Evaluación del comportamiento de sistemas informáticos

Benchmark

Pablo Riutort Grande & Alfredo Ucendo Arroyo

5 de febrero de 2015

Resumen

***Benchmark**[1]: A level of quality that can be used as a standard when comparing other things.*
Un Benchmark es punto de referencia establecido sobre el que ordenadores o programas pueden ser medidos para comparar su rendiminto u otras características.

1. Introducción

En esta práctica [2] se pretende diseñar, construir y aplicar algunos criterios para establecer un benchmark para la paralelización de tareas. Para la realización del programa primero se han diseñado algunas pruebas que el ordenador debe realizar, después se ha construido el programa que ejecutará dichas pruebas sobre la máquina y finalmente se han analizado estos resultados.
En este informe se recojen los procesos anteriormente mencionados.

2. Diseño

Para el diseño de este benchmark se ha optado por una diversificación de tareas, de esta forma pretendemos medir el rendimiento del computador ante ciertos niveles de estrés con diferentes dificultades y tareas.

2.1. Contar

Contar es una función elemental del computador, por eso creemos que el primer nivel de dificultad debe incluir este tipo de operaciones.
Para esta primera prueba hemos aplicado los programas de *Fibonacci* y calcular el factorial de un número, ambos de manera recursiva.

2.2. Ordenar

La ordenación de elementos y vectores propone el siguiente nivel de dificultad. Aunque también es una tarea básica, la ordenación de vectores con según qué técnicas puede suponer un coste computacional alto. Para esta prueba en concreto ordenamos vectores con los algoritmos *sort* de Python y *merge sort*, este último también con una aproximación recursiva y ambos con coste $O(n \log n)$.

2.3. Matrices

El cálculo de matrices es nuestro último nivel de dificultad para este benchmark. Supone una tarea más compleja que las anteriores y una tarea compleja para el procesador si nos referimos a matrices grandes.
En este nivel proponemos la suma, la multiplicación y la potencia de matrices cuadradas de números no inferiores a 1000.

3. Construcción

Este benchmark se compone de cinco programas:

3.1. Benchmark.py

Es el programa central, en este programa se inicializan los hilos que ejecutarán los métodos para realizar cálculos, se llamarán a las diferentes funciones y finalmente terminará los hilos. Por último recogerá los resultados de la ejecución.
Este programa se ejecutará un total de cinco veces con un número de 20 hilos. Para cada ejecución llamará a todos los métodos que conforman nuestro benchmark pasando como parámetro el parámetro que la función necesite, este pueden ser un límite, una lista o una dimensión. Finalmente el programa leerá los archivos intermedios y efectuará una media mediante un comando **awk** generando el resultado en el archivo resultados.txt.

3.2. timing.py

Dada una función por parámetro calcula el tiempo de ejecución que ha conllevado dicha llamada, luego escribe los resultados en un fichero y lo cierra.

3.3. count.py

Contiene las funciones que implementan los métodos de *Fibonacci* y el cálculo factorial de un número respectivamente.
Recibe como parámetro un número que será el número factorial a calcular o la secuencia de *Fibonacci* asociada, concretamente 30.

3.4. sort.py

Contiene las funciones de ordenación, el primero es un método simple de ordenación que viene implementado en python, el segundo es un método de *merge sort* recursivo.
Estas funciones reciben como parámetro una lista de números aleatorios de rango 1000.

3.5. matrix.py

En este programa se hace uso de la librería ***NumPy***[3] para las construcción y el cálculo de matrices.
Implementa varias funciones que construyen filas y columnas de matrices de números aleatorios entre 1000 y 10000000 y de dimensión 20x20. Estas matrices luego se pueden utilizar en los métodos que suman, multiplican y elevan matrices respectivamente.

4. Conjunto de pruebas

Los ordenadores seleccionados para el conjunto de pruebas reunen las siguientes características:

Nombre	Arquitectura	Modelo
Ordenador 1	64 bits	AMD E1-1200 APU with Radeon(tm) HD Graphics
Ordenador 2	64 bits	Pentium(R) Dual-Core CPU T4300 @ 2.10GHz
Ordenador 3	32 bits	Intel(R) Atom(TM) CPU N270 @ 1.60GHz

Sobre estos ordenadores fue ejecutado el programa Benchmark.py.

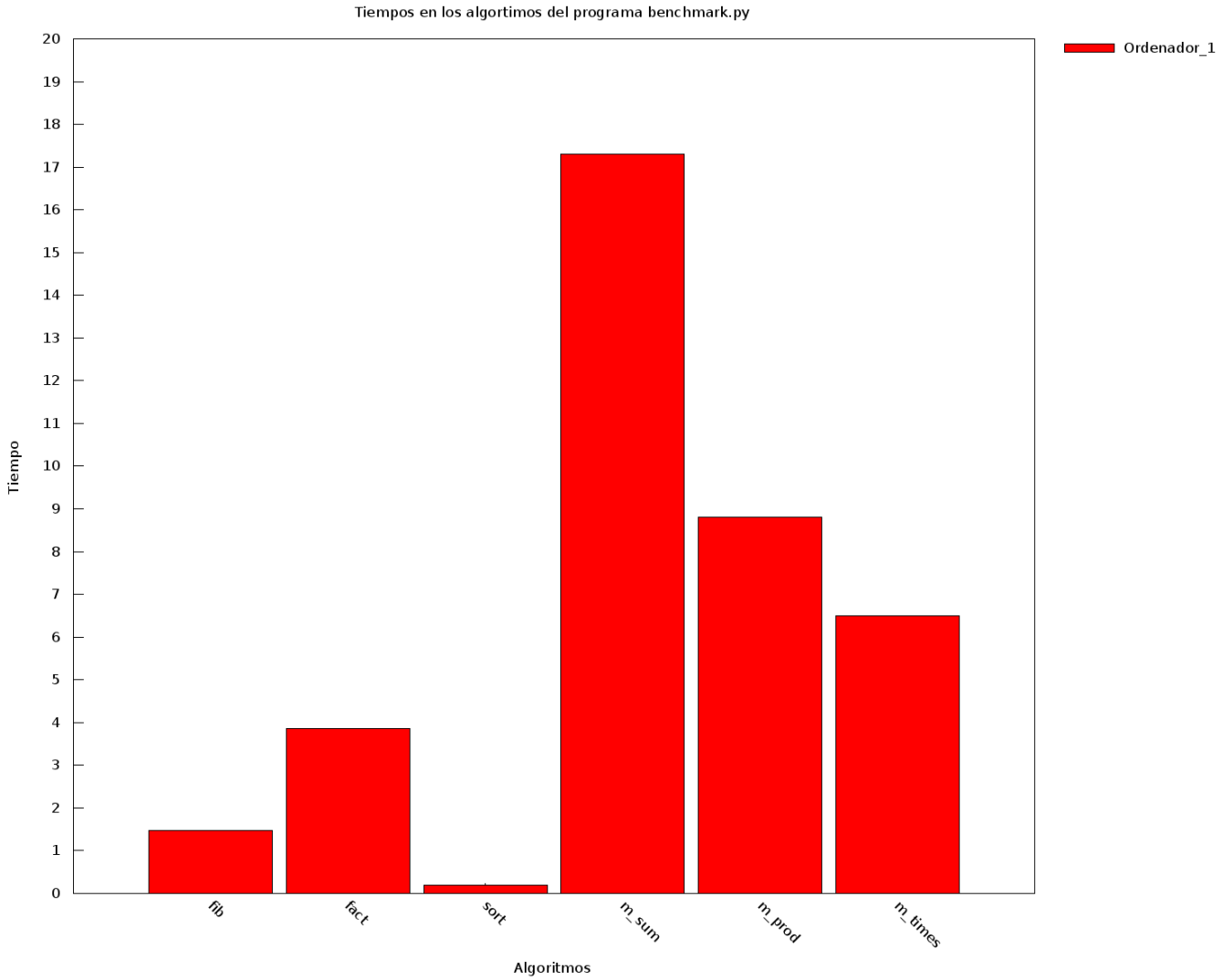
5. Análisis de los Resultados

Los resultados obtenidos son los siguientes:

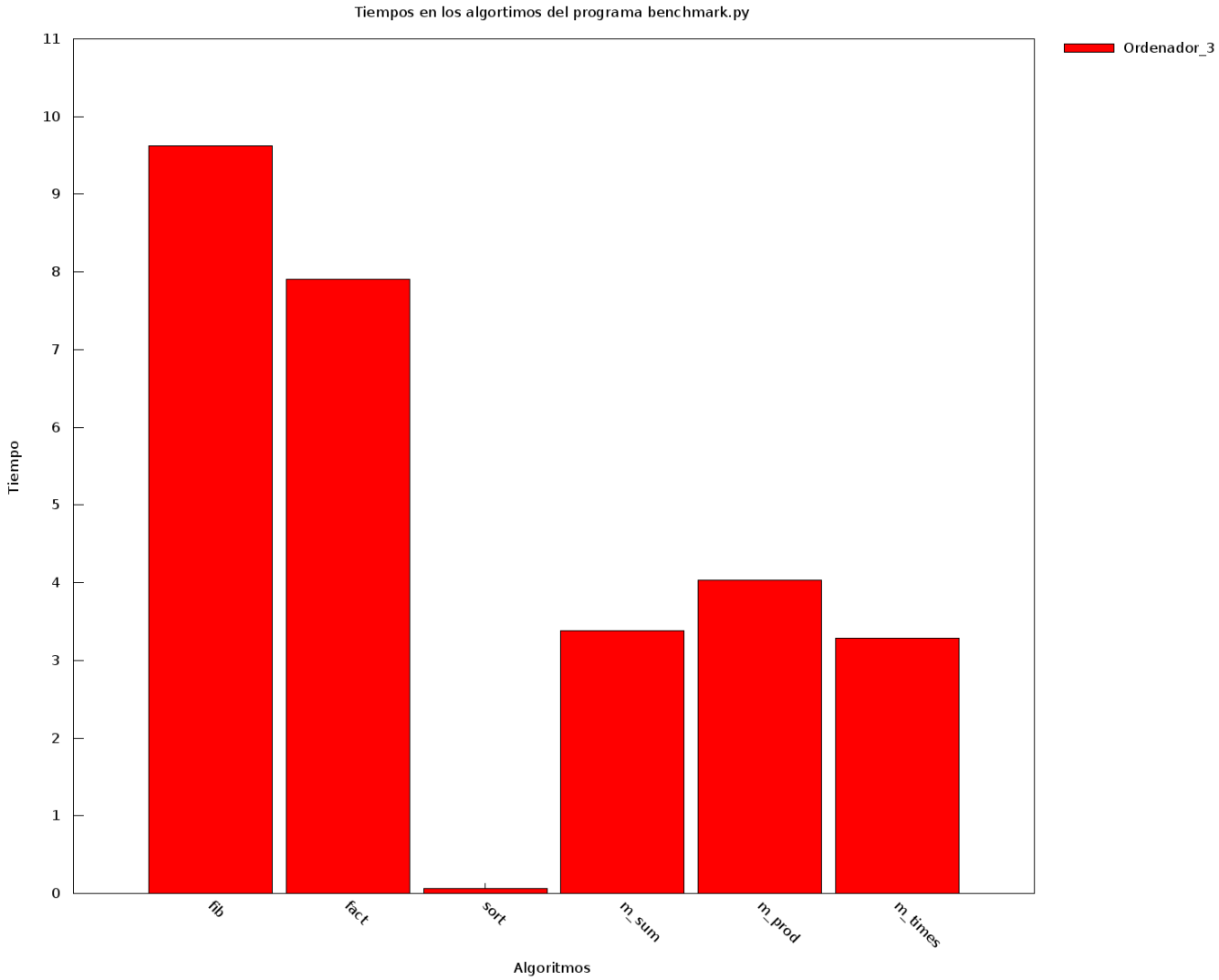
Máquina	Algoritmo	Medias (ms)
Ordenador 1	Fibonacci	1.47333
	factorial	3.86452
	sort	0.2
	merge	73.3
	suma	17.3
	producto	8.8
	potencia	6.5
Ordenador 2	Fibonacci	9.62627
	factorial	7.90744
	sort	0.06128
	merge	187.571
	suma	3.38264
	producto	4.03547
	potencia	3.28818
Ordenador 3	Fibonacci	69.47
	factorial	35.4645
	sort	0.2
	merge	657.2
	suma	16.4
	producto	19.1
	potencia	15.8

A continuación una serie de gráficas que describen el tiempo que ha tardado cada programa en ejecutarse en cada uno de los ordenadores.

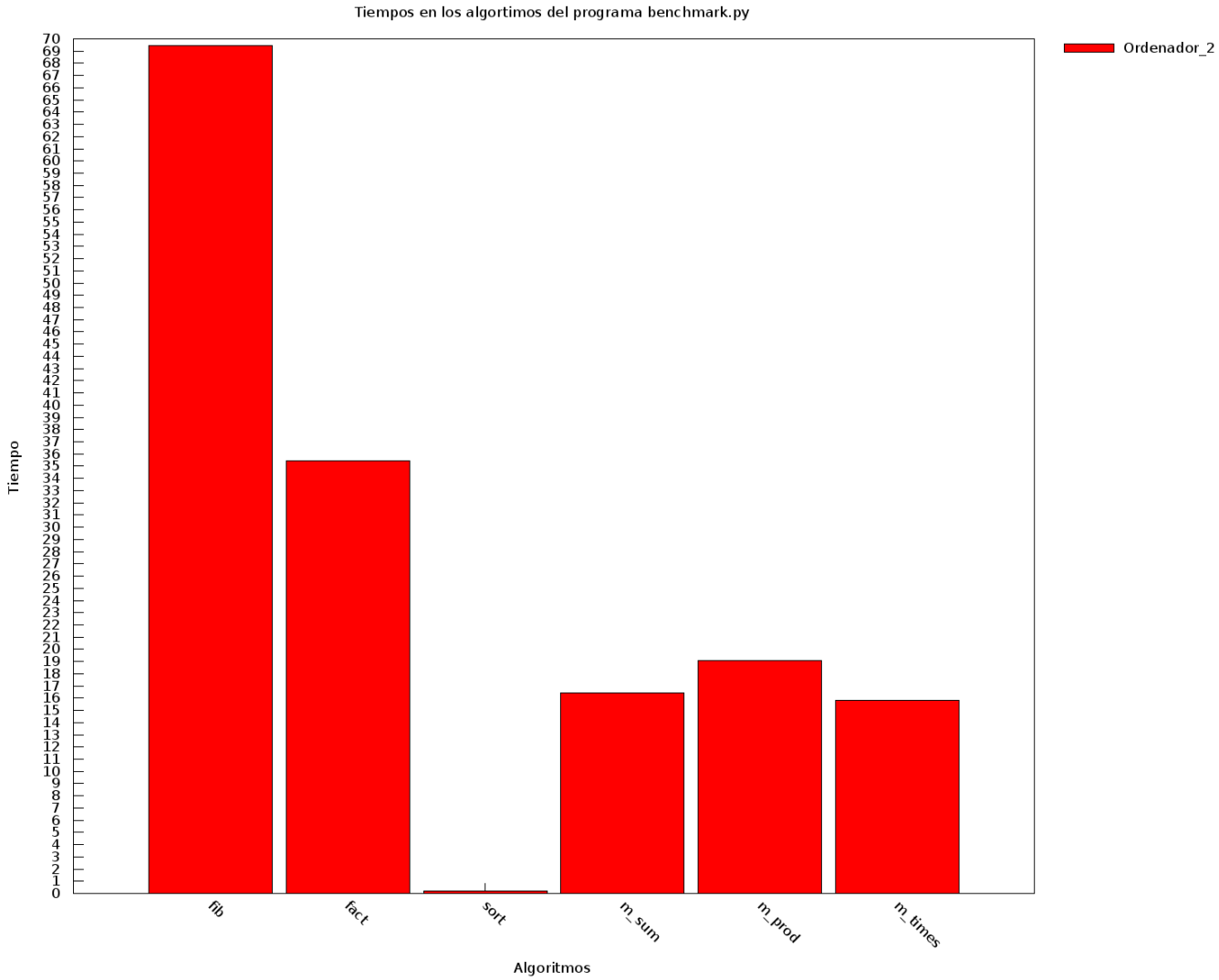
Ordenador 1


Tiempos en los algortimos del programa benchmark.py
Ordenador_1
Tiempo
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
fib
fact
sort
m_sum
m_prod
m_times
Algoritmos

Ordenador 2

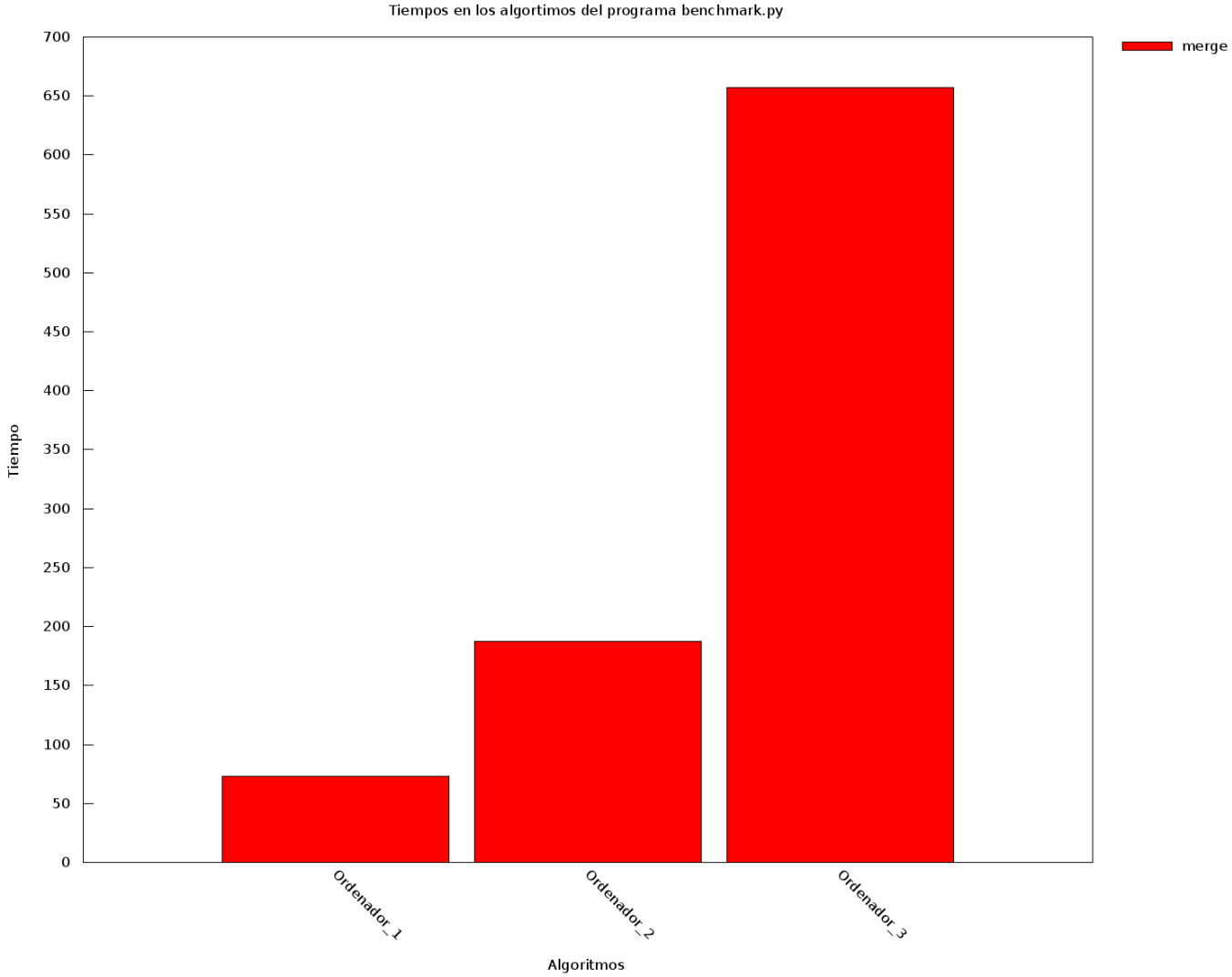

Tiempos en los algortimos del programa benchmark.py
Ordenador_3
Tiempo
0
1
2
3
4
5
6
7
8
9
10
11
fib
fact
sort
m_sum
m_prod
m_times
Algoritmos

Ordenador 3


Tiempos en los algortimos del programa benchmark.py
Ordenador_2
Tiempo
Algoritmos
fib
fact
sort
m_sum
m_prod
m_times

En el caso del algoritmo merge, los tiempos de ejecución eran tan elevados que menguaban la calidad visual de las otras gráficas. Por este motivo, hemos decidido analizar los resultado del merge por separado.

Merge



6. Conclusiones

Si observamos nuestra gráficas vemos claramente cómo sorprende ver el algoritmo *merge sort* como el que más tiempo ha tardado. Esto puede debserse a su complejidad algorítmica $O(n \log n)$, a la cantidad de números que le pasábamos por parámetro o al hecho de que sea recursivo. Recordemos que un algoritmo recursivo crea una instancia de su propio método cuando es llamado formando así un nuevo entorno de ejecución, esto hace que se cree otro proceso y sea menos eficiente que desde una perspectiva iterativa; puesto que el algoritmo hace uso de la recursividad múltiple, esto es, se llama a sí mismo más de una vez en cuerpo de ejecución, este coste aumenta, dando así los resultados tan elevados anteriormente observados.
En comparación con el otro método recursivo, *Fibonacci*, que implementa una recursividad simple vemos que la diferencia de tiempos es muy grande, lo cual nos dice que la manera de implementar un algoritmo influye en el tiempo, sin embargo, *Fibonacci* en las muestras 2 y 3 ha quedado en segundo puesto como algoritmo más tardío, por lo tanto la recursividad puede ser una característica importante a la hora de comparar métodos.
Otro dato interesante es lo poco que tarda la implementación de python del algoritmo *sort*, esto puede deberse a que este método ya implementado por el lenguaje es de por sí muy eficiente y está pensado y desarrollado de tal forma que su uso sea de un coste óptimo. De igual forma podemos observar cómo la implemenación de matrices de nuestro benchmark ha dado unos resultados bastante bajos en comparación con nuestra implementación del *merge sort*; en ese caso entra en juego la librería ***NumPy*** que es una librería dedicada al cáculo y suponemos que ya tiene implementados de maneras óptimas los cálculos correspondientes a las matrices.
Desde el punto de vista de hardware vemos una clara relación entre las arquitecturas de procesador y el tiempo que tardan los algoritmos. El ordenador 3, de 32 bits muestra tiempos muy elevados en comparación a las otras dos máquinas, que son de 64.
En definitiva vemos que en los tiempos influye la implementación del método, si está implementado de manera eficiente y si es recursivo, por otra, la arquitectura de la máquina también influye pero en términos mucho más globales

Referencias

[1] Diccionario de Cambridge http://dictionary.cambridge.org/es/diccionario/britanico/benchmark

[2] La memoria y el programa se pueden encontrar en https://github.com/pabloriutort/Benchmark

[3] ***NumPy*** *is the fundamental package for scientific computing with Python* http://www.numpy.org/