	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURAS DE DATOS Y ALGORITMOS II

Grupo: 5

No de Práctica(s): 01

Integrante(s): GÓMEZ LUNA ALEJANDRO

No. de Equipo de cómputo empleado: 43

No. de Lista o Brigada:

Semestre: 2020-1

Fecha de entrega: 15/Agosto/2019

Observaciones:

CALIFICACIÓN: _____

- Objetivo de la práctica
  EL ESTUDIANTE IDENTIFICARÁ LA ESTRUCTURA DE LOS ALGORITMOS DE ORDENAMIENTO POR SELECCIÓN Y POR INSERCIÓN

- Desarrollo
  a) **Ejercicios de la guía:** Para este caso, se buscó la forma en cómo se implementan los algoritmos de ordenamiento Insertion-Sort y Selection-Sort en Python, pues estos no se encuentran en la guía. Empezaremos con Insertion-Sort.
  La implementación encontrada de Insertiorn-Sort en Python nos muestra el siguiente código:

[3]:
```
def instertion_sort(seq):

    for slice_end in range(len(seq)):
        pos = slice_end
        # inserting elements at correct position
        while pos > 0 and seq[pos] < seq[pos - 1]:
            (seq[pos], seq[pos - 1]) = (seq[pos - 1], seq[pos])
            pos = pos - 1


# test input
l = [54, 20, 93, 17, 70, 31, 42, 55, 21]
instertion_sort(l)
print(l)
```

```
[17, 20, 21, 31, 42, 54, 55, 70, 93]
```

Con base en este código encontrado, se presentan varias características distintas en cuanto al pseudocódigo visto en clase. Para empezar, no se utiliza una variable para almacenar el tamaño de la lista, pues el resultado de la función len() se pone directamente en el valor frontera del primer for. Además de esto, la variable slice_end tendrá un valor inicial de 0, mientras que en clase vimos que se empieza con un valor de 1. Otra característica distinta que observamos es en cuanto a la variable pos, la cual toma directamente el valor de la variable slice_end, sin decrementarle en uno su valor y luego almacenarlo, como vimos en clase. Además, no se usa una variable auxiliar para almacenar el elemento que se encuentra en el índice correspondiente al número de iteración del for. De igual forma, en la comprobación del while, se observa una especie de mejora con respecto a la vista en clase, pues, si el próximo elemento a comparar en la lista es mayor que su antecesor, entonces no tiene que seguir con los demás elementos a su izquierda, porque ya está en su respectiva posición.
Por último, se observa que, como Python permite la asignación paralela, en caso de cumplirse la condición del while, se irán intercambiando los elementos que aún no ha sido ordenado a la izquierda hasta llegar a su posición correspondiente.

La implementación encontrada de Selection-Sort en Python nos muestra el siguiente código:

[8]:
```
# Sorting function
def selection_sort(l):

    for start in range(len(l)):
        min_pos = start

        for i in range(start, len(l)):
            if l[i] < l[min_pos]:
                min_pos = i
        # swap values
        (l[start], l[min_pos]) = (l[min_pos], l[start])

# Test input
l = [52, 24, 99, 11, 70, 33, 44, 55, 13]
selection_sort(l)
print(l)
```

```
[11, 13, 24, 33, 44, 52, 55, 70, 99]
```

Con base en este código encontrado se presentan ligeras características diferentes en cuanto al pseudocódigo visto en clase. Lo primero es que, al igual que en el código anterior, no se hace uso de una variable para almacenar la longitud de la lista.
La otra característica diferente es que, como Python permite la asignación paralela, no se tiene que hacer uso de la función intercambio, pues esta acción se puede realizar en una sola línea de código,
Respecto a lo demás, el código se apega al pseudocódigo lo visto en clase, mediante el uso de dos for, uno anidado dentro del otro, se irá recorriendo la lista elemento por elemento, hasta encontrar el mínimo, intercambiarlo con el valor que se encuentra con su posición correspondiente y así, ir ordenando la lista, sin pasar por aquellos elementos que ya se encuentran ordenados.

b) **Ejercicios propuestos:**

a. Ejercicio1: En la biblioteca de ordenamientos.h encontramos dos funciones: selectionSort e insertionSort. La función selectionSort nos permite implementar el algoritmo de ordenamiento Selection sort que se vió en clase. La función empieza por declarar tres variables de tipo entero; indiceMenor, i y j. La variable i servirá para ir contabilizando y controlando las iteraciones que se vayan realizando. Se tienen dos for, uno dentro del otro. El for más externo irá almacenando el número de iteración en el que nos encontramos en la variable indiceMenor, con la finalidad que en el for interno, se puedan ir comparando todos los elementos restantes a la derecha de este. Cabe resaltar que los elementos se irán ordenando de izquierda a derecha, sin pasar por aquellos que ya han sido puestos en su posición correspondiente. En cada recorrido del arreglo se escogerá el elemento más pequeño que aún no esté en su posición correcta, y se guardará el índice en el que se encuentra, para que, una vez que finalice el for interno, se pueda posicionar correctamente. Siempre que se realice este posicionamiento, se llamará a la función swap, la cual sirve para cambiar de posición a dos elementos dentro del arreglo.

La función insertionSort nos permite implementar el algoritmo de ordenamiento Insertion sort que se vió en clase. La función empieza por declarar tres variables de tipo entero; i, j y aux. La variable i será la que lleve el número de iteración del for externo.

Es importante remarcar que la variable i empezará desde 1, pues también llevará el control de los índices del arreglo, y como el primer elemento que está en el índice cero ya se toma como "ordenado", entonces empezará a partir de este. Posteriormente se hace uso de la variable auxiliar j, la cual almacenará el número de iteración en el que nos encontramos, y de igual forma la variable aux almacenará el elemento que estamos ordenando en esa iteración. Una vez almacenados estos valores, se presenta un ciclo de repetición while, el cual solamente se ejecutará cuando la variable j sea mayor a 0 y cuando los elementos a la izquierda del elemento que estamos ordenando sean menores a este, pues esto nos indica que el elemento en cuestión no se encuentra en su posición adecuada. Por último, posicionaremos al elemento en su posición correspondiente.

En cuanto a las funciones que vienen en la biblioteca utilidades.h, encontramos tres. La primera es la función swap, la cual, mediante el paso por referencia de dos variables y el uso de apuntadores, nos permite intercambiar los valores de estas dos variables.

La segunda es la función printArray, la cual nos permite imprimir todos los elementos en un arreglo. Los parámetros que se le pasan es el arreglo que queramos imprimir y su tamaño. La tercera es la función printSubArray, la cual nos permite imprimir todos los elementos que se encuentran dentro de una porción del arreglo. Nosotros le pasamos como parámetros el arreglo en cuestión y, mediante valores frontera, delimitar que porción del arreglo queremos imprimir.

b. Ejercicio 2: Para este ejercicio, se pedía crear un arreglo con 10 elementos y llenarlo con elementos de maneara aleatoria. Para llenar el arreglo de forma aleatoria se hizo uso de la función srand y rand, dentro de la biblioteca time.h y stdlib.h. Una vez llenado el arreglo con elementos aleatorios entre 1 y 30, se imprimía en pantalla los elementos del arreglo, haciendo uso de la función printArray proporcionada en la biblioteca de utilidades.h. Después, se creó un pequeño menú en donde se le pedía al usuario que seleccionara la opción correspondiente al algoritmo de ordenamiento con el que quería ordenar el arreglo.

Con la finalidad de mostrar la manera en como se iba ordenando el arreglo con la función insertionSort y selectionSort, cada que finalizara la última instrucción del for externo, se imprimía en pantalla el arreglo, pues de esta manera se pueda visualizar como se va ordenando el arreglo cada que se recorre.

Para finalizar, se muestra en pantalla el arreglo ya ordenado.
Algo que sucedió cuando se quiso implementar la función de insertionSort, es que se mostró un error de compilación, pues no se estaba utilizando el identificador correcto en algunas partes de la función.

Es importante notar que, aunque existan elementos repetidos dentro del arreglo,no fallan los algoritmos de ordenamiento.

```
El arreglo creado tiene los elementos:
9 25 24 26 29 26 9 28 17 7
    ***Bienvenido al programa de ordenamiento***
Seleccione el numero de la opcion que desee para ordenar el arreglo:
1) Insertion Sort
2) Selection sort
1
Ordenando el arreglo...
9 25 24 26 29 26 9 28 17 7
9 24 25 26 29 26 9 28 17 7
9 24 25 26 29 26 9 28 17 7
9 24 25 26 29 26 9 28 17 7
9 24 25 26 26 29 9 28 17 7
9 9 24 25 26 26 29 28 17 7
9 9 24 25 26 26 28 29 17 7
9 9 17 24 25 26 26 28 29 7
7 9 9 17 24 25 26 26 28 29
El arreglo ordenado es:
7 9 9 17 24 25 26 26 28 29
Program ended with exit code: 175
```

```
El arreglo creado tiene los elementos:
25 21 30 1 2 21 30 9 21 4
    ***Bienvenido al programa de ordenamiento***
Seleccione el numero de la opcion que desee para ordenar el arreglo:
1) Insertion Sort
2) Selection sort
2
Ordenando el arreglo...
1 21 30 25 2 21 30 9 21 4
1 2 30 25 21 21 30 9 21 4
1 2 4 25 21 21 30 9 21 30
1 2 4 9 21 21 30 25 21 30
1 2 4 9 21 21 30 25 21 30
1 2 4 9 21 21 30 25 21 30
1 2 4 9 21 21 21 25 30 30
1 2 4 9 21 21 21 25 30 30
1 2 4 9 21 21 21 25 30 30
El arreglo ordenado es:
1 2 4 9 21 21 21 25 30 30
Program ended with exit code: 175
```

c. Ejercicio 3: Para este ejercicio se puede observar que los mismos algoritmos de ordenamiento de Insertion sort y Selection sort se han implementado en el lenguaje de programación Java. Debido a esta implementación en Java, se pueden observar varias características importantes del paradigma orientado a objetos. Una de estas características importantes son las clases, las cuales nos ayudan a modelar entidades de la vida real y, para este caso, se observa que cada algoritmo de ordenamiento está siendo modelado a partir de una clase, y dicha clase tiene un solo comportamiento, el cual es la función que ordena el arreglo. Otra característica importante es la creación de objetos a partir de una clase, como podemos observar en el método main, en donde se crea una instancia de la clase selección, es decir, se crea un objeto a partir de la clase selección. De igual forma podemos observar como los modificadores de acceso nos permiten acceder a métodos de otras clases sin limitación alguna.

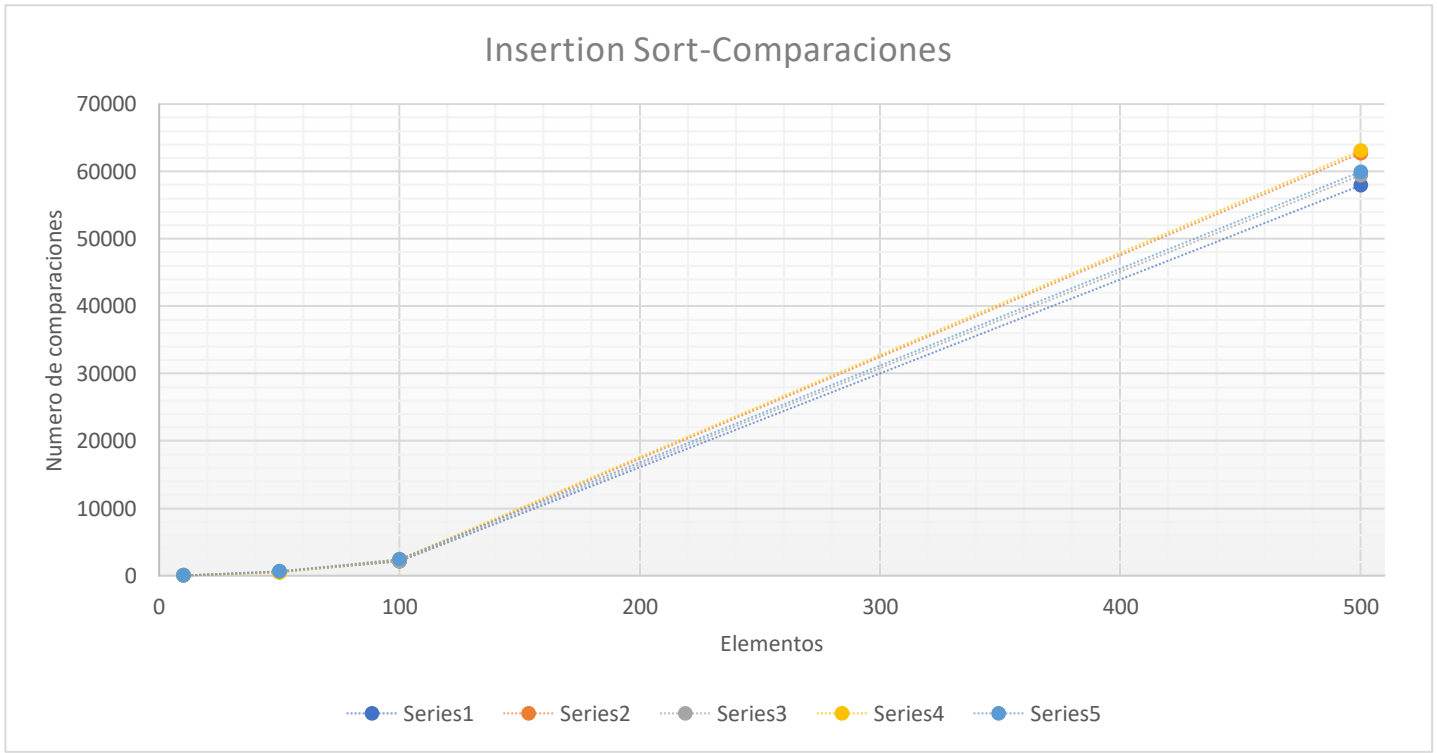
En cuanto al código, no existen muchas diferencias. Solamente existen pequeñas variaciones por el cambio de lenguaje, como el uso de for each, el cual es usado por la función imprimirArreglo y nos permite realizar la función de imprimir el arreglo en pantalla de manera más simple. También observamos como el uso de la palabra reservada “static” nos permite utilizar funciones de una clase sin necesidad de crear una instancia a ella, lo que asemeja más a un paradigma estructurado.

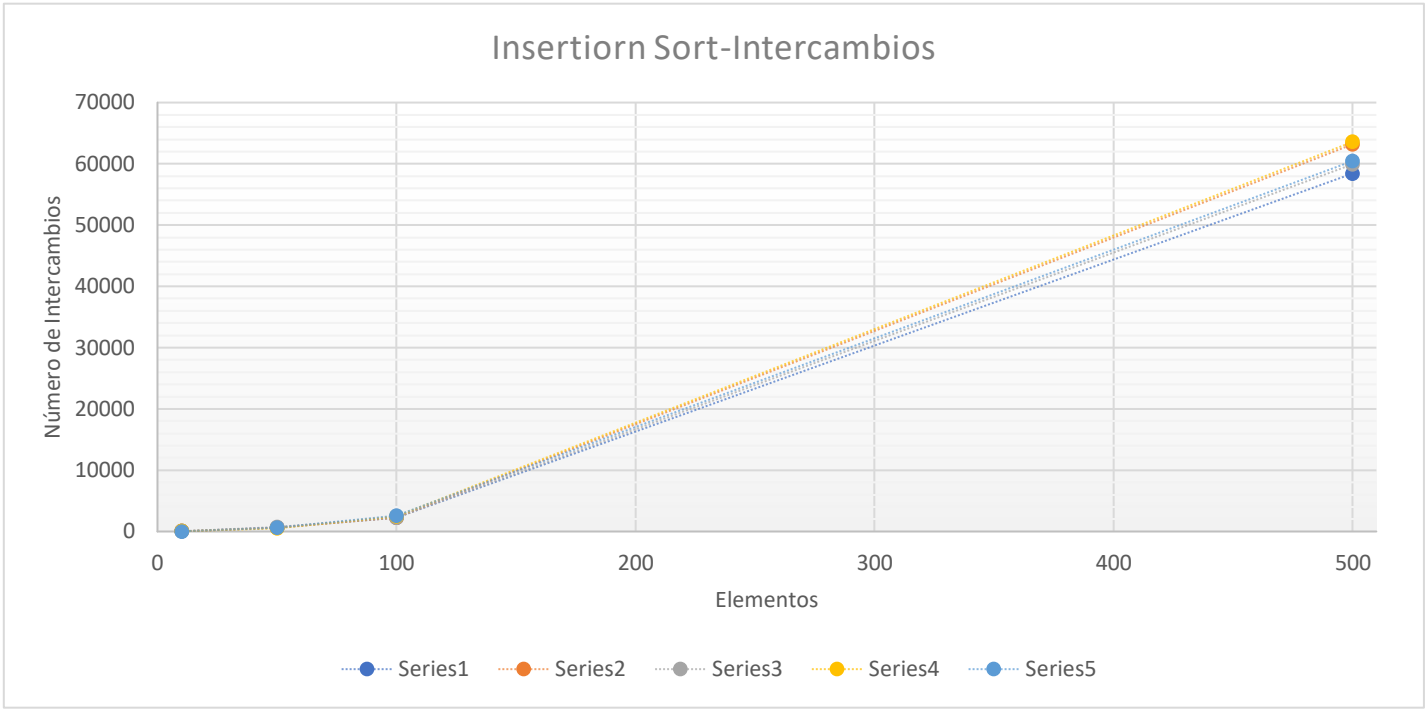
d. Ejercicio4: Para este ejercicio se nos pidió modificar las funciones de insertionSort y selectionSort, de manera que se hiciera la cuenta de cuantos intercambios, así como de cuantas comparaciones se realizaban para ordenar por completo un arreglo. Se realizó un total de 5 pruebas, con los siguientes resultados y sus correspondientes gráficas:

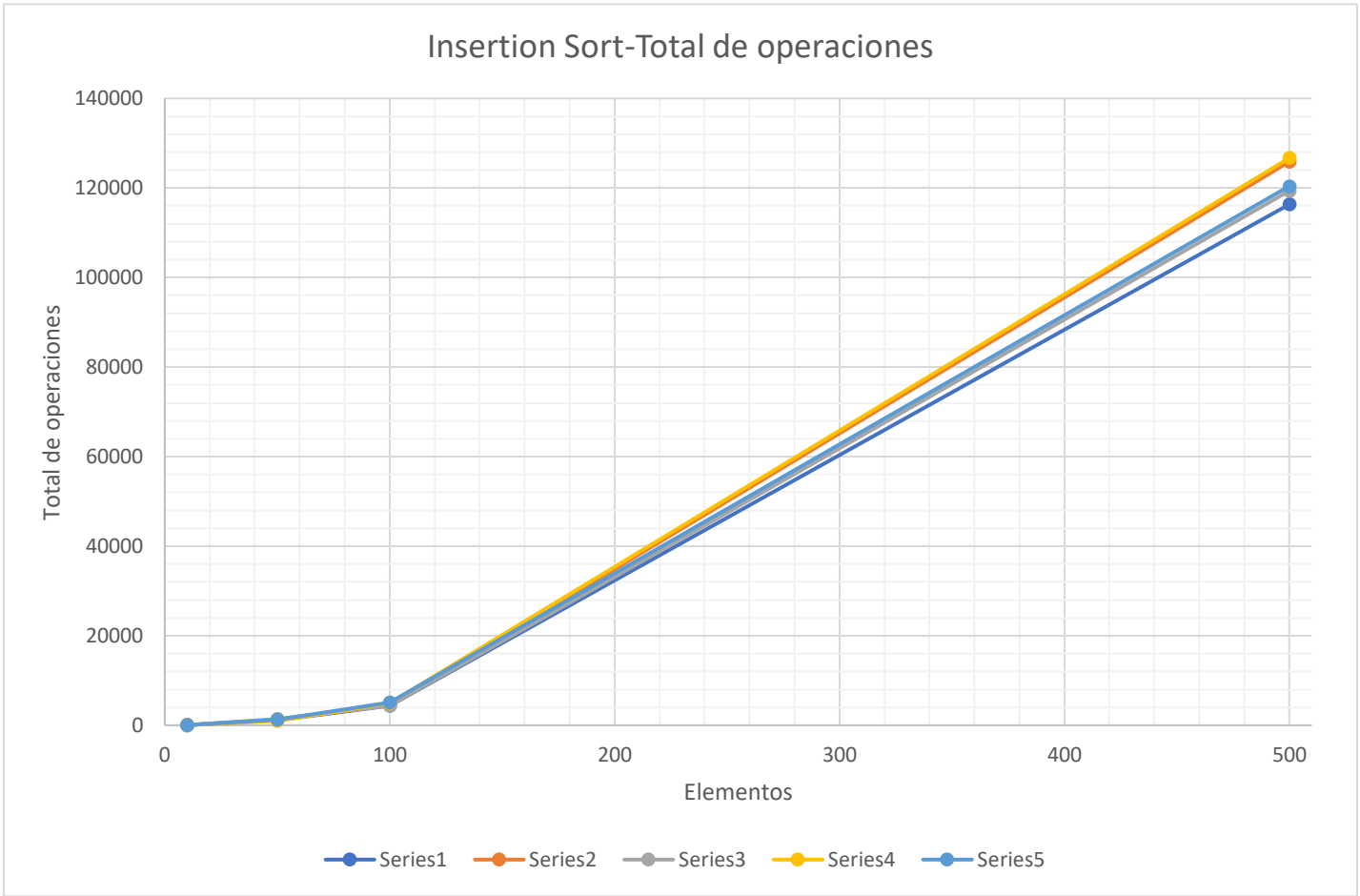
Para Insertion Sort:

Número de Prueba	*Cantidad de elementos*	*Comparaciones*	*Intercambios*	*Total de Operaciones*
1	10	24	33	57
	50	540	589	1129
	100	2107	2206	4313
	500	57913	58412	116325

2	10	21	30	51
	50	581	630	1211
	100	2190	2289	4479
	500	62684	63183	125867
3	10	23	32	55
	50	663	712	1375
	100	2176	2275	4451
	500	59426	59925	119351
4	10	21	30	51
	50	487	536	1023
	100	2407	2506	4913
	500	63084	63583	126667
5	10	19	28	47
	50	627	676	1303
	100	2462	2561	5023
	500	59948	60447	120395

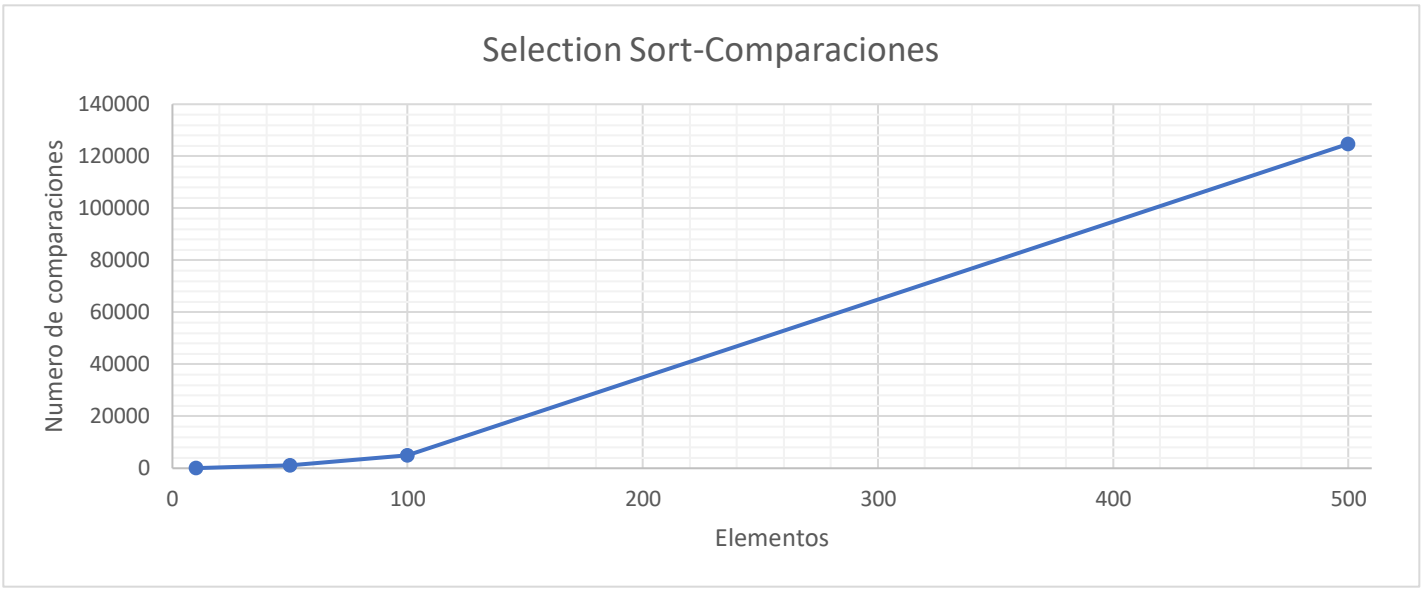


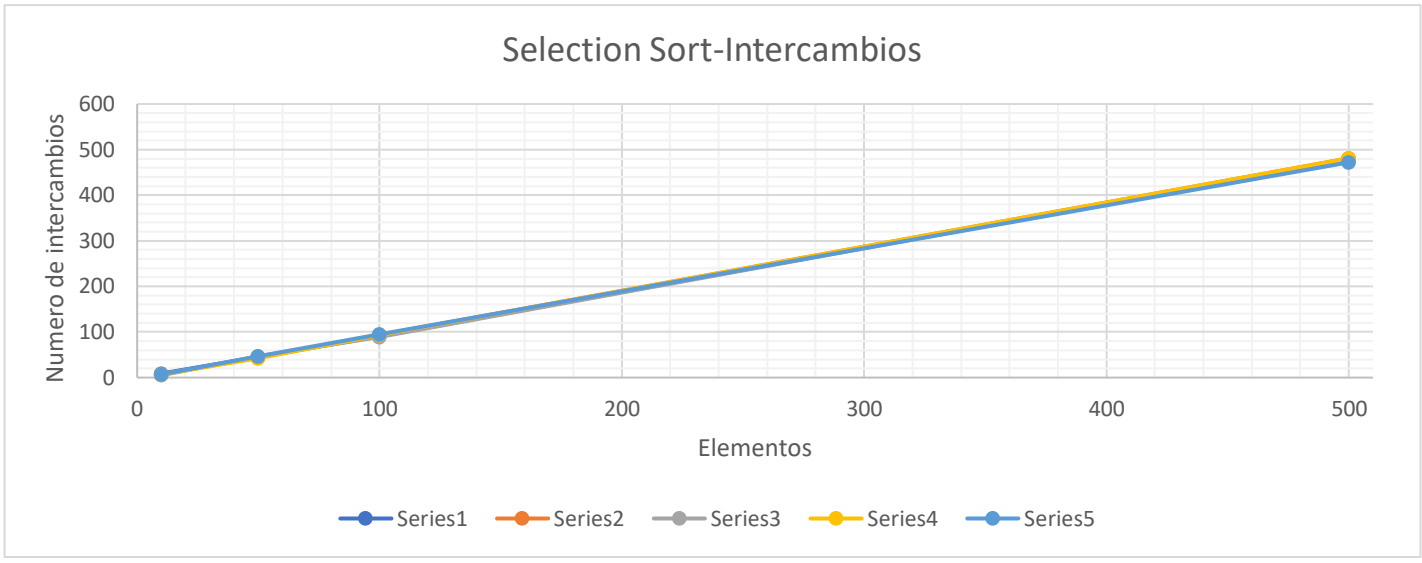

Insertiorn Sort-Intercambios
70000
60000
50000
40000
30000
20000
10000
0
Número de Intercambios
0
100
200
300
400
500
Elementos
Series1
Series2
Series3
Series4
Series5

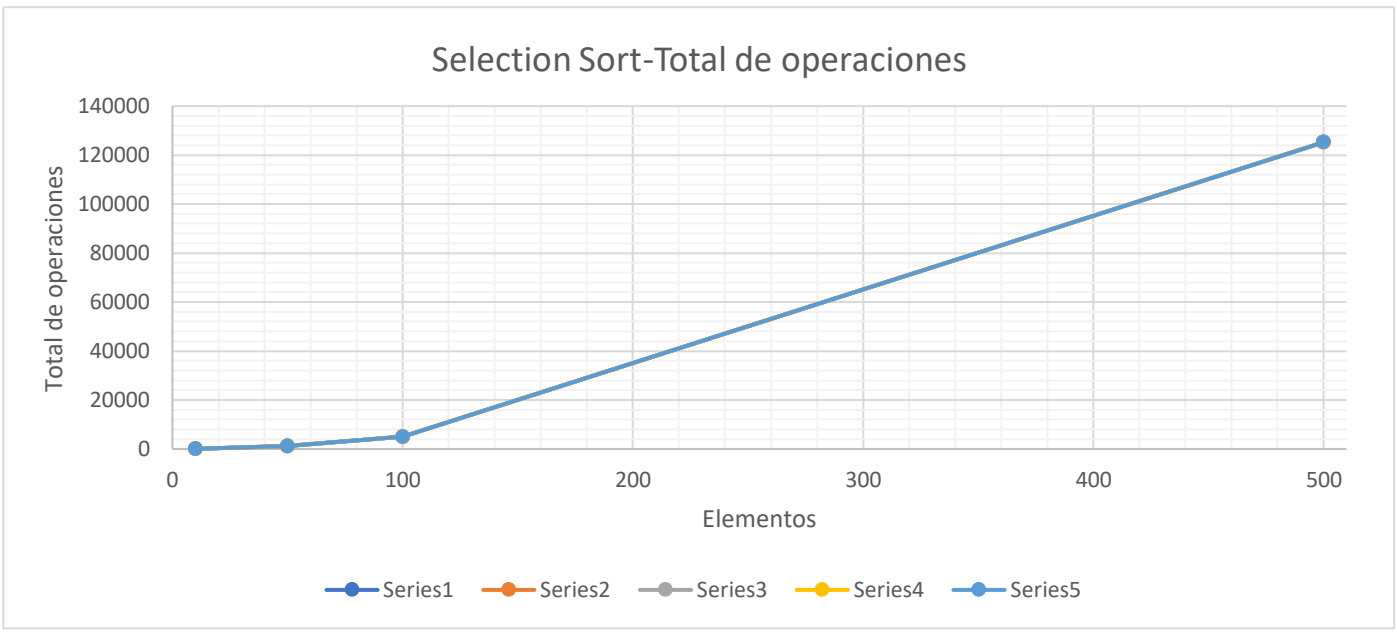

Insertion Sort-Total de operaciones
140000
120000
100000
80000
60000
40000
20000
0
Total de operaciones
0
100
200
300
400
500
Elementos
Series1
Series2
Series3
Series4
Series5

Para Selection Sort:

Número de Prueba	*Cantidad de elementos*	*Comparaciones*	*Intercambios*	*Total de Opraciones*
1	10	45	9	54
	50	1225	45	1270
	100	4950	90	5040
	500	124750	479	125229
2	10	45	7	52
	50	1225	44	1269
	100	4950	93	5043
	500	124750	481	125231
3	10	45	5	50
	50	1225	47	1272
	100	4950	89	5039
	500	124750	478	125228
4	10	45	7	52
	50	1225	42	1267
	100	4950	94	5044
	500	124750	481	125231
5	10	45	7	52
	50	1225	47	1272
	100	4950	95	5045
	500	124750	472	125222


Selection Sort-Comparaciones
Numero de comparaciones
140000
120000
100000
80000
60000
40000
20000
0
0
100
200
300
400
500
Elementos


Selection Sort-Intercambios
Numero de intercambios
600
500
400
300
200
100
0
0
100
200
300
400
500
Elementos
Series1
Series2
Series3
Series4
Series5


Selection Sort-Total de operaciones
Total de operaciones
140000
120000
100000
80000
60000
40000
20000
0
0
100
200
300
400
500
Elementos
Series1
Series2
Series3
Series4
Series5

Con base en estos resultados podemos determinar que:

- Insertion Sort tiene un número mucho mayor de intercambios que Selection Sort, el cual tiene un número siempre menor al tamaño total del arreglo. Esto se debe a que el algoritmo, cada que se recorre el arreglo, solo se intercambian dos valores como máximo, el mínimo sin ordenar y el elemento en la posición que le corresponde al mínimo, por lo que, como máximo, los intercambios que realizará serán el tamaño del arreglo menos uno, pues el último elemento que quede ya estará ordenado.

  Debido a esta estrecha relación entre el tamaño del arreglo y la cantidad de intercambios, es que se observa una gráfica bastante parecida a una de una función lineal.

- Selection Sort, en las cinco pruebas que se realizaron, tuvo siempre el mismo número de comparaciones realizadas. Esto es debido a que la función Selection Sort describe un polinomio de la forma: ((n^2) /2) - (n/2), en donde n es la cantidad de elementos que se le están pasando como entrada, con respecto al número de comparaciones realizadas.

- Ambos algoritmos toman una cantidad total de operaciones parecidas, sin embargo, en Selection Sort, las que más suman en ese total son la cantidad de comparaciones, mientras que, para Insertion Sort, la cantidad de comparaciones e intercambios son bastante parecidas en cantidades.

- En Insertion Sort, la cantidad de comparaciones e intercambios tiene más variación, pues estás tienen una mayor dependencia en la forma en como estén distribuidos los elementos que en el tamaño de arreglo proporcionado. Además, como en el algoritmo se tiene un while, este podría o no cumplirse, lo que da como resultado una fluctuación mayor en la cantidad de comparaciones e intercambios realizados para ordenar un arreglo.

- Conclusiones

  En esta práctica se lograron identificar las principales características de los algoritmos de ordenamiento Selection Sort e Insertion Sort, así como las maneras de implementarlo en distintos lenguajes de programación, como fue el caso de Python (se buscó la implementación en Python), en C y en Java. De esta manera, el hecho de ver la codificación del algoritmo en distintos lenguajes de programación permitía observar y analizar de mejor manera las características fundamentales de estos dos algoritmos de ordenamiento.

  Aunado a esto, se pudieron comparar los dos algoritmos de ordenamiento, en cuanto al número de comparaciones, número de intercambios y el total de operaciones realizadas, pudiendo observar como la cantidad de operaciones realizadas es casi la misma en ambos, mientras que el número de intercambios y operaciones variaban más en uno y en otro, debido al algoritmo que usan para ordenar el arreglo.

  Estos algoritmos de ordenamiento sirven como una buena introducción para entender algoritmos de ordenamiento más complejos, además de que para el mejor caso y el caso promedio tienen una complejidad lineal, lo que los coloca como unos algoritmos de ordenamiento intermedio.

  Esta práctica me resultó bastante interesante y adecuada, ya que se implementaros los dos algoritmos de ordenamiento en varios lenguajes, además de que se visualizaron características importantes de estos, como lo es la cantidad de operaciones realizadas y la manera en como estas operaciones aumentan a medida que la cantidad de elementos de entrada aumenta, los ligeros cambios que pueden presentarse al codificarse en otros lenguajes, pero manteniendo la misma estructura.

- Referencias

  - https://djangocentral.com/insertion-sort-in-python/. Recuperado el 12/08/2019 a las 18:47
  - https://djangocentral.com/selection-sort-in-python/ Recuperado el 12/08/2019 a las 19:05