Python con Jupyter

May 14, 2017

In [2]:
```
# Ejemplo de Python con Jupyter
```

In [5]:
```
class Algoritmos(object):
    """
    Clase que contiene los siguientes algoritmos de ordenacion
        - Ordenamiento por seleccion
        - Ordenamiento por insercion
    """

    @staticmethod
    def minimos_sucesivos(lista):
        """
        Ordena una lista de números
        por el algoritmo minimos sucesivos.
        Ordenamiento por selección (Selection Sort en inglés).
        :param lista: Lista con los números a ordenar
        """
        for i in range(len(lista) - 1):
            for j in range(i + 1, len(lista)):
                if lista[i] > lista[j]:
                    aux = lista[i]
                    lista[i] = lista[j]
                    lista[j] = aux

    @staticmethod
    def insertion_sort(lista):
        """
        Ordena una lista de números
        por el algoritmo ordenamiento por inserción
        (InsertionSort en inglés)
        :param lista: Lista con los números a ordenar
        """
        for i in range(1, len(lista) - 1):
            Algoritmos.__inserta(lista, i + 1, lista[i + 1])

    @staticmethod
    def __inserta(lista, k, v):
```

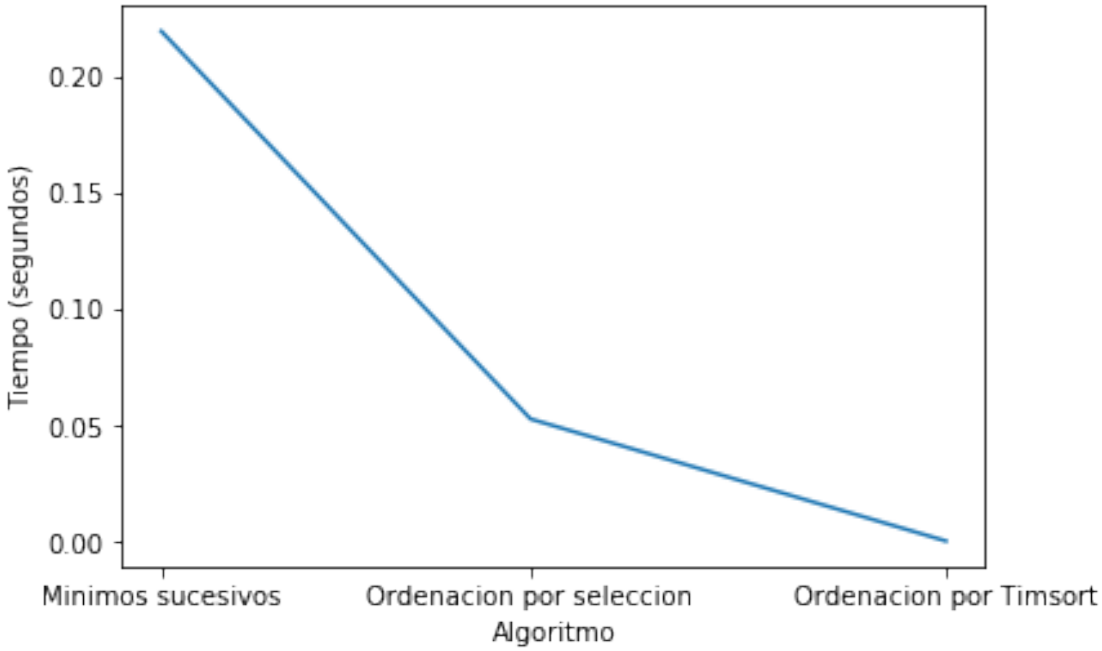
```
            """
            Metodo auxiliar para el algoritmo de ordenacion
            insertion_osrt
            :param lista: Lista con los números a ordenar
            :param k: Posición del siguiente elemento de la lista
            :param v: Siguiente elemento de la lista
            """
            for i in range(k):
                if lista[i] > v:
                    lista.pop(k)
                    lista.insert(i, v)
                    return
```

In [6]:
```
import random
import time
import multiprocessing
import numpy as np
import matplotlib.pyplot as plt
from src.algoritmos_ordenacion import Algoritmos


def process_1(lista, q):
    """
    Funcion para el proceso 1
    :param lista: Lista de números a ordenar
    """
    start = time.time()
    print("Empezando: %s \n" % multiprocessing.current_process().name)
    Algoritmos.minimos_sucesivos(lista)
    end = time.time()
    q.put({
        'time': end - start,
        'algoritmo': 1,
    })
    print("Finalizado: %s en %f \n" % (multiprocessing.current_process().name, end - sta
    print(lista)


def process_2(lista, q):
    """
    Funcion para el proceso 2
    :param lista: Lista de números a ordenar
    """
    print("Empezando: %s \n" % multiprocessing.current_process().name)
    start = time.time()
    Algoritmos.insertion_sort(lista)
    end = time.time()
    q.put({
```

```
        'time': end - start,
        'algoritmo': 2,
    })
    print("Finalizado: %s en %f \n" % (multiprocessing.current_process().name, end - sta
    print(lista)


def process_3(lista, q):
    print("Empezando: %s \n" % multiprocessing.current_process().name)
    start = time.time()
    lista = sorted(lista)
    end = time.time()
    q.put({
        'time': end - start,
        'algoritmo': 3,
    })
    print("Finalizado: %s en %f \n" % (multiprocessing.current_process().name, end - sta
    print(lista)


def main():
    try:
        size = int(input("Introduce el tamaño de la lista: "))
    except ValueError:
        size = random.randint(1, 10000)
        print('No has introducido un número, el tamaño de la lista será de: {0}'.format(

    lista = []  # Para algoritmo minimos sucesivos
    for n in range(size):
        lista.append(random.randint(1, 1000))

    lista_2 = list(lista)  # Para algoritmo ordenacion por insercion
    lista_3 = list(lista)  # Para algoritmo ordenacion por insercion

    queue = multiprocessing.Queue()  # Cola para los resultados

    p1 = multiprocessing.Process(name='Minimos sucesivos', target=process_1, args=(lista

    p2 = multiprocessing.Process(name='Ordenacion por seleccion (Algoritmo inestable)',
                                 args=(lista_2, queue))

    p3 = multiprocessing.Process(name='Ordenacion por Timsort', target=process_3,
                                 args=(lista_3, queue))

    # Iniciar procesos
    p1.start()
    p2.start()
    p3.start()
```

```
            # Espera hasta que el proceso haya terminado su trabajo
            p1.join()
            p2.join()
            p3.join()

            times = []
            times.append(queue.get())
            times.append(queue.get())
            times.append(queue.get())
            times = sorted(times, key=lambda k: k['algoritmo'])  # Ordenamos la lista por algori

            x = np.array([1, 9, 18])  # Eje x
            y = np.array([times[0]['time'], times[1]['time'], times[2]['time']])  # Eje y
            my_xticks = ['Minimos sucesivos', 'Ordenacion por seleccion', 'Ordenacion por Timsor
            plt.xticks(x, my_xticks)  # Asociamos nombres de algoritmos a eje x

            plt.plot(x, y)  # Creamos el grafico
            plt.xlabel('Algoritmo')  # Titulo del eje x
            plt.ylabel('Tiempo (segundos)')  # Titulo del eje y

            plt.show()  # Mostramos el grafico


        if __name__ == "__main__":
            main()
```

```
Introduce el tamaño de la lista: 2000
Empezando: Minimos sucesivos

Empezando: Ordenacion por Timsort

Finalizado: Ordenacion por Timsort en 0.000453

[1, 1, 2, 3, 3, 4, 4, 5, 7, 8, 8, 8, 8, 9, 10, 11, 11, 11, 12, 12, 13, 13, 14, 14, 14, 15, 15, 1
Empezando: Ordenacion por seleccion (Algoritmo inestable)

Finalizado: Ordenacion por seleccion (Algoritmo inestable) en 0.052893

[1, 1, 2, 3, 4, 4, 5, 7, 8, 8, 8, 8, 9, 10, 11, 11, 11, 12, 12, 13, 13, 14, 14, 14, 15, 15, 15,
Finalizado: Minimos sucesivos en 0.219540

[1, 1, 2, 3, 3, 4, 4, 5, 7, 8, 8, 8, 8, 9, 10, 11, 11, 11, 12, 12, 13, 13, 14, 14, 14, 15, 15, 1
```


0.20
0.15
0.10
0.05
0.00
Tiempo (segundos)
Minimos sucesivos
Ordenacion por seleccion
Ordenacion por Timsort
Algoritmo

In [ ]: