Algoritmos de Ordenamiento en Mips 32: BubbleSort y SelectionSort - Practica 2

Orlando Bonilla 30869545 - Enyerber Flores 27157754

July 14, 2025

Bubble Sort - Ordenamiento Burbuja

```
.data
vector:      .word 5, 3, 8, 1, 9   #Insertamos un vector cualquiera
tam:        .word 5                #Le asignamos un tamano
mensaje_ordenado: .asciiz "Vector ordenado: "
espacio:       .asciiz " "
nueva_linea:     .asciiz "\n"

.text
.globl main

main:
    # Ordenamos el vector
    jal bubble_sort

    # Imprimir vector ordenado
    la $a0, mensaje_ordenado
    li $v0, 4
    syscall

    jal mostrar_vector

    # Salir del programa
    li $v0, 10
    syscall

# Subrutina para imprimir el vector
mostrar_vector:
    la $t0, vector      # Direccion base del vector
    lw $t1, tam         # Tamano del vector
    li $t2, 0           # Contador i=0

ciclo_imprimir:
    beq $t2, $t1, f_imprimir  # Si i == tam, terminar

    # Calcular direccion del elemento i
    sll $t3, $t2, 2      # i*4 (tamano de palabra)
    add $t3, $t3, $t0    # Direccion de vector[i]

    # Imprimir elemento
    lw $a0, 0($t3)
    li $v0, 1
    syscall

    # Imprimir espacio
```

```
for1:
    bge $t2, $t1, f_sort  # Si i >= tam-1, terminar
    li $t3, 0             # j = 0 (contador ciclo2)
    sub $t4, $t1, $t2     # tam-1-i (limite para j)

for2:
    bge $t3, $t4, f_for2  # Si j >= tam-1-i, terminar ciclo2

    # Calcular direcciones de vector[j] y vector[j+1]
    sll $t5, $t3, 2       # j*4
    add $t5, $t5, $t0     # Direccion de vector[j]
    addi $t6, $t5, 4      # Direccion de vector[j+1]

    # Cargar valores
    lw $t7, 0($t5)        # vector[j]
    lw $t8, 0($t6)        # vector[j+1]

    # Comparar
    ble $t7, $t8, sin_intercambio  # Si vector[j] <= vector[j+1], no intercam

    # Intercambiar valores
    sw $t8, 0($t5)
    sw $t7, 0($t6)

sin_intercambio:
    addi $t3, $t3, 1      # j++
    j for2

f_for2:
    addi $t2, $t2, 1      # i++
    j for1

f_sort:
    jr $ra
```

Selection Sort - Ordenamiento por Selección

```
.data
vector:     .word   5, 1, 7, 4, 9    # Vector a ordenar
tam:        .word   5                # Tamano del vector
espacio:      .asciiz " "            # Espacio para separar numeros
n_linea:    .asciiz "\n"             # Salto de linea
```

```
.text
.globl main

main:
    # Ordenar el vector
    la      $a0, vector
    lw      $a1, tam
    jal     selection_sort

    # Imprimir vector ordenado
    la      $a0, vector
    lw      $a1, tam
    jal     mostrar_vector

    # Terminar programa
    li      $v0, 10
    syscall

# Subrutina selection_sort
# $a0: direccion del vector
# $a1: tamano del vector
selection_sort:
    addi    $sp, $sp, -16     # Guardar registros en la pila
    sw      $ra, 12($sp)
    sw      $s2, 8($sp)
    sw      $s1, 4($sp)
    sw      $s0, 0($sp)

    move    $s0, $a0          # $s0 = direccion del vector
    move    $s1, $a1          # $s1 = tamano del vector
    li      $t0, 0            # $t0 = i = 0

ciclo1:
    addi    $t1, $t0, 1       # $t1 = j = i + 1
    move    $t2, $t0          # $t2 = min = i

    # Calcular direccion de vector[i]
    sll     $t3, $t0, 2       # $t3 = i * 4
    add     $t3, $s0, $t3     # $t3 = direccion de vector[i]
    lw      $s2, 0($t3)       # $s2 = vector[i] (valor minimo actual)

ciclo2:
    bge     $t1, $s1, f_ciclo_2  # Si j >= tam, terminar ciclo2

    # Calcular direccion de vector[j]
```

```
    sll     $t4, $t1, 2         # $t4 = j * 4
    add     $t4, $s0, $t4       # $t4 = direccion de vector[j]
    lw      $t5, 0($t4)         # $t5 = vector[j]

    bge     $t5, $s2, omitir    # Si vector[j] >= minimo actual, saltar
    move    $t2, $t1            # min = j
    move    $s2, $t5            # minimo actual = vector[j]

omitir:
    addi    $t1, $t1, 1         # j++
    j       ciclo2

f_ciclo_2:
    beq     $t2, $t0, no_intercambio  # Si min == i, no intercambiar

    # Intercambiar vector[i] y vector[min]
    sll     $t3, $t0, 2         # $t3 = i * 4
    add     $t3, $s0, $t3       # $t3 = direccion de vector[i]
    lw      $t6, 0($t3)         # $t6 = vector[i]

    sll     $t4, $t2, 2         # $t4 = min * 4
    add     $t4, $s0, $t4       # $t4 = direccion de vector[min]
    lw      $t7, 0($t4)         # $t7 = vector[min]

    sw      $t7, 0($t3)         # vector[i] = $t7
    sw      $t6, 0($t4)         # vector[min] = $t6

no_intercambio:
    addi    $t0, $t0, 1         # i++
    addi    $t8, $s1, -1        # $t8 = tam - 1
    blt     $t0, $t8, ciclo1 # Si i < tam-1, continuar ciclo1

    lw      $s0, 0($sp)         # Restaurar registros de la pila
    lw      $s1, 4($sp)
    lw      $s2, 8($sp)
    lw      $ra, 12($sp)
    addi    $sp, $sp, 16
    jr      $ra

# Subrutina mostrar_vector
# $a0: direccion del vector
# $a1: tamano del vector
mostrar_vector:
    move    $t0, $a0            # $t0 = direccion del vector
    move    $t1, $a1            # $t1 = tamano del vector
    li      $t2, 0              # $t2 = indice (i)
```

```
ciclo_imprimir:
    bge     $t2, $t1, f_imprimir  # Si i >= tam, terminar

    # Calcular direccion de vector[i] y cargar valor
    sll     $t3, $t2, 2         # $t3 = i * 4
    add     $t3, $t0, $t3       # $t3 = direccion de vector[i]
    lw      $a0, 0($t3)         # $a0 = vector[i]

    # Imprimir numero
    li      $v0, 1
    syscall

    # Imprimir espacio (excepto despues del ultimo elemento)
    addi    $t4, $t2, 1         # i+1
    bge     $t4, $t1, prox  # Si es el ultimo elemento, saltar

    li      $v0, 4
    la      $a0, espacio
    syscall

prox:
    addi    $t2, $t2, 1         # i++
    j       ciclo_imprimir

f_imprimir:
    # Imprimir nueva linea al final
    li      $v0, 4
    la      $a0, n_linea
    syscall
    jr      $ra
```

Respuestas

Registros Temporales y Registros Guardados

Los Registros Temporales ($t0 - $t9) son aquellos que almacenan valores temporalmente durante la ejecución de una función, y su valor no se preserva entre llamadas, pueden verse como variables locales. En caso contrario, **los Registros Guardados ($s0 - $s7**) mantienen sus valores a través de llamadas a distintas funciones y, para no perderse, se guardan en la pila. En este caso, **Bubble Sort** utiliza principalmente registros temporales porque no llama a otras subrutinas durante su ordenamiento y no se preserva ningún valor, distinto al **Selection Sort**, que debe buscar cuál es el menor para luego ordenar,

preservando este número y también la dirección y tamaño del vector.

Registros de Argumentos, Retorno y de Dirección de Retorno

Los Registros de Argumentos ($a0-$a3) sirven para pasar parametros a subrutinas, **los Registros de Retorno ($v0-$v1)** son los que devuelven un valor desde dicha subrutina al ámbito principal y **los Registro de Dirección de Retorno ($ra)** almacenan el lugar desde donde fue llamada la subrutina para regresar ahí. En ambos algoritmos usamos $ra para volver en la función de ordenado y en la de imprimir, usamos $v0 como salida para mostrar mensajes y los elementos del vector ordenado, la diferencia está en que para Selection Sort usamos parámetros para la función pero en Bubble Sort no fueron usados.

¿Cómo afecta el uso de registros frente a memoria en el rendimiento de los algoritmos?

Mientras más repeticiones o búsquedas deba hacer un algoritmo (recorrido), tenderá a ser más ineficiente, en este caso, Bubble Sort accede muy poco a memoria, de hecho, solo lo hace para intercambios y su camino siempre es secuencial pero podía empeorar si hay muchos intercambios, mientras que Selection Sort usa pila y compara en la prueba del menor con todo el vector desordenado, varias veces, para poder ordenar, teniendo que acceder a memoria.

¿Qué impacto tiene el uso de estructuras de control (bucles anidados, saltos) en la eficiencia de estos algoritmos?

En Mips 32 la eficiencia depende más de reducir saltos y accesos a memoria que de la complejidad teórica, a partir de esto podemos deducir que aunque el Bubble Sort sufre por saltos frecuentes, aprovecha su rendimiento en uso de registros temporales y su localidad espacial, mientras que el Selection Sort tiene menos intercambios pero más accesos a memoria y saltos impredecibles, siguen siendo dos ciclos for pero tiene más que hacer que el primero porque este no va comparando valor por valor sino una posición por todo el resto del vector, haciendo un ciclo interno posiblemente más largo.

Diferencias de Complejidad Computacional entre estos dos algoritmos

- Ambos poseen la misma complejidad de orden $O(n^2)$, sin embargo, para Selection Sort esto también aplica para el peor caso, porque realiza comparaciones aunque el arreglo ya esté ordenado, en Bubble Sort eso no pasa, ya que si el arreglo está ordenado, se puede adaptar para que no haga recorridos innecesarios y pase a orden $O(n)$.
- El Bubble Sort podría tener muchos intercambios, lo que haría que tenga muchos accesos a memoria (sw/lw) en el peor caso, mientras que en Selection Sort siempre accederá a n-1 intercambios, porque coloca un pivote (número mínimo fijo) que se compara con el resto de elementos, menos ese mismo.

Fases del ciclo de ejecución de instrucciones en la arquitectura MIPS32

1. **IF - Búsqueda de Instrucciones:** Se carga la instrucción desde la memoria y el PC se actualiza a PC + 4 (cada instrucción ocupa 4 bytes).
2. **ID - Decodificación:** Se interpreta la instrucción (opcode) junto con su tipo (R,I,J), prepara los operandos y envía señales para las etapas siguientes.
3. **EX - Ejecución:** La ALU ejecuta las operaciones aritmetica/logicas, ya sea operando con los valores o para saltos, calculando la dirección de destino.
4. **MEM - Acceso a Memoria:** Lee o escribe en memoria, depende de si la instrucción es sw o lw. Para otras instrucciones, este paso no cambia nada.
5. **WB - Escritura:** Guarda resultados en el banco de registros, si es sw o una instrucción de salto, no es afectado en nada.

¿Qué tipo de instrucciones se usaron predominantemente en la práctica?

- **R:** Se usaron para incrementar indices (add), calcular direcciones del arreglo (sll) y, en el caso del Selection Sort, hacer comparaciones (slt/slti).
- **I:** Para cargar elementos del vector a reistros (lw) y para guardar esos valores al intercambiarlos (sw).
- **J:** Para finalizar bucles (bge), para evitar intercambios (ble), pero también para buscar el mínimo (blt/bgt).

¿Cómo se ve afectado el rendimiento si se abusa del uso de instrucciones de salto?

El abuso de saltos degrada el rendimiento de algoritmos en MIPS 32 debido a la presencia de stalls en el pipeline por riesgos de control o a pérdida de localidad en fetch de instrucciones, y no solo esto, también podría afectar la legibilidad de un algoritmo por código espagueti. En algoritmos de ordenamiento, priorizar estructuras lineales podría reducir su tiempo de ejecución hasta en un 30 por ciento.

¿Qué ventajas ofrece el modelo RISC de MIPS en la implementación de algoritmos básicos como los de ordenamiento?

Al ser algortimos que presentan varias repeticiones, las intrucciones RICS permiten que operaciones en MIPS se ejecuten en un solo ciclo de reloj, acelerando comparaciones e intercambios. También minimiza el acceso a memoria con solamente dos instrucciones (sw/lw) sin necesidad de acceder a ella. Todo esto hace que el pipeline sea más eficiente y varias instrucciones puedan ejecutarse a la vez en diferentes etapas.

Modo de ejecución paso a paso

En el copilador Mars, al ensamblar un conjunto de instrucciones, podemos ver el paso a paso del algoritmo al recorrer cada instrucción, y ver cómo se va afectando cada registro. Para estos ejercicios de ordenamiento, hemos creado una función que realiza la acción de ordenar y otra para imprimir los elementos ordenados. En el Bubble Sort va comparando cada elemento y lo intercambia si uno es menor que el otro, en Seleccion Sort hace algo similar, la diferencia es que planta como pivote al menor elemento del vector y al finalizar el recorrido, lo añade a la primera casilla para volverlo a hacer pero n-1 veces. una vez ambos están ordenados, se les hace un último recorrido a ambos para imprimirlos.

¿Qué herramienta de MARS fue más útil para observar el contenido de los registros y detectar errores lógicos?

Yo utilicé la tabla de registros ubicada a la derecha, le dí en ejecutar instrucciones pero le bajé velocidad para ir viendo poco a poco cómo iban cambian los datos.

¿Cómo puede visualizarse en MARS el camino de datos para una instrucción?

Ya sea instrucción tipo R, I o J, podemos visualizar el camino de datos usando una herramienta llamada MIPS X-Ray, una vez adentro, conectamos el código y copilamos, ahí podremos ver el camino que toman los datos en cada instrucción, con colores.

¿Por qué Selection Sort y no otro?

Quise comparar con un algoritmo casi similar al Bubble Sort porque tenía la duda: Si ambos son similares, ¿no debe haber ningún problema en usar alguno de los dos? Pero hallamos, en este camino, que aunque se parezcan y hagan lo mismo, incluso, aunque tengan la misma complejidad computacional, el rendimiento no es igual, ambos podrían presentar problemas distintos y no solucionan el problema de la misma forma, ¿Ambos son funcionales? Si, pero me quedaría 100 por ciento con el Bubble Sort por el rendimiento de recursos.

Análisis y Discusión de los Resultados

- **Complejidad Computacional:** Ambos poseen el mismo orden pero Bubble Sort puede ser adaptado a un solo ciclo con una bandera que verifique si el vector está ordenado, caso contrario, Selection Sort no porque siempre deberá comparar el menor elemento.
- **Registros:** Selection Sort obligatoriamente debe usar un registro guardado para preservar el valor a comparar para agregar a una casilla luego de todo el procedimiento mientras que Bubble Sort puede ir comparando e intercambiando de una vez, usando registros temporales.
- **Saltos:** Bubble Sort utiliza muchos saltos condicionales porque podría llegar a hacer muchos intercambios, mientras que en Selection Sort solo hace saltos en la busqueda del menor.
- **Depende del tamaño del arreglo:** Si hay arreglos grandes siempre será mejor un algoritmo que no acceda tanto a memoria para evitar cuellos de botellas, y también uno que ofrezca siempre el mismo rendimiento eficiente, este sería el perfil perfecto que podría llenar el Selection Sort; ahora, si el arreglo es pequeño y no hay límites de memoria, Bubble Sort es la mejor opción.