Ordiales Caballero, Iñaky
N.L. 33
EDA I

# Tarea 10
# (Eficiencia en tiempo de algoritmos de ordenamiento)

**Instrucciones:**
Ejecuten y realicen análisis de tiempos, comparando los 4 algoritmos con sus respectivas gráficas. Córranlo para valores de n hasta 500, hasta 2000, hasta 4000 y hasta 6000.

## Código:

```python
import matplotlib.pyplot as plt


import random
import time


def sortBurbuja(lista):
    n = len(lista)
    i=0
    while i<n-1:
        j=i+1
        #print ("pasada #: {}\n".format(j))
        while j<n:
            if lista[i]>lista[j]:
                aux = lista[i]
                lista[i]=lista[j]
                lista[j]=aux
            j=j+1
        i=i+1


def insertionSort(n_lista):
    for index in range(1,len(n_lista)):
        actual = n_lista[index]
        posicion = index
        #print("valor a ordenar = {}".format(actual))
        while posicion>0 and n_lista[posicion-1]>actual:
            n_lista[posicion]=n_lista[posicion-1]
            posicion = posicion-1
        n_lista[posicion]=actual
    return n_lista

```

```python
def quicksort(lista):
    quicksort_aux(lista,0,len(lista)-1)

def quicksort_aux(lista,inicio, fin):
    if inicio < fin:

        pivote = particion(lista,inicio,fin)

        quicksort_aux(lista, inicio, pivote-1)
        quicksort_aux(lista, pivote+1, fin)
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

```python
def particion(lista, inicio, fin):
    #Se asigna como pivote en número de la primera localidad
    pivote = lista[inicio]
    #print("Valor del pivote {}".format(pivote))
    #Se crean dos marcadores
    izquierda = inicio+1
    derecha = fin
    #print("Índice izquierdo {}".format(izquierda))
    #print("Índice derecho {}".format(derecha))


    bandera = False
    while not bandera:
        while izquierda <= derecha and lista[izquierda] <= pivote:
            izquierda = izquierda + 1
        while lista[derecha] >= pivote and derecha >=izquierda:
            derecha = derecha -1
        if derecha < izquierda:
            bandera= True
        else:
            temp=lista[izquierda]
            lista[izquierda]=lista[derecha]
            lista[derecha]=temp

    #print(lista)

    temp=lista[inicio]
    lista[inicio]=lista[derecha]
    lista[derecha]=temp
    return derecha
```

```python
def mergeSort(lista, inicio, fin):
    if len(lista)==1 or len(lista)==0:
        return lista
    else:
        arrBajo = mergeSort(lista[inicio:(fin//2)], 0,len(lista)-1)
        arrAlto = mergeSort(lista[(fin//2):fin], 0,len(lista)-1)
        com = merge(arrBajo, arrAlto)
    return com

def merge(arrBajo,arrAlto):
    # print("Merge in :", arrBajo, arrAlto)
    t=[]
    while(len(arrBajo)>0 and len(arrAlto)>0):
        if(arrBajo[0] <= arrAlto[0] ):
            t.append(arrBajo.pop(0))
        else:
            t.append(arrAlto.pop(0))

    # agregar los elemntos faltantes del primer sub arreglo
    if(len(arrBajo)>0):
        t=t+arrBajo

    # agregar los elemntos faltantes del segundo sub arreglo
    if(len(arrAlto)>0):
        t=t+arrAlto

    # print("Merge out:",t)
    return t
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

```python
def graficarTiempos(numDatos,tiemposBurbuja, tiemposInsertionSort,tiemposQuick,tiemposMerge):
    fig, ax= plt.subplots()
    ax.plot(numDatos, tiemposBurbuja, label ="Ordenamiento de la Burbuja", marker="*", color ="r")
    ax.plot(numDatos, tiemposInsertionSort, label ="Ordenamiento de Insertion", marker=".", color ="g")
    ax.plot(numDatos, tiemposQuick, label ="Ordenamiento de Quick", marker="x", color ="b")
    ax.plot(numDatos, tiemposMerge, label ="Ordenamiento de Merge", marker="+", color ="y")

    ax.set_xlabel("Numero de Datos")
    ax.set_ylabel("Tiempo")
    ax.grid(True)
    ax.legend(loc=2)

    plt.title("Tiempos de ejecución [seg]")
    plt.show()
```

```python
def eficienciaAlgoritmos(numDatos):
    tiemposBurbuja= []
    tiemposQuick = []
    tiemposInsertionSort = []
    tiemposMerge = []
    for n in numDatos:
        #lista_Burbuja = random.sample(range(1000), n)
        lista_Burbuja = random.sample(range(0, 100000), n)    #genera(n)
        lista_Quick = lista_Burbuja.copy()
        lista_insertionSort = lista_Burbuja.copy()
        lista_merge = lista_Burbuja.copy()

        print("\n=========\n")
        #print (lista_Burbuja)
        t0 = time.monotonic()
        sortBurbuja(lista_Burbuja)
        dt = round(time.monotonic() -t0,6)
        tiemposBurbuja.append(dt)
        print("\nBurbuja(n={}): \tTiempo transcurrido: {} seg".format(n,round(dt,6)))
        #print("\nLISTA ORDENADA\n")
        #print (lista_Burbuja)

        t0 = time.monotonic()
        insertionSort(lista_insertionSort)
        dt = round(time.monotonic() -t0,6)
        tiemposInsertionSort.append(dt)
        print("InsertionSort(n={}): \tTiempo transcurrido: {} seg".format(n,round(dt,6)))
        #print("\nLISTA ORDENADA\n")
        #print (lista_insertionSort)
```

```python
        t0 = time.monotonic()
        # lista_Quick.sort()
        quicksort(lista_Quick)
        dt = round(time.monotonic() -t0,6)
        tiemposQuick.append(dt)
        print("QuickSort(n={}):\tTiempo transcurrido: {} seg".format(n,round(dt,6)))

        #COLOCAR AQUI EL CODIGO FALTANTE
        t0 = time.monotonic()
        mergeSort(lista_merge,0,len(lista_merge)-1)
        dt = round(time.monotonic() -t0,6)
        tiemposMerge.append(dt)
        print("MergeSort(n={}):\tTiempo transcurrido: {} seg".format(n,round(dt,6)))

    print("\n=========\n")
    return tiemposBurbuja,tiemposInsertionSort,tiemposQuick,tiemposMerge
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

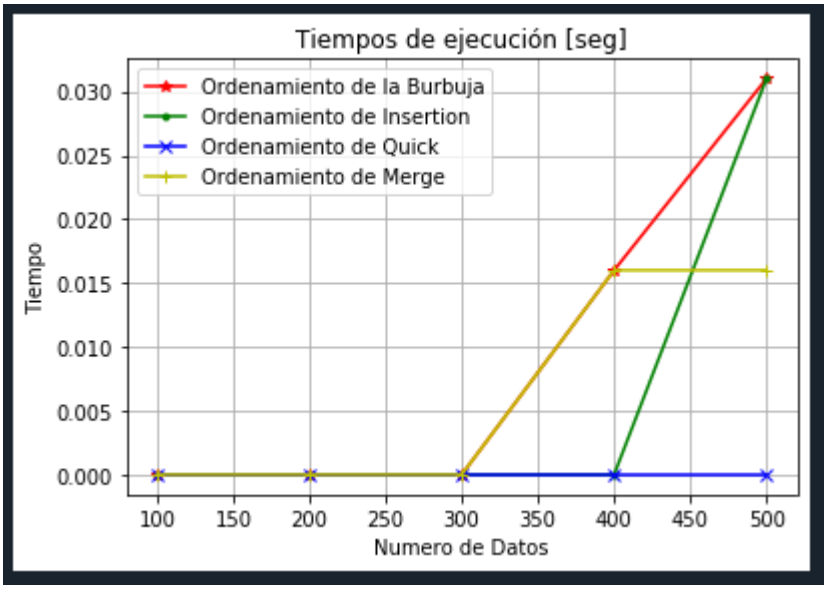
```
def main():

    #Tamaños de la lista de números aleatorios a generar
    #numDatos = [100,200,300,400,....,2000]
    numDatos = [n*100 for n in range(1,21)]

    tiemposBurbuja= []
    tiemposInsertionSort = []
    tiemposQuick = []
    tiemposMerge = []
    tiemposBurbuja, tiemposInsertionSort, tiemposQuick, tiemposMerge = eficienciaAlgoritmos(numDatos)
    print ("Tiempos de ejecucion burbuja: \n{}\n".format(tiemposBurbuja))
    print ("Tiempos de ejecucion InsertionSort: \n{}\n".format(tiemposInsertionSort))
    print ("Tiempos de ejecucion QuickSort: \n{}\n".format(tiemposQuick))
    print ("Tiempos de ejecucion MergeSort: \n{}\n".format(tiemposMerge))

    # Se imprimen los tiempos totales de ejecución
    # Para calcular el tiempo total se aplica la función sum() a las listas de tiempo
    print("\nANALISIS DE TIEMPOS:")
    print("======================")
    print("Tiempo total de ejecución en burbuja sort \t{} [s]".format(sum(tiemposBurbuja)))
    print("Tiempo total de ejecución en insert sort \t{} [s]".format(sum(tiemposInsertionSort)))
    print("Tiempo total de ejecución en quick sort \t{} [s]".format(sum(tiemposQuick)))
    print("Tiempo total de ejecución en merge sort \t{} [s]".format(sum(tiemposMerge)))

    graficarTiempos(numDatos,tiemposBurbuja,tiemposInsertionSort,tiemposQuick,tiemposMerge)

main()
```

## Resultados:

### Para n hasta 500:

```
Tiempos de ejecucion burbuja:
[0.0, 0.0, 0.0, 0.016, 0.031]

Tiempos de ejecucion InsertionSort:
[0.0, 0.0, 0.0, 0.0, 0.031]

Tiempos de ejecucion QuickSort:
[0.0, 0.0, 0.0, 0.0, 0.0]

Tiempos de ejecucion MergeSort:
[0.0, 0.0, 0.0, 0.016, 0.016]


ANALISIS DE TIEMPOS:
======================
Tiempo total de ejecución en burbuja sort       0.047 [s]
Tiempo total de ejecución en insert sort        0.031 [s]
Tiempo total de ejecución en quick sort         0.0 [s]
Tiempo total de ejecución en merge sort         0.032 [s]
```

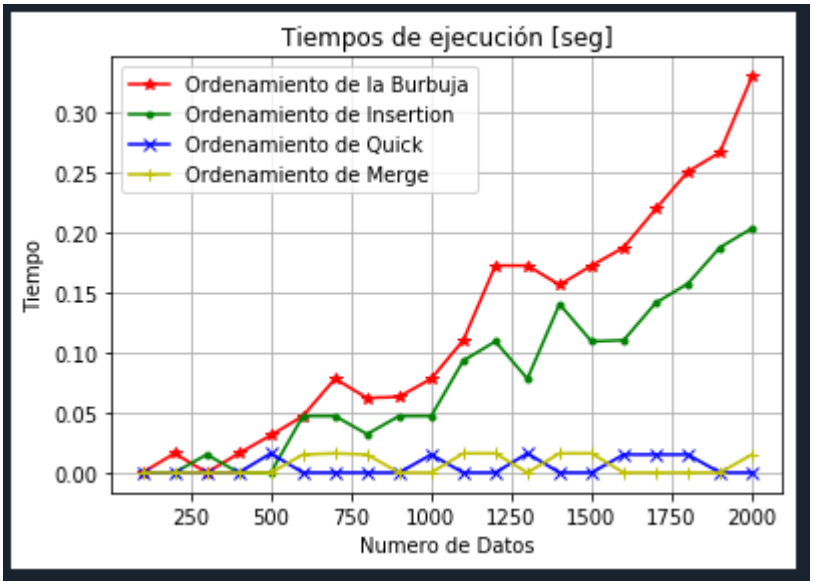
Ordiales Caballero, Iñaky
N.L. 33
EDA I




**Para n hasta 2000:**

```
Tiempos de ejecucion burbuja:
[0.0, 0.016, 0.0, 0.016, 0.031, 0.047, 0.078, 0.062, 0.063, 0.078,
0.11, 0.172, 0.172, 0.156, 0.172, 0.187, 0.219, 0.25, 0.266, 0.329]

Tiempos de ejecucion InsertionSort:
[0.0, 0.0, 0.015, 0.0, 0.0, 0.047, 0.047, 0.032, 0.047, 0.047,
0.093, 0.109, 0.078, 0.14, 0.109, 0.11, 0.141, 0.157, 0.187, 0.203]

Tiempos de ejecucion QuickSort:
[0.0, 0.0, 0.0, 0.0, 0.016, 0.0, 0.0, 0.0, 0.0, 0.015, 0.0, 0.0,
0.016, 0.0, 0.0, 0.015, 0.015, 0.015, 0.0, 0.0]

Tiempos de ejecucion MergeSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.015, 0.016, 0.015, 0.0, 0.0, 0.016,
0.016, 0.0, 0.016, 0.016, 0.0, 0.0, 0.0, 0.0, 0.015]


ANALISIS DE TIEMPOS:
======================
Tiempo total de ejecución en burbuja sort       2.424 [s]
Tiempo total de ejecución en insert sort        1.562 [s]
Tiempo total de ejecución en quick sort         0.092 [s]
Tiempo total de ejecución en merge sort         0.125 [s]
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

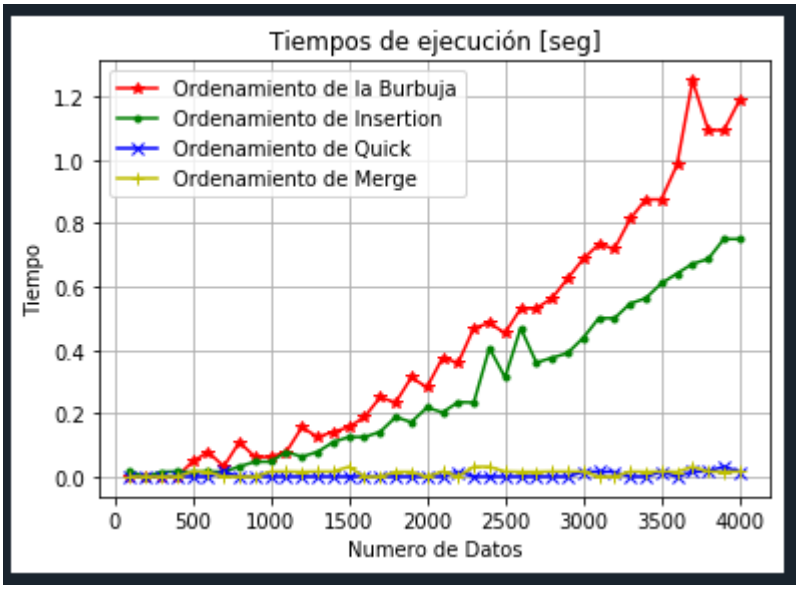



**Para n hasta 4000:**

```
Tiempos de ejecucion burbuja:
[0.0, 0.0, 0.0, 0.0, 0.047, 0.078, 0.032, 0.109, 0.062, 0.062, 0.078,
0.156, 0.125, 0.141, 0.156, 0.188, 0.25, 0.234, 0.313, 0.282, 0.375,
0.359, 0.469, 0.485, 0.453, 0.531, 0.531, 0.563, 0.625, 0.687, 0.734,
0.719, 0.813, 0.875, 0.875, 0.984, 1.25, 1.094, 1.093, 1.188]

Tiempos de ejecucion InsertionSort:
[0.016, 0.0, 0.015, 0.016, 0.015, 0.016, 0.015, 0.032, 0.047, 0.047,
0.078, 0.063, 0.078, 0.109, 0.125, 0.125, 0.141, 0.188, 0.172, 0.218,
0.203, 0.235, 0.234, 0.406, 0.313, 0.469, 0.36, 0.375, 0.39, 0.438, 0.5,
0.5, 0.546, 0.563, 0.61, 0.641, 0.672, 0.687, 0.75, 0.75]

Tiempos de ejecucion QuickSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.016, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.015, 0.016, 0.015, 0.0, 0.0, 0.015, 0.0, 0.016, 0.016, 0.032,
0.015]

Tiempos de ejecucion MergeSort:
[0.0, 0.0, 0.0, 0.0, 0.016, 0.015, 0.0, 0.0, 0.0, 0.016, 0.016, 0.015,
0.016, 0.016, 0.031, 0.0, 0.0, 0.015, 0.015, 0.0, 0.016, 0.0, 0.031,
0.031, 0.016, 0.015, 0.015, 0.016, 0.016, 0.016, 0.0, 0.0, 0.016, 0.015,
0.016, 0.015, 0.031, 0.016, 0.015, 0.016]
```

```
ANALISIS DE TIEMPOS:
========================
Tiempo total de ejecución en burbuja sort       17.016 [s]
Tiempo total de ejecución en insert sort        11.158 [s]
Tiempo total de ejecución en quick sort         0.17099999999999999 [s]
Tiempo total de ejecución en merge sort         0.4830000000000002 [s]
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

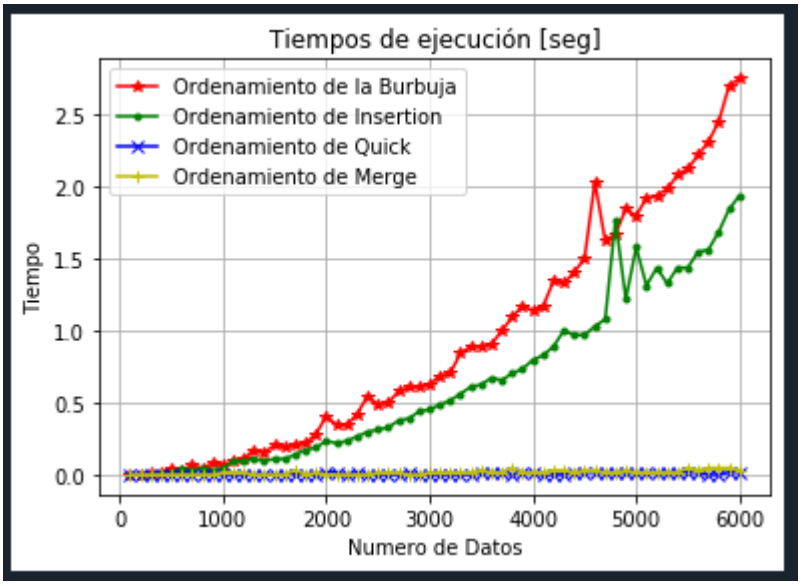



**Para n hasta 6000:**

```
Tiempos de ejecucion burbuja:
[0.0, 0.0, 0.016, 0.015, 0.047, 0.032, 0.063, 0.047, 0.078, 0.062, 0.094, 0.109, 0.172,
0.156, 0.203, 0.188, 0.203, 0.218, 0.281, 0.406, 0.344, 0.344, 0.422, 0.547, 0.484, 0.5,
0.578, 0.609, 0.609, 0.625, 0.687, 0.704, 0.844, 0.89, 0.891, 0.907, 1.0, 1.094, 1.172,
1.141, 1.172, 1.343, 1.328, 1.406, 1.5, 2.032, 1.625, 1.672, 1.844, 1.797, 1.922, 1.938,
1.984, 2.079, 2.125, 2.219, 2.312, 2.453, 2.703, 2.75]

Tiempos de ejecucion InsertionSort:
[0.0, 0.0, 0.0, 0.016, 0.015, 0.046, 0.031, 0.047, 0.031, 0.047, 0.093, 0.094, 0.109, 0.094,
0.11, 0.109, 0.14, 0.172, 0.188, 0.234, 0.218, 0.234, 0.265, 0.297, 0.313, 0.328, 0.375,
0.391, 0.438, 0.453, 0.485, 0.515, 0.562, 0.61, 0.625, 0.672, 0.656, 0.703, 0.734, 0.797,
0.828, 0.891, 1.0, 0.969, 0.969, 1.031, 1.078, 1.766, 1.218, 1.578, 1.313, 1.437, 1.328,
1.437, 1.438, 1.547, 1.563, 1.687, 1.844, 1.937]

Tiempos de ejecucion QuickSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.016, 0.016, 0.0, 0.016, 0.0, 0.0, 0.0, 0.0, 0.015, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.015, 0.015, 0.016, 0.0, 0.016, 0.015, 0.0, 0.0, 0.0, 0.015, 0.015, 0.016, 0.016, 0.015,
0.016, 0.016, 0.015, 0.016, 0.016, 0.016, 0.015, 0.016, 0.0, 0.0, 0.016, 0.016]

Tiempos de ejecucion MergeSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.016, 0.016, 0.016, 0.0, 0.0, 0.0, 0.0,
0.032, 0.0, 0.016, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015, 0.016, 0.016, 0.0, 0.0, 0.016, 0.015,
0.016, 0.016, 0.016, 0.031, 0.016, 0.015, 0.047, 0.015, 0.016, 0.016, 0.031, 0.032, 0.016,
0.031, 0.031, 0.015, 0.016, 0.031, 0.016, 0.016, 0.016, 0.015, 0.015, 0.047, 0.031, 0.047,
0.047, 0.047, 0.031]
```

```
ANALISIS DE TIEMPOS:
======================
Tiempo total de ejecución en burbuja sort       54.986000000000004 [s]
Tiempo total de ejecución en insert sort        38.105999999999995 [s]
Tiempo total de ejecución en quick sort         0.3750000000000002 [s]
Tiempo total de ejecución en merge sort         0.9090000000000006 [s]
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I



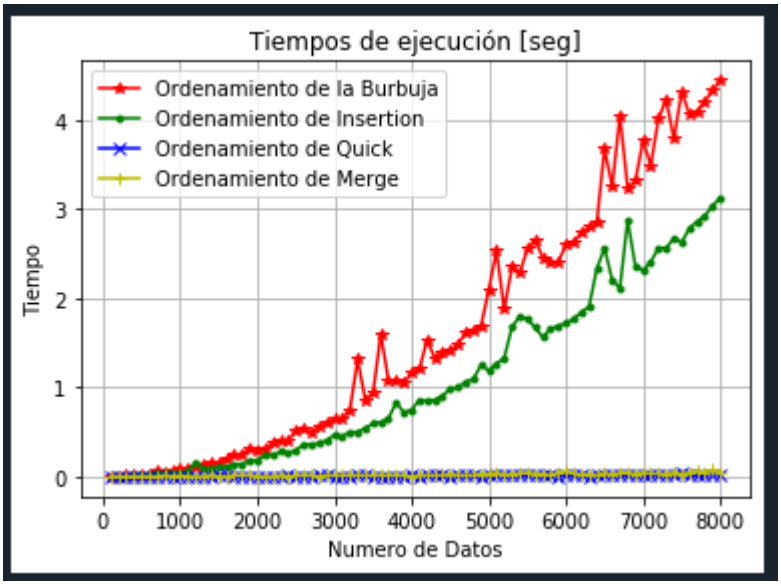

**Para n hasta 8000:**

```
Tiempos de ejecucion burbuja:
[0.0, 0.0, 0.016, 0.015, 0.016, 0.016, 0.063, 0.047, 0.062, 0.078, 0.094, 0.11, 0.125,
0.156, 0.157, 0.203, 0.25, 0.25, 0.313, 0.297, 0.313, 0.375, 0.406, 0.407, 0.515, 0.531,
0.5, 0.562, 0.593, 0.64, 0.656, 0.75, 1.313, 0.859, 0.938, 1.594, 1.078, 1.078, 1.047,
1.172, 1.203, 1.532, 1.328, 1.391, 1.407, 1.485, 1.609, 1.64, 1.688, 2.094, 2.531, 1.875,
2.344, 2.281, 2.547, 2.657, 2.453, 2.407, 2.406, 2.594, 2.625, 2.734, 2.797, 2.844, 3.672,
3.25, 4.047, 3.234, 3.313, 3.766, 3.485, 4.015, 4.219, 3.782, 4.312, 4.062, 4.094, 4.203,
4.328, 4.437]

Tiempos de ejecucion InsertionSort:
[0.0, 0.0, 0.0, 0.016, 0.015, 0.031, 0.046, 0.047, 0.047, 0.032, 0.062, 0.156, 0.094, 0.094,
0.109, 0.109, 0.14, 0.141, 0.172, 0.188, 0.234, 0.234, 0.282, 0.265, 0.297, 0.36, 0.36,
0.375, 0.407, 0.469, 0.453, 0.5, 0.5, 0.547, 0.594, 0.609, 0.656, 0.828, 0.719, 0.75, 0.859,
0.843, 0.859, 0.906, 0.984, 1.0, 1.062, 1.094, 1.25, 1.187, 1.266, 1.328, 1.672, 1.797,
1.765, 1.671, 1.563, 1.656, 1.688, 1.719, 1.766, 1.844, 1.906, 2.328, 2.562, 2.203, 2.109,
2.875, 2.359, 2.312, 2.406, 2.563, 2.562, 2.671, 2.625, 2.781, 2.86, 2.922, 3.032, 3.125]
```

```
Tiempos de ejecucion QuickSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.016, 0.016, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.016, 0.0, 0.0, 0.0, 0.016, 0.015, 0.0, 0.0, 0.0, 0.016,
0.016, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015, 0.0, 0.016, 0.016, 0.016, 0.0, 0.015, 0.016, 0.016,
0.0, 0.016, 0.016, 0.015, 0.016, 0.015, 0.016, 0.016, 0.015, 0.016, 0.0, 0.016, 0.015,
0.016, 0.0, 0.016, 0.016, 0.016, 0.016, 0.016, 0.016, 0.016, 0.016, 0.015, 0.016, 0.016,
0.032, 0.016, 0.015, 0.015, 0.031, 0.016]

Tiempos de ejecucion MergeSort:
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.016, 0.0, 0.015, 0.0, 0.0, 0.0, 0.015, 0.0, 0.0,
0.016, 0.015, 0.015, 0.0, 0.0, 0.0, 0.015, 0.0, 0.016, 0.015, 0.015, 0.0, 0.016, 0.016,
0.016, 0.015, 0.016, 0.015, 0.015, 0.016, 0.016, 0.016, 0.015, 0.0, 0.016, 0.016, 0.015,
0.015, 0.031, 0.016, 0.016, 0.015, 0.031, 0.031, 0.047, 0.016, 0.031, 0.032, 0.047, 0.031,
0.031, 0.015, 0.046, 0.062, 0.032, 0.015, 0.031, 0.015, 0.047, 0.015, 0.047, 0.046, 0.031,
0.047, 0.032, 0.032, 0.031, 0.047, 0.031, 0.047, 0.063, 0.047, 0.078, 0.047]
```

Ordiales Caballero, Iñaky
N.L. 33
EDA I

```
ANALISIS DE TIEMPOS:
=======================
Tiempo total de ejecución en burbuja sort       130.28599999999997 [s]
Tiempo total de ejecución en insert sort        86.918 [s]
Tiempo total de ejecución en quick sort         0.7090000000000005 [s]
Tiempo total de ejecución en merge sort         1.6660000000000001 [s]
```




## Conclusiones:

Los diferentes algoritmos de ordenamiento presentan diferentes ventajas y desventajas. Algunos son más sencillos de implementar, otros son más eficientes en proceso y otros son más eficientes en tiempo. En esta tarea, nos enfocaremos a en la eficiencia de tiempo de los programas.

En general para una cantidad de datos hasta 500, los cuatro algoritmos de ordenamiento tienen la misma eficiencia en tiempo. Con una diferencia menor de 0.05 s entre el que más tiempo y el que menos hizo. Por lo que la mejor elección de ordenamiento sería en mi opinión la de más fácil implementación.

Pata datos de hasta 2000 la diferencia sigue siendo mínima, de menos de 0.3 s para el más rápido y el más lento. Igual sería el de más fácil implementación.

Ya en datos de hasta 4000 la diferencia empieza a ser de más de un segundo, lo que al repetir posiblemente el proceso varias veces, ya es considerable. Además, vemos que casi desde un principio **bubble sort** es el que más tiempo conlleva, pero también **insertion sort** se despega en comparación a **quick** y **merge** sort.

En el caso de hasta 6000 y de hasta 8000 datos, se aprecia en las gráficas claramente que **bubble sort** es el de peor eficiencia, seguido con una diferencia

considerable por **insertive sort**, y muy por debajo casi pegados a cero están **merge** y **quick sort**. Ordenar los 8,000 datos le tomó 0.047 s a **merge sort** y 0.016 s a **quick sort**. En todos los datos quick sort, estuvo casi constante en 0.015 – 0.016 s, algo bastante impresionante para la cantidad de datos que fueron.

Como conclusión para cantidades grandes de datos es decir mayores a 2000 datos, en definitiva, es mucho más eficiente ya sea el **merge** o el **quick sort.** Siendo el último el de mayor eficiencia.