

ORGANIZACIÓN DE COMPUTADORES: LABORATORIO 1

NICOLÁS OLIVARES

Profesores:

Felipe Garay

Erika Rosas

Nicolás Hidalgo

TABLA DE CONTENIDOS

| | |
|--|-----------|
| ÍNDICE DE FIGURAS..... | v |
| ÍNDICE DE CUADROS | vi |
| CAPÍTULO 1. INTRODUCCIÓN..... | 7 |
| 1.1 MOTIVACIÓN | 7 |
| 1.2 OBJETIVOS | 7 |
| 1.2.1 Objetivo general | 7 |
| 1.2.2 Objetivos específicos | 7 |
| 1.3 PROBLEMA | 7 |
| 1.4 ORGANIZACIÓN DEL DOCUMENTO | 7 |
| 1.5 HERRAMIENTAS | 8 |
| CAPÍTULO 2. MARCO TEÓRICO | 9 |
| CAPÍTULO 3. DESARROLLO..... | 11 |
| 3.1 PRELIMINARES | 11 |
| 3.2 PROGRAMAS DE ORDENAMIENTO | 11 |
| 3.2.1 Bubblesort | 11 |
| 3.2.2 Quicksort | 12 |
| 3.3 BENCHMARK | 13 |
| 3.3.1 Requerimientos para la construcción del benchmark | 13 |
| 3.3.2 Análisis previo | 13 |
| 3.3.3 Abordando el problema | 13 |
| Cálculo de MIPS | 13 |
| CAPÍTULO 4. ANÁLISIS..... | 15 |
| 4.1 EXPERIMENTOS | 15 |
| 4.1.1 Experimento No 1 | 16 |
| Hipótesis | 16 |
| Resultados | 16 |

| | | |
|--------------------------------------|--|-----------|
| 4.1.2 | Experimento No 2 | 17 |
| | Hipótesis | 17 |
| | Resultados | 17 |
| 4.1.3 | Experimento No 3 | 17 |
| | Hipótesis | 17 |
| | Resultados | 17 |
| 4.2 | Análisis de resultados | 18 |
| 4.2.1 | Con respecto a... | 18 |
| | Nivel de optimización | 18 |
| | Tiempo real vs Tiempo estimado | 18 |
| | Experimentos | 18 |
| CAPÍTULO 5. CONCLUSIÓN..... | | 21 |
| 5.1 | RESPUESTA A LA INTERROGANTE | 21 |
| CAPÍTULO 6. BIBLIOGRAFÍA..... | | 23 |

ÍNDICE DE FIGURAS

| | |
|--|----|
| Figura 3-1: Proporción de instrucciones dinámicas según su tipo para Algoritmo 1 | 14 |
| Figura 3-2: Proporción de instrucciones dinámicas según su tipo para Algoritmo 2 | 14 |
| Figura 3-3: Porcentaje de instrucciones dinámicas afectado por las iteraciones en el Algoritmo 1 . . | 14 |
| Figura 3-4: Porcentaje de instrucciones dinámicas afectado por las recursiones en el Algoritmo 2 . | 14 |
| Figura 4-1: Datos de salida de Benchmark | 15 |
| Figura 4-2: Resultados Experimento 01 | 16 |
| Figura 4-3: Resultados Experimento 02 | 17 |
| Figura 4-4: Resultados Experimento 03 | 18 |

ÍNDICE DE CUADROS

CAPÍTULO 1. INTRODUCCIÓN

1.1 MOTIVACIÓN

En el contexto del curso de Organización de Computadores, se invita al ávido lector a desarrollar las habilidades críticas y analíticas en un tema rico en dominio técnico, que le permitirá adentrarse en el concepto y mundo del rendimiento de los procesadores y cómo los métodos de medición operan.

1.2 OBJETIVOS

1.2.1 Objetivo general

El presente informe tiene como objetivo principal el análisis de un benchmark basado en MIPS en base a la aplicación de tareas de ordenamiento.

1.2.2 Objetivos específicos

Dentro de los objetivos para este informe están:

1. Generar conjunto de programas, el cuál cuenta con dos programas de ordenamiento y uno de benchmark.
2. Fase de experimentación.
3. Análisis de los resultados.

1.3 PROBLEMA

¿De qué manera se puede medir el rendimiento de un procesador?

Este informe presenta el planteamiento y solución a esta pregunta.

1.4 ORGANIZACIÓN DEL DOCUMENTO

El lector puede acceder a este documento y guiarse a través de los capítulos siguientes:

1. Introducción: la sección actual da al lector la información necesaria de preámbulo al tema en cuestión, motivaciones y objetivos esperados.
2. Marco Teórico: en esta sección se hará un resumen explicativo de conceptos que el lector necesita entender en cierta medida para la mejor comprensión del fondo de este informe.

3. Desarrollo: en esta sección el lector podrá conocer en detalle el proceso de construcción de este laboratorio.
4. Análisis: el lector tendrá acceso a los razonamientos obtenidos a través de la realización de experimentos sobre los programas construidos.
5. Para finalizar existe una sección de conclusiones para el informe, acerca del trabajo realizado y de la comparación entre objetivos y logros.

1.5 HERRAMIENTAS

Para la realización de este laboratorio se utilizaron las siguiente herramientas:

- Linux Mint 18 “*Sarah*” [1]
- Docker version 1.11.2, build b9f10c9 [2]
- Sublime Text Build 3124 [3]

CAPÍTULO 2. MARCO TEÓRICO

El lector debe entender este documento contiene términos técnicos que se explican en este capítulo, para expresarlos a manera de terminar en concenso para su lectura. Sépase que la mención de instrucciones hace referencia a dos conceptos en este informe. Instrucciones estáticas corresponde a las instrucciones que un programador escribe en sus programas como líneas de código, llamense for, whiles u otros elementos que controlan el flujo del programa. Por otro lado las instrucciones dinámicas se establecen a nivel de máquina y procesador, como aquellas instrucciones que ejecuta el procesador y que dependen del conjunto de instrucciones en el cuál se trabaja.

CAPÍTULO 3. DESARROLLO

3.1 PRELIMINARES

Los programas se realizan en el lenguaje de programación C, con un paradigma de programación imperativo procedural.

Se utiliza docker engine para generar contenedores con la imagen y sistema de archivos proporcionado por el profesor Felipe Garay, de esta manera, dentro del container se tienen las herramientas de compilación necesarias como son mipsel-linux-gnu y sus diferentes opciones.

3.2 PROGRAMAS DE ORDENAMIENTO

Utilizando los conocimientos entregados en los cursos anteriores de la carrera sobre el manejo de algoritmos, se implementaron dos algoritmos de ordenamiento de números enteros, *ordenamiento de burbuja o bubblesort* y *ordenamiento rapido o quicksort*.

3.2.1 Bubblesort

El Algoritmo 1 corresponde al algoritmo de bubblesort, este algoritmo cuenta con una complejidad de instrucciones de orden $O(n^2)$. [4]

Data: Este algoritmo se encarga de ordenar un conjunto de números de menor a mayor.

Input: Conjunto A desordenado, n número de datos

Output: Conjunto A ordenado

bubblesort (A,n) **for** $i = 1$ **to** n **do**

```
    for  $j = n - i$  to  $0$  do
        if  $A[j] > A[j + 1]$  then
             $aux = A[j]$ 
             $A[j] = A[j + 1]$ 
             $A[j + 1] = aux$ 
        end
    end
end
```

Algorithm 1: Algoritmo bubblesort

3.2.2 Quicksort

El Algoritmo 2 corresponde a un algoritmo de caracter recursivo el cual presenta un orden de complejidad de instrucciones $O(n \log n)$ para el caso promedio y $O(n^2)$ para el peor caso.[5][6]

Data: Este algoritmo se encarga de ordenar un conjunto de números de menor a mayor.

Input: Conjunto A desordenado, inicio, fin

Output: Conjunto A ordenado

quicksort ($A, inicio, f$) $i = inicio$

$f = fin$

$pivote = arreglo[(i + f)/2]$

while $i \leq f$ **do**

while $A[i] < pivote$ **do**
 | $i = i + 1$

end

while $A[f] > pivote$ **do**
 | $f = f + 1$

end

if $i \leq f$ **then**

$aux = A[i]$
 $A[i] = A[f]$
 $A[f] = aux;$
 $i = i + 1$
 $f = f - 1$

end

end

if $inicio < f$ **then**

 | **quicksort** ($A, inicio, f$)

end

if $fin > i$ **then**

 | **quicksort** (A, i, fin)

end

Algorithm 2: Algoritmo quicksort

Luego de tener ambos algoritmos e implementarlos se pasa a la fase de construcción del benchmark.

3.3 BENCHMARK

3.3.1 Requerimientos para la construcción del benchmark

Con el uso de Docker Engine se utiliza un contenedor para cargar la imagen que el profesor Felipe Garay nos facilitó vía hub. Esta imagen corresponde a un sistema UFS o Unix File System, donde esta disponible una herramienta de emulación llamada qemu

3.3.2 Análisis previo

Para el cálculo de MIPS se da la siguiente fórmula en el enunciado del laboratorio

$$MIPS = \frac{NI}{T \times 10^6} \quad (3.1)$$

En la fórmula 3.1, NI representa el *número de instrucciones* de un programa, mientras que T representa el *Tiempo de ejecución* del programa.[7] Ahora bien, ¿Cómo obtener tanto NI y T para la fórmula dada? En base al código C de los programas de ordenamiento, se obtienen, mediante decompilación las instrucciones del conjunto MIPS que son ejecutadas a nivel de procesador. Se contabilizan de estas instrucciones, separando aquellas que pertenecen al tipo I, tipo R y tipo J,

3.3.3 Abordando el problema

Cálculo de MIPS

En base al análisis previo realizado, se describen a continuación una serie de hechos y supuestos para el cálculo de MIPS en el programa de Benchmark.

- El cálculo de MIPS y tiempos de ejecución se realizan sólo para las instrucciones de la subrutina de ordenamiento, utilizando la librería "sys/time.h".
- Considerando que se trabaja con el set de instrucciones de MIPS, esto implica que existen 3 tipos de instrucciones, I, R, J.

Luego de la decompilación del programa, se contabilizan las instrucciones según su tipo. A continuación se presentan las proporciones obtenidas de la contabilización de las instrucciones.

| Nivel de Optimización | Ordenamiento Iterativo | | |
|-----------------------|------------------------|--------|--------|
| | Tipo I | Tipo R | Tipo J |
| O0 | 46 | 26 | 2 |
| O1 | 14 | 9 | 2 |
| O2 | 13 | 11 | 0 |
| O3 | 13 | 11 | 0 |
| Os | 11 | 8 | 1 |

Figura 3-1: Proporción de instrucciones dinámicas según su tipo para Algoritmo 1

| Nivel de Optimización | Ordenamiento Recursivo | | |
|-----------------------|------------------------|--------|--------|
| | Tipo I | Tipo R | Tipo J |
| O0 | 72 | 39 | 4 |
| O1 | 39 | 40 | 7 |
| O2 | 40 | 40 | 4 |
| O3 | 294 | 297 | 29 |
| Os | 27 | 29 | 4 |

Figura 3-2: Proporción de instrucciones dinámicas según su tipo para Algoritmo 2

Para generar la Figura 3-1 y la Figura 3-2 se contabilizaron las instrucciones dinámicas de la subrutina de ordenamiento iterativa y recursiva. El objetivo de hacer esta identificación, es obtener las *proporciones* de las instrucciones con respecto al total de instrucciones del algoritmo.

| Nivel de Optimización | Iterativo | Total instrucciones | Porcentaje |
|-----------------------|-----------|---------------------|------------|
| O0 | 60 | 74 | 81,08% |
| O1 | 21 | 25 | 84,00% |
| O2 | 17 | 24 | 70,83% |
| O3 | 17 | 24 | 70,83% |
| Os | 15 | 20 | 75,00% |

Figura 3-3: Porcentaje de instrucciones dinámicas afectado por las iteraciones en el Algoritmo 1

| Nivel de Optimización | Recursivo | Total instrucciones | Porcentaje |
|-----------------------|-----------|---------------------|------------|
| O0 | 85 | 115 | 73,91% |
| O1 | 65 | 86 | 75,58% |
| O2 | 63 | 84 | 75,00% |
| O3 | 595 | 620 | 95,96% |
| Os | 34 | 60 | 56,66% |

Figura 3-4: Porcentaje de instrucciones dinámicas afectado por las recursiones en el Algoritmo 2

En las tablas de las figuras Figura 3-4 y Figura 3-3, se observa el resultado de la lectura e interpretación de los archivos decompilados con las instrucciones de assembly MIPS. Esta interpretación representa una aproximación a que fracción de las instrucciones dinámicas del programa se ven afectadas o se ejecutan mas de una vez debido a que las instrucciones estáticas así lo demandan debido a loops, whiles u otros métodos de control involucrados en la programación de los programas.[8].

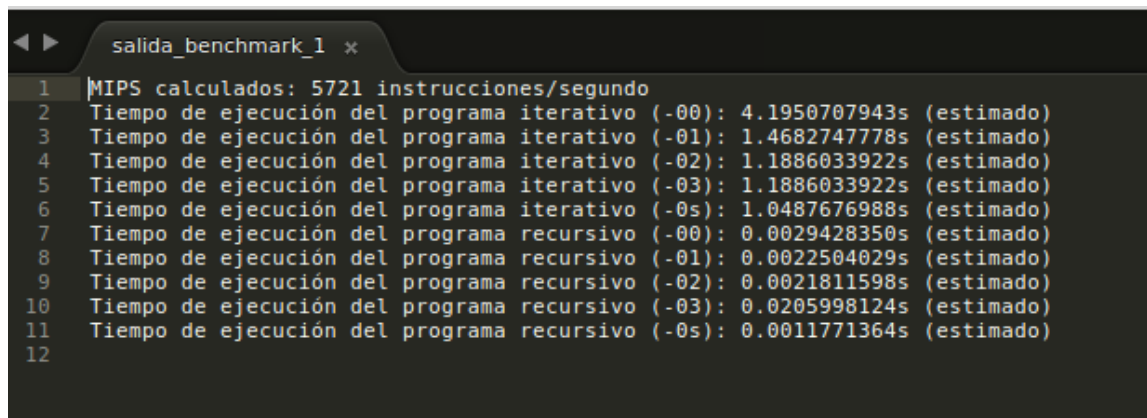
Dicho todo esto, se procede a construir el programa de benchmark para obtener los Millones de Instrucciones Por Segundo mediante la ecuación 3.1 y también los tiempos estimados de ejecución del programa, y específicamente, de la porción que realiza el ordenamiento.

CAPÍTULO 4. ANÁLISIS

Para este capítulo se declaran a continuación algunos hechos y supuestos que deben estar presentes durante el análisis.

- La máquina usada para las pruebas es un laptop personal, dentro de sus características están.
 - Procesador: Intel(R) Core(TM) i5-2450M CPU @2.50GHz de cuatro núcleos de proceso.
 - Memoria: 8GB RAM.
 - Sistema Operativo: Linux Mint "Sarah".
 - Dispositivos de entrada estandar para laptop.
- El cálculo aproximado de tiempos, representa una aproximación a las cantidades reales medibles de parámetros.
- Para la prueba de los experimentos y su análisis, se utiliza una entrada de 20000 datos.

La figura Figura 4-1 muestra los resultados obtenidos del benchmark ejecutado en mi computador.

A screenshot of a terminal window with a dark background. The title bar shows 'salida_benchmark_1 x'. The terminal output is as follows:

```
1 MIPS calculados: 5721 instrucciones/segundo
2 Tiempo de ejecución del programa iterativo (-00): 4.1950707943s (estimado)
3 Tiempo de ejecución del programa iterativo (-01): 1.4682747778s (estimado)
4 Tiempo de ejecución del programa iterativo (-02): 1.1886033922s (estimado)
5 Tiempo de ejecución del programa iterativo (-03): 1.1886033922s (estimado)
6 Tiempo de ejecución del programa iterativo (-0s): 1.0487676988s (estimado)
7 Tiempo de ejecución del programa recursivo (-00): 0.0029428350s (estimado)
8 Tiempo de ejecución del programa recursivo (-01): 0.0022504029s (estimado)
9 Tiempo de ejecución del programa recursivo (-02): 0.0021811598s (estimado)
10 Tiempo de ejecución del programa recursivo (-03): 0.0205998124s (estimado)
11 Tiempo de ejecución del programa recursivo (-0s): 0.0011771364s (estimado)
12
```

Figura 4-1: Datos de salida de Benchmark

4.1 EXPERIMENTOS

Se definen los tres experimentos a realizar:

- *EXP 01*: Medición de los tiempos reales de ejecución con `time.h` para bubblesort y quicksort, comparación con tiempo estimado de benchmark, con la máquina en el siguiente estado:
 - Prendida por más de 16 hrs consecutivas.

- Reproduciendo streaming de video y música.
- Conectada a corriente eléctrica.
- *EXP 02*: Medición de los tiempos reales de ejecución con `time.h` para bubblesort y quicksort, comparación con tiempo estimado de benchmark, con la máquina en el siguiente estado:
 - No más de 10 minutos desde su encendido, sin correr ningún programa o tarea extra además de los programas de inicio.
 - Conectada a corriente eléctrica.
- *EXP 03*: Medición de los tiempos reales de ejecución con `time.h` para bubblesort y quicksort, comparación con tiempo estimado de benchmark, con la máquina en el siguiente estado:
 - No más de 10 minutos desde su encendido, sin correr ningún programa o tarea extra además de los programas de inicio.
 - Solo con suministro de batería.

4.1.1 Experimento No 1

Hipótesis

Dada la acumulación de tareas en el procesador durante el tiempo en que la máquina permanece encendida, sumado al uso de buffers y procesamiento para streaming, el valor de los tiempos de ejecución deberían estar a un nivel más bien cercano al estimado, con una diferencia de a lo más 0.5 segundos

Resultados

| Nivel de Optimización | Tiempo de ejecución <time.h> | |
|-----------------------|---------------------------------|---------------------|
| | Bubbles ort [seg] | Quick sort [seg] |
| O0 | 3.125747 | 0.005645 |
| O1 | 1.445154 | 0.004145 |
| O2 | 1.428959 | 0.003857 |
| O3 | 1.432600 | 0.005200 |
| Os | 1.6602248 | 0.004032 |

Figura 4-2: Resultados Experimento 01

4.1.2 Experimento No 2

Hipótesis

Dado que al encender la máquina luego de un estado de apagado, las tareas ejecutadas en un inicio son las mínimas, el valor de los tiempos de ejecución deberá estar alejado del tiempo esperado en a lo más 1 segundo

Resultados

| Nivel de Optimización | Tiempo de ejecución <time.h> | |
|-----------------------|---------------------------------|--------------------|
| | Bubblesort [seg] | Quicksort [seg] |
| O0 | 2.888990 | 0.006790 |
| O1 | 1.347416 | 0.003896 |
| O2 | 1.344475 | 0.003573 |
| O3 | 1.551029 | 0.006888 |
| Os | 0.006790 | 0.005601 |

Figura 4-3: Resultados Experimento 02

4.1.3 Experimento No 3

Hipótesis

Dado que al encender la máquina luego de un estado de apagado, las tareas ejecutadas en un inicio son las mínimas, y considerando que la máquina con batería entra en modo de eficiencia, el valor de los tiempos de ejecución deberá estar alejado del tiempo esperado en a lo más 1.2 segundos

Resultados

| Nivel de Optimización | Tiempo de ejecución <time.h> | |
|-----------------------|---------------------------------|--------------------|
| | Bubblesort [seg] | Quicksort [seg] |
| O0 | 2.891646 | 0.006360 |
| O1 | 1.328143 | 0.005461 |
| O2 | 1.340710 | 0.004602 |
| O3 | 1.335966 | 0.004854 |
| Os | 1.534886 | 0.004342 |

Figura 4-4: Resultados Experimento 03

4.2 Análisis de resultados

4.2.1 Con respecto a...

Nivel de optimización

Según Figura 4-2, Figura 4-3, y Figura 4-4, se puede decir que el nivel O0 presenta un menor grado de optimización con respecto a los demás niveles para el algoritmo de bubblesort, ya que de acuerdo a la decompilación aplicada, este nivel presentaba cerca de 74 instrucciones dinámicas que son cerca de 45 instrucciones más que los demás niveles de bubblesort que rondaban entre las 20 a 25 instrucciones, véase Figura 3-1.

Para el caso del algoritmo recursivo de quicksort, aquella optimización que entrega un menor grado de optimización y por consiguiente un mayor número de instrucciones es el nivel O3; para este nivel en particular se generaron 620 instrucciones dinámicas en el decompilado, véase Figura 3-2. Todo esto se debe a que los algoritmos fueron compilados con 5 niveles de optimización distintos, según el compilador, esto generó más o menos instrucciones de MIPS según el nivel.

Tiempo real vs Tiempo estimado

Por parte del algoritmo bubblesort, este se aproxima cercanamente al resultado del benchmark para la máquina, quedando unos más o menos entre 0.5 a 1 segundo por debajo o arriba del tiempo estimado. Así como también el algoritmo de quicksort donde el fallo de la estimación es de unos más o menos 0.003 segundos en cada nivel.

Experimentos

Evaluando el desempeño de los niveles y algoritmos en relación a los resultados obtenidos nos lleva a las siguiente análisis:

1. En el caso del EXP01 existe evidencia para rechazar la hipótesis ya que el valor real obtenido difiere del estimado en 1.13 segundos en el caso del algoritmo de bubblesort. ¿A qué se debe esto?, por una parte es posible y probable que el cálculo del benchmark para los tiempos estimados dado los MIPS obtenidos, sea más abultado que la realidad, ya que el cálculo de los tiempos de ejecución aproximados en base a las proporciones, no están implementados correctamente. Pero se puede aceptar la hipótesis si se evalúa el desempeño con los otros niveles de optimización, los cuales difieren del estimado en 0.3 segundos aproximadamente, como lo refleja la Figura 4-2.
2. En base a lo explicado para el experimento 1 y con la evidencia de la Figura 4-3 se puede aceptar la hipótesis como verdadera, ya que los valores reales comparados con la estimación presentan una variación de menos de 1 segundo, descartando el nivel O0 dado bajo grado de optimización que posee. Se destaca además que los tiempos del experimento son más bajos que los tiempos obtenidos en el experimento 1, Figura 4-2, ya que el procesador tenía una menor sobrecarga de tareas durante el momento de ejecución del experimento 2.
3. Pese a lo planteado en la hipótesis, en donde se esperaba que dado la falta de corriente eléctrica directa de la red, se produjera un ascenso en el tiempo real debido a gran parte de las máquinas actuales, tienen frecuencias de reloj variable, y que en momentos de *sólo batería* presentan frecuencias más bajas por lo tanto una disminución en la capacidad de procesar más instrucciones por segundo. Los resultados en la Figura 4-4 indican un rechazo a esta hipótesis. Aunque esto puede deberse a que la distribución de Linux que se usó para la experimentación no maneja de manera eficiente el control de frecuencias del procesador.

CAPÍTULO 5. CONCLUSIÓN

Ya para finalizar, este informe de análisis sobre el uso de la estrategia de benchmarking MIPS o Millions Instructions Per Second. A continuación se compara el estado previo al desarrollo de este documento, con los objetivos y logros.

Se presentan las comparaciones con los objetivos específicos planteados.

1. Se cumplió el objetivo de generar el conjunto de programas de pedidos; dos programas de ordenamiento fueron creados de manera exitosa, cumpliendo su objetivo final. Por otro lado el programa de benchmark cumple su función de referencia, pero algunos aspectos dentro del programa no representan una correcta implementación.
2. Se realizó una fase de experimentación, abordando aspectos de cómo los tiempos responderían ante el estado del procesador dado la carga de trabajo que sostiene. Además se comparó con el benchmarking principal del documento, el uso de MIPS.
3. El análisis realizado fue en base a la experimentación con sus resultados comparativos, y la evaluación del desempeño de los algoritmos como también la eficacia y precisión del benchmark MIPS generado

5.1 RESPUESTA A LA INTERROGANTE

Se planteó en el inicio una pregunta, *¿De qué manera se puede medir el rendimiento de un procesador?*. En base al desarrollo de este documento de carácter analítico y descriptivo, se pudo responder esa duda. Con el uso del benchmarking MIPS se puede obtener una referencia de cuán eficiente es el rendimiento de un procesador. Ahora bien, este tipo de mediciones representa ventajas y desventajas. Las ventajas que se destacan está en la efectividad de la medición. Pero este sistema que actualmente está obsoleto, tiene más vívidas desventajas, el uso de este método de medición solo sirve para comparar máquinas que poseen el mismo tipo de ISA o conjunto de instrucciones, además las tareas que deben ser compiladas con el mismo procesador para todas las máquinas, ya que esta medición se hace en base a las instrucciones y sus proporciones comparativas al desempeño de las otras máquinas similares

CAPÍTULO 6. BIBLIOGRAFÍA

- [1] L. M. project, 2016. [Online]. Available: <https://www.linuxmint.com/>.
- [2] S. Hykes, 2013. [Online]. Available: <https://www.docker.com/products/docker-engine>.
- [3] J. Skinner, 2008. [Online]. Available: <https://www.sublimetext.com/>.
- [4] Wikipedia, *Ordenamiento de burbuja - wikipedia, la enciclopedia libre*, [Online; accessed 1-Septiembre-2016], 2016. [Online]. Available: https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja.
- [5] D. B. Thomas Cormen, *Analysis of quicksort*, [Online; accessed 1-Septiembre-2016]. [Online]. Available: <https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort>.
- [6] I. Zempoaltecatl, *Método de ordenamiento quicksort*, [Online; accessed 1-Septiembre-2016], 2014. [Online]. Available: https://www.youtube.com/watch?v=NsVX_aUzJug.
- [7] J. J. R. Ortiz, *Apunte curso 11-12 estructura de computadores universidad de complutense de madrid*. [Online]. Available: <http://www.fdi.ucm.es/profesor/jjruiz/web2/temas/EC4.pdf>.
- [8] U. of Pittsburgh, *Computer organization and assembly language*, [Online; accessed 29-Septiembre-2016]. [Online]. Available: <http://www.pitt.edu/~kmram/CoE0147/lectures/concepts.pdf>.
- [9] J. H. Gerry Kane, [Online; accessed 27-Septiembre-2016]. [Online]. Available: <https://www.d.umn.edu/~gshute/mips/single-cycle-summary.pdf>.
- [10] linux die, *System(3): Execute shell command - linux man page*. [Online]. Available: <https://linux.die.net/man/3/system>.
- [11] J. A. G. Pulido, *Repertorio de instrucciones mips*, [Online; accessed 27-Septiembre-2016], 2000. [Online]. Available: <https://www.d.umn.edu/~gshute/mips/single-cycle-summary.pdf>.