PONTIFICIA UNIVERSIDAD CATÓLICA MADRE Y MAESTRA
FACULTAD DE CIENCIAS DE LA INGENIERÍA
ESCUELA DE INGENIERÍA EN COMPUTACIÓN Y TELECOMUNICACIONES

Asignatura:
Laboratorio de Estructuras de Datos.
Grupo:
ST-ISC-423-171
Periodo Académico:
2020-2021

Práctica #1:
Asignación #1-A

Estudiante:
Junior Hernández 2018-0999

Profesor:
Ing. Jorge A. Luna M.

Fecha de entrega:
6 de noviembre de 2020

Santiago de los caballeros, República Dominicana.

Índice

Introducción

En el siguiente trabajo se presentará y comprobará el funcionamiento de los algoritmos de ordenamiento:

Selection Sort

Insertion Sort

Bubble Sort

Para así poder confirmar que tanto su aplicación práctica comprueba su definición lógica de su forma teórica, con el objetivo de introducir los algoritmos de ordenamiento y su orden de crecimiento, comportamiento a observarse durante la ejecución de los mismo y realización del experimento todo a través de la tabulación de datos y realización de sus graficas para demostrar lo propuesto por lo anterior.

Metodología de los experimentos

Para realizar este análisis sobre los diversos algoritmos de ordenamiento; Selection Sort, Insertion Sort y Bubble Sort, ya que se debe de tomar en cuenta con que se está trabajando, las especificaciones de con lo trabajado son:

- ✓ Laptop Dell
- ✓ Sistema Operativo: Windows 10 Home Single Language
- ✓ Procesador: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz 2.20GHz
- ✓ Memoria instalada (RAM): Sistema operativo de 64 bits, procesador x64
- ✓ Lenguaje de programa: C
- ✓ Compilador: CLion con el IDE por defecto.
- ✓ Microsoft Excel para la representación gráfica

Para generar y tomar el tiempo de trabajo y ejecución de los algoritmos de ordenamiento se procedió a utilizar la librería `#include <time.h>` que viene por defecto, solo teniendo que llamarla y esa generar variables tipo `clock();` o "reloj" en español, para así poder utilizar el siguiente patrón:

```
clock_t start = clock();
//Algoritmos
clock_t stop = clock();
double elapsed = (double) (stop - start) / CLOCKS_PER_SEC;
```

El procedimiento del experimento consistió en dar valores a un puntero con 8 tamaños distintos que van desde 1000 a un 1000000 de elementos, en este caso se eligieron los valores:1000, 5000, 10000, 25000, 50000, 100000, 500000 y 10000000. Para la obtención de resultados un poco más confiables o más exacto, se decidió tomar varias repeticiones para cada uno de los algoritmos y así sacar un promedio de comparaciones, intercambios y sobre todo tiempo. En todo caso para facilitar la hora de obtención de cálculos, se procedió a utilizar la función swap para los intercambios de los arreglos, la función promedioDeArreglo para sacar el promedio de los arreglos y permutation que permite generar valores aleatorios para el arreglo:

```
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void permutation(int array[], long size)
{
    srand(time(NULL));
    int i, j;
    for (i = size - 1; i > 0; i--)
    {
        j = rand() % (i + 1);
        swap( &array[i], &array[j]);
    }
}
double promedioDeArreglo(const double arreglo[], int cantidadDeElementos)
{
    double suma = 0;
    for (int x = 0; x < cantidadDeElementos; x++)
    {
        suma = suma + arreglo[x];
    }
    return suma / cantidadDeElementos;
}
```

Realización de los experimentos

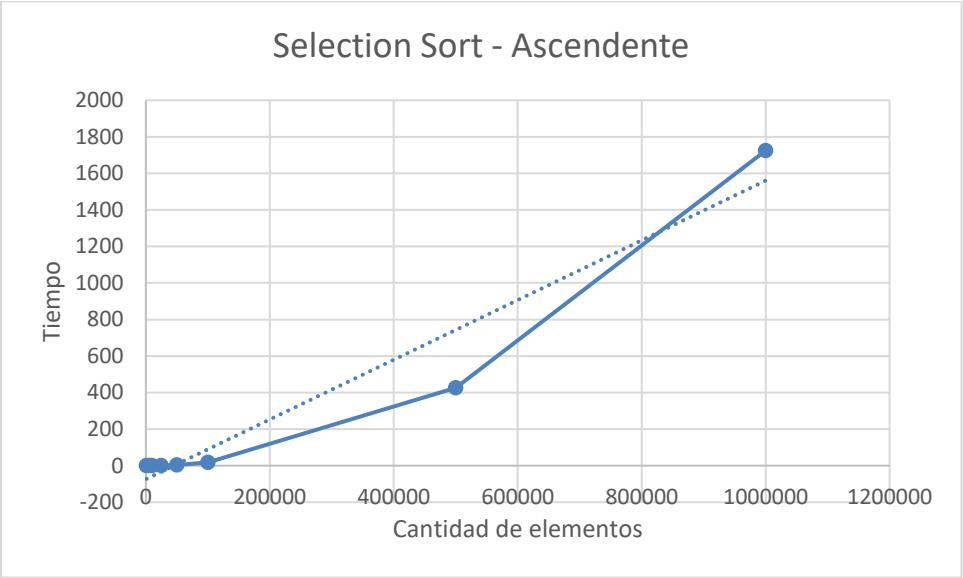
Generación de tiempos de ejecución

Para promediar el mejor de los casos el arreglo no se mezcló, sino que se mantuvo ordenado y los algoritmos solo lo recorrían deshabilitando la función de `permutation`.

```
for(i = 0; i<n; i++) {
    vector[i] = i+1;
}
//permutation(vector,n);
```

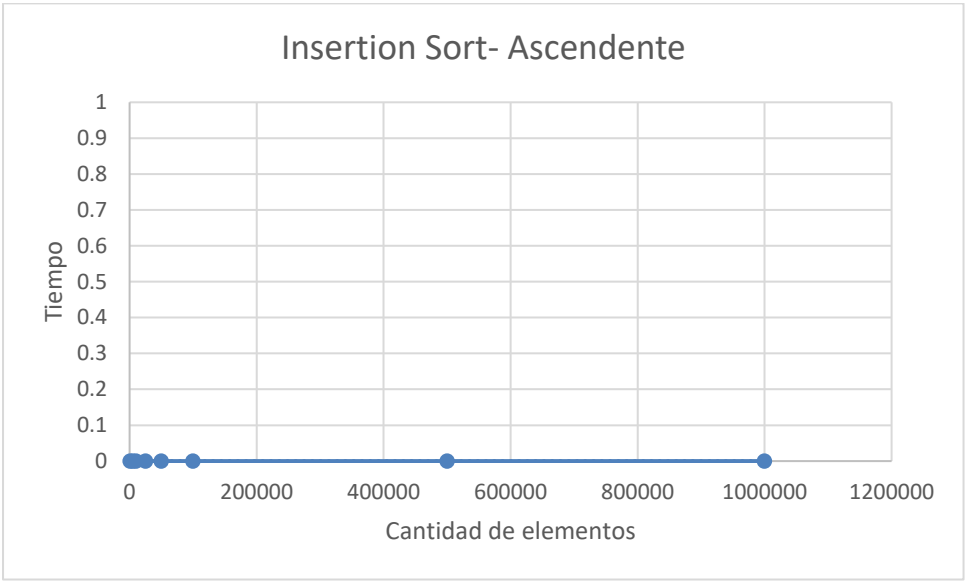
```
void SelectionSort(int *array, int n, double *arreglo, double *arreglo2)
{
    int i, j, position;
    double inter = 0, comp = 0, key;
    for(i = 0; i < (n - 1); i++)
    {
        position=i;
        for(j = i + 1; j < n; j++)
        {
            if(array[position]>array[j]){
                position=j;
                comp ++;
            }
        }
        if(position != i)
        {
            /*key=array[i];
            array[i]=array[position];
            array[position]=key;*/
            swap(&array[i], &array[position]);
            inter ++;
        }
    }
    *arreglo = inter;
    *arreglo2 = comp;
}
```

	Mejor de los casos			
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.002	0	0
2	5000	0.1164	0	0
3	10000	0.2068	0	0
4	25000	1.0836	0	0
5	50000	4.3768	0	0
6	100000	17.3572	0	0
7	500000	425.3664	0	0
8	1000000	1724.7532	0	0



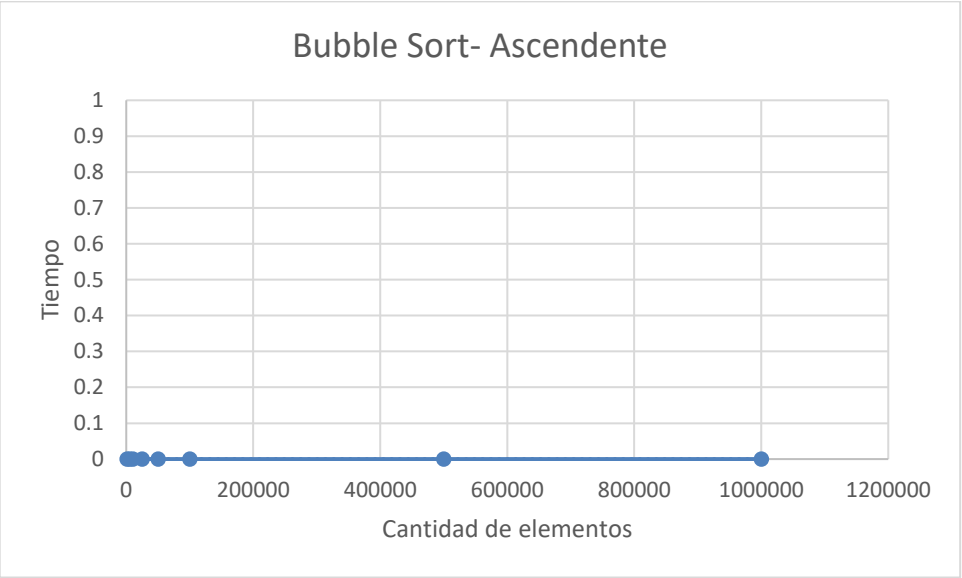

```
void IDirecta(int *array, int n, double *arreglo, double *arreglo2)
{
    int i, key, j;
    double inter = 0, comp = 0;
    for (i = 1; i < n; i++)
    {
        key = array[i];
        j = i - 1;

        while (j >= 0 && array[j] > key)
        {
            array[j + 1] = array[j];
            j = j - 1;
            comp ++;
            inter ++;
        }
        array[j + 1] = key;
        comp ++;
    }
    *arreglo = inter;
    *arreglo2 = comp;
}
```

	Mejor de los casos			
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0	0	2997
2	5000	0	0	19992
3	10000	0	0	59987
4	25000	0	0	154982
5	50000	0	0	354977
6	100000	0	0	754972
7	500000	0	0	2454967
8	1000000	0	0	6454962




```
void Bubblesort(int arreglo[], int tamano,double *arreglo1, double *arreglo2)
{
    long int i,j,flag = 0;
    double inter = 0, comp = 0;
    for(i = 0;i < tamano;i++)
    {
        for(j = 0;j < (tamano-i-1);j++)
        {
            if(arreglo[j] > arreglo[j+1])
            {
                swap(&arreglo[j],&arreglo[j+1]);
                comp++;
                inter++;
                flag = 1;
            }
            comp++;
        }
        if(flag ==0)
        {
            break;
        }
    }
    *arreglo1 = inter;
    *arreglo2 = comp;
}
```

	Mejor de los casos			
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0	0	999
2	5000	0	0	4999
3	10000	0	0	9999
4	25000	0	0	24999
5	50000	0	0	49999
6	100000	0	0	99999
7	500000	0	0	499999
8	1000000	0	0	999999


Bubble Sort- Ascendente
Tiempo
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
200000
400000
600000
800000
1000000
1200000
Cantidad de elementos

Tiempos de ejecución en casos intermedios

Para este si se mezclaron los elementos en el arreglo de forma aleatoria para que los algoritmos los organizaran de manera ascendente habilitando la función de permutaciones.

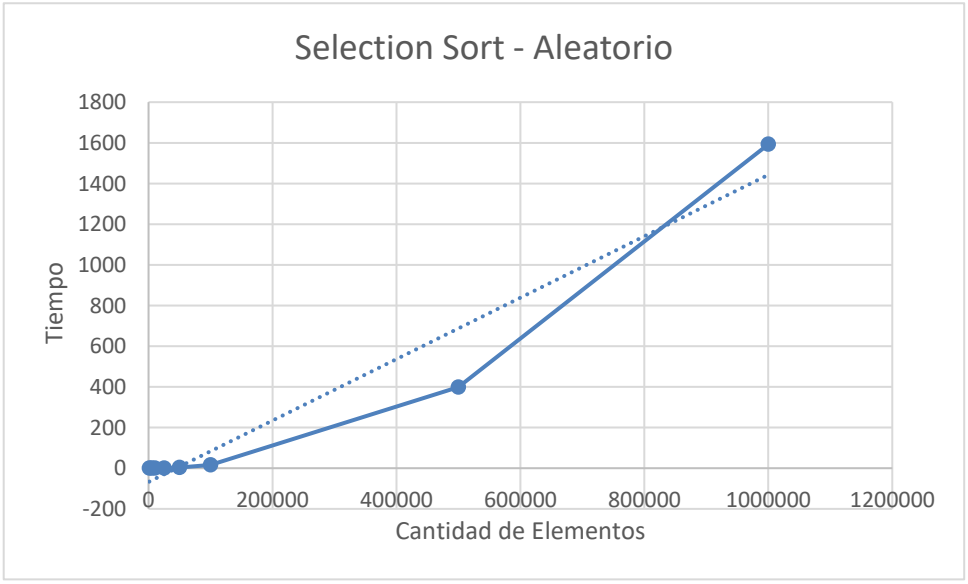
```
for (i = 0; i < n; i++){
    vector[i] = i + 1;
}
permutation(vector,n);
```

Prueba No.1 -Aleatoria				
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.002	2976	16941
2	5000	0.042	19932	134701.4
3	10000	0.1634	59902	442757
4	25000	0.9886	154852	1259265
5	50000	3.9822	354794	3136412
6	100000	15.8382	754738	7200094
7	500000	396.1096	2454684	25891614
8	1000000	1587.3458	6454636	69631333

Prueba No.2 -Aleatoria				
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0026	2976	16059
2	5000	0.0422	19930	132050
3	10000	0.1608	59868	436310
4	25000	1.0122	154797	1246377
5	50000	3.9658	354738	3116860
6	100000	15.8686	754691	7183197
7	500000	396.232	2454643	25791437
8	1000000	1589.604	6454588	69531021

Prueba No.3 -Aleatoria				
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0016	2979	16428
2	5000	0.041	19942.8	134672
3	10000	0.1622	59893	443503
4	25000	0.977	154840	1258774
5	50000	4.138	354789	3128065
6	100000	17.0496	754747	7194349
7	500000	401.8562	2454694	25916446
8	1000000	1600.6892	6454643	69609607

Promedio -Aleatoria				
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.00206	2977	16476
2	5000	0.0417	19935	133808
3	10000	0.1621	59888	440858
4	25000	0.9926	154830	1254805
5	50000	4.0287	354774	3127112
6	100000	16.2521	754725	7192546
7	500000	398.0659	2454674	25866499
8	1000000	1592.5463	6454622	69590654

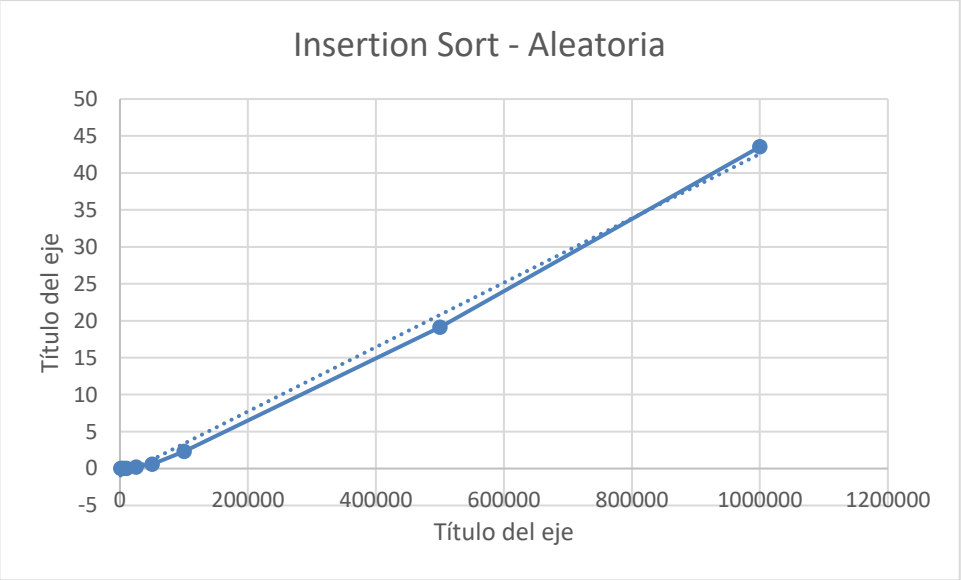


Prueba No.1 -Aleatoria				
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0002	51693	52692
2	5000	0.0152	1255276	1260275
3	10000	0.0358	5133476	5143475
4	25000	0.1456	33150350	33175349
5	50000	0.59	135768154	135818153
6	100000	2.2938	526750347	526850346
7	500000	19.2386	4428888934	4429388933
8	1000000	48.0194	9345135563	9346135562

Prueba No.2 -Aleatoria				
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0002	49376	50375
2	5000	0.0076	1255545	1260544
3	10000	0.0306	5141545	5151544
4	25000	0.1824	33142033	33167032
5	50000	0.6064	136104531	136154530
6	100000	2.3334	525814563	525914562
7	500000	19.0382	4431422796	4431922795
8	1000000	40.4298	9344822208	9345822207

Prueba No.3 -Aleatoria				
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0004	49209	50208
2	5000	0.0284	1256423	1261422
3	10000	0.0356	5116696	5126697
4	25000	0.1734	33035359	33060358
5	50000	0.5994	136220381	136270380
6	100000	2.266	527857025	527957024
7	500000	19.0382	4433296565	4433796564
8	1000000	42.1548	9343359282	9344359281

Promedio -Aleatoria				
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.00026667	50092.6	51091.6
2	5000	0.01706667	1255748	1260747
3	10000	0.034	5130572	5140571
4	25000	0.16713333	33109247.4	33134246.4
5	50000	0.5986	136031021.9	136081020.9
6	100000	2.29773333	526807311.9	527957024.4
7	500000	19.105	4431202765	4431702764
8	1000000	43.5346667	9344439018	9345439017



Prueba No.1 -Aleatoria				
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0038	755106	1498500
2	5000	0.1644	20207398	39990000
3	10000	0.6568	109400401	214970000
4	25000	3.5138	657922658	1252422500
5	50000	13.0228	3027747092	5627322500
6	100000	48.7676	1.23E+10	2.3127E+10
7	500000	752.8252	8.4041E+10	4.0813E+11
8	1000000	2513.2202	2.6855E+11	2.1581E+12

Prueba No.2 -Aleatoria				
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.005	727983	1498500
2	5000	0.1196	20035864	39990000
3	10000	0.4812	109149902	214970000
4	25000	2.8518	656454451	1252422500
5	50000	11.5332	3029727686	5627322500
6	100000	48.035	1.2305E+10	2.3127E+10
7	500000	783.7554	8.4072E+10	4.0813E+11
8	1000000	2571.0888	2.6862E+11	2.1581E+12

Prueba No.3 -Aleatoria				
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0056	744354	1498500
2	5000	0.202	20080545	39990000
3	10000	0.6932	108813864	214970000
4	25000	3.5108	656187324	1252422500
5	50000	15.0294	3026843899	5627322500
6	100000	49.7122	12299658575	2.3127E+10
7	500000	795.4736	84075590865	4.0813E+11
8	1000000	2558.1422	2.68636E+11	2.1581E+12

Promedio-Aleatoria				
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0048	742481	1498500
2	5000	0.162	20107935.67	39990000
3	10000	0.6104	109121388.9	214970000
4	25000	3.29213333	656854811.2	1252422500
5	50000	13.1951333	3028106225	5627322500
6	100000	48.8382667	12301354234	2.3127E+10
7	500000	777.3514	84062883360	4.0813E+11
8	1000000	2547.48373	2.686E+11	2.1581E+12

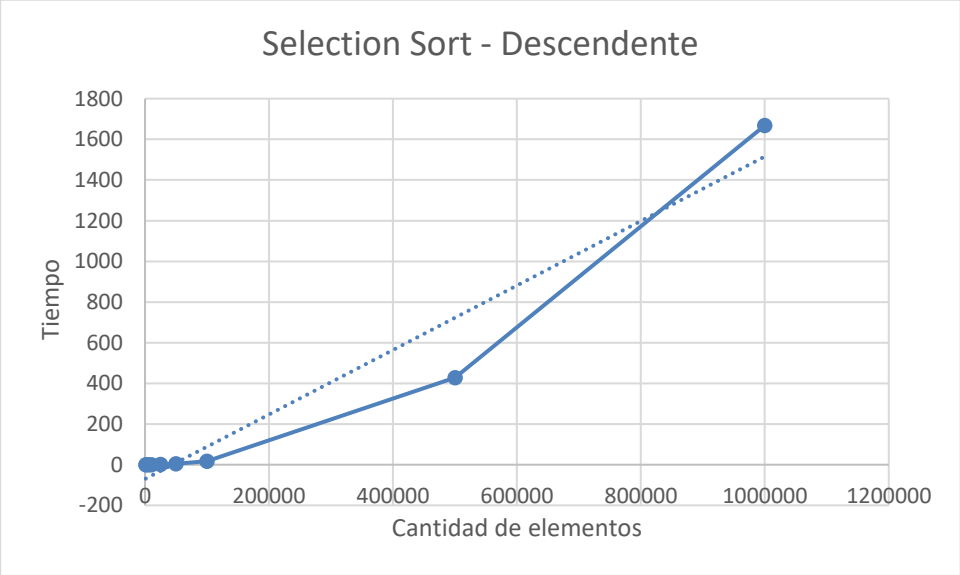

Bubble Sort - Aleatorio
3000
2500
2000
1500
1000
500
0
-500
Tiempo
0
200000
400000
600000
800000
1000000
1200000
Cantidad de elementos

Tiempos de ejecución en casos extremos o peores.

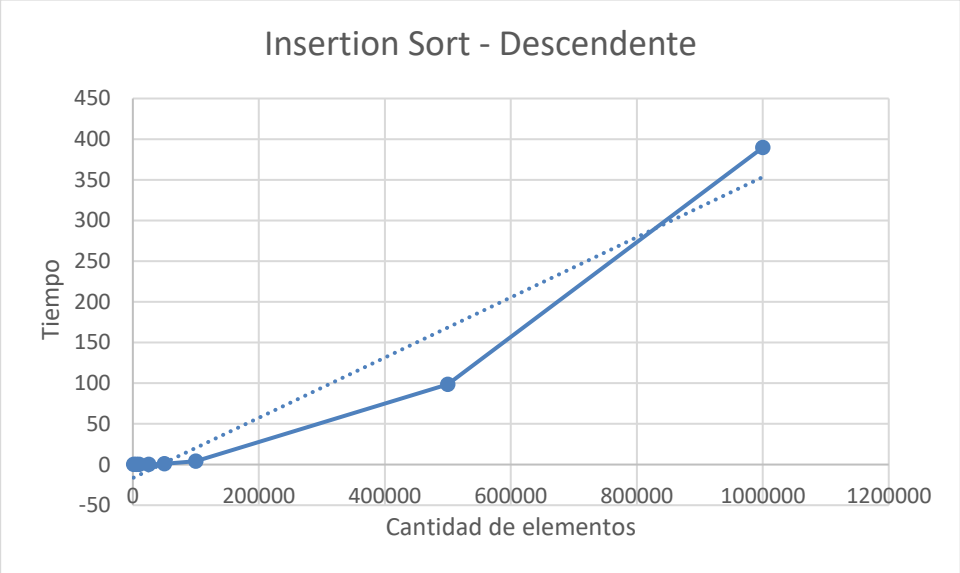
Para este se creó un arreglo con los elementos del mismo de forma descendente para que los algoritmos los organizara de forma ascendente deshabilitando la función de `permutation` y modificando el for que genera los elementos del arreglo y creando una variable nueva llamada baja.

```
/*for (i = 0; i < n; i++)*/for(i = n, baja = 0; i > 0; i--) {
    //vector[i] = i + 1;
    vector[baja] = i - 1;
    baja++;
}
//permutation(vector,n);
```

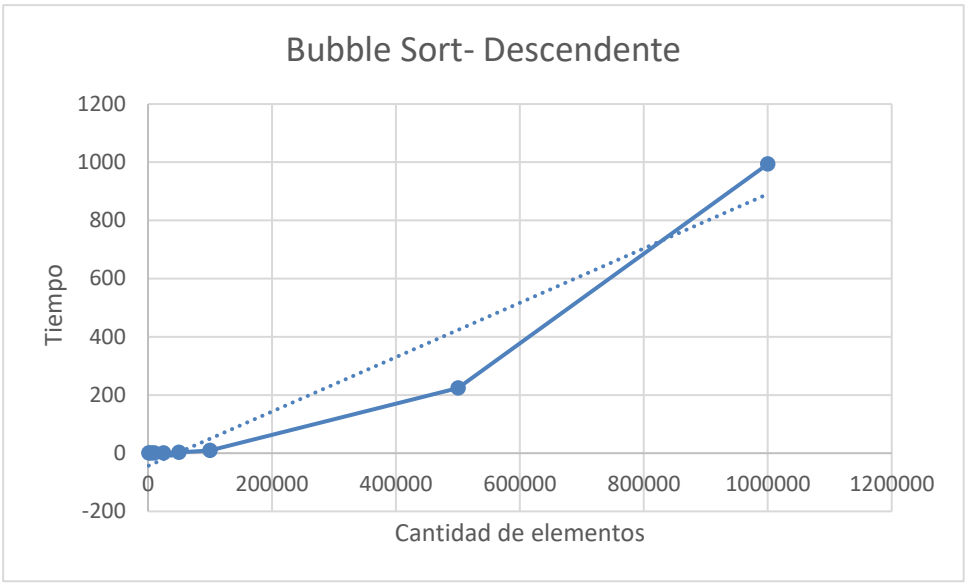
	Peor de los casos			
Selection Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.005	100	50000
2	5000	0.0652	500	1250000
3	10000	0.1778	1000	5000000
4	25000	1.105	2500	31250000
5	50000	4.2254	5000	125000000
6	100000	16.963	10000	500000000
7	500000	428.573	50000	12500000000
8	1000000	1667.9964	100000	50000000000



Peor de los casos				
Insertion Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.0004	99900	100899
2	5000	0.013	2499500	2504499
3	10000	0.0498	9999000	10008999
4	25000	0.2688	62497500	62522499
5	50000	1.0068	249995000	250044999
6	100000	3.9636	999990000	1000089999
7	500000	98.4564	24999950000	25000449999
8	1000000	389.7194	99999900000	1.00001E+11



Peor de los casos				
Bubble Sort				
Num. Muestra	Num. Elementos	Tiempo de ejecución	Swap elements	Comparation
1	1000	0.001	99900	200599
2	5000	0.0272	2499500	5002999
3	10000	0.0942	9999000	20005999
4	25000	0.5564	62497500	125014999
5	50000	2.229	249995000	500029999
6	100000	8.9706	999990000	2000059999
7	500000	223.4018	24999950000	50000299999
8	1000000	993.6042	99999900000	2.00001E+11


Bubble Sort- Descendente
1200
1000
800
600
400
200
0
-200
Tiempo
0
200000
400000
600000
800000
1000000
1200000
Cantidad de elementos

Análisis de resultados.

Comparación de Resultados Teóricos y Prácticos

Para poder observar el comportamiento expuesto de forma teórica por el maestro, se propuso y se realizó la práctica de graficar los datos obtenidos para mostrar el crecimiento del tiempo proporcional al número de elementos que se encuentra bajo el estudio de los algoritmos de ordenamientos, Se puedo ver esta relación con las gráficas mostradas anteriormente en el procedimiento.

Algoritmo de ordenamiento Selection Sort

	Complejidad		
	Mejor de los casos	Caso intermedio	Peor de los casos
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
	Intercambios (Swaps)		
	O(n)	O(n)	O(n)

Observando los datos obtenidos del algoritmo de ordenamiento Selection Sort durante el procedimiento de realización de cada complejidad temporal se puede observar cómo lo teórico correspondió a los resultados prácticos ya que se puede ver en los tres caos de complejidad temporal se observa como su grafico corresponde a una curva de la función cuadrática matemática $O(n^2)$ siendo esta misma constante en los 3 casos. Incluso en el intermedio que hasta su promedio resulto siendo esta grafica cuadrática demostrando que su definición teórica corresponde a su aplicación práctica.
También observando como en el caso promedio con 100000 elementos el tiempo que tomo fue de 16.2521 y su con su siguiente caso que es 500000 tiene 5 veces más sus elementos su tiempo tendría que ser de $t = 16.2521(5^2)$ cuyo resultado es 406.3025 solo teniendo de margen de error de un 0.92% al resultado práctico de 398.0659 demostrando como es consecuente uno con otro.

Algoritmo de ordenamiento Insertion Sort

	Complejidad		
	Mejor de los casos	Caso intermedio	Peor de los casos
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
	Intercambios (Swaps)		
	O(n)	O(n)	O(n)

Al observar los datos que se lograron obtener se puede observar cómo los mismo se encuentran apegados a la teoría, ya que ya tabulados y graficados fueron los que se esperaban obtener tras su prueba de corrida. Observando como dice la teoría que su única complejidad temporal se viene a dar con el mejor de los casos es decir O(n) dando así una línea recta como se observa en la gráfica anterior siendo sus peores casos de complejidad intermedia y ya con el arreglo organizado de forma descendente $O(n^2)$ lo cual es su representación conceptual de su forma teórica. También se puede observar su comportamiento con el caso intermedio donde el margen de error es de un 24.32% demostrando ser uno muy grande en comparación a su “mejor de los casos”, siendo consecuente y apegándose a su forma teórica.

Algoritmo de ordenamiento Bubble Sort

	Complejidad		
	Mejor de los casos	Caso intermedio	Peor de los casos
	O(n)	$O(n^2)$	$O(n^2)$
Bubble Sort	Intercambios (Swaps)		
	O(n)	O(n)	O(n)

En este algoritmo de ordenamiento se apega a su forma teórica ya que en el mismo caso que en el Insertion Sort su mejor de los casos es O(n), ya que este siendo de los algoritmos el que más tiempo llevo de corrida, siendo su diferencia es decir más rápido en el mejor de los casos O(n) observando como en sus otros casos de complejidad aunque sus graficas son cuadráticas se puede ver esas alteraciones que concuerdan a su teoría demostradas tanto en sus tabulaciones como en sus graficas.

Mejor de los casos, peor de los casos y caso intermedio.

Se puede observar la verificación teórica de cada uno de los algoritmos de ordenamiento cumple con su definición teórica, en donde se puede distinguir que en el Selection Sort no existe ningún mejor caso de O(n) siendo solo sus resultados $O(n^2)$ como se observa en sus datos tabulados y graficados tomando la misma duración de tiempo en su hora de corrida, a diferencia de los algoritmos Bubble Sort e Insertion Sort, que aunque Bubble Sort tomaba una gran cantidad de tiempo de duración tomando menos tiempo con la función O(n) mismo caso que pasa con el Insertion Sort que está a diferencia del Bubble Sort tomaba menos tiempo que los dos algoritmos pero fue mucho más rápido más o menos duro lo mismo que el Bubble Sort cuando era la función O(n), es decir cuando el arreglo ya estaba ordenado y su función resultaba esta misma (O(n)) representada tanto en sus tabulaciones como en sus graficas.

Numero de comparaciones e intercambios

Como se observa en los datos de las tablas, el algoritmo que tuvo el mayor número de comparaciones del algoritmos fue Bubble Sort aunque siendo sus números de intercambios más pequeños ya que en el bucle el if se fue ejecutando cada vez que se recorría el algoritmo resultando también el porqué de su duración tan larga de tiempo en terminar de ejecutarse a diferencia del Insertion Sort en donde este solo recorría el algoritmo para intercambiar los elementos y ordenarlos, si estos estaban desordenados en el caso de su comparación no era muy necesaria ya que se hacía en el bucle creado y el bucle solo recorría, en este decide colocar el comparador en los dos bucles ya que tomando en cuenta la falta de if del algoritmo en tema de comparación solo se realizaban a través de bucles por ello es que se ve su sumatoria. En el Selection Sort su caso los swaps y comparaciones se puede ver sus alteraciones en los datos en donde si el arreglo estaba ya ordenado este no realizaba ninguna comparación ni intercambio y si estaba de forma descendente sus comparaciones tomaba dependían del tamaño del arreglo por ello sus resultados eran más o menos del mismo tamaño del arreglo y solo realizaba la menor cantidad de intercambios, pero siempre durando la misma cantidad de tiempo en ejecución.

Conclusión

Con la realización de esta práctica se logró cumplir con el objetivo planteado de la misma, a través de los elementos especificados en el ambiente de trabajo en donde se ejecutaron y se tomó el tiempo de los algoritmos de ordenamiento, además del análisis de los resultados logrando comparar los resultados prácticos con los resultados teóricos siendo los mismos los esperados con la realización de la misma.

Se pudo afirmar los conceptos teóricos de los algoritmos de ordenamiento Bubble Sort, Selection Sort e Insertion Sort, a través de su aplicación práctica y toma de tiempo donde se distingue que el Selection Sort no posee mejor de los casos, ya que consiste en recorrer todo el algoritmo sin importar que este ordenado o no para encontrar el menor de los números en donde se implementación siempre constante será de $O(n^2)$ como se muestra en sus tabulaciones y la curva representada en sus gráficas, a diferencia de sus compañeros algorítmicos Insetion, cada elemento se va insertando en su lugar para organizarlo y Bubble Sort, que revisa cada elemento de la lista, en donde estos algoritmos demostraron que el mejor de los caos para los dos siempre será O(n), confirmado lo propuesto por el docente donde el mismo relativa que “el mejor de los casos siempre será O(n) y que se notara en la duración de los algoritmos, pero OJO! esto no aplica de igual forma para los 3” confirmando lo dicho a través de sus gráficas y tomas de tiempo.