Informe Práctica de Laboratorio 2

José Ángel Masías
Edael Duque

Naguanagua, 24 de enero de 2026

República Bolivariana De Venezuela
Ministerio Del Poder Popular Para la Educación
Universidad De Carabobo
Facultad Experimental De Ciencias Y Tecnología
Departamento De Computación

Profesor:
José Canache

Estudiantes:
José Ángel Masías (C.I. 30.281.906)
Edael Duque (C.I. 30.800.814)

Naguanagua, 24 de enero de 2026

INFORME

1. ¿QUÉ DIFERENCIAS EXISTEN ENTRE REGISTROS TEMPORALES ($T0–$T9) Y REGISTROS GUARDADOS ($S0-$S7) Y CÓMO SE APLICÓ ESTA DISTINCIÓN EN LA PRÁCTICA?

Los registros $t0–$t9 son "caller-saved" (el que llama a la función no asume que mantendrán su valor), mientras que los $s0–$s7 son "callee-saved" (si una función los usa, debe salvar su valor original en la pila y restaurarlo antes de terminar). Su aplicación práctica reside en algoritmos de ordenamiento: se usan los $s para variables críticas (como el puntero del arreglo o el tamaño) y los $t para cálculos rápidos dentro de bucles (como índices i, j o valores temporales para el swap).

2. ¿QUÉ DIFERENCIAS EXISTEN ENTRE LOS REGISTROS $a0–$a3, $v0–$v1, $ra Y CÓMO SE APLICÓ ESTA DISTINCIÓN EN LA PRÁCTICA?

Los $a0-$a3 se pasan como parámetros de las funciones. Los $v0-$v1 devuelven los resultados de una función. El $ra almacena la dirección de retorno para volver al código principal después de un *jal*. Se utilizaron para mostrar en pantalla la dirección base de un arreglo, y para retornar al nivel anterior sin perder el camino.

3. ¿CÓMO AFECTA EL USO DE REGISTROS FRENTE A MEMORIA EN EL RENDIMIENTO DE LOS ALGORITMOS DE ORDENAMIENTO IMPLEMENTADOS?

El acceso a registros es un proceso casi instantáneo (1 ciclo de reloj), mientras que el acceso a memoria (*lw, sw*) es considerablemente más lento, en gran medida debido a la latencia del bus de datos y la espera de RAM. El ordenamiento Bubblesort será drásticamente más veloz que su contraparte Quicksort.

4. ¿QUÉ IMPACTO TIENE EL USO DE ESTRUCTURAS DE CONTROL (BUCLES ANIDADOS, SALTOS) EN LA EFICIENCIA DE LOS ALGORITMOS EN MIPS32?

El impacto de las estructuras de control en MIPS32 es fundamental debido a los riesgos de control que introducen en el pipeline segmentado. Cuando el procesador encuentra un salto condicional (branch), no puede determinar la siguiente instrucción hasta resolver la condición, lo que provoca burbujas en el pipeline y ciclos perdidos. Los bucles anidados multiplican este efecto, ya que cada iteración del bucle interno ejecuta saltos que, a su vez, se repiten por cada iteración del bucle externo. Esto genera un impacto exponencial en el rendimiento. Los bucles pequeños son especialmente problemáticos porque el costo del salto representa un porcentaje muy alto del tiempo total de ejecución. La eficiencia depende críticamente de la predictibilidad de los saltos. Los patrones regulares como contadores permiten alta precisión en la predicción, mientras que patrones

aleatorios o datos impredecibles causan fallos frecuentes. Las llamadas a funciones mediante *jal* y especialmente *jr* presentan desafíos adicionales, mitigados mediante "return address stacks" que predicen direcciones de retorno. La estructura del código con múltiples if-else anidados o switches extensos puede saturar la tabla de historia de saltos y aumentar la tasa de fallos.

5. ¿CUÁLES SON LAS DIFERENCIAS DE COMPLEJIDAD COMPUTACIONAL ENTRE EL ALGORITMO QUICKSORT Y EL ALGORITMO ALTERNATIVO? ¿QUÉ IMPLICACIONES TIENE ESTO PARA LA IMPLEMENTACIÓN EN UN ENTORNO MIPS32?

En términos de complejidad con respecto al tiempo, sorprendentemente Quicksort es superior a Bubblesort ya que, en caso promedio, el primero tiene una complejidad de orden $O(n \log n)$. Esta eficiencia es gracias al ordenamiento por el pivote; a pesar de que tiene operaciones más tardadas como *lw* y *sw*, la eficiencia clave está en su estrategia de ordenamiento. En contraparte, Bubblesort tiene una complejidad con respecto al tiempo en caso promedio de orden $O(n^2)$, lo que alarga el proceso debido a que el algoritmo tiene dos ciclos anidados. En términos de complejidad de memoria, Bubblesort es superior, ya que solo usa registros y el espacio en pila del vector, pero esto es algo que también tiene Quicksort, a diferencia de que este posee una pila recursiva: mientras más crece n, el espacio es proporcionalmente mayor en memoria.

6. ¿CUÁLES SON LAS FASES DEL CICLO DE EJECUCIÓN DE INSTRUCCIONES EN LA ARQUITECTURA MIPS32 (CAMINO DE DATOS)? ¿EN QUÉ CONSISTEN?

Las fases del ciclo de ejecución de instrucciones son las siguientes:

- **IF (Instruction Fetch):** Se busca la instrucción en la memoria, usando el PC.
- **ID (Instruction Decode):** Se decodifica la instrucción y se leen los registros.
- **EX (Execute):** La ALU realiza operaciones de tipo aritmético, cálculo direccional o de comparación.
- **MEM (Memory Access):** De ser necesario, lee o escribe en la memoria de datos.
- **WB (Write Back):** Escribe el resultado final de vuelta en el banco de registros.

7. ¿QUÉ TIPO DE INSTRUCCIONES SE USARON PREDOMINANTEMENTE EN LA PRÁCTICA (R, I, J) Y POR QUÉ?

- **Tipo R:** Se usaron bastante *add, slt*, entre otros, para sumar registros, calcular direcciones y verificar condiciones.
- **Tipo I:** Instrucciones como *lw* y *sw* para guardar o cargar elementos de los vectores, y *beq* o *bne* para hacer comparaciones.
- **Tipo J:** Se usó *j* o *jal* para saltos y llamadas, especialmente para la recursión.

8. ¿CÓMO SE VE AFECTADO EL RENDIMIENTO SI SE ABUSA DEL USO DE INSTRUCCIONES DE SALTO (J, BEQ, BNE) EN LUGAR DE USAR ESTRUCTURAS LINEALES?

El abuso de saltos (*j, beq*) rompe la ejecución secuencial. En arquitecturas segmentadas, esto causa burbujas en el pipeline, reduciendo el CPI (Ciclos por Instrucción) efectivo. El código lineal es siempre preferible para mantener el flujo de datos constante.

9. ¿QUÉ VENTAJAS OFRECE EL MODELO RISC DE MIPS EN LA IMPLEMENTACIÓN DE ALGORITMOS BÁSICOS COMO LOS DE ORDENAMIENTO?

El modelo RISC de MIPS usa instrucciones de tamaño fijo (32 bits) y un conjunto reducido de comandos simples. Esto facilita la optimización del compilador y permite que el hardware ejecute instrucciones muy rápido (un ciclo por fase), ideal para operaciones repetitivas como el movimiento de datos en ordenamientos.

10. ¿CÓMO SE USÓ EL MODO DE EJECUCIÓN PASO A PASO (STEP, STEP INTO) EN MARS PARA VERIFICAR LA CORRECTA EJECUCIÓN DEL ALGORITMO?

Step ejecuta la siguiente instrucción. Si es una función, la ejecuta completa de un solo golpe, para ver rápidamente si el resultado es correcto. Como dice un programador: "Si funciona, no lo toques"; gracias a esta opción se puede verificar de forma directa si la salida es la correcta. En caso de que no lo sea, se procede a usar *Step Into* para verificar línea a línea cuál es el fallo en la lógica. *Step Into* es vital para revisar a detalle cada línea del algoritmo y así analizar su desempeño, como una corrida en frío. Fue vital para depurar la recursión en Quicksort y verificar que el puntero de pila ($sp) se manejara correctamente.

11. ¿QUÉ HERRAMIENTA DE MARS FUE MÁS ÚTIL PARA OBSERVAR EL CONTENIDO DE LOS REGISTROS Y DETECTAR ERRORES LÓGICOS?

El "Registers Display" y el "Data Segment". Permiten ver en tiempo real cómo cambian los valores de los registros y cómo se reordenan físicamente los datos en la memoria RAM, facilitando la detección de errores de lógica en el intercambio de valores (swap). También el *Step Into* ayuda bastante a observar si la lógica del ejercicio es correcta.

12. ¿CÓMO PUEDE VISUALIZARSE EN MARS EL CAMINO DE DATOS PARA UNA INSTRUCCIÓN TIPO R? (POR EJEMPLO: ADD).

Primero debes ejecutar el código. Una vez estés en la pestaña de ejecución, accede en la barra superior al apartado "Tools". Se despliega un cuadro con una serie de herramientas; haz click en

"MIPS X-Ray". Se desplegará una pestaña encima de la pestaña de ejecución y ahí podrás ver el flujo de datos de las instrucciones de tipo R, manipulando los botones de navegación de corrida "Step Into" y "Last Step".

13. ¿CÓMO PUEDE VISUALIZARSE EN MARS EL CAMINO DE DATOS PARA UNA INSTRUCCIÓN TIPO I? (POR EJEMPLO: LW).

De igual forma que las instrucciones de tipo R, primero debes ejecutar el código. Una vez estés en la pestaña de ejecución, accede en la barra superior al apartado "Tools". Se despliega un cuadro con una serie de herramientas; haz click en "MIPS X-Ray". Se desplegará una pestaña encima de la pestaña de ejecución y ahí podrás ver el flujo de datos de las instrucciones de tipo I, manipulando los botones de navegación de corrida "Step Into" y "Last Step".

14. JUSTIFICAR LA ELECCIÓN DEL ALGORITMO ALTERNATIVO.

Elegimos el Bubblesort porque queríamos contrastar los dos tipos de ordenamientos y ponerlos a prueba, ya que estos son relativamente similares, pero uno tiene enfoque recursivo y otro iterativo. Esta elección busca darle continuidad al análisis de la práctica anterior, que consistía en la comparación iterativa y recursiva de la función Paridad en MIPS. Aquí tenemos una segunda oportunidad de comparativa para expandir nuestro conocimiento sobre qué es mejor para cada caso y escenario. Logramos obtener comparaciones destacables, como ver su eficiencia en tiempo y memoria, qué complejidad lógica guardan y si son razonables en un entorno de un grupo de programadores compartiendo sus aportes. Fue una elección del algoritmo alternativo acertada.

15. ANÁLISIS Y DISCUSIÓN DE LOS RESULTADOS.

Para finalizar podemos dar un pequeño análisis de lo visto, que funciona como forma de síntesis de los datos obtenidos de los dos ordenamientos. Concluimos lo siguiente:

- **Eficiencia:** El Quicksort es notablemente más eficiente. Cuando pasamos un vector de tamaño n, por ejemplo ocho, no se nota la diferencia en eficiencia, pero cuando elevamos el valor de n a 100, podemos ver una mejora con el Quicksort. En el caso de n mayor a mil, el Bubblesort empieza a "sufrir", tardando varios segundos; en contraparte, el Quicksort seguirá tardando unos milisegundos.
- **Memoria:** El ganador es Bubblesort, ya que no requiere de una pila y el uso de su memoria es constante y bajo.
- **Simplicidad:** Bubblesort vuelve a ganar en este apartado; es mucho más fácil de comprender y de usar, ya sea para usos laborales o educativos.
- **Estabilidad:** Quicksort también tiene problemas, ya que un tamaño de vector muy grande puede generar desbordamientos en la pila. A pesar de eso, debe ser un vector muy grande y se pueden generar contramedidas para evitar esa situación.

Parece ser que la eficiencia sí tiene sus costos en este caso (para el ámbito del uso como programador, para la memoria y estabilidad del algoritmo). Ambos algoritmos fueron diseñados para admitir vectores ordenados (de menor a mayor, de mayor a menor, o en posiciones aleatorias), admiten números grandes y números negativos, y admiten números repetidos de igual forma.