FACULTAD DE
INGENIERÍA

Tarea: Burbuja y comparación tiempos de ejecución
Materia: Algoritmos y Complejidad
Maestro: Juan Carlos Cueva Tello
Alumno: Rojas Aranda Jose Luis
Fecha: 12 de Febrero de 2019

Ing. Sistemas Inteligentes
Facultad de Ingeniería
UASLP

Análisis de complejidad: Bubble Sort

En este caso vamos a analizar la versión del algoritmo que no está optimizado, este no hace diferencia para cuando analizamos el peor de los casos, pero sí habrá una diferencia si analizamos el mejor de los casos

Pseudocodigo:

Bubble-Sort(A)	*cost*	*times*
1 For i = 0 to A.length - 1	*c1*	*n*
2 For j = 0 to A.length - 1 - i	*c2*	$\sum_{i=0}^{n-1}(n-1-t_i)$
3 if A[j] < A[j-1]	*c3*	$\sum_{i=0}^{n-1}(n-1-t_i)$
4 exchange A[j] with A[j-1]	*c4*	$\sum_{i=0}^{n-1}(n-1-t_i)$

Sumatoria total para el pero de los casos:

$$T(n) = c_1 n + c_2 \sum_{i=0}^{n-1}(n-1-t_i) + c_3 \sum_{i=0}^{n-1}(n-1-t_i) + c_4 \sum_{i=0}^{n-1}(n-1-t_i)$$

Tenemos que:

$$\sum_{i=0}^{n-1}(n-1-t_i) = \frac{n(n+1)}{2} - 1 - n$$

Si desarrollamos toda la operación nos da una ecuación no lineal:

$$= an^2 + bn + c$$

Solo se toma el factor de exponente más alto, y los demás se consideran despreciables lo que nos da una complejidad de:

$$= an^2$$
$$= \Theta(n^2)$$

Comparación de tiempo de ejecución:

Los algoritmos de ordenación están programados en python, decidí python debido a que es un lenguaje fácil y bueno para hacer análisis matemáticos.

Bubble sort python:

```
# Algoritmo de ordenación burbuja
def BubbleSort(lista):
    for num in range(len(lista)-1, 0, -1):
        for i in range(num):
            if lista[i] > lista[i+1]:
                temp = lista[i]
                lista[i] = lista[i+1]
                lista[i+1] = temp
```

Para analizar los tiempos de ejecución de los algoritmos se utilizaron las siguientes bibliotecas de python:

```
import datetime as time
import numpy as np
from matplotlib import pyplot as plt
```

Datetime: Nos sirve para poder medir el tiempo dentro del código
Numpy: Biblioteca de python para hacer operaciones matemáticas, la utilizamos para generar el arreglo aleatorio
Matplotlib: Permite graficar valores de un arreglo

Tiempo de ejecución (nanosegundos):

N	Bubble Sort	Insertion Sort	Merge Sort
100	2493	904	536
1,000	258323	91271	5898
10,000	28373173	9474124	80475
100,000	2.8e+10	7e+9	95095
1,000,000	3.055e+12	6.9e+11	1e+9

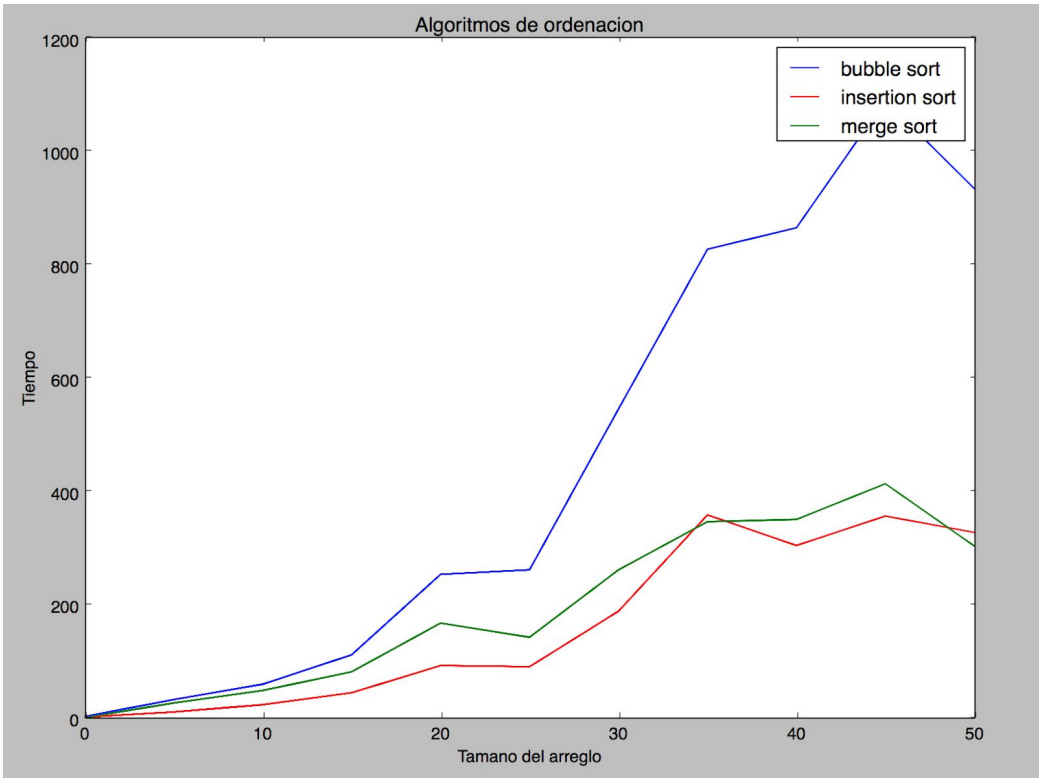
Graficación de tiempos de ejecución

Debido a que una vez que el número de elementos del arreglo es muy alto, la diferencia es muy grande no se alcance a apreciar las asíntotas de la complejidad de los algoritmos. Por lo tanto se van a graficar tamaños de n no muy grandes con un incremento bajo. El tiempo es medido en nanosegundos y se utiliza matplotlib para graficar los valores.

Especificaciones de la computadora en la cual se realizaron las corridas:

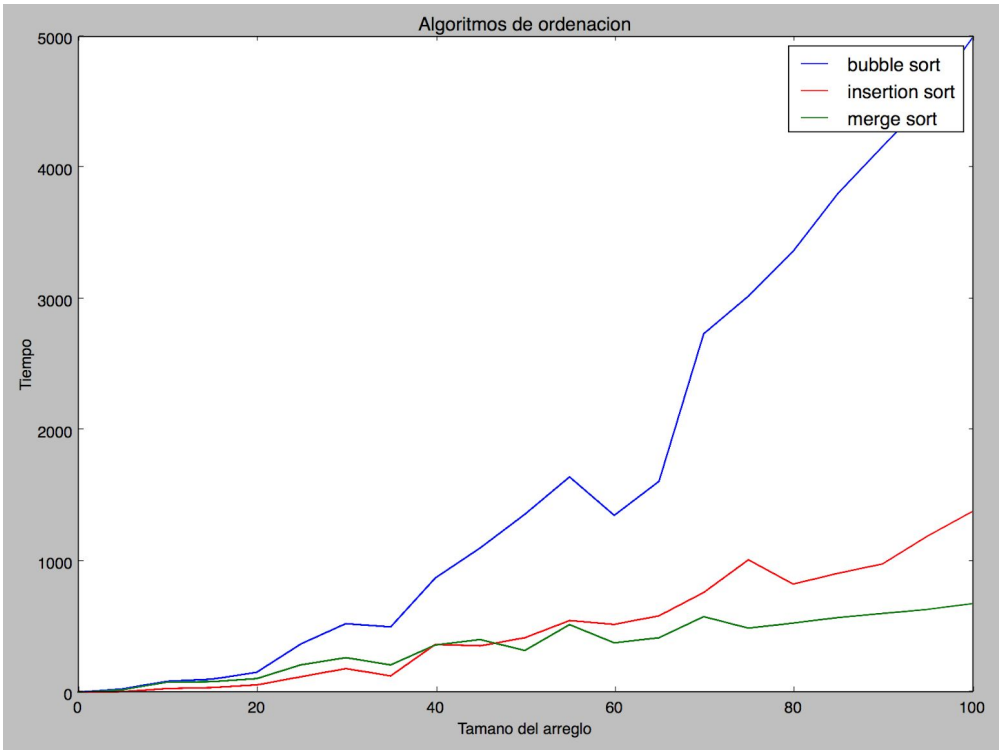
MacBook Pro (Retina, 15-inch, Mid 2014)
Processor 2.5 GHz Intel Core i7
Memory 16 GB 1600 MHz DDR3
Startup Disk Macintosh HD
Graphics Intel Iris Pro 1536 MB

Tamaño máximo de N = 50, con incrementos de 5:

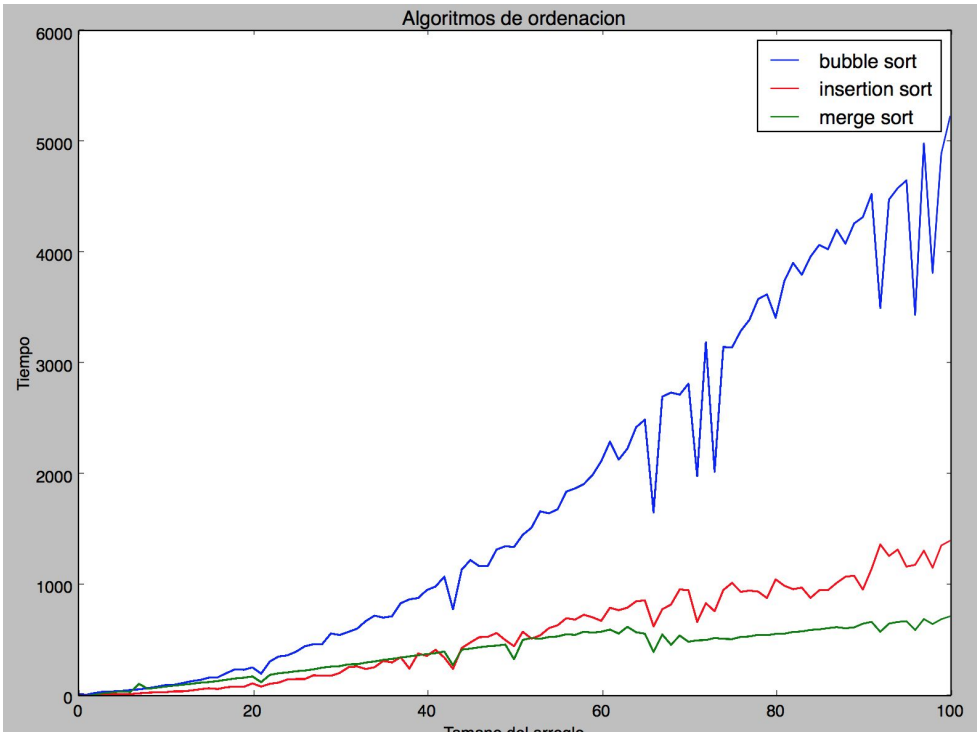


En esta corrida se puede observar claramente que en tamaños de arreglo pequeños no siempre el algoritmo de mejor complejidad es el más rápido. Siendo al inicio el insertion sort el más eficiente de los tres.

Tamaño máximo de N = 100, con incrementos de 5:

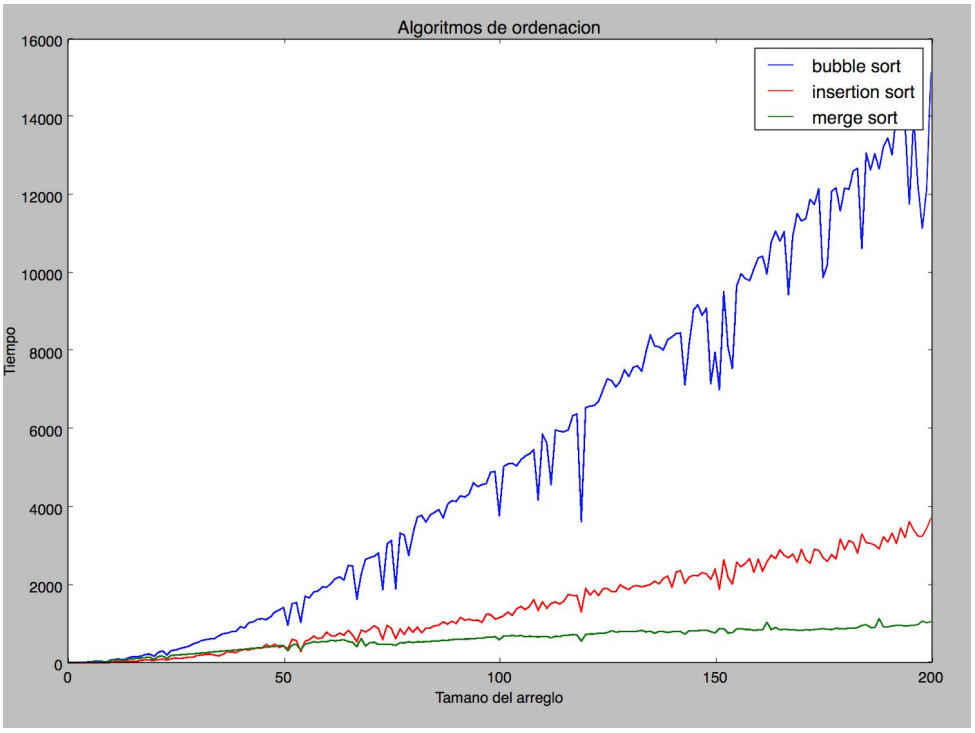


Tamaño máximo de N = 100, con incrementos de 1:



En estas corridas se observa como conforme el tamaño del arreglo va aumentando, el tiempo de ejecución de los algoritmos bubble sort e insertion van teniendo un crecimiento exponencial, mientras que el merge sort tiene un crecimiento más estable.

Tamaño máximo de N = 200, con incrementos de 1:

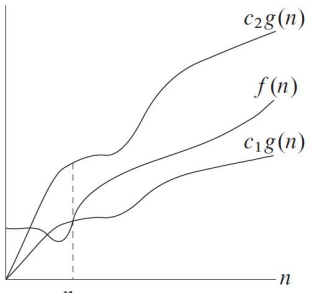


Encontrando el valor de n_0

El valor de n0, es aquel en el cual a partir de él no hay una intersección con otra función:

$$f_1(n_0) > f_2(n_0) > f_3(n_0)$$

Como se muestra en la siguiente imagen:



Para obtener este valor se tiene que analizar las gráficas. Debido a que los resultados no son constantes ya que el caso cambia cada vez, se van a hacer varias corridas y obtener el promedio como resultado

Corrida	**Valor de** n_0
1	55
2	50
3	53
4	53
Promedio	**52.75**