

# Computer Architecture HW2 Report

## 1. Bubble Sort:

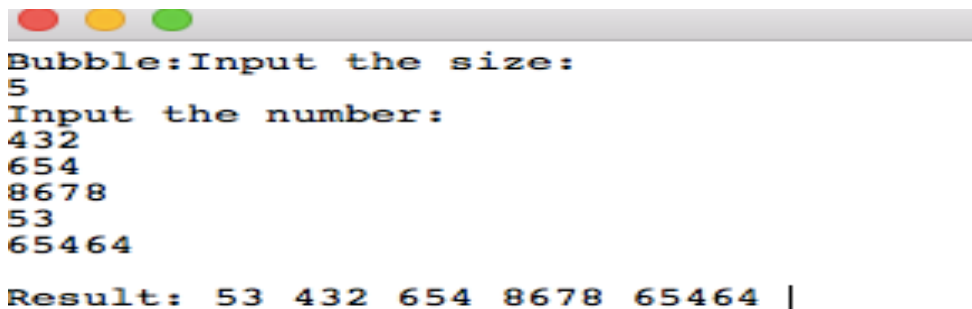
這次的作業第一題是 bubble sort，也是自己第一次寫 assembly，而 bubble sort 的演算法很簡單，就是在 input 的資料裡從第一筆開始，和第二、第三等等的資料做比較，一旦第一筆大於後面的資料就做 swap，完成第一輪之後，最大值會在最右邊了，接下來就是做和 array size 一樣多次後，完成 sorting。

課本上的 C++ code:

```
void sort (int v[], int n)
{
    int i, j;
    for (i = 0; i < n; i += 1) {
        for (j = i - 1; j >= 0 && v[j] > v[j + 1]; j -= 1) {
            swap(v, j);
        }
    }
}
```

(要跑  $n^2$  次)

值得注意的是，因為 bubble sort 的 space complexity 只有  $O(1)$ ，所以在一開始 input 放的地方移動值就好了，課本也有提供 assembly code，就是從 c++ 翻譯來的，在這裡學到了 stack 的復原、if 的寫法、迴圈的寫法，那接下來就是要考慮 input 和 output 的問題，這裡一開始自己的寫法是直接 allocate 一塊很大的記憶體給它，然後把記憶體一開始的 address 存起來，print 也直接從這個 address 開始 print，但是這樣如果要 sorting 的數不多，就很浪費記憶體空間，所以後來自己改成用可以自己輸入 array 的 size 並且把 size 的值存起來，這樣再去要求要 allocate 多少記憶體，從這個地方去看 spim 的時候也學會去觀察 data 部分存了什麼 (debug 有用！配合 single step 來看記憶體存值是不是在你希望的位置)



```
Bubble:Input the size:
5
Input the number:
432
654
8678
53
65464

Result: 53 432 654 8678 65464 |
```

## 2.Quick sort:

Quick Sort 是一個用 Divide and Conquer 來做的 sorting，就是選了 pivot 之後開始依照大小放到 small\_list 和 great\_list，利用遞迴來繼續分，終止條件是當 small\_list 或 great\_list 的大小為 0 或 1 停止。

因為之前修演算法有 Quick Sort 的 C++ code，swap 也在上一題寫好了，所以一開始打算是直接把 C++ 的 code 翻成 assembly，但是這裡自己犯了一個很大的錯誤，就是沒有考慮到遞迴的 address、size、value 都應該存起來、沒有考慮到它的 Space complexity，就直接進遞迴，所以 bug 很多，試著加上會存起來 code 也 de 不出 bug，下圖是當時第一版寫的：

```
.data
str1: .asciiz "Input the size: \n"
str2: .asciiz "Input the number: \n"
str3: .asciiz "\nResult: "
space: .asciiz " "

.text
main: li $v0, 4          #print string "Input the size \n"
      la $a0, str1
      syscall

      li $v0, 5
      syscall
      move $t5, $v0 # t5 is the size
      move $s3, $v0

      move $t6, $t5
      sll $t6, $t6, 2
      move $a0, $t6
      li $v0, 9
      syscall

      move $t8, $v0
      move $v1, $v0
      move $s2, $v0

      li $v0, 9
      syscall
      move $t9, $v0
      move $a3, $v0

      li $v0, 4          #print string "Input the number \n"
      la $a0, str2
      syscall

      move $t0, $zero
      move $t1, $zero

input: li $v0, 5
      syscall
      sll $t0, $t1, 2
      add $t8, $v1, $t0
      sw $v0, 0($t8)
      addi $t1, $t1, 1
      bne $t1, $t5, input

      move $t0, $zero
      move $t1, $zero

      li $v0, 4          #print string "\nResult: "
      la $a0, str3
      syscall

      move $s0, $zero
      move $s1, $zero
      move $s4, $zero
      move $s5, $zero

      jal quick_sort

print: li $v0, 1          #print integer
      lw $a0, 0($t9)
      syscall

      li $v0, 4;          # print string
      la $a0, space;      # space
      syscall;            # call

      sll $t0, $t1, 2
      add $t9, $a3, $t0
      addi $t1, $t1, 1

      bne $t1, $t5, print

finish: li $v0, 10
        syscall

exit4:
      jal swap_left_j
      addi $s3, $s1, -1
      jal quick_sort
      addi $s4, $s1, -1
      jal quick_sort
exit1:
      j finish

swap_left_j: sll $t1, $s4, 2
             add $t1, $s2, $t1

             sll $t2, $s1, 2
             add $t2, $s2, $t2

             lw $t3, 0($t1)
             lw $t4, 0($t2)

             sw $t4, 0($t1)
             sw $t3, 0($t2)

             jr $ra

swap_ij: sll $t1, $s0, 2
         add $t1, $s2, $t1

         sll $t2, $s1, 2
         add $t2, $s2, $t2

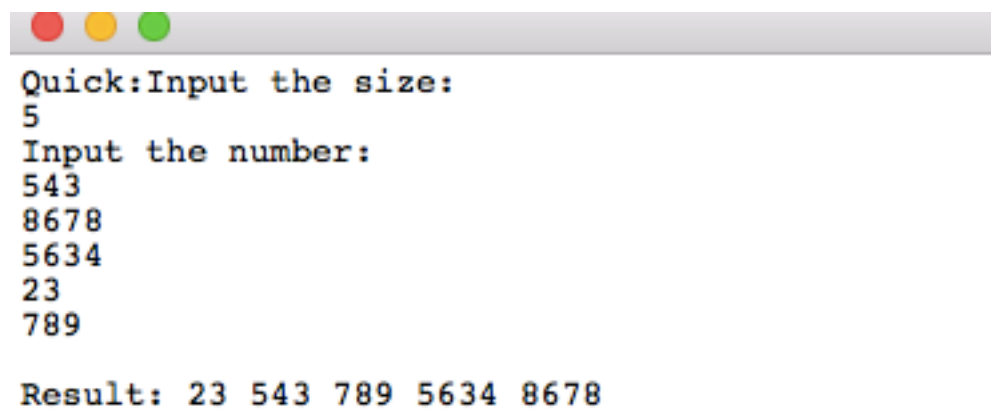
         lw $t3, 0($t1)
         lw $t4, 0($t2)

         sw $t4, 0($t1)
         sw $t3, 0($t2)

         jr $ra
```

後來這個方法行不通之後，就和同學討論要如何能夠保存這些每次分出來的 `small_list`、`great_list` 的 `address` 和 `size` 來讓遞迴可以呼叫呢？我們的答案是，在每次開始之前都 `allocate` 兩塊和 `size` 一樣大的記憶體，並且分別用 `$s6` 和 `$s7` 來存起始 `address`、`$s5` 和 `$s3` 則紀錄 `size`，從講解投影片中 Fibonacci Recurrence 看到可以 `sw` 在 `stack` 裡，這邊也遇到過分開來存會有問題，所以一併儲存。

接下來是做 `partition` 的動作，選定該 `list` 的第一個 `element` 為 `pivot`，比較小的分配到 `small_list`，比較大的分配到 `great_list`，一旦分完的話就跳到 `small_recur`，這個是在檢查是否已經到終止條件了，如果 `size` 大於 1，就把 `address` 和 `size` 存起來要再跳回 `recursive`；如果 `small_list` `size` 為 1，就表示到最小值了，可以放回去要印出來的地址那裡存起來了，再跳到 `great_recur`；如果 `size` 是 0，就直接跳到 `great_recur`（意義和上面差不多），`great_recur` 則會把 `pivot` 記得放回去，然後去檢查 `great_list` `size`，如果大於 1，就存起來繼續跳回 `recursive`；如果等於 1 就放回要印的地址；如果等於零，就準備跳出去，印出之前會先檢查一下 `size` 符不符合（+1 是考慮 `pivot`），最後再從預留的地址印出。



```
Quick:Input the size:
5
Input the number:
543
8678
5634
23
789

Result: 23 543 789 5634 8678
```

Quick sort 在 `time complexity` 比較少因為是用 `divide and conquer`，如果不是 `worst case` 的話，花的時間可以比較少。