I. Question

1.bubble_sort

(a)先看組語

```
1 .data
2 argument: .word 10
3 data: .word 5, 3, 6, 7, 31, 23, 43, 12, 45, 1
4 str1: .string "Array: "
5 str2: .string "Sorted: "
6 str3: .string " "
7 str4: .string "\n"

8 .text
9 main:
10    addi t1, zero, 5        #store array to x22
11    la      x22, data
12    lw      s0, argument #N = 10
13    la      a1, str1          #print Array
14    li      a0, 4
15    ecall
16    la      a1, str4
17    li      a0, 4
18    ecall
19    jal     ra, printArray
20    jal     ra, bubblesort
21    la      a1, str2      #print Sorted
22    li       a0, 4
23    ecall
24    la      a1, str4
25    li      a0, 4
26    ecall
27    jal     ra, printArray
28    li      a0, 10
29    ecall

30 bubblesort:
31    addi sp, sp, -48
32    sw      x23, 0(sp)
```

```
33  sw      x24, 8(sp)
34  sw      t1, 16(sp)
35  sw      t2, 24(sp)
36  sw      x20, 32(sp)
37  sw      ra, 40(sp)
38  li      x23, 0      #i = x23
39  for1loop:
40  bge     x23, s0, for1exit
41  addi x24, x23, -1       #j = x24
42  for2loop:
43  blt     x24, zero, for2exit
44  slli x10, x24, 2
45  add     x10, x10, x22
46  lw      t1, 0(x10)    #v[j]
47  lw      t2, 4(x10)        #v[j+1]
48  bge     t2, t1, for2exit
49  mv      x20, x24
50  jal     ra, swap
51  addi x24, x24, -1
52  j       for2loop
53  for2exit:
54  addi x23, x23, 1
55  j       for1loop
56  for1exit:
57  lw      x23, 0(sp)
58  lw      x24, 8(sp)
59  lw      t1, 16(sp)
60  lw      t2, 24(sp)
61  lw      x20, 32(sp)
62  lw      ra, 40(sp)
63  addi sp, sp, 48
64  ret

65 swap:
66  slli x10, x20, 2        #k = x20
67  add     x10, x10, x22
68  lw      t3, 0(x10)
69  lw      t4, 4(x10)
```

```
70  sw      t3, 4(x10)
71  sw      t4, 0(x10)
72  ret

73 printArray:
74  li      x19, 0          #i = x19
75  loop:
76  bge     x19, s0, exit
77  slli x10, x19, 2
78  add     x10, x10, x22
79  lw      t0, 0(x10) #print Array[i]
80  mv      a1, t0
81  li      a0, 1
82  ecall
83  la      a1, str3
84  li      a0, 4
85  ecall
86  addi x19, x19, 1
87  j       loop
88  exit:
89  la      a1, str4
90  li      a0, 4
91  ecall
92  ret
```

(1)執行行 10~行 19	19-10+1=10
(2)進入 printArray，執行行 74	1
(3)執行行 76~行 87，x19=0~9 共 10 次	(87-76+1)*10=120
(4)當 x19=10，執行行 76，接著進入 exit 執行行 89~行 92	92-89+1+1=5
(5)執行行 20	1
(6)進入 bubblesort，執行行 31~38	38-31+1=8
(7)接著執行雙重迴圈，為了方便計算迴圈，就不照著順序講了	
第一層為行 40~行 41，x23=0~9 共執行 10 次	(41-40+1)*10=20
行 40 在 x23=10 時會執行一次	1
for1exit 只會執行 1 次，行 57~行 64	64-57+1=8
for2exit 只會執行 10 次，行 54~行 55	(55-54+1)*10=20
在算第二層前 先看他是怎麼 sort	
5 3 6 7 31 23 43 12 45 1 swap 0 次，j=-1 for2exit	
3 5 6 7 31 23 43 12 45 1 swap 1 次，j=-1 for2exit	

3 5 6 7 31 23 43 12 45 1 swap 0 次，data[j] > data[j+1]for2exit
3 5 6 7 31 23 43 12 45 1 swap 0 次，data[j] > data[j+1]for2exit
3 5 6 7 31 23 43 12 45 1 swap 0 次，data[j] > data[j+1]for2exit
3 5 6 7 23 31 43 12 45 1 swap 1 次，data[j] > data[j+1]for2exit
3 5 6 7 23 31 43 12 45 1 swap 0 次，data[j] > data[j+1]for2exit
3 5 6 7 12 23 31 43 45 1 swap 3 次，data[j] > data[j+1]for2exit
3 5 6 7 12 23 31 43 45 1 swap 0 次，data[j] > data[j+1]for2exit
1 3 5 6 7 12 23 31 43 45 swap 9 次，j=-1 for2exit
統計下來
在第二層成功 swap 共 14 次
因為 j=-1 而 for2exit 共 3 次
因為 data[j] > data[j+1]而 for2exit 共 7 次

接著算第二層我們分 3 個 case:
Case1:在第二層執行 swap
執行行 43~行 52 和行 66~行 72，共 14 次　　(52-43+72-66+2)*14=238
Case2:在第二層因 j=-1 而 for2exit
執行行 43，共 3 次　　1*3=3
Case3:因為 data[j] > data[j+1]而 for2exit
執行行 43~行 48，共 7 次　　(48-43+1)*7=42
(8)執行行 21~行 27　　27-21+1=7
(9)執行 PrintArray，同(2)~(4)　　1+120+5=126
(10)執行行 28~行 29　　29-28+1=2

將(1)~(10)總和，總共有 612 個 instructions

(b)行 32~行 37 最多同時有 37-32+1=6 個 variable in the stack

2.gcd
(a)先看組語

```
1 .data
2 N1: .word 4
3 N2: .word 8
4 str1: .string "GCD value of "
5 str2: .string " and "
6 str3: .string " is "
```

```
7 .text
8 main:
9     lw          s0, N1
10    lw          s1, N2
11    jal         ra, gcd
12    jal         ra, printResult
13    li          a0, 10
14    ecall

15 gcd:
16    addi        sp, sp, -32
17    sw          ra, 24(sp)
18    sw          s2, 16(sp)
19    sw          s1, 8(sp)
20    sw          s0, 0(sp)
21    mv          t0, s1
22    blt         zero, t0, ngcd
23    mv          a0, s0
24    addi        sp, sp, 32
25    jalr        x0, x1, 0

26 ngcd:
27    rem             s2, s0, s1
28    mv          s0, s1
29    mv          s1, s2
30    jal         ra, gcd
31    lw          s0, 0(sp)
32    lw          s1, 8(sp)
33    lw          s2, 16(sp)
34    lw          ra, 24(sp)
35    addi        sp, sp, 32
36    ret

37 printResult:
38    mv          t0, a0
39    mv          t1, a1
40    la          a1, str1
41    li          a0, 4
```

```
42  ecall
43  mv          a1, s0
44  li          a0, 1
45  ecall
46  la          a1, str2
47  li          a0, 4
48  ecall
49  mv          a1, s1
50  li          a0, 1
51  ecall
52  la          a1, str3
53  li          a0, 4
54  ecall
55  mv          a1, t0
56  li          a0, 1
57  ecall
58  ret
```

---

(1)main 執行行 9~行 14 14-9+1=6

(2)main 中執行了一次 printResult，printResult 從行 38~行 58 58-38+1=21

(3)在計算 gcd 的 instruction 前，先看 gcd 演算法怎麼執行

首先先在 main 執行 gcd(4,8)，gcd(4,8) return gcd(8,4)，gcd(8,4) return gcd(4,0)，而 gcd(4,0) return 4

了解脈絡後就來計算 gcd

(4)gcd 首先先執行行 16~行 22 22-16+1=7

因為 gcd(4,8)，8!=0 所以跳到 ngcd 執行行 27~行 30 30-27+1=4

走到行 30 後又回去執行 gcd，因為 4!=0，執行行 16~行 22 22-16+1=7

因為 gcd(8,4)，4!=0 所以跳到 ngcd 執行行 27~行 30 30-27+1=4

走到行 30 後又回去執行 gcd，因為 0=0，所以執行行 16~行 25 25-16+1=10

走完行 25，跳回行 31，執行行 31~行 36，之後跳回行 25 36-31+1=6

走完行 25，跳回行 31，執行行 31~行 36，之後跳回行 12 36-31+1=6

將(1)~(4)總和，總共有 71 個 instructions

(b)gcd(4,8)和 gcd(8,4)和 gcd(4,0)都各執行行 17~行 20 一次

最多同時有(20-17+1)*3=12 個 variable in the stack

3.fibonacci

(a)先看組語

```
1 .data
2 N: .word 7
3 str1: .string "th number in the Fibonacci sequence is "
4 str2: .string "\n"

5 .text
6 main:
7     lw s0, N
8     mv s1, s0
9     addi s2, zero, 1
10    jal ra, fibonacci
11    mv a1, s0
12    li a0, 1
13    ecall
14    la a1, str1
15    li a0, 4
16    ecall
17    mv a1, a2
18    li a0, 1
19    ecall
20    la a1, str2
21    li a0, 4
22    ecall
23    li a0, 10
24    ecall

25 fibonacci:
26    addi sp, sp, -32
27    sw      ra, 24(sp)
28    sw      s2, 16(sp)
29    sw      s1, 8(sp)
30    sw      s0, 0(sp)
31    blt    s2, s1, nfibonacci
32    beq    s1, s2, ro
33    addi sp, sp, 32
34    jalr x0, x1, 0
35 ro:
```

```
36    addi a2, a2, 1
37    addi sp, sp, 32
38    ret
39 nfibonacci:
40    addi s1, s1, -1
41    jal    ra, fibonacci
42    lw      s0, 0(sp)
43    lw      s1, 8(sp)
44    lw      s2, 16(sp)
45    lw      ra, 24(sp)
46    addi sp, sp, 32
47    addi s1, s1, -2
48    ble    zero, s1, fibonacci
49    ret
```

---

(1)main 執行行 7~行 24　　24-7+1=18

(2)因為中間執行行 10，所以有一度跳去行 26，不過計算 fibonacci 前，我們先看他 c code 演算法怎麼執行的，顯然

Fibonacci(7)會 return Fibonacci(6)+ Fibonacci(5)

Fibonacci(6)會 return Fibonacci(5)+ Fibonacci(4)

Fibonacci(5)會 return Fibonacci(4)+ Fibonacci(3)

Fibonacci(4)會 return Fibonacci(3)+ Fibonacci(2)

Fibonacci(3)會 return Fibonacci(2)+ Fibonacci(1)

Fibonacci(2)會 return Fibonacci(1)+ Fibonacci(0)

Fibonacci(1)會 return 1

Fibonacci(0)會 return 0

所以可以總結:

每執行 1 次 Fibonacci(2)時，會附帶執行 Fibonacci(1)，Fibonacci(0)各 1 次

每執行 1 次 Fibonacci(3)時，會附帶執行 Fibonacci(2)，Fibonacci(1)各 1 次，因為執行 Fibonacci(2) 1 次，所以又附帶執行 Fibonacci(1)，Fibonacci(0)各 1 次，所以實際上附帶 Fibonacci(2)執行 1 次，Fibonacci(1)執行 2 次，Fibonacci(0)執行 1 次

每執行 1 次 Fibonacci(4)時，會附帶執行 Fibonacci(3)，Fibonacci(2)各 1 次，同上面的邏輯，計算後實際上附帶 Fibonacci(3)執行 1 次，Fibonacci(2)執行 2 次，Fibonacci(1)執行 3 次，Fibonacci(0)執行 2 次

每執行 1 次 Fibonacci(5)時，會附帶執行 Fibonacci(4)，Fibonacci(3)各 1 次，同上面的邏輯，計算後實際上附帶 Fibonacci(4)執行 1 次，Fibonacci(3)執行 2 次，

Fibonacci(2)執行 3 次，Fibonacci(1)執行 5 次，Fibonacci(0)執行 3 次

每執行 1 次 Fibonacci(6)時，會附帶執行 Fibonacci(5)，Fibonacci(4)各 1 次，同上面的邏輯，計算後實際上附帶 Fibonacci(5)執行 1 次，Fibonacci(4)執行 2 次，Fibonacci(3)執行 3 次，Fibonacci(2)執行 5 次，Fibonacci(1)執行 8 次，Fibonacci(0)執行 5 次

根據上面的推論，當執行唯一 1 次 Fibonacci(7)時，實際上會附帶 Fibonacci(6)執行 1 次，Fibonacci(5)執行 2 次，Fibonacci(4)執行 3 次，Fibonacci(3)執行 5 次，Fibonacci(2)執行 8 次，Fibonacci(1)執行 13 次，Fibonacci(0)執行 8 次

(3)實際計算 instruction，進入 fibonacci 後可以分 3 個 case
Case1:s1>1，也就是 Fibonacci(2~7)，總和共 20 次
執行行 26~行 31 和行 40~行 49　　(31-26+49-40+2)*20=320

Case2: s1=1，也就是 Fibonacci(1) ，總和共 13 次
執行行 26~行 32 和行 36~行 38　　(32-26+38-36+2)*13=130

Case3: s1=0，也就是 Fibonacci(0) ，總和共 8 次
執行行 26~行 34　　(34-26+1)*8=72

將(1)~(3)總和，總共有 540 個 instructions

(b)觀察 fibonacci 和 nfibonacci，當執行行 26~行 30 時，會在 stack 中放入 4 個 variable，如果沒跳去 nfibonacci。而在 nfibonacci 中，執行完行 40，也就是 Fibonacci(n-1)後，行 41~行 45 會收回 stack 中的 4 個 variable，接著才會執行行 47，也就是 Fibonacci(n-2)。
舉 Fibonacci(7)為例，走進 nfibonacci 中，如果執行完 Fibonacci(6)且執行 load，那變數就都全還完了，不過顯然 Fibonacci(7)的 stack 變數比 Fibonacci(7-2)的 stack 變數還多，所以我們不用管行 47，只要管行 40 的 recursive 就行。
也就是 Fibonacci(1~7)，每次都放 4 個 variable，所以同時最多有 7*4=28 個 variable 在 stack 中。

II. experience

這次組語還好可以用各種方式生出來，所以我就找了其他會組語的學長幫忙指導。一開始要讀懂組語還真不容易，網路上也找不到合適的 tutorial，不過熬了幾天後，就慢慢漸入佳境，最後還抓出了學長的一些 bug，真的收穫良多。雖然說學會讀懂組語，還有那套寫組語的邏輯是件好事。不過我在寫作業中感受

到高階語言的美好，至少哪裡寫錯還有人提醒，還好這次沒有真的要手刻一份出來，不然完全沒學過組語的我大概又要交屍體了。