	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	M.C. JOSÉ MAURICIO MATAMOROS DE MARIA Y CAMPOS
Asignatura:	ESTRUCTURA DE DATOS Y ALGORITMOS I
Grupo:	12
No de Práctica(s):	11
Integrante(s):	ARELLANES CONDE ESTEBAN
No. de Equipo de cómputo empleado:	10
No. de Lista o Brigada:	01
Semestre:	2022-2
Fecha de entrega:	16 mayo, 23:59
Observaciones:	

CALIFICACIÓN: _____

Práctica 11: Estrategias

Estructuras de Datos y Algoritmos I

Autor: Arellanes Conde Esteban

1. Objetivo

El alumno aprenderá a identificar las diferentes estrategias para la construcción de algoritmos (fuerza bruta, codiciosos, programación dinámica y divide y vencerás) y sus paradigmas (Top-down y Bottom-up).

2. Introducción

El uso de estrategias y paradigmas de programación son necesarios para elaborar algoritmos. Durante esta práctica se usaron en específico los paradigmas y las estrategias *top-down*, *bottom-up* y *algoritmos ávidos* (o *greedy algorithms*) respectivamente.

3. Actividad 1:

Los programas ‘‘`fact1.py`’’ y ‘‘`fact2.py`’’ calculan el factorial de un número entero positivo n. Ejecutándolos con las líneas: `python3 fact1.py 10` `python3 fact2.py 10`

Compare los programas `fact1.py` y `fact2.py` y responda las siguientes preguntas:
¿Qué paradigma utiliza el programa `fact1.py`? Explique [1 punto]: El programa `fact1.py` utiliza el paradigma *Bottom-up*, este es visto en el ciclo *“for”* que tiene el código.

¿Qué paradigma utiliza el programa `fact2.py`? Explique [1 punto]: Utiliza el paradigma *Top-down* puesto que parte un problema complejo en uno más simple con una iteración condicional comparativa.

4. Actividad 2:

El programa ‘‘`time.py`’’ mide e imprime el tiempo que tarda en ejecutarse la función *foo()*. Ejecutándolo con la línea: `python3 time.py`

Combine el programa `time.py` con los programas `fact1.py` y `fact2.py` para poder medir el tiempo que tarda cada uno de los programas en calcular el factorial de un número entero. Con la información obtenida llene la siguiente tabla.

n	Prog1.py	Prog2.py
10	0.01 ms - 0.02 ms	0.01 ms - 0.02 ms
100	0.01 ms - 0.02 ms	0.01 ms - 0.02 ms
500	0.01 ms - 0.02 ms	0.01 ms - 0.02 ms
1000	0.01 ms - 0.02 ms	no compila o tarda mucho n>999
10000	0.01 ms - 0.03 ms	no compila o tarda mucho n>999

Cuadro 1: Tabla 1.

¿Existe alguna diferencia en la ejecución de los programas? ¿Alguno es más rápido? ¿Es posible llenar la tabla? Explique: [1 punto]: Sí, es distinta la ejecución debido a que el segundo programa aunque es más corto toma más espacio en memoria porque por defecto python asigna sólo memoria por debajo de 999.

5. Actividad 3:

El programa ‘‘`change.py`’’ recibe una cantidad numérica como argumento que representa el vuelto o cambio a entregar, y devuelve el número de monedas que deben entregarse. Ejecutándolo con la línea:

```
python3 change.py 23.50
```

Este programa (`change.py`) tiene una limitación muy grave: asume que tiene disponible cambio infinito, por lo que siempre da un número mínimo de monedas que podrían no estar disponibles. Modificando el programa `change.py` para que entregue la menor cantidad de monedas posibles usando sólo las monedas disponibles. Para tal fin, hay que reemplazar la línea 10 (variable `coins`) con:

```
1 coins = {
2 0.1 : 3,
3 0.2 : 3,
4 0.5 : 2,
5 1 : 0,
6 2 : 3,
7 5 : 0,
8 10 : 3,
9 20 : 3,
10 50 : 3,
11 100 : 3,
12 }
```

Probando el programa con:

```
python3 change.py 23.50
python3 change.py 171.30
python3 change.py 19.90
```

n	isort.py	qsort.py
10	0.01 ms	0.01 ms
100	0.35 ms	0.17 ms
1000	41.45 ms	2.18 ms
10000	4324.42 ms	32.31 ms
100000	5102801.56 ms	382.12 ms
1000000	----------	4846.89 ms

Cuadro 2: Tabla 2.

¿Qué paradigma y qué estrategia utiliza el programa? Explique [1 punto]: Usa el paradigma *Top-down* y la estrategia de *Búsqueda Exhaustiva* debido a que simplificamos algo complejo a problemas menores. Con respecto a la estrategia, esta busca la cantidad de iteraciones de monedas de tal cantidad para dar el cambio.

¿Qué modificaciones se realizaron al programa? Explique [2 puntos]: Implementar una lista de listas o bien adaptar el código para leer un diccionario. En nuestro caso, implementamos la primera opción y para ello en el código fue necesario adaptarlo para leer el primer elemento de la sublista, es decir, la denominación del cambio.

6. Actividad 4:

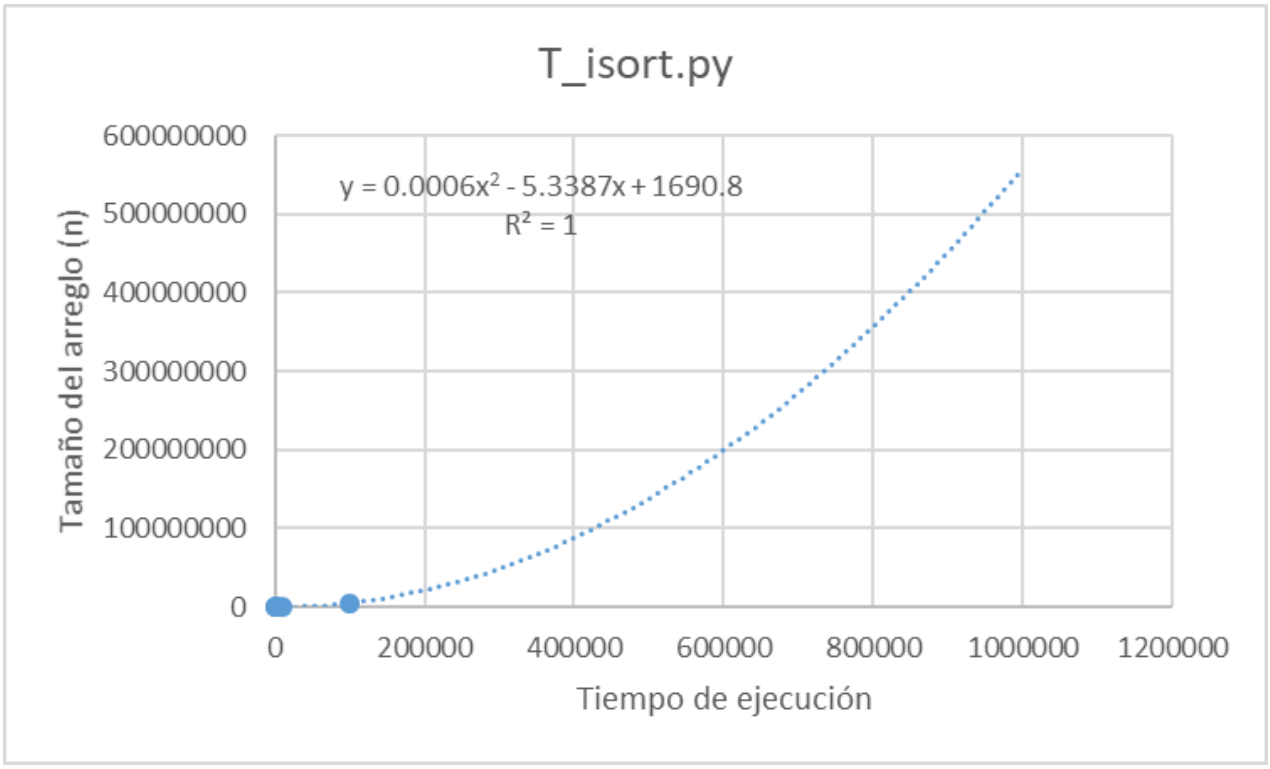
Los programas ‘‘isort1.py’’ y ‘‘qsort2.py’’ ordenan un arreglo de números generados aleatoriamente usando dos algoritmos: InsertionSort y QuickSort. Ejecútelos con las líneas:

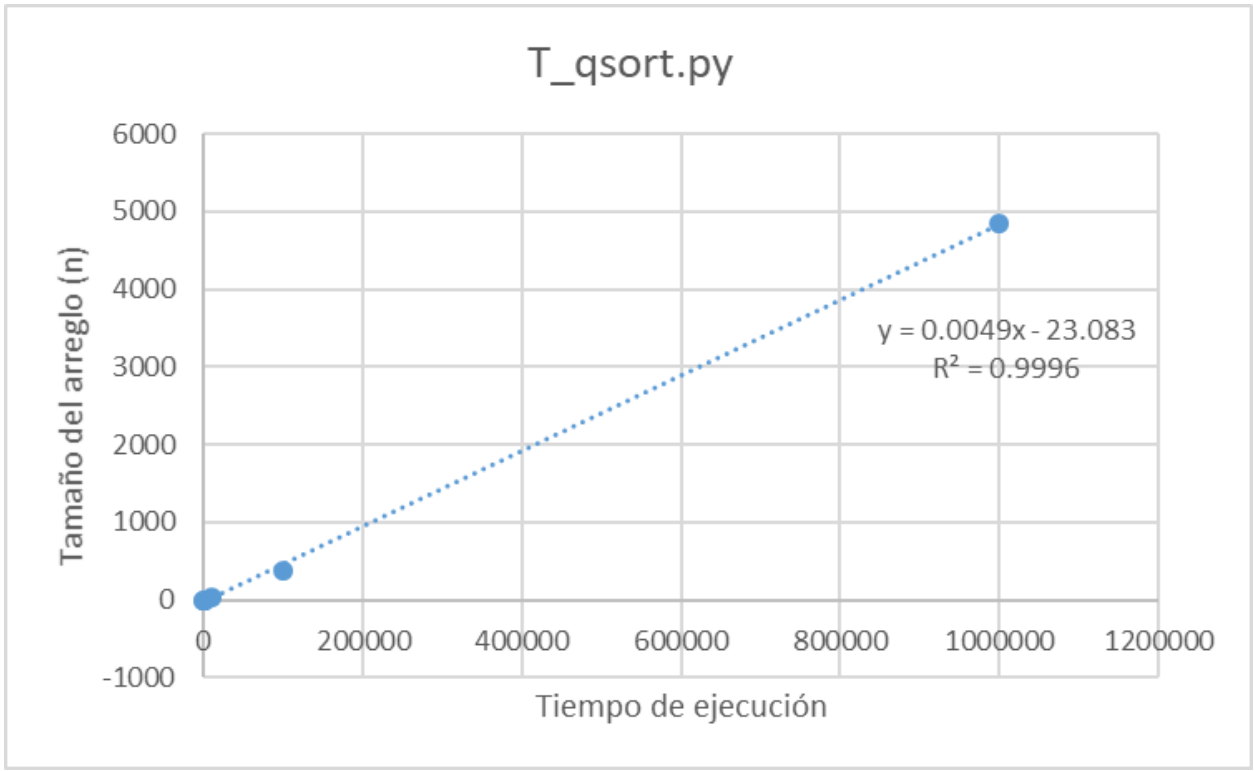
```
python3 isort1.py 100
python3 qsort2.py 100
```

Posteriormente complete la siguiente tabla anotando los tiempos de ordenamiento:

¿Qué paradigma y qué estrategia utiliza *QuickSort*? Explique [1 punto]:
Emplea el paradigma Top-Down porque simplificamos de algo complejo a sub-problemas, y la estrategia Divide y Vencerás debido a que reacomoda el arreglo en sub-arreglos para ordemarlos.

Con base en la información de la tabla, genere una gráfica de dispersión n, ti(n), tq(n) donde compare los tiempos de ejecución de los algoritmos QuickSort e Insertion Sort en función del tamaño del arreglo (n). Analice los resultados y explique las diferencias [3 puntos]: Las diferencias radican en el número de elementos en función del tiempo. En otras palabras, mientras más tiempo pase más elementos tendrá Quicksort en comparación con Insertion Sort.


T_isort.py
y = 0.0006x² - 5.3387x + 1690.8
R² = 1
Tamaño del arreglo (n)
Tiempo de ejecución
600000000
500000000
400000000
300000000
200000000
100000000
0
0
200000
400000
600000
800000
1000000
1200000


T_qsort.py
y = 0.0049x - 23.083
R² = 0.9996
Tamaño del arreglo (n)
Tiempo de ejecución
6000
5000
4000
3000
2000
1000
0
-1000
0
200000
400000
600000
800000
1000000
1200000

[2 puntos extra:]: Utilice matplotlib para generar las gráficas.

[3 puntos extra:]: Una los programas ‘‘`change.py`’’ y ‘‘`isort1.py`’’ y automatice la generación de las gráficas con matplotlib y numpy, iterando en el intervalo $n \in [0, 10000]$ con un incremento $k = 10$ (n in range(10, 10001, 10)).

6.1. Referencias:

- `Thomas H. Cormen. (2009). Introduction to Algorithms 3e. United States of America: Massachusets Institute of Technology.`
- `Shovic, John and Simpson, Alan (2019). Python All-In-One for Dummies. John Wiley & Sons, Inc., Hoboken, New Jersey`