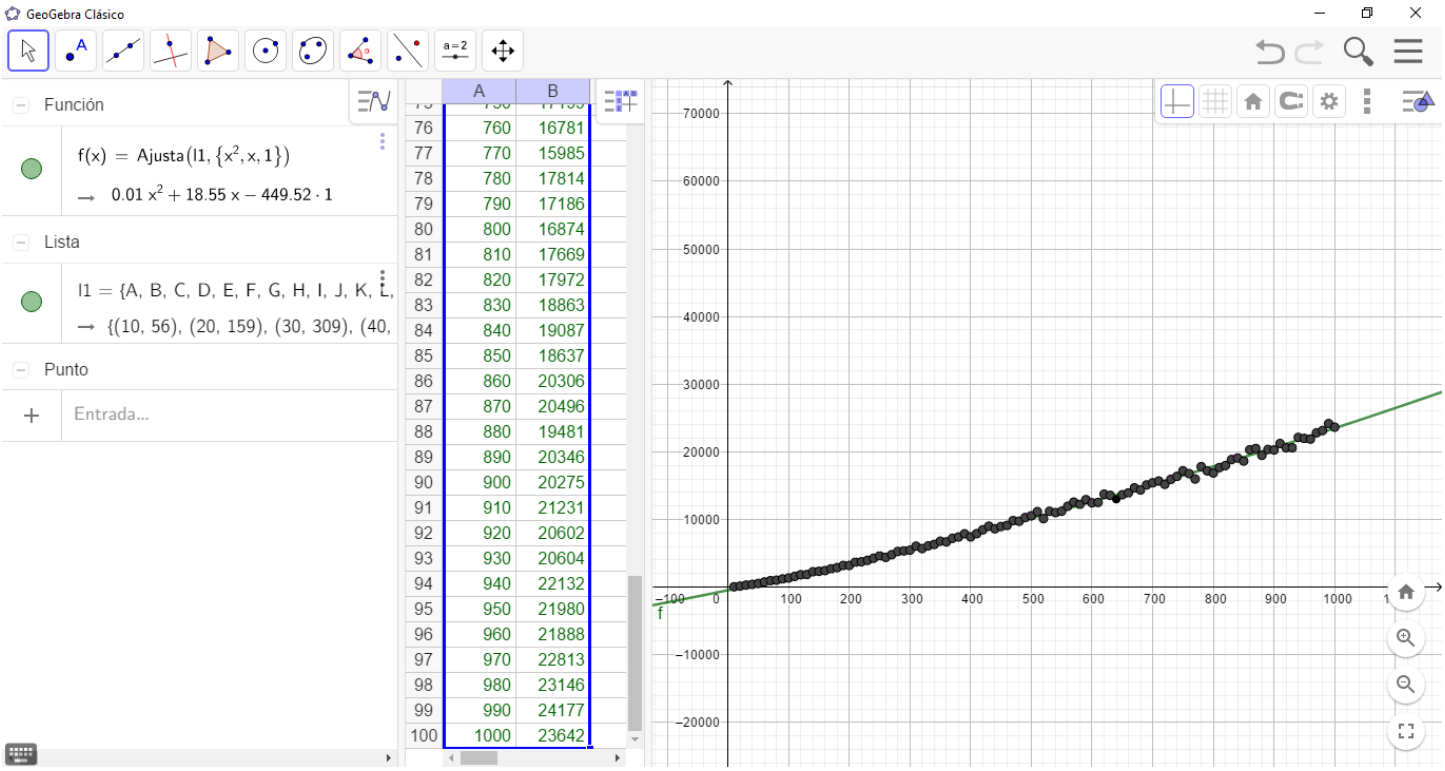
Se copiaron los valores obtenidos en el código y se copiaron en GeoGebra haciendo una lista de puntos con la cual se hizo el análisis. Después de crear la lista se uso este comando par que se mostraran gráficamente:

Ajustar(l1,{x^2,x,1})

NOTA: Solo se uso una lista y no más dado que mi caso era el de orden inverso no se podían obtener resultados distintos a comparación de si me hubiera tocado el caso de aleatorio






GeoGebra Clásico
Función
f(x) = Ajusta(l1, {x², x, 1})
→ 0.01 x² + 18.55 x − 449.52 · 1
Lista
l1 = {A, B, C, D, E, F, G, H, I, J, K, L,
→ {(10, 56), (20, 159), (30, 309), (40,
Punto
Entrada...
A
B
76 760 16781
77 770 15985
78 780 17814
79 790 17186
80 800 16874
81 810 17669
82 820 17972
83 830 18863
84 840 19087
85 850 18637
86 860 20306
87 870 20496
88 880 19481
89 890 20346
90 900 20275
91 910 21231
92 920 20602
93 930 20604
94 940 22132
95 950 21980
96 960 21888
97 970 22813
98 980 23146
99 990 24177
100 1000 23642
70000
60000
50000
40000
30000
20000
10000
−10000
−20000
−100
0
100
200
300
400
500
600
700
800
900
1000

CONCLUSIONES:

¿Qué diferencia hay entre cambiar el tamaño del arreglo a ordenar o el número de datos para promediar?

La diferencia en mi caso no existe pues no realice un promedio de comparaciones y de cambios ya que mi caso era el de orden inverso siempre salen los mismos valores cuando se ejecuta otra vez.

Lo que cambia en mi caso es que si se cambia el tamaño de datos estos siguen sumando a las comparaciones, pero no cambian los valores antes obtenidos, ósea solo se le agregan los nuevos valores que se aumentaron.

• ¿Es posible calcular cuantas milésimas de segundo se tarda en hacer una operación de comparación o intercambio?

Yo diría que si se hace unos cálculos con n cantidad de valores y luego se divide entre la cantidad de comparaciones.

En mi casi calcule que mi programa se tardo 6.5 s en calcular e imprimir las comparaciones de 1000 valores.

Y además en mis impresiones de la salida.txt las comparaciones que se hicieron en esos 1000 valores es de 23642

Entonces solo tengo que dividir los 6.5s que se tardo en hacer 23642

Que me da que una comparación se tarda: 0.000274934439 s

• ¿Cuántos datos habría que ordenar para que el algoritmo tarde 10s? ¿En general T-segundos?

Haciendo cálculos retomando la anterior pregunta:

Si

23642→6.5s

X→10s

Se necesitan 36372 comparaciones

Ahora 1000→23642

X→36372

Se necesitan 1538 datos más o menos

Entonces se necesitará 15.38 datos por cada segundo

• ¿Cómo se puede comparar los datos que obtengo con otro caso: ¿el peor, el mejor?

Se supone que mi caso es el peor pues están del lado opuesto a como deben de terminar, pero en si mi algoritmo se las arregla para que siempre den los mismos resultados pues como siempre es en orden inverso no cambian los valores. Lo único que puedo cambiar es la cantidad de datos a organizar.

Supongo que el peor es cuando son datos aleatorios pues siempre se contienen diferentes cantidades de comparaciones.

• ¿Cuál creo que es el peor y el mejor caso sustentando en mis experimentos?

El peor caso seria el de valores aleatorios pues se obtendrían diferentes cantidades cada vez.

El mejor creo que es cuando ya están completamente ordenados o casi ordenados y entonces solo se tiene que hacer una cantidad mínima de cambios y comparciones