

Computer Architecture

B03901022 卓伯鴻

1. Bubble Sort

1.1. Implementation

```
void bubble_sort(T arr[], int len) {  
    int i, j;  
    for (i = 0; i < len - 1; i++)  
        for (j = 0; j < len - 1 - i; j++)  
            if (arr[j] > arr[j + 1])  
                swap(arr[j], arr[j + 1]);  
}
```

我採用是先寫出C++的程式，再逐行轉換成assembly code的方法，左圖為我使用的C++程式。

而在parsing的部分，我的使用者介面(console)是會先印出要求輸入需要排序的數字數量，再者，一次輸入一個數字，直至達到一開始輸入的數字數量為止。而在輸出結果的部分我也使用了一個迴圈來印出所有在記憶體中的數字序列。

1.2. Design

1. Parser

```
main:  
    #print msg1 on console  
    li $v0, 4  
    la $a0, msg1  
    syscall  
  
    #get the number and save in t0 (n = length(A) = t0)  
    li $v0, 5  
    syscall  
    move $t0, $v0  
  
    #print msg2 on console  
    li $v0, 4  
    la $a0, msg2  
    syscall  
  
    #enter the number: A[]  
    sll $t1, $t0, 2  
    li $t2, 0  
loop: #t2(count = 0)  
    #loop until input number is filled aka = $t0  
input:  
    beq $t2, $t0, endinput  
    li $v0, 5  
    syscall  
    #store input number into stack pointer  
    sub $sp, $sp, 4  
    sw $v0, 0($sp)  
    addi $t2, $t2, 1  
    j input  
endinput:  
    add $sp, $sp, 4  
    li $t3, 1
```

由左圖可見，我先印出了第一行msg1在console上，此時console上會印出“Enter the Length of Array:”的字樣。

再來，在操作者輸入一個數字之後，會將這個輸入吃入並存在\$t0的register中，為求不要被洗掉，我就將\$t0設為length(Array) = n這個值。在輸入一個值之後，會印出“Input the number of each element one by one:”的字樣，再來使用者就需要輸入n個數字，才會繼續進行到下一個bubble sort的部分。

2. bubble sort function

```
endinput:  
    add $sp, $sp, 4  
    li $t3, 1  
loop_i:  
    #repeat until nothing swapped  
    beq $t2, $t0, loop_i_end  
    li $t4, 0  
loop_j:  
    sub $t5, $t0, $t3  
    beq $t4, $t5, loop_j_end  
    #load word in a0 from A[i-1]  
    lw $a0, 0($t2)  
    #load word in a1 from A[i]  
    lw $a1, 4($t2)  
    #compare a0 > a1  
    slt $t6, $a0, $a1  
    #swap if true  
    beq $t6, 0, loop_j_continue  
    #swap  
    sw $a1, 0($t2)  
    sw $a0, 4($t2)  
    #jump to next j_loop  
    addi $t4, $t4, 1  
    j loop_j_continue  
loop_j_end:  
    #jump to next i_loop  
    addi $t3, $t3, 1  
    j loop_i
```

在這個function內，我只有兩個迴圈，第一個迴圈是for i in range(n)的迴圈，在更內一層的迴圈是for j in range(n - i)的迴圈，而在這個迴圈之下，又有一個if的判斷式，判斷如果A[i-1]中存的值比A[i]中存的值還要大的話，就會進行swap這個function。

```

i_loop_end:
    #print sorted array
    li $t2, 0 #count = 0
    #jump to end if i > j
    li $v0, #j4: to end if i > j
    la $a0, #msg3 A[i]
    syscall

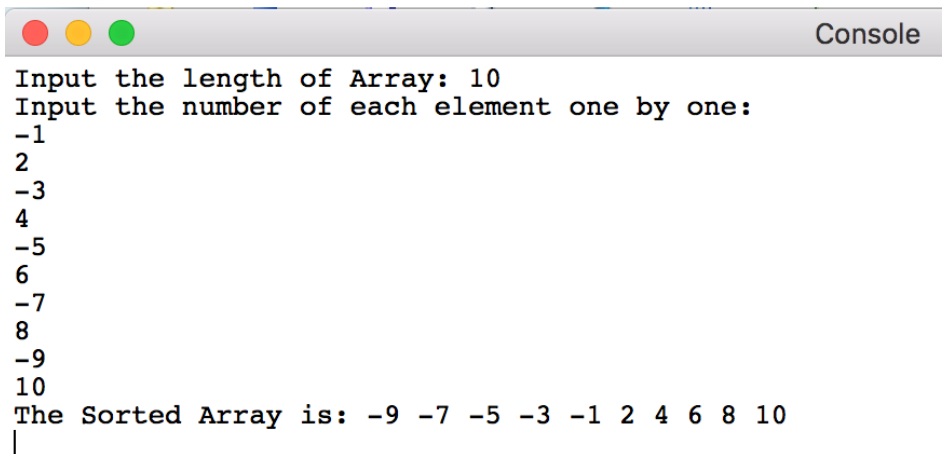
print:
    #tmp = A[i]
    beq $t2, #ac$0, A[i] print_end
    addi $s0, $t2, 1
    sll $s0, $s0, 2
    sub $s0, #t7$sp,[] $s0
    li $v0, #A[1] - tmp #print_int
    lw $a0, #A[0($s0)]
    syscall
    li $v0, 4 #print_string
    la $a0, space
    syscall #keep looping
    addi $t2, $t2, 1 #count++
    j print
    #jump to next if left >= j
    #false
print_end:
    li $v0, 4
    la $a0, nl
    syscall
    li $v0, 10 #syscall exit
    syscall #left = left
    #right = j

swap:
    sw $a0, 0($s3)
    sw $a1, 0($s2)
    j swapped

```

在這個部分，我把已經排列好的序列A依次印出，也只需要一個迴圈就足以全部印出，而在全部數字印出之後，會跳到結束function的system call，而此程式將終止執行。

1.3. Result



```

Input the length of Array: 10
Input the number of each element one by one:
-1
2
-3
4
-5
6
-7
8
-9
10
The Sorted Array is: -9 -7 -5 -3 -1 2 4 6 8 10
|

```

左圖是我執行完程式的結果

2.Quick Sort

2.1. Implementation

```

void quickSort(int arr[], int left, int right) {
    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];

    /* partition */
    while (i <= j) {
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    };

    /* recursion */
    if (left < j)
        quickSort(arr, left, j);
    if (i < right)
        quickSort(arr, i, right);
}

```

與上一個bubble sort所使用的方法一樣，我也是先做出C++版本的程式，再將它逐行翻譯成assembly code，左圖及為我使用的C++程式。

而為求方便，在parsing與print的部分，我使用的assembly code與我在bubble sort 所打的code是一模一樣的。(這之中也參雜了，避免這兩部分在quick sort中產生bug而與我其他的quick sort function bug混淆而增加debug的難度)

而quick sort 的function我是採用遞迴的方式將時間複雜度壓在 $O(n \cdot \log n)$ 。

2.2. Design

由於parsing & printing 部分的code 與 bubble sort 的部分大同小異，所以就不再貼上。直接進入quick sort function 的部分。

1.Start

```
start:
    li    $a0, 0           #left = 0
    sub   $a1, $t0, 1      #right = n - 1
    sub   $sp, $sp, 12     201 #move up 3 blocks
    sw    $ra, 8($sp)
    sw    $a0, 4($sp)
    sw    $a1, 0($sp)
    jal   sort
    lw    $ra, 8($sp)
    lw    $a0, 4($sp)
    lw    $a1, 0($sp)
    add   $sp, $sp, 12     2017-01-23 3:32:15
```

在這個部分，我把quick sort所需的陣列邊界條間先宣告好，也就是left = 0，right = n - 1這兩件事，並把他們的邊界條件及現在執行位置分別存在\$a0,\$a1,\$ra這三個地方，就直接進入遞迴式中。

2.sort預設

```
sort:
    move  $s0, $a0         #i = left
    move  $s1, $a1         #j = right
    add   $t4, $a0, $a1
    srl   $t4, $t4, 1
    add   $t4, $t4, 1
    srl   $t4, $t4, 2
    sub   $s4, $s6, $t4
    lw    $t5, 0($s4)
    lw    $s3, 0($s4)      #access s3 = pivot
    lw    $s3, 0($s4)
```

在這個部分，我把quick sort 所需的幾個變數宣告好分別是i = left，j = right，pivot我取當前Array的最中間，這三件事做完之後就可以進入判斷是否需要交換位置的迴圈中

3.while(i<j)迴圈

```
while:
    bgt    $s0, $s1, while_end

i_while:
    addi   $t4, $s0, 1
    sll    $t4, $t4, 2
    sub    $s4, $s6, $t4
    lw     $t5, 0($s4)
    bge    $t5, $s3, j_while
    addi   $s0, $s0, 1
    j      i_while

j_while:
    addi   $t4, $s1, 1
    sll    $t4, $t4, 2
    sub    $s4, $s6, $t4
    lw     $t5, 0($s4)
    #access A[j]
    #t5=A[j]
    ble    $t5, $s3, j_while_end
    sub    $s1, $s1, 1
    #j--
    j      j_while

j_while_end:
    bgt    $s0, $s1, if_false
    addi   $t4, $s0, 1
    sll    $t4, $t4, 2
    sub    $s4, $s6, $t4
    lw     $t5, 0($t4)
    #t5 = A[i]
    addi   $t6, $s1, 1
    sll    $t6, $t6, 2
    sub    $s4, $s6, $t6
    lw     $t7, 0($t6)
    #access A[i] at 0($t6)
    #t7 = A[j]
    sw     $t5, 0($t6)
    sw     $t7, 0($t4)
    addi   $s0, $s0, 1
    sub    $s1, $s1, 1
    #i++
    #j--
    if_false:
    j      while
```

在這個迴圈中，我會依序從最外邊的兩側i = left，j = right這兩個值開始向中央pivot夾，一旦左邊的A[i]值小於pivot右邊的A[j]值小於pivot就進行換位，直到其中一邊的值被換完為止。在演算法中，這個找尋交換對象的配對過程被叫做petition，而在這個過程結束之後就可以跳到遞迴式recursion的部分。

左圖中的i_while就是來找尋符合條件的A[i]的，而j_while就是來找尋符合條件的A[j]，而j_while_end中所寫的就是交換位置的部分，如果做完還有其他未找完的配對對象就會在進入一次迴圈。

4.recursion

```

while_end:
    bge    $a0, $s1, if_j_false
    sub    $sp, $sp, 12
    sw     $ra, 8($sp)
    sw     $a0, 4($sp)
    sw     $a1, 0($sp)
    move   $a0, $a0
    move   $a1, $s1
    jal    sort
    lw     $ra, 8($sp)
    lw     $a0, 4($sp)
    lw     $a1, 0($sp)
    add    $sp, $sp, 12

if_j_false:
    bge    $s0, $a1, if_i_false
    sub    $sp, $sp, 12
    sw     $ra, 8($sp)
    sw     $a0, 4($sp)
    sw     $a1, 0($sp)
    move   $a1, $a1
    move   $a0, $s0
    jal    sort
    lw     $ra, 8($sp)
    lw     $a0, 4($sp)
    lw     $a1, 0($sp)
    add    $sp, $sp, 12

if_i_false:
    jr     $ra

```

在recursion的部分，我會先檢查left是否小於j，如果小於j的話表示為二分至最細，可以透過binary的方法切至最小，反之，right大於i的部分亦然。若是兩者都切到最細，就代表這個recursion已經做到最底層，可以透過jr \$ra的方法，回到上層的迴圈，並繼續進行類似binary search的過程。

2.3. Result

```

Input the length of Array: 10
Input the number of each element one by one:
-1
2
-3
4
-5
6
-7
8
-9
10
The Sorted Array is: -9 -7 -5 -3 -1 2 4 6 8 10

```

上圖為我的程式執行的結果。

3. Comparing performance between quick sort from bubble sort

從演算法的層面來看，quick sort為較有時間效率的排序方法，也不需要再多花空間來儲存，排序結果，與merge sort 不同，bubble sort的優點則在於，程式簡單，方便建構，但是速度就較慢。

4. Problems Encountered

我在做bubble sort 的時候幾乎沒有遇到什麼太麻煩的bug，大部分的bug都出在quick sort 上面，其中我覺得最難以解決的bug是在遇到\$sp的位置，在進入遞迴時，不應該會有所更動，所以對於一些固定pointer的掌握仍不夠熟悉，而導致\$sp的位置被推移到，並間接影響進入迴圈之後，去Array中取出來的值是錯誤的。果然，在C++上遇到最麻煩的問題是pointer的處理，在assembly上仍然靈驗呢。