	Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia	

Laboratorios de computación salas A y B

Profesor : Edgar Tista García

Asignatura: EDA 2

Grupo: 4

No. Práctica : 1

Integrante: Chagoya Gonzalez Leonardo

No. Equipo de Cómputo: N/A

No. de Lista o Brigada: N/A

Semestre: 2022-2

Fecha de entrega: 14 de febrero del 2022

Observaciones:

CALIFICACIÓN: _____

Objetivo general: El estudiante identificará la estructura de los algoritmos de ordenamiento InsertionSort, SelectionSort y BubbleSort

Ejercicio 1. Análisis Inicial

El concepto de capas de abstracción permite al programador elaborar sus propias bibliotecas basado en la solución de problemas específicos para poder elaborar soluciones más complejas.

a) Verifica los archivos proporcionados, junto con sus respectivas implementaciones:

- ✓ `utilerias.h`
- ✓ `utilerías.c`
- ✓ `ordenamientos.h`
- ✓ `ordenamientos.c`

utilerias.h : Contiene los prototipos de las funciones `swap`, `printArray`, `printSubArray`.

ordenamientos.h : Contiene los prototipos de las funciones `selectionSort` e `insertionSort`.

utilerías.c Contiene las funciones:

swap: Recibe como parámetros las direcciones de dos variables del tipo `int` e intercambia el contenido haciendo uso de la indirección.

printArray: Recibe como parámetros la dirección del primer elemento de un arreglo de enteros y su tamaño, imprime el contenido de todo arreglo.

printSubArray: Recibe como parámetros la dirección del primer elemento de un arreglo de enteros, dos variables enteras `low` y `high`, imprime el contenido del arreglo desde `low` hasta `high`.

ordenamientos.c Contiene las funciones

selectionSort: Dicha función recibe como parámetros la dirección del primer elemento de un arreglo de enteros y su tamaño, realiza el funcionamiento de `selectionSort` ascendente revisado en clase (recorre el arreglo buscando la posición del elemento más pequeño y lo intercambia con el elemento ubicado en la posición dada por el número de iteración correspondiente).

insertionSort: Recibe como parámetros la dirección del primer elemento de un arreglo de entero y su tamaño, realiza el funcionamiento de `insertionSort` ascendente revisado en clase (todos los elementos a la izquierda del elemento en turno están ordenados y lo que se busca es la posición en la cual debe ser insertado el elemento en turno).

b) Agrega en los archivos correspondientes el código del algoritmo BubbleSort

El prototipo de la función bubbleSort fue agregado en ordenamiento.h y el código de bubble Sort en ordenamiento.c

```
43
44 void bubbleSort(int a[], int size) {
45     int i,j,n;
46     n = size;
47     for(i=n-1; i>0; i--) {
48         for(j=0; j<i; j++) {
49             if(a[j]>a[j+1])
50                 swap(&a[j], &a[j+1]);
51         }
52     }
53 }

1 void selectionSort(int arreglo[], int n);
2 void insertionSort(int a[], int n);
3 void bubbleSort(int a[], int size);
```

c) Modifica el código para que bubble sort se detenga en el momento que la lista se encuentre ordenada

```
void bubbleSort(int a[], int size) {
    int i,j,n,ordenado;
    ordenado = 0;
    n = size;
    while(i>0 && ordenado == 0) {
        ordenado = 1;
        for(j=0; j<i; j++) {
            if(a[j]>a[j+1]) {
                swap(&a[j], &a[j+1]);
                ordenado = 0;
            }
        }
        i--;
    }
}
```

Se colocó la variable entera “ordenado” y se inicializó con un valor = 0 esto para que funcione como indicador si el arreglo ya esté ordenado, para ello el primer ciclo se cambió por un while y verifica en cada iteración si el arreglo se encuentra desordenado (ordenado ==0) en caso afirmativo seguirá con la siguiente iteración.

Al inicio de cada iteración se parte de la premisa que el arreglo ya está ordenado (ordenado = 1) en caso afirmativo no realiza ningún intercambio y en caso de que sea negativo se realizará un intercambio es aquí donde el valor de ordenado cambiará a 0.

Ejercicio 2. Probando los ordenamientos

a) Crea un nuevo proyecto para lenguaje c (Puede ser un proyecto de Xcode, codeblocks, o Dev c++). Elabora un programa en el que, utilizando las bibliotecas proporcionadas, utilices los algoritmos que se encuentran en la biblioteca de ordenamientos

- ✓ Se deberá crear un arreglo de 20 elementos, para ello se dará al usuario la opción de ingresar los valores o bien, que se llenan con valores aleatorios (0-999)
- ✓ Mediante un menú de usuario, se podrá elegir el algoritmo de ordenamiento deseado. (insertion, selection, bubble)
- ✓ Una vez elegido, el programa deberá mostrar las modificaciones que se hacen al arreglo en cada iteración y el arreglo final ordenado

Mediante una función *menuArray* se da las opciones de carga para el arreglo y según sea el caso realiza alguna de las dos siguientes funciones:

La carga del array aleatorio se realizó en una función llamada *cargarArrayAleatorio* por medio de la semilla rand (para utilizar esta función se agregó la biblioteca time.h y stdlib.h en utilidades.c), para delimitar los resultados de 0 a 999 se utilizó el operador % que devuelve el resto de la división entera entre el número que arroja rand() y 1000

```
]void cargarArrayAleatorio(int *array) {  
    srand(time(NULL));  
    int i;  
    for(i=0; i<20; i++) {  
        array[i] = rand()%1000;  
    }  
}
```

La carga del array por el usuario se realiza en la función *cargarArrayManual*

Una vez cargado el array se muestra al usuario un menú de opciones para elegir el método de ordenamiento de su preferencia: bubbleSort , insertionSort o selectionSort, únicamente se hace uso de las funciones con los nombres de dichos algoritmos que fueron proporcionados y el algoritmo bubbleSort mejorado, mostrado en este reporte previamente

Pruebas de ejecución

Carga del arreglo por teclado

```
-----EJERCICIO 2-----  
Creacion de arreglo, elija una opcion  
1)Cargar numeros ingresandolos tu mismo      2)Generar numeros aleatorios      1  
Digite el numero de la posicion 1: 123  
Digite el numero de la posicion 2: 67  
Digite el numero de la posicion 3: 17  
Digite el numero de la posicion 4: 10  
Digite el numero de la posicion 5: 11  
Digite el numero de la posicion 6: 333  
Digite el numero de la posicion 7: 69  
Digite el numero de la posicion 8: 80  
Digite el numero de la posicion 9: 2  
Digite el numero de la posicion 10: 40  
Digite el numero de la posicion 11: 1  
Digite el numero de la posicion 12: 999  
Digite el numero de la posicion 13: 3  
Digite el numero de la posicion 14: 77  
Digite el numero de la posicion 15: 55  
Digite el numero de la posicion 16: 30  
Digite el numero de la posicion 17: 666  
Digite el numero de la posicion 18: 152  
Digite el numero de la posicion 19: 30  
Digite el numero de la posicion 20: 77_
```

```
El arreglo es: 123 67 17 10 11 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77
```

```
-----Menu-----  
Elija el algortimo de ordenamiento  
1)InsertionSort  2)SelectionSort  3)BubbleSort
```

Para efectos visuales una vez que se termina de ingresar por teclado el último elemento del arreglo hace una limpieza de pantalla y únicamente muestra el arreglo ya cargado, junto con el menú de ordenamientos

Elección del ordenamiento insertionSort

```
-----Menu-----
Elija el algortimo de ordenamiento
1)InsertionSort  2)SelectionSort  3)BubbleSort  1

Iteracion numero 1
67 123 17 10 11 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 2
17 67 123 10 11 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 3
10 17 67 123 11 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 4
10 11 17 67 123 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 5
10 11 17 67 123 333 69 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 6
10 11 17 67 69 123 333 80 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 7
10 11 17 67 69 80 123 333 2 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 8
2 10 11 17 67 69 80 123 333 40 1 999 3 77 55 30 666 152 30 77

Iteracion numero 9
2 10 11 17 40 67 69 80 123 333 1 999 3 77 55 30 666 152 30 77

Iteracion numero 10
1 2 10 11 17 40 67 69 80 123 333 999 3 77 55 30 666 152 30 77

Iteracion numero 11
1 2 10 11 17 40 67 69 80 123 333 999 3 77 55 30 666 152 30 77

Iteracion numero 12
1 2 3 10 11 17 40 67 69 80 123 333 999 77 55 30 666 152 30 77

Iteracion numero 13
1 2 3 10 11 17 40 67 69 77 80 123 333 999 55 30 666 152 30 77

Iteracion numero 14
1 2 3 10 11 17 40 55 67 69 77 80 123 333 999 30 666 152 30 77

Iteracion numero 15
1 2 3 10 11 17 30 40 55 67 69 77 80 123 333 999 666 152 30 77
```

```
Iteracion numero 16
1 2 3 10 11 17 30 40 55 67 69 77 80 123 333 666 999 152 30 77

Iteracion numero 17
1 2 3 10 11 17 30 40 55 67 69 77 80 123 152 333 666 999 30 77

Iteracion numero 18
1 2 3 10 11 17 30 30 40 55 67 69 77 80 123 152 333 666 999 77

Iteracion numero 19
1 2 3 10 11 17 30 30 40 55 67 69 77 77 80 123 152 333 666 999

El arreglo ordenado es: 1 2 3 10 11 17 30 30 40 55 67 69 77 77 80 123 152 333 666 999
```

Al final se observa que fue correctamente ordenado (orden ascendente) nuestro arreglo con números aleatorios

Elección de generar números aleatorios

```
-----EJERCICIO 2-----  
  
Creacion de arreglo, elija una opcion  
  
1)Cargar numeros ingresandolos tu mismo      2)Generar numeros aleatorios    2_
```

Elección del ordenamiento selectionSort

```
El arreglo es: 20 769 703 435 244 530 310 343 180 768 33 483 649 771 554 34 531 641 734 910  
  
-----Menu-----  
Elija el algoritmo de ordenamiento  
1)InsertionSort  2)SelectionSort  3)BubbleSort    2_
```

```
Iteracion numero 1  
20 769 703 435 244 530 310 343 180 768 33 483 649 771 554 34 531 641 734 910  
  
Iteracion numero 2  
20 33 703 435 244 530 310 343 180 768 769 483 649 771 554 34 531 641 734 910  
  
Iteracion numero 3  
20 33 34 435 244 530 310 343 180 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 4  
20 33 34 180 244 530 310 343 435 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 5  
20 33 34 180 244 530 310 343 435 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 6  
20 33 34 180 244 310 530 343 435 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 7  
20 33 34 180 244 310 343 530 435 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 8  
20 33 34 180 244 310 343 435 530 768 769 483 649 771 554 703 531 641 734 910  
  
Iteracion numero 9  
20 33 34 180 244 310 343 435 483 768 769 530 649 771 554 703 531 641 734 910  
  
Iteracion numero 10  
20 33 34 180 244 310 343 435 483 530 769 768 649 771 554 703 531 641 734 910  
  
Iteracion numero 11  
20 33 34 180 244 310 343 435 483 530 531 768 649 771 554 703 769 641 734 910  
  
Iteracion numero 12  
20 33 34 180 244 310 343 435 483 530 531 554 649 771 768 703 769 641 734 910
```

```

Iteracion numero 13
20 33 34 180 244 310 343 435 483 530 531 554 641 771 768 703 769 649 734 910

Iteracion numero 14
20 33 34 180 244 310 343 435 483 530 531 554 641 649 768 703 769 771 734 910

Iteracion numero 15
20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 768 769 771 734 910

Iteracion numero 16
20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 734 769 771 768 910

Iteracion numero 17
20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 734 768 771 769 910

Iteracion numero 18
20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 734 768 769 771 910

Iteracion numero 19
20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 734 768 769 771 910

El arreglo ordenado es: 20 33 34 180 244 310 343 435 483 530 531 554 641 649 703 734 768 769 771 910

```

Al final podemos verificar el funcionamiento del algoritmo de selectionSort en forma ascendente

b) Realiza la verificación con su respectiva evidencia de que la modificación realizada a bubble sort funciona correctamente

Se verifica que ingresando un arreglo ordenado bubbleSort realiza solo 1 iteración y termina pues detecta que el arreglo ya está ordenado

```

El arreglo es: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

-----Menu-----
Elija el algortimo de ordenamiento
1)InsertionSort  2)SelectionSort  3)BubbleSort  3

Iteracion numero 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

El arreglo ordenado es: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

```


Ejercicio 3. Análisis de la Complejidad computacional

✓ Agrega en los algoritmos de ordenamiento, las instrucciones necesarias para contabilizar el número de operaciones del algoritmo (comparaciones, intercambios, inserciones) que sean considerables en el análisis de complejidad temporal.

✓ Realiza pruebas con arreglos de diferente tamaño (llenar los valores con números

aleatorios) 50, 100, 500, 800, 1000, 2000*, 5000*, 10'000*

*si tu computadora lo permite.

✓ Realiza al menos 5 ejecuciones en cada uno de los tamaños mencionados y muestra una tabla general con promedio del número de operaciones en cada uno.

✓ Realiza gráficas (en el programa que tu prefieras) de la cantidad de operaciones que se realizan de acuerdo con el tamaño de la entrada y explica tus resultados (eje x: tamaño del arreglo, eje y: número de operaciones).

✓ En estas gráficas observarás la complejidad de cada algoritmo.

✓ Deberás agregar capturas de pantalla de los aspectos más relevantes de las pruebas realizadas.

Para el conteo del número de operaciones realizadas se tomaron en cuenta comparaciones, asignaciones e intercambios, la función swap involucra 3 asignaciones pues realiza un intercambio. La variable que estará realizando el conteo de operaciones realizadas se colocó en las funciones insertionSort selectionSort y bubbleSort definida como *contador*

InsertionSort

Se realizaron 5 pruebas por cada tamaño mencionado

InsertionSort						
Tamaño del arreglo	50	100	500	800	1000	2000
Prueba1(Num Operaciones)	1317	4697	126359	302895	500661	1983905
Prueba2 (Num operaciones)	1325	6087	123605	324365	494217	2013951
Prueba 3(Num Operaciones)	1265	5643	125685	321281	499619	2038663
Prueba4(Num Operaciones)	1499	4591	121341	314395	499681	1976865
Prueba5(Num operaciones)	1339	5219	123251	337263	498493	2000037
Promedio	1349	5247,4	124048,2	320039,8	498534,2	10013421

Cabe destacar que se pudo correr el algoritmo con un tamaño de 5000 sin embargo el tiempo de ejecución supera los 15 minutos y contando.

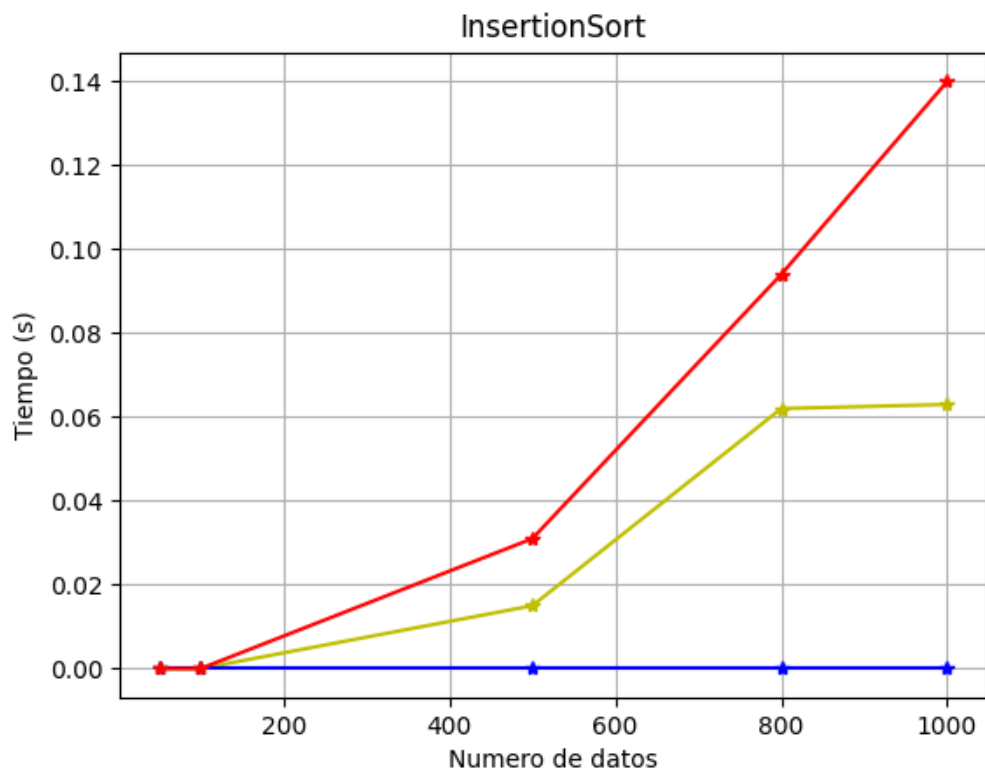


Tamaño del arreglo	Num operaciones
50	1349
100	5247
500	124048
800	320040
1000	498534
2000	2002684

En esta gráfica podemos ver el comportamiento de un algoritmo con orden de complejidad $O(n^2)$. A medida que el tamaño del arreglo incrementa el número de operaciones crece de notable forma haciendo que la curva que representa dicho comportamiento se asemeje a una parábola.

El algoritmo también fue implementado en python y se realizaron pruebas con los tamaños de arreglos de 50,100,200,500,800,1000 tomando el tiempo de ejecución para cada uno de estos

```
def insertionSort(array):
    print("-----Algoritmo de insertion-----")
    n = len(array)
    for i in range(1,n):
        j=i
        aux = array[i]
        while(j>0 and aux<array[j-1]):
            array[j]=array[j-1]
            j=j-1
        array[j]=aux
        #print("Iteracion numero ",i)
        #imprimir(array)
```



MEJOR CASO(LISTA ORDENADA DE FORMA ASCENDENTE)

CASO PROMEDIO (GENERADO NÚMEROS ALEATORIOS)

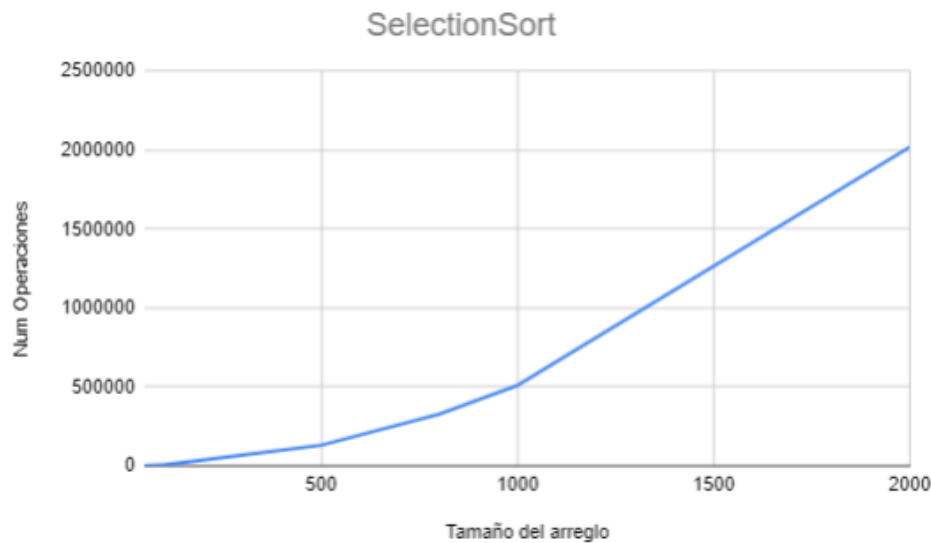
PEOR CASO(LISTA ORDENADA DE FORMA DESCENDENTE)

En esta gráfica de número de datos contra tiempo podemos observar un comportamiento esperado donde el mejor caso no haga muchas operaciones por lo que su línea de tiempo es mucho menor que la del peor caso y el caso promedio se encuentra entre estas dos.

SelectionSort

Se realizaron 5 pruebas por cada tamaño mencionado

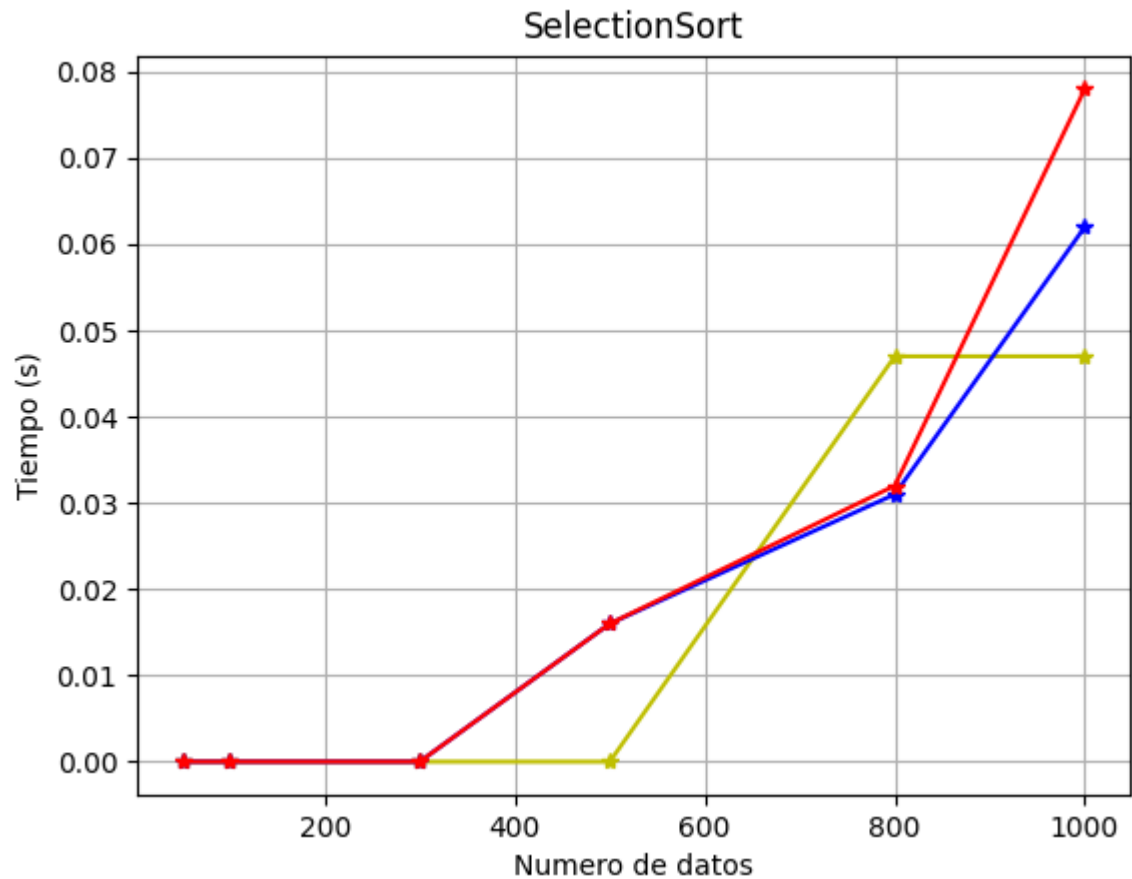
SelectionSort						
Tamaño del arreglo	50	100	500	800	1000	2000
Prueba1(Num Operaciones)	1584	5744	129602	327540	509610	2020534
Prueba2 (Num operaciones)	1570	5752	129480	327823	509862	2020342
Prueba 3(Num Operaciones)	1584	5799	129598	327742	509819	2020647
Prueba4(Num Operaciones)	1588	5796	129644	327571	509911	2020959
Prueba5(Num operaciones)	1589	5735	129537	327996	509684	2020556
Promedio	1583	5765,2	129572,2	327734,4	509777,2	2020607,6



En esta gráfica se observa el comportamiento del algoritmo con orden de complejidad $O(n^2)$ pues a medida que el tamaño del arreglo incrementa se puede observar el comportamiento de una parábola de mejor manera. Tanto la curva y el número de operaciones de insertionSort y SelectionSort se parecen demasiado en casos promedio.

Algoritmo en python

```
def selectionSort(array):
    n = len(array)
    for i in range(0,n-1):
        indiceMenor=i
        for j in range(i+1,n):
            if(array[j]<array[indiceMenor]):
                indiceMenor=j
        if(i!=indiceMenor):
            array[i],array[indiceMenor] = array[indiceMenor],array[i] #realiza el intercambio de valores
        #print("Iteracion numero ",i)
        #imprimir(array)
```



MEJOR CASO(LISTA ORDENADA DE FORMA ASCENDENTE)

CASO PROMEDIO (GENERADO NÚMEROS ALEATORIOS)

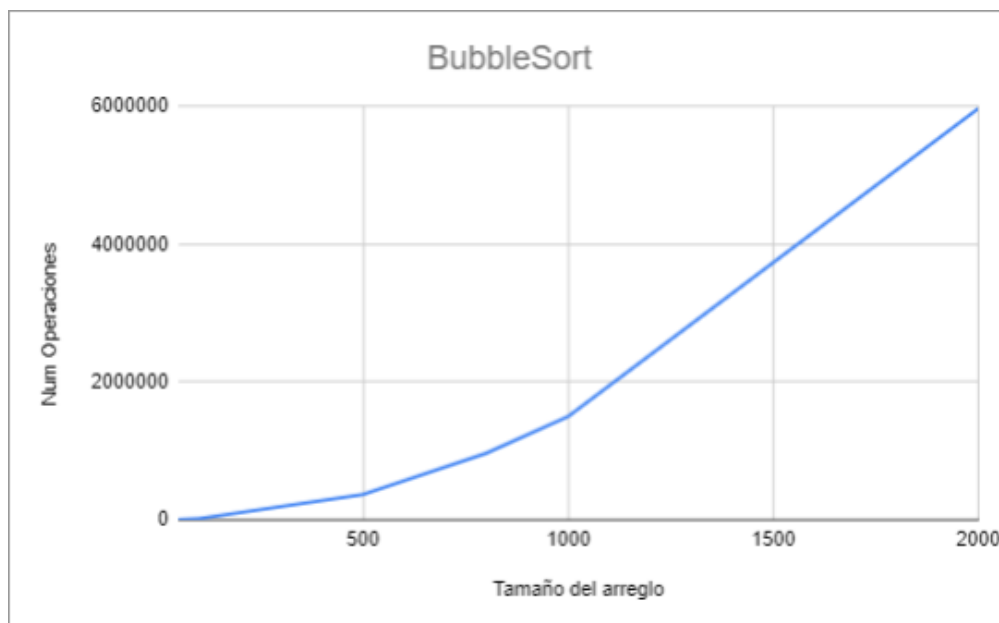
PEOR CASO(LISTA ORDENADA DE FORMA DESCENDENTE)

Hice varias pruebas del tiempo que se llevaba ejecutar el algoritmo sin embargo siempre arroja resultados diferentes no eran constantes las líneas de mejor caso, peor caso y caso promedio.

BubbleSort

Se realizaron 5 pruebas por cada tamaño de elementos del array y se calculó su promedio

Tamaño del arreglo	50	100	500	800	1000	2000
Prueba1(Num Operaciones)	3825	15173	369180	949293	1496499	5986819
Prueba2 (Num operaciones)	4206	15253	365117	970912	1493572	5909851
Prueba 3(Num Operaciones)	3945	14364	378768	972130	1509383	6000339
Prueba4(Num Operaciones)	4041	14764	364712	974402	1493973	6066031
Prueba5(Num operaciones)	3702	14675	356109	960147	1521820	5908168
Promedio	3943,8	14845,8	366777,2	965376,8	1503049,4	5974241,6

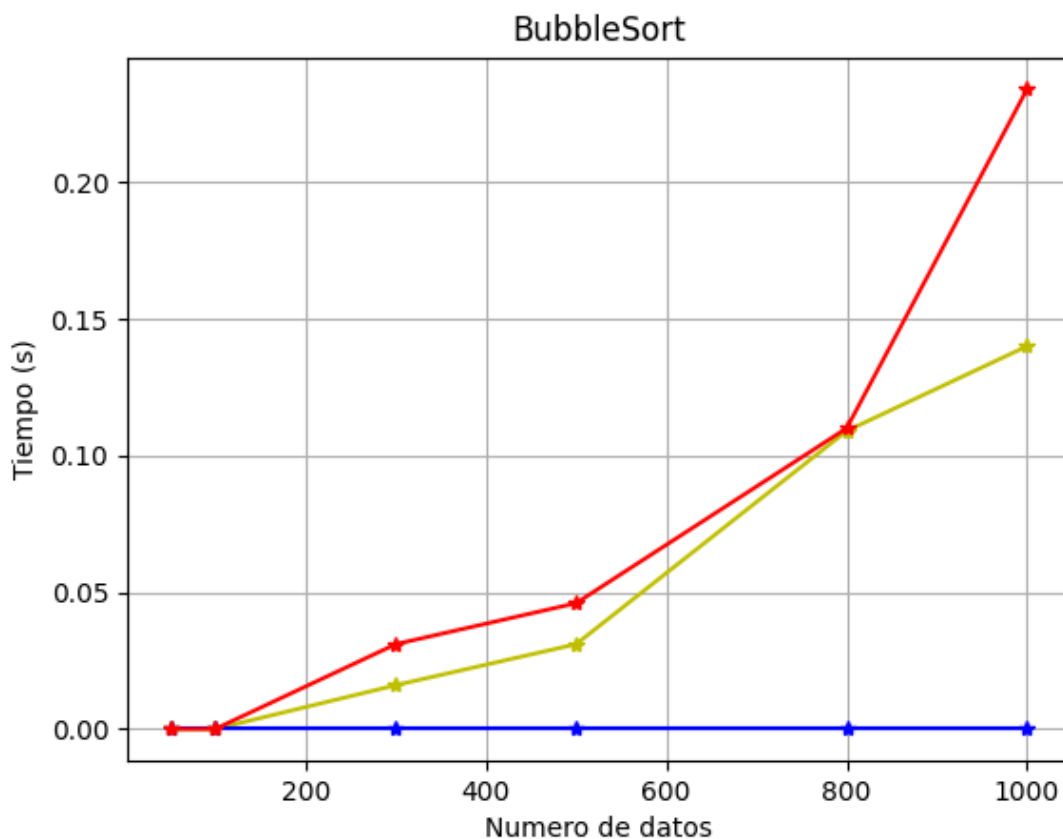


Tamaño del arreglo	Num Operaciones
50	3943
100	14845
500	366777
800	965376
1000	1503049
2000	5974241

En esta gráfica así como en las anteriores verificamos el comportamiento de un algoritmo con orden de complejidad $O(n^2)$, solo que en este caso el número de operaciones promedio incrementa considerablemente a comparación de insertionSort y selectionSort, además se observa que el rango de número de operaciones en cada pasada dentro de un mismo tamaño varía aún más ya que al ser el bubbleSort mejorado y terminar el proceso cuando el arreglo está ordenado (en otras palabras puede que un proceso termine mucho antes que otro) habrá casos en los que se haga más iteraciones en comparación con otros. Por ejemplo tenemos un caso con un tamaño de array de 500 en donde el número de operaciones es 378,768 y otro caso donde el número de operaciones es 356,109

Algoritmo en Python

```
def bubbleSort(array):  
    ordenado = 0  
    i = len(array)  
    while i>0 and ordenado==0:  
        ordenado = 1  
        for j in range(0,i-1):  
            if array[j]>array[j+1]:  
                array[j],array[j+1]=array[j+1],array[j]#intercambio de valores  
            ordenado = 0  
        i=i-1  
        #print("Iteracion numero ",len(array)-1)  
        #imprimir(array)
```



MEJOR CASO(LISTA ORDENADA DE FORMA ASCENDENTE)

CASO PROMEDIO (GENERADO NÚMEROS ALEATORIOS)

PEOR CASO(LISTA ORDENADA DE FORMA DESCENDENTE)

Se realizaron pruebas en donde se puede observar el comportamiento del algoritmo con distintos tamaños de elementos tanto para mejor, peor y caso promedio, la gráfica cumple con lo esperado ya que el mejor caso sólo realiza una iteración y el tiempo es mínimo, el caso promedio estará variando y el peor caso supera a los anteriores 2 en cuestión de tiempo de ejecución.

Ejercicio 4. Ahora en Java

Nota: para el ejercicio en java se sugiere el uso del IDE NetBeans, sin embargo el alumno es libre de utilizar el entorno de desarrollo que considere conveniente

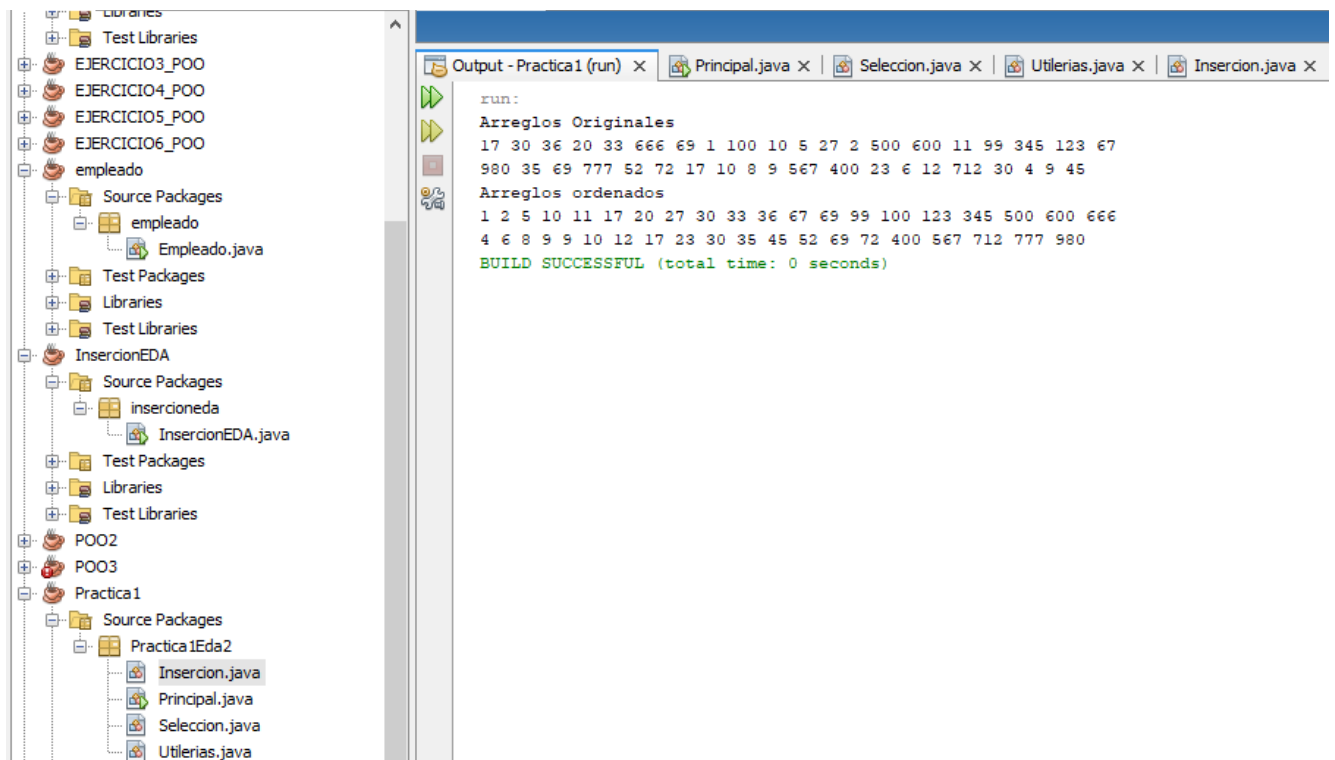
✓ Abre el proyecto “OrdenamientoP1” utilizando el editor NetBeans, compila y ejecuta el programa para verificar el funcionamiento.

[Agrega captura de pantalla de la ejecución]

✓ En caso de tener problemas de compatibilidad con el editor de Netbeans, puedes crear un proyecto nuevo en tu editor, o bien, ejecutar los programas desde la consola. En cualquier caso, debes explicar lo realizado y colocar las evidencias respectivas en el reporte

✓ Revisa el código de las clases proporcionadas.

Ejecución del programa.



The screenshot shows the NetBeans IDE interface. On the left, the 'Project Explorer' displays the project structure, including source packages like 'empleado' and 'InsercionEDA', and test packages. The main editor window shows the 'Output - Practica1 (run)' console. The output text is as follows:

```
run:
Arreglos Originales
17 30 36 20 33 666 69 1 100 10 5 27 2 500 600 11 99 345 123 67
980 35 69 777 52 72 17 10 8 9 567 400 23 6 12 712 30 4 9 45
Arreglos ordenados
1 2 5 10 11 17 20 27 30 33 36 67 69 99 100 123 345 500 600 666
4 6 8 9 9 10 12 17 23 30 35 45 52 69 72 400 567 712 777 980
BUILD SUCCESSFUL (total time: 0 seconds)
```


o Explica las diferencias entre la forma en la que se manda llamar el algoritmo de selección y el de inserción

R = Para el algoritmo de selectionSort se crea un objeto llamado *selección* de la clase Selección y posteriormente mediante este objeto se ocupa el método selectionSort dándole como argumento *arr2* y para insertion sort es como si a través de direcciones se llamará al método insertionSort de la clase Inserción sin crear ningún objeto.

o ¿Por qué en la clase principal se pueden utilizar las otras sin tener que llamarlas explícitamente?

R= Porque se encuentran todas dentro de un mismo *paquete* que se puede comparar a un folder que almacena clases.

o Realiza los comentarios de otros aspectos que te llamen la atención o te causen dudas del código java (codificación, ejecución, clases, etc).

Me causa dudas la forma en como que llama al algoritmo de inserción `Inserción.insertion Sort(arr1);` ya que con lo poco que he estudiado de POO no estaba acostumbrado que un método se ejecutará de dicha forma, sino a través de un objeto.

Conclusiones

Durante la realización de la práctica se pudieron reafirmar conceptos vistos en EDA 1 y fundamentos de programación tales como el uso de funciones, paso de parámetros y complejidad computacional. Gracias a dichos conceptos pudimos cumplir con el objetivo de la práctica pues nos permitió observar de mejor forma el comportamiento de los algoritmos vistos en clase - insertion Sort, bubble Sort y selectionSort - comparando el número de operaciones realizadas y el tamaño del arreglo, analizando que aun teniendo la complejidad de $O(n^2)$ existen grandes diferencias entre ellos y que el uso de un algoritmo de ordenamiento depende mucho de la entrada de datos y el equipo que se utilice.

De igual manera dimos un vistazo a dichos algoritmos implementados en Java lo cual considero que es un buen ejercicio pues no se necesita modificar más que los elementos de los arreglos enfocándonos más en interpretar el funcionamiento utilizando el paradigma orientado a objetos.

Como comentario final supongo que es posible implementar algún algoritmo mejorado en selection Sort que nos permita detener el algoritmo si la lista de

elementos se encuentra ordenada, algo parecido a lo que realiza bubble Sort con la mejora.