Universidad de Murcia

Ingeniería Informática

tecnologías específicas de la ingeniería informática - dis.

Comparación de algoritmos de ordenación con Python.

Alumno:
Adrián Carayol Orenes

Profesor:
Alberto Ruíz García

Índice

Resumen

Esto es una pequeña comparación en Python entre tres algoritmos de búsqueda, en concreto, los siguientes:

- Ordenación por selección (mínimos sucesivos).
- Ordenación por inserción.
- Ordenación Timsort.

1. Introducción.

A continuación, vamos a comparar tres algoritmos de ordenación, entre ellos está el que usa **Python** (Timsort) en el método **sorted()** para ordenar listas, por ejemplo.

Pediremos al usuario que introduzca un número, que será el tamaño de la lista a ordenar.

La lista se rellenará automáticamente con números aleatorios, y se usará la misma lista para los tres algoritmos.

Finalmente, lanzaremos tres procesos en paralelo (uno con cada algoritmo de ordenación) y se compararán los tiempos de cada uno, y se mostrará por pantalla las listas ordenadas.

Para llevar un control de las versiones del programa, y poder tener acceso al programa desde cualquier sitio, usaremos **Git** junto con **GitHub**.

2. Algoritmos utilizados.

En concreto, los algoritmos utilizados son siguientes:

- **Ordenación por selección** (mínimos sucesivos).

  El código está inspirado en el libro [**?**]Python Fácil.

  • Es un doble bucle anidado, en el que se compara el primer número con todos los demás, y si alguno es menor, los intercambia de posición, luego se repite la misma operación con el segundo etcétera.

- **Ordenación por inserción**.

  El código está inspirado en el libro [**?**]Python Fácil.

  • Este algoritmo, tiene un orden de ejecución de: $O(n^2)$, además de ser inestable, es decir, puede que algún número no quede ordenado. Consiste en insertar un elemento de la lista en la parte "izquierda" de la misma (los menores se desplazan a la izquierda), se repite este proceso desde el segundo hasta el último elemento de la lista.

- **Ordenación Timsort**.

  El código está implementado por los desarrolladores de Python.

  • Timsort es un algoritmo de clasificación estable (ordena siempre bien las listas) híbrido, derivado de la ordenación por ordenación y clasificación por inserción, diseñado para funcionar bien en muchos tipos de datos del mundo real

3. Comparación de algoritmos

A continuación, se muestra una tabla comparativa con los tiempos de ejecución de los 3 algoritmos, con diferentes tamaños de lista. (tabla 1):

Tamaño lista	Minimos sucesivos	Ordenacion por seleccion	Ordenacion por Timsort
200	0.002418	0.000693	0.000051
2000	0.190324	0.052366	0.000613
20000	14.562728	5.272498	0.005289

Cuadro 1: Tabla con 2000 elementos

Como podemos observar, vemos que, el que mejor se comporta para todos los tamaños es el algoritmo de ordenación Timsort, además, el algoritmo de ordenación por selección, ha fallado en la ordenación de 200 elementos, ya que ha dejado un número sin ordenar.

También podemos observar, que, conforme crece el tamaño de la lista, el algoritmo que menos perjudicado se ve es el de Timsort, por ello se usa en la vida real para trabajar con grandes cantidades de datos de una forma eficiente.

4. Listados de código

A continuación, el código del programa, que está dividido en dos ficheros, uno con el módulo de los algoritmos y otro, con el programa principal.

– program.py –

```
0  import random
1  import time
2  import multiprocessing
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from algoritmos_ordenacion import Algoritmos
6
7
8  def process_1(lista, q):
9      """
10     Funcion para el proceso 1
11     :param lista: Lista de únmeros a ordenar
12     """
13     start = time.time()
14     print("Empezando: %s \n" % multiprocessing.current_process().name)
15     Algoritmos.minimos_sucesivos(lista)
16     end = time.time()
17     q.put({
18         'time': end - start,
19         'algoritmo': 1,
20     })
21     print("Finalizado: %s en %f \n" % (multiprocessing.current_process().name, end -
           start))
22     print(lista)
23
24
25 def process_2(lista, q):
26     """
27     Funcion para el proceso 2
28     :param lista: Lista de únmeros a ordenar
29     """
30     print("Empezando: %s \n" % multiprocessing.current_process().name)
31     start = time.time()
32     Algoritmos.insertion_sort(lista)
33     end = time.time()
34     q.put({
35         'time': end - start,
36         'algoritmo': 2,
37     })
38     print("Finalizado: %s en %f \n" % (multiprocessing.current_process().name, end -
           start))
39     print(lista)
```

– program.py –

```
41 úú
42
43
44 def process_3( lista , q):
45     print("Empezando: %s \n" % multiprocessing.current_process().name)
46     start = time.time()
47     lista = sorted( lista )
48     end = time.time()
49     q.put({
50         'time': end - start,
51         'algoritmo': 3,
52     })
53     print(" Finalizado : %s en %f \n" % (multiprocessing.current_process().name, end -
           start))
54     print( lista )
55
56
57 def main():
58     try:
59         size = int(input("Introduce el ñtamao de la lista : "))
60     except ValueError:
61         size = random.randint(1, 10000)
62         print('No has introducido un únmero, el ñtamao de la lista á ser de: {0}'.
               format(size))
63
64     lista = [] # Para algoritmo minimos sucesivos
65     for n in range(size):
66         lista .append(random.randint(1, 1000))
67
68     lista_2 = list( lista ) # Para algoritmo ordenacion por insercion
69     lista_3 = list( lista ) # Para algoritmo ordenacion por insercion
70
71     queue = multiprocessing.Queue() # Cola para los resultados
72
73     p1 = multiprocessing.Process(name='Minimos sucesivos', target=process_1, args=(
           lista , queue))
74
75     p2 = multiprocessing.Process(name='Ordenacion por seleccion (Algoritmo inestable)',
           target=process_2,
76                                  args=( lista_2 , queue))
```

– program.py –

```
76 úúñúñá
77
78     p3 = multiprocessing.Process(name='Ordenacion por Timsort', target=process_3,
79                                  args=(lista_3, queue))
80
81     # Iniciar procesos
82     p1.start()
83     p2.start()
84     p3.start()
85     # Espera hasta que el proceso haya terminado su trabajo
86     p1.join()
87     p2.join()
88     p3.join()
89
90     times = []
91     times.append(queue.get())
92     times.append(queue.get())
93     times.append(queue.get())
94     times = sorted(times, key=lambda k: k['algoritmo'])  # Ordenamos la lista por
           algoritmo
95
96     x = np.array([1, 9, 18])  # Eje x
97     y = np.array([times[0]['time'], times[1]['time'], times[2]['time']])  # Eje y
98     my_xticks = ['Minimos sucesivos', 'Ordenacion por seleccion', 'Ordenacion por
           Timsort']  # Nombres de los algoritmos
99     plt.xticks(x, my_xticks)  # Asociamos nombres de algoritmos a eje x
100
101    plt.plot(x, y)  # Creamos el grafico
102    plt.xlabel('Algoritmo')  # Titulo del eje x
103    plt.ylabel('Tiempo (segundos)')  # Titulo del eje y
104
105    plt.show()  # Mostramos el grafico
106
107
108 if __name__ == "__main__":
109     main()
```

– algoritmos˙ordenacion.py –

```
0  class Algoritmos(object):
1      """
2      Clase que contiene los siguientes algoritmos de ordenacion
3          – Ordenamiento por seleccion
4          – Ordenamiento por insercion
5      """
6
7      @staticmethod
8      def minimos_sucesivos( lista ):
9          """
10         Ordena una lista de únmeros
11         por el algoritmo minimos sucesivos.
12         Ordenamiento por óseleccin (Selection Sort en éingls).
13         :param lista: Lista con los únmeros a ordenar
14         """
15         for i in range(len( lista ) - 1):
16             for j in range(i + 1, len( lista )):
17                 if lista [i] > lista [j]:
18                     aux = lista [i]
19                     lista [i] = lista [j]
20                     lista [j] = aux
21
22     @staticmethod
23     def insertion_sort ( lista ):
24         """
25         Ordena una lista de únmeros
26         por el algoritmo ordenamiento por óinsercin
27         (InsertionSort en éingls)
28         :param lista: Lista con los únmeros a ordenar
29         """
30         for i in range(1, len( lista ) - 1):
31             Algoritmos.__inserta ( lista , i + 1, lista [i + 1])
32
33     @staticmethod
34     def __inserta ( lista , k, v):
```

– algoritmos˙ordenacion.py –

```
36 úóéúúóéú
37         """
38         Metodo auxiliar para el algoritmo de ordenacion
39          insertion_osrt
40         :param lista : Lista con los únmeros a ordenar
41         :param k: óPosicin del siguiente elemento de la lista
42         :param v: Siguiente elemento de la lista
43         """
44         for i in range(k):
45             if lista[i] > v:
46                 lista.pop(k)
47                 lista.insert(i, v)
48                 return
```

5. Flujo y entorno de trabajo.

Hemos utilizado las siguientes ramas en **Git**:

- Master $\mapsto$ Emula la rama maestra
- Develop $\mapsto$ Emula la rama de desarrollo
- Pro $\mapsto$ Emula la rama de producción

A continuación, se muestra los distintos cambios realizados en el repositorio, utilizando la herramienta **UpSource**.

6. Manual de uso.

1. Abrimos el terminal.
2. Introducimos en el terminal el comando:

```
python3 program.py
```

3. Indicamos el tamaño de la lista que queremos que ordene.

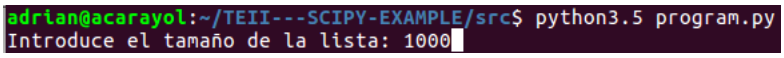

Figura 1: Ejecución de programa.

4. Esperamos a que el programa acabe.

5. Se visualizará un gráfico con los tiempos comparativos de los 3 algoritmos, además de la lista ordenada por consola.

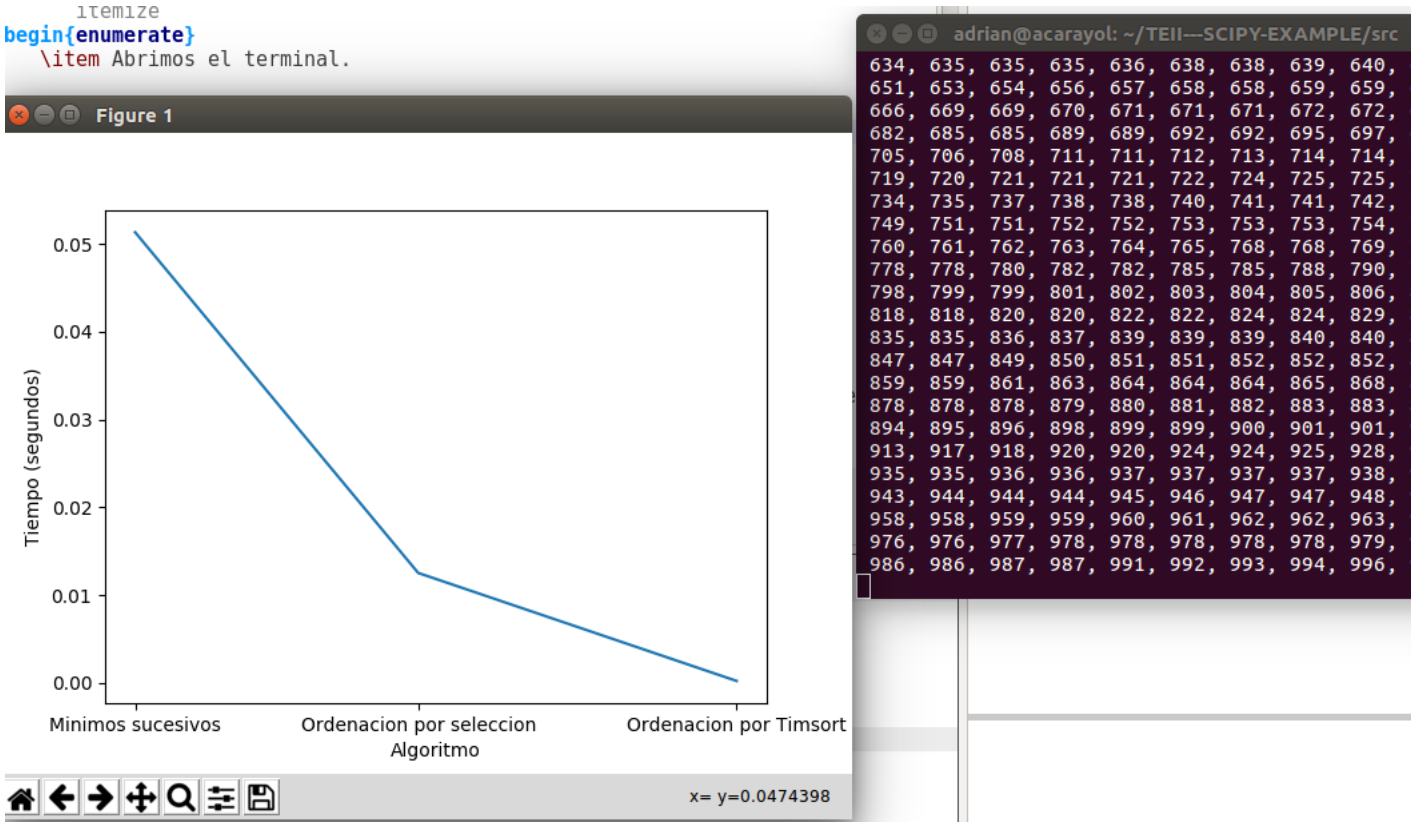

Figura 2: Resultados de la ejecución.

6. Cerramos el gráfico y el programa habrá finalizado.

7. Conclusión

El uso de **Latex** para preparar documentación es muy útil, ya que proporciona herramientas más completas y sofisticadas que las típicas herramientas de oficina.

Respecto a **Git**, me parece una herramienta ideal para trabajar en equipo, remotamente, estés donde estés, y te permite poder realizar revisiones de todas las versiones que has subido a tu repositorio, por lo que te permite llevar un mayor control sobre todos los archivos.

Finalmente, **Python**, me parece un lenguaje muy interesante y con una sintaxis sencilla, que, junto a sus cientos de módulos, te permite programar de una manera muy rápida.