



**Instituto  
Politécnico  
Nacional**



# **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  $n=10, 20, 30, \dots, 10,000$ . 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

```
1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include <stdlib.h>
5 int main(){
6     int arreglo[100] = {0};
7     int i;
8
9     /*Inicializar*/
10    for(i=0;i<25;i++){
11        arreglo[i]=rand();
12    }
13    for(i=0;i<25;i++){
14        printf("%d\t",arreglo[i]);
15    }
16    return 0;
17 }
```

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

### VERSION 0.1

```
1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include <stdlib.h>
5 #define NUM_MAX 100//Tamaño del arreglo
6 #define NUM_A_ANA 25//datos a ser analizados
7 int main(){
8     int arreglo[NUM_MAX] = {0};
9     int i;
10
11    /*Inicializar*/
12    for(i=0;i<NUM_A_ANA;i++){
13        arreglo[i]=rand();
14    }
15    for(i=0;i<NUM_A_ANA;i++){
16        printf("%d\t",arreglo[i]);
17    }
18    return 0;
19 }
```

- Ponerle nombres a las constantes

## VERSION 0.2

```
1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include "shell.h"
6
7 #define NUM_MAX 100//Tamaño del arreglo
8 #define NUM_A_ANA 25//datos a ser analizados
9 int main(){
10     int arreglo[NUM_MAX] = {0};
11     int i;
12
13     /*Inicializar*/
14     for(i=0;i<NUM_A_ANA;i++){
15         arreglo[i]=NUM_A_ANA - i;
16     }
17
18     if(shell(arreglo,NUM_A_ANA)!=OK){
19         printf("Error al ordenar\n");
20     }
21
22     for(i=0;i<NUM_A_ANA;i++){
23         printf("%d\t",arreglo[i]);
24     }
25
26     return 0;
27 }
```

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

```
1 #ifndef _SHELL_H
2 #define _SHELL_H
3
4 #define OK 0 //No hay error
5
6 int shell(int* arr, int tam);
7
8 #endif
```

Se declaro la función shell

```
1 #include<stdio.h>
2 #include "shell.h"
3
4 int shell(int* arr, int tam){
5     return OK;
6 }
```

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

## VERSION 0.3

```
1  #ifndef _SHELL_H
2  #define _SHELL_H
3  /*****
4  /*      Constantes      */
5  /*****
6  /*No hay error*/
7  #define OK 0
8  /*Apuntador nulo*/
9  #define AP_INV 1
10 /*Tamaño de arreglo invalido*/
11 #define TA_INV 2
12 /*Tamaño Maximo del arreglo*/
13 #define TAM_MAX 10000
14
15 /*****
16 /*      Macros      */
17 /*****
18 /*Apuntador valido, No nulo */
19 #define AP_VAL(a) ((a)!=NULL)
20 /*Tamaño de arreglo valido*/
21 #define TA_VAL(a) (a)>0&&(a<TAM_MAX)
22
23 /*****
24 /*      Ordena Los datos      */
25 /*****+**/
26 #/*Recibe:
27     arr: Arreglo con Los datos
28     tam: número de datos a ordenar
29 Regresa:
30     Costante de error*/
31 int shell(int* arr, int tam);
32 #/*
33 Intercambia Los valores de dos variables
34 Recibe:
35     Apuntador a las variables
36     Intercambiar de valor
37 Regresa:
38     Constante de error*/
39 int swap(int* a, int* b);
40 #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 de la función que intercambia el valor de dos variables

```

1  #include<stdio.h>
2  #include "shell.h"
3  /*Intercambia los valores de dos variables
4  Recibe:
5      Apuntador a las variables
6      Intercambiar de valor
7  Regresa:
8      Constante de error*/
9  int swap(int *a, int *b){
10     int tmp;
11     tmp = *a;
12     *a = *b;
13     *b = tmp;
14     return OK;
15 }
16 /*Ordena Los numeros usando el algoritmo shell
17 en el peor caso que es de forma descendente/orden inverso
18 Recibe: arr-->Arreglo con los datos
19         tam-->Número de datos a ordenar
20 Regresa: Constante de error*/
21 int shell(int* arr, int tam){
22     if(!AP_VAL(arr)){
23         return AP_INV;
24     }
25     if(!TA_VAL(tam)){
26         return TA_INV;
27     }
28     int cont,temp,i;
29     for(cont=tam/2;cont!=0;cont/=2)
30     {
31         for(i=cont;i<tam;i++)
32         {
33             if(arr[i-cont]<arr[i])
34             {
35                 swap(&arr[i],&arr[i-cont]);
36             }
37         }
38     }
39     return OK;
40 }

```

Agregar comentarios similares a los del archivo cabecera.

- Código de la función que intercambia dos valores
- Código de la función que ordena, en este caso el ordenamiento shell

```

1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include "shell.h"
6
7 #define NUM_MAX 100//Tamaño del arreglo
8 #define NUM_A_ANA 25//datos a ser analizados
9 int main(){
10     int arreglo[NUM_MAX] = {0};
11     int i;
12     srand(3);
13     /*Inicializar*/
14     for(i=0;i<NUM_A_ANA;i++){
15         arreglo[i]=NUM_A_ANA - i;
16     }
17
18     if(shell(arreglo,NUM_A_ANA)!=OK){
19         printf("Error al ordenar\n");
20     }
21
22     shell(arreglo,NUM_A_ANA);
23
24     for(i=0;i<NUM_A_ANA;i++){
25         printf("%d\t",arreglo[i]);
26     }
27     return 0;
28 }

```

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

```
1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso:10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include "shell.h"
5
6 #define NUM_MAX 1000//Tamaño del arreglo
7 #define NUM_A_ANA 1000//datos a ser analizados
8
9 int main(){
10     int arreglo[NUM_MAX] = {0};
11     int i,j;
12     int num;
13     /*Inicializar*/
14     for(i=0;i<NUM_A_ANA;i++){
15         //num=rand()%1000;
16         arreglo[i]=NUM_A_ANA - i;
17     }
18     //Imprimir la secuencia de numeros
19     //de forma decreciente que es el peor caso
20     int k;
21     for(k=0;k<NUM_A_ANA;k++){
22         printf("%d\t",arreglo[k]);
23     }
24     printf("\n\n");
25     //Llama a la funcion shell y verifica
26     //si todo salio bien
27     if(shell(arreglo,NUM_A_ANA)!=OK){
28         printf("Error al ordenar\n");
29     }
30     //Llama a la funcion verifica que checa que
31     //todos los numero esten ordenados
32     verifica(arreglo,NUM_A_ANA);
33     return 0;
34 }
```

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.

```

1  #include<stdio.h>
2  #include "shell.h"
3  /*Ordena los numeros usando el algoritmo shell
4  en el peor caso que es de forma descendente/orden inverso
5  Recibe: arr-->Arreglo con los datos
6         tam-->Número de datos a ordenar
7  Regresa: Constante de error*/
8  int shell(int* arr, int tam){
9      if(!AP_VAL(arr)){
10         return AP_INV;
11     }
12     if(!TA_VAL(tam)){
13         return TA_INV;
14     }
15
16     int h, i, j, temp;
17     for (h=tam/2; h>0; h=h/2) {
18         for (i=h; i<tam; i++) { //aplica inserción a partir de h
19             temp=arr[i];
20             for (j=i-h; j>=0 && arr[j]>temp; j-=h) {
21                 arr[j+h]=arr[j];
22             }
23             arr[j+h]=temp; //inserta
24         }
25     }
26     int k;
27     for(k=0;k<tam;k++){
28         printf("%d\t",arr[k]);
29     }
30     return OK;
31 }
32

```

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

```

33  /*Verifica que los datos estan ordenados,
34  en lugar de desplegar los datos,
35  asi evitamos un error al visualizar
36  los datos en pantalla*/
37  int verifica(int* arr, int tam){
38     int verificar=0,i;
39     for(i=0;i<tam - 1;i++){
40         if(arr[i]<=arr[i+1]){
41             verificar++;
42         }
43     }
44     if(verificar!=tam - 1){
45         printf("Error al ordenar %d\n",verificar);
46     }
47     return OK;
48 }

```

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



```

1  #ifndef _SHELL_H
2  #define _SHELL_H
3  /*****
4  /*          Constantes          */
5  *****/
6  /*No hay error*/
7  #define OK 0
8  /*Apuntador nulo*/
9  #define AP_INV 1
10 /*Tamaño de arreglo invalido*/
11 #define TA_INV 2
12 /*Tamaño Maximo del arreglo*/
13 #define TAM_MAX 10000
14
15 /*****
16 /*          Macros          */
17 *****/
18 /*Apuntador valido, No nulo */
19 #define AP_VAL(a)  ((a)!=NULL)
20 /*Tamaño de arreglo valido*/
21 #define TA_VAL(a)  (a)>0&&(a<TAM_MAX)
22
23 /*****
24 /*          Ordena Los datos          */
25 *****/
26 /*Recibe:
27     arr: Arreglo con los datos
28     tam: número de datos a ordenar
29 Regresa:
30     Costante de error*/
31 int shell(int* arr, int tam);
32 /*
33 Intercambia los valores de dos variables
34 Recibe:
35     Apuntador a las variables
36     Intercambiar de valor
37 Regresa:
38     Constante de error*/
39 int swap(int* a, int* b);
40 /*
41 Verifica que los datos estan ordenados,
42 en lugar de desplegar los datos,
43 asi evitamos un error al visualizar
44 los datos en pantalla*/
45 int verifica(int* arr,int tam);
46 #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

```
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
10 9 8 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 9 10
```

```
Símbolo del sistema
C:\Users\Cesar Hernández\Desktop\AlgoEstrucDat\P12\SCR>a
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6
5 4 3 2 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20
```

```
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
30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
15 14 13 12 11 10 9 8 7 6 5 4 3 2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29
30
```

```
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
1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986
985 984 983 982 981 980 979 978 977 976 975 974 973 972
971 970 969 968 967 966 965 964 963 962 961 960 959 958
957 956 955 954 953 952 951 950 949 948 947 946 945 944
943 942 941 940 939 938 937 936 935 934 933 932 931 930
929 928 927 926 925 924 923 922 921 920 919 918 917 916
915 914 913 912 911 910 909 908 907 906 905 904 903 902
901 900 899 898 897 896 895 894 893 892 891 890 889 888
887 886 885 884 883 882 881 880 879 878 877 876 875 874
873 872 871 870 869 868 867 866 865 864 863 862 861 860
```

...

173	172	171	170	169	168	167	166	165	164	163	162	161	160
150	158	157	156	155	154	153	152	151	150	149	148	147	146
145	144	143	142	141	140	139	138	137	136	135	134	133	132
131	130	129	128	127	126	125	124	123	122	121	120	119	118
117	116	115	114	113	112	111	110	109	108	107	106	105	104
103	102	101	100	99	98	97	96	95	94	93	92	91	90
89	88	87	86	85	84	83	82	81	80	79	78	77	76
75	74	73	72	71	70	69	68	67	66	65	64	63	62
61	60	59	58	57	56	55	54	53	52	51	50	49	48
47	46	45	44	43	42	41	40	39	38	37	36	35	34
33	32	31	30	29	28	27	26	25	24	23	22	21	20
19	18	17	16	15	14	13	12	11	10	9	8	7	6
5	4	3	2	1									

1

2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39	40	41	42	43
44	45	46	47	48	49	50	51	52	53	54	55	56	57
58	59	60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83	84	85
86	87	88	89	90	91	92	93	94	95	96	97	98	99
100	101	102	103	104	105	106	107	108	109	110	111	112	113

...

702	703	704	705	706	707	708	709	710	711	712	713	714	715
716	717	718	719	720	721	722	723	724	725	726	727	728	729
730	731	732	733	734	735	736	737	738	739	740	741	742	743
744	745	746	747	748	749	750	751	752	753	754	755	756	757
758	759	760	761	762	763	764	765	766	767	768	769	770	771
772	773	774	775	776	777	778	779	780	781	782	783	784	785
786	787	788	789	790	791	792	793	794	795	796	797	798	799
800	801	802	803	804	805	806	807	808	809	810	811	812	813
814	815	816	817	818	819	820	821	822	823	824	825	826	827
828	829	830	831	832	833	834	835	836	837	838	839	840	841
842	843	844	845	846	847	848	849	850	851	852	853	854	855
856	857	858	859	860	861	862	863	864	865	866	867	868	869
870	871	872	873	874	875	876	877	878	879	880	881	882	883
884	885	886	887	888	889	890	891	892	893	894	895	896	897
898	899	900	901	902	903	904	905	906	907	908	909	910	911
912	913	914	915	916	917	918	919	920	921	922	923	924	925
926	927	928	929	930	931	932	933	934	935	936	937	938	939
940	941	942	943	944	945	946	947	948	949	950	951	952	953
954	955	956	957	958	959	960	961	962	963	964	965	966	967
968	969	970	971	972	973	974	975	976	977	978	979	980	981
982	983	984	985	986	987	988	989	990	991	992	993	994	995
996	997	998	999	1000									

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

```
1  #ifndef _SHELL_H
2  #define _SHELL_H
3  /*****
4  /*      Constantes      */
5  /*****
6  /*No hay error*/
7  #define OK 0
8  /*Apuntador nulo*/
9  #define AP_INV 1
10 /*Tamaño de arreglo invalido*/
11 #define TA_INV 2
12 /*Tamaño Maximo del arreglo*/
13 #define TAM_MAX 10000
14
15 /*****
16 /*      Macros      */
17 /*****
18 /*Apuntador valido, No nulo */
19 #define AP_VAL(a) ((a)!=NULL)
20 /*Tamaño de arreglo valido*/
21 #define TA_VAL(a) (a)>0&&(a<TAM_MAX)
22 /*****
23 /*      Ordena Los datos      */
24 /*****+**/
25 #/*Recibe:
26     arr: Arreglo con Los datos
27     tam: número de datos a ordenar
28 Regresa:
29     Costante de error*/
30 int shell(int* camb, int* comp, int* arr, int tam);
31 /*****
32 /*      Cambia dos datos      */
33 /*****+**/
34 #/*Recibe:
35     a*:Primer valor del arreglo
36     b*:segundo valor del arreglo
37 Regresa:
38     Costante de error*/
39 int swap(int* c, int* a, int* b);
40 //Compara dos datos del arreglo y le suma
41 //uno al contador de comparaciones
42 int comp_c(int* c, int a, int b);
43 //Imprime el arreglo en pantalla
44 void Imprime_Arr(int* arr, int tam);
45 #endif
```

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

```

1  #include<stdio.h>
2  #include "shell.h"
3  //intercambia dos valores del arreglo
4  int swap(int* camb, int* a, int* b){
5      int tmp;
6      (*camb)++;
7      tmp= *a;
8      *a = *b,
9      *b = tmp;
10     return OK;
11 }
12 /*Ordena los numeros usando el algoritmo shell
13 en el peor caso que es de forma descendente/orden inverso
14 Recibe: arr-->Arreglo con los datos
15         tam-->Número de datos a ordenar
16 Regresa: Costante de error*/
17 int shell(int* camb,int* comp,int* arr, int tam){
18     if(!AP_VAL(arr)){
19         return AP_INV;
20     }
21     if(!TA_VAL(tam)){
22         return TA_INV;
23     }
24
25     int salto,pos,cambios;
26     salto = tam/2;
27
28     do{
29         do{
30             cambios = 0;
31             for(pos = salto; pos < tam; pos++){
32                 if(comp_c(comp,arr[pos - salto],arr[pos])){
33                     swap(camb,&arr[pos],&arr[pos - salto]);
34                     cambios++;
35                 }
36             }while(cambios != 0);
37             salto = salto/2;
38         }while(salto != 0);
39         return OK;
40     }
41     //compara dos valores del arreglo
42     int comp_c(int* comp, int a, int b){
43         (*comp)++;
44         return a>b;
45     }
46     //Imprime el arreglo por pantalla
47     void Imprime_Arr(int* arr, int tam){
48         int i;
49         for(i=0;i<tam;i++){
50             printf("%d\t",arr[i]);
51         }
52     }

```

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.

```

1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso(Orden Inverso):10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include <time.h>
5 #include <stdlib.h>
6 #include "shell.h"
7
8 #define NUM_MAX 100//Tamaño del arreglo
9 #define CANT_NUM 100//datos a ser analizados
10 int main(){
11     srand(time(NULL));
12     int arreglo[NUM_MAX] = {0};
13     int i,j,k,n,num,comparacion,cambio,verificar;
14     float promedio;
15     /*Inicializar en Orden Inverso*/
16     for(i=0;i<CANT_NUM;i++){
17         //num=rand()%1000;
18         arreglo[i]=CANT_NUM - i;//CANT_NUM - i;
19     }
20     //Llama a la funcion shell y verifica
21     //si todo salio bien
22     cambio=0;
23     comparacion=0;
24     if(shell(&cambio,&comparacion,arreglo,CANT_NUM)!=OK){
25         printf("Error al ordenar\n");
26     }
27     //Impresion de arreglo
28     Imprime_Arr(arreglo,CANT_NUM);
29     //Verificar que se ordenaron los datos
30     verificar=0;
31     for(i=0;i<CANT_NUM - 1;i++){
32         if(arreglo[i]<=arreglo[i+1]){
33             verificar++;
34         }
35     }
36     if(verificar!=CANT_NUM - 1){
37         printf("Error al ordenar %d\n",verificar);
38     }
39
40     printf("\n\nCambios:%d\n",cambio);
41     printf("Comparaciones:%d\n",comparacion);
42     return 0;
43 }

```

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

## VERSION DEL ANALISIS / VERSION FINAL

```
1 //Algoritmo 3 --> Shell
2 //Caso 2-->Peor caso(Orden Inverso):10,9,8,7,6,5,4,3,2,1
3 #include <stdio.h>
4 #include "shell.h"
5
6 #define NUM_MAX 10010//Tamaño del arreglo
7 int main(){
8     int arreglo[NUM_MAX] = {0};
9     int i,j,k,n,num,comparacion,cambio,verificar,promedio;
10    for(n=10;n<NUM_MAX;n+=10)
11    {
12        /*Inicializar en Orden Inverso*/
13        for(i=0;i<n;i++){
14            //num=rand()%1000;
15            arreglo[i]=n - i;//CANT_NUM - i;
16        }
17        //Llama a la funcion shell y verifica
18        //si todo salio bien
19        cambio=0;
20        comparacion=0;
21        if(shell(&cambio,&comparacion,arreglo,n)!=OK){
22            printf("Error al ordenar\n");
23        }
24        //Impresion de arreglo
25        //Imprime_Arr(arreglo,n);
26        //Verificar que se ordenaron Los datos
27        verificar=0;
28        for(i=0;i<n - 1;i++){
29            if(arreglo[i]<=arreglo[i+1]){
30                verificar++;
31            }
32        }
33        if(verificar!=n - 1){
34            printf("Error al ordenar %d\n",verificar);
35        }
36
37        //printf("\n\nCambios:%d\n",cambio);
38        //printf("Comparaciones:%d\n",comparacion);
39        promedio= cambio + comparacion;
40        printf("%d\t%d\n",n,promedio);
41    }
42    return 0;
43 }
44 }
```

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



```

1  #include<stdio.h>
2  #include "shell.h"
3  //intercambia dos valores del arreglo
4  int swap(int* camb, int* a, int* b){
5      int tmp;
6      (*camb)++;
7      tmp= *a;
8      *a = *b,
9      *b = tmp;
10     return OK;
11 }
12 /*Ordena los numeros usando el algoritmo shell
13 en el peor caso que es de forma descendente/orden inverso
14 Recibe: arr-->Arreglo con Los datos
15         tam-->Número de datos a ordenar
16 Regresa: Costante de error*/
17 int shell(int* camb,int* comp,int* arr, int tam){
18     if(!AP_VAL(arr)){
19         return AP_INV;
20     }
21     if(!TA_VAL(tam)){
22         return TA_INV;
23     }
24
25     int salto,pos,cambios;
26     salto = tam/2;
27     do{
28         do{
29             cambios = 0;
30             for(pos = salto; pos < tam; pos++){
31                 if(comp_c(comp,arr[pos - salto],arr[pos])){
32                     swap(camb,&arr[pos],&arr[pos - salto]);
33                     cambios++;
34                 }
35             }
36         }while(cambios != 0);
37         salto = salto/2;
38     }while(salto != 0);
39     return OK;
40 }
41 //compara dos valores del arreglo
42 int comp_c(int* comp, int a, int b){
43     (*comp)++;
44     return a>b;
45 }
46 //Imprime el arreglo por pantalla
47 void Imprime_Arr(int* arr, int tam){
48     int i;
49     for(i=0;i<tam;i++){
50         printf("%d\t",arr[i]);
51     }
52 }

```

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

Se agregó otra función que cuenta las comparaciones entre valores.

```

1  #ifndef _SHELL_H
2  #define _SHELL_H
3  /*****
4  /*          Constantes          */
5  /*****
6  /*No hay error*/
7  #define OK 0
8  /*Apuntador nulo*/
9  #define AP_INV 1
10 /*Tamaño de arreglo invalido*/
11 #define TA_INV 2
12 /*Tamaño Maximo del arreglo*/
13 #define TAM_MAX 10000
14
15 /*****
16 /*          Macros          */
17 /*****
18 /*Apuntador valido, No nulo */
19 #define AP_VAL(a)  ((a)!=NULL)
20 /*Tamaño de arreglo valido*/
21 #define TA_VAL(a)  (a)>0&&(a<TAM_MAX)
22 /*****
23 /*          Ordena Los datos          */
24 /*****+**/
25 /*Recibe:
26     arr: Arreglo con Los datos
27     tam: número de datos a ordenar
28 Regresa:
29     Costante de error*/
30 int shell(int* camb, int* comp, int* arr, int tam);
31 /*****
32 /*          Cambia dos datos          */
33 /*****+**/
34 /*Recibe:
35     a*:Primer valor del arreglo
36     b*:segundo valor del arreglo
37 Regresa:
38     Costante de error*/
39 int swap(int* c, int*a, int* b);
40 //Compara dos datos del arreglo y le suma
41 //uno al contador de comparaciones
42 int comp_c(int* c, int a, int b);
43 //Imprime el arreglo en pantalla
44 void Imprime_Arr(int* arr, int tam);
45 #endif

```

Se agregaron los  
prototipos de las  
funciones que  
faltaban.



```
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>_
```

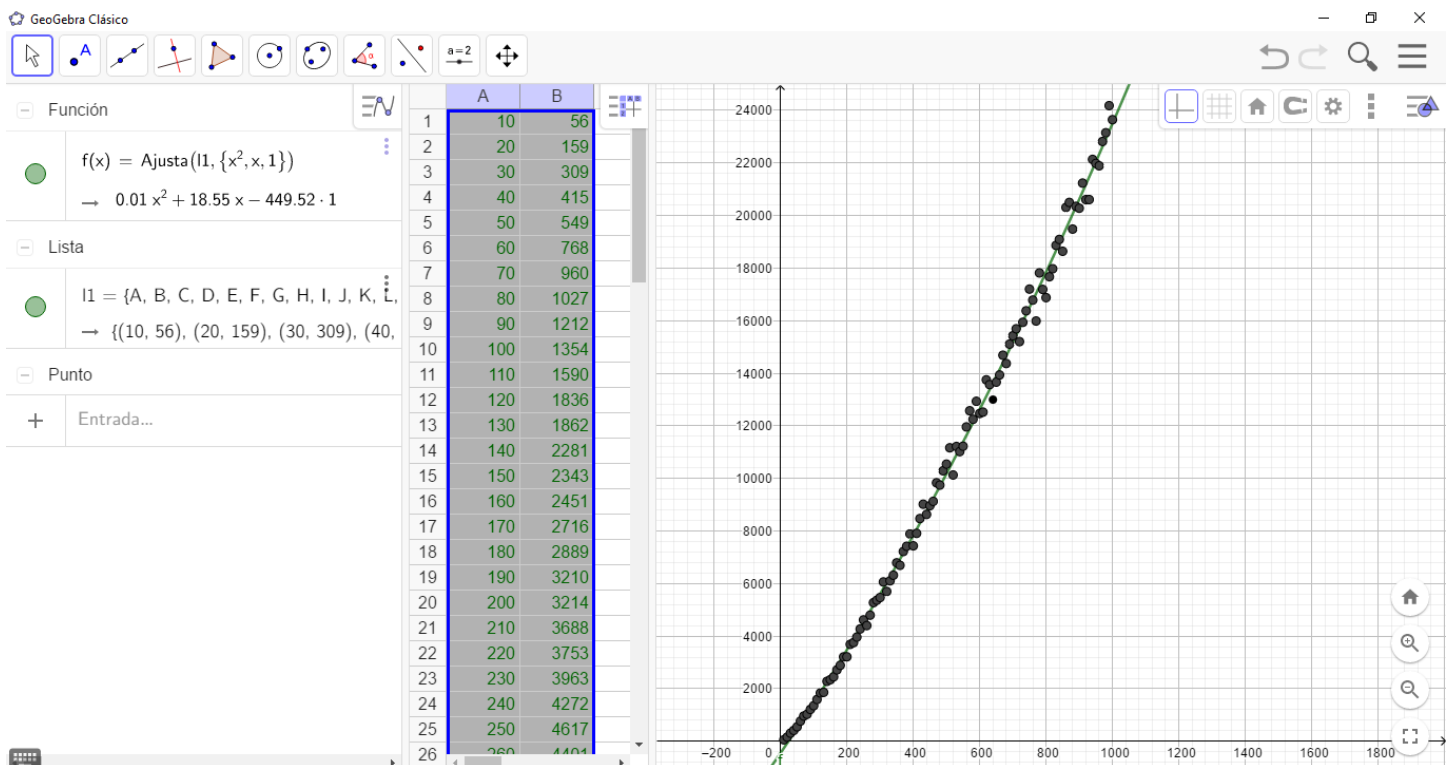
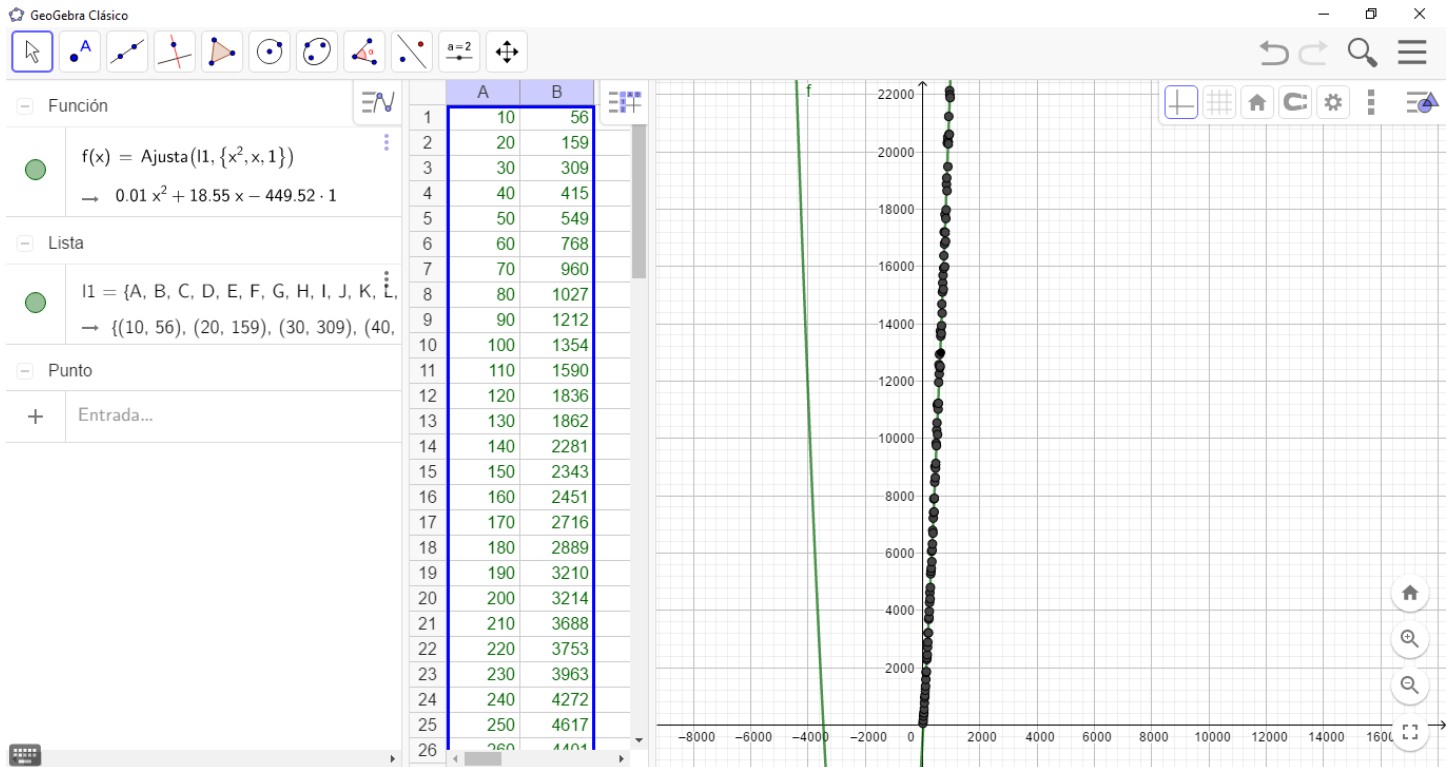
1	10	56	956	9560	303446
2	20	159	957	9570	301908
3	30	309	958	9580	317062
4	40	415	959	9590	311814
5	50	549	960	9600	295968
6	60	768	961	9610	303443
7	70	960	962	9620	304926
8	80	1027	963	9630	313047
9	90	1212	964	9640	306102
10	100	1354	965	9650	317929
11	110	1590	966	9660	315456
12	120	1836	967	9670	318763
13	130	1862	968	9680	307574
14	140	2281	969	9690	315119
15	150	2343	970	9700	320837
16	160	2451	971	9710	329033
17	170	2716	972	9720	318054
18	180	2889	973	9730	296896
19	190	3210	974	9740	313744
20	200	3214	975	9750	306937
21	210	3688	976	9760	300125
22	220	3753	977	9770	307720
23	230	3963	978	9780	309223
24	240	4272	979	9790	317474
25	250	4617	980	9800	316057
26	260	4401	981	9810	309078
27	270	4797	982	9820	325556
28	280	5273	983	9830	324846
29	290	5365	984	9840	312122
30	300	5468	985	9850	319787
31	310	6061	986	9860	317165
32	320	5699	987	9870	325481
33	330	6104	988	9880	313318
34	340	6317	989	9890	311720
35	350	6788	990	9900	328162
36	360	6693	991	9910	321936
37	370	7218	992	9920	320308
38	380	7417	993	9930	328033
39	390	7887	994	9940	329566
40	400	7434	995	9950	337957
41	410	7909	996	9960	330782
42	420	8471	997	9970	323903
43	430	9017	998	9980	340446
44	440	8629	999	9990	315354
45	450	8960	1000	10000	322249
46	460	9126			

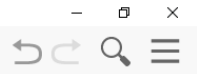
Para correr este código solo se espera como impresión la cantidad de valores y la suma de los cambios y comparaciones que se realizaron. Se redirigió la impresión del código a un archivo salida.txt

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:

$\text{Ajustar}(l1, \{x^2, x, 1\})$

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





Función

$f(x) = \text{Ajusta}(\text{I1}, \{x^2, x, 1\})$   
 $\rightarrow 0.01 x^2 + 18.55 x - 449.52 \cdot 1$

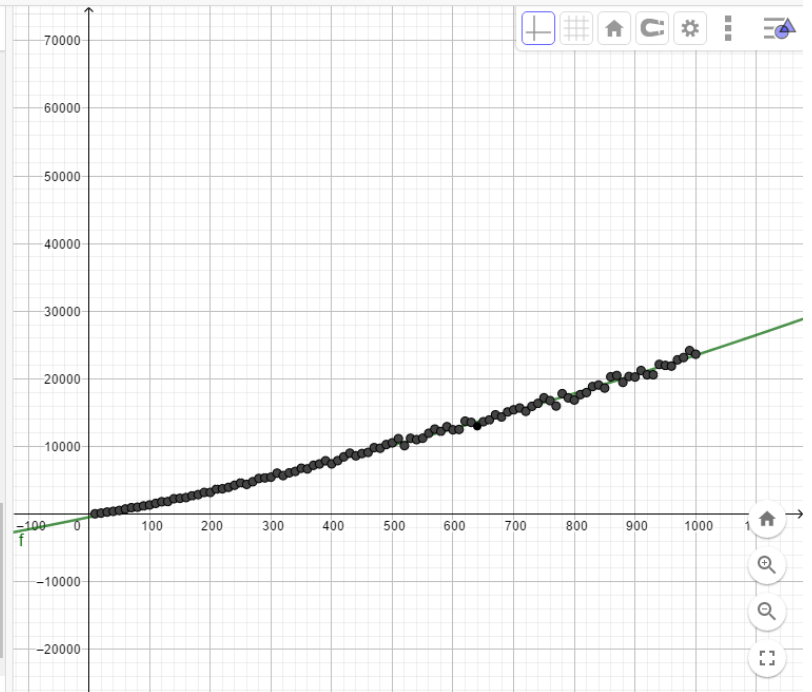
Lista

$\text{I1} = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}, \text{G}, \text{H}, \text{I}, \text{J}, \text{K}, \text{L}, \dots\}$   
 $\rightarrow \{(10, 56), (20, 159), (30, 309), (40, \dots)\}$

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



## 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 caso calculé que mi programa se tardó 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 tardó 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 sería 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 comparaciones