Important Information

- Please use the reading time to review the questions and decide which questions you should approach first, *questions might not be in order of difficulty*
- If you aren't sure how to approach something, at least write down your current thoughts on it in case we can award marks for this
- write in **BLUE** or **BLACK** pen only, no pencil
  - if you make a mistake, put a line through the error and continue beneath or on another page

Write down any assumptions you make.

Question 1: [20 marks]

Consider the naive implementation of bubble sort given below

```
def swap(the_list,a,b):
        tmp = the_list[a]
        the_list[a] = the_list[b]
        the_list[b] = tmp

def bubbleSort(aList):
        for i in range(len(aList)):
                for j in range(len(aList)-1):
                        if aList[j] > aList[j+1]:
                                swap(aList,j,j+1)
        return aList
```

(a) (6 marks) We can modify this implementation to replace `range(len(aList)-1)` with `range(len(aList)-i-1)` in the inner (`j` dependent) loop. Will this have any impact on its:

(i) ***time complexity***?
(ii) ***actual runtime***?

Justify your answer.

There would be no impact on the time complexity as the inner loop remains n dependant, however the true run time would halve. This is as rather than running N-1 iterations for each iteration, it runs N-1 and then N-2 then N-3 etc. ($\sum_{i=1}^{N}(N-i) = \frac{N\times(N-1)}{2}$) which is still $O(N^2)$ as big O notation only cares about scaling, multiples and smallers terms are ignored..

- 2 marks for halves true run time
- 2 marks for no change in complexity
- 2 marks for valid explanation

(b) (6 marks) Explain in words how the implementation of bubble sort given can be updated to have a best case complexity of $O(N)$. Justify your answer.

Currently, this implementation has a best and worst case complexity of $O(N^2)$ as regardless on input it must consider **all** elements in the list. In order to optimise this we recognise that a sorted list *should* require no further sorting. In this case, we can allow bubble sort to track whether any swaps occur and in such a case terminate at the end of the inner loop. E.g. if the inner loop causes no swaps to occur, the list is sorted and hence no need to look through again (as nothing has changed anyway). In this way, the first run of bubbleSort given a sorted list terminates in $O(N)$ time.

- 4 marks for a clear explanation
  - 2 marks where vague
- 2 marks for clearly detailing why or in what situation the new implementation is $O(n)$

Note: any explanation resulting in $O(N)$ best case is valid (only best case complexity matters; effect on true run-time is inconsequential)

(c) (8 marks) Consider the sorting (non-)algorithm known as bogoSort. One implementation of this is defined as follows:

```
terribleApproachBogoSort...
... get the_list
... while not inOrder(the_list)...
... ... set pos1 to a random index of the_list
... ... set pos2 to a random index of the_list
... ... swap elements at pos1 and pos2 in the_list
```

(i) What is sorting stability?

(ii) Assuming bogoSort terminated in finite time, would it be stable?

- Justify your answer with an example.

bogoSort is ***not*** a stable approach to sorting. Stability (for sorting) is defined as leaving duplicate elements within a sorted list in the same relative order after sorting as they were before sorting. If we assume that bogoSort terminates we could easily find a situation where the stability is broken. Given it 'sorts' the list by randomly swapping elements within the list, there could be two duplicate items which are exchanged or moved past each other.

for instance, given the list `[1,6,2a,5,2b,5]` we could easily randomly swap `2a` with `5` resulting in `[1,6,5,5,2b,2a]`. This shows stability can be broken at any point by this algorithm.

- 2 marks for explaining stability
- 2 marks for not stable
- 2 marks for valid example
- 2 marks for explanation of example

0

Question 2: [20 marks]

This question is about MIPS programming. Translate the following MIPS code into Python. Note that in this piece of code all variables are global variables. Use the space to the right of the MIPS code for your answer.

```
.data
a:  .word 20
b:  .word 41
c:  .word 0

.text
        lw $t0, a
        lw $t1, b

        add $t2, $t1, $t0
        sw $t2, c
        addi $t3, $0, 2
        lw $t2, c
        div $t2, $t3
        mfhi $t4

        beq $t4, $0, ed

jUp:    lw $t6, c
        addi $t6, $t6, 1
        sw $t6, c

ed:     lw $t5, c
        addi $t3, $0, 2
        div $t5,$t3
        mflo $t5
        sw $t5, c
        addi $v0, $0, 1
        lw $a0, c
        syscall
```


20

equivalent python would be

```
a=20
b=41
c=0
c = (a+b)
if c%2==1:
    c+=1
c=c//2
print(c)
```

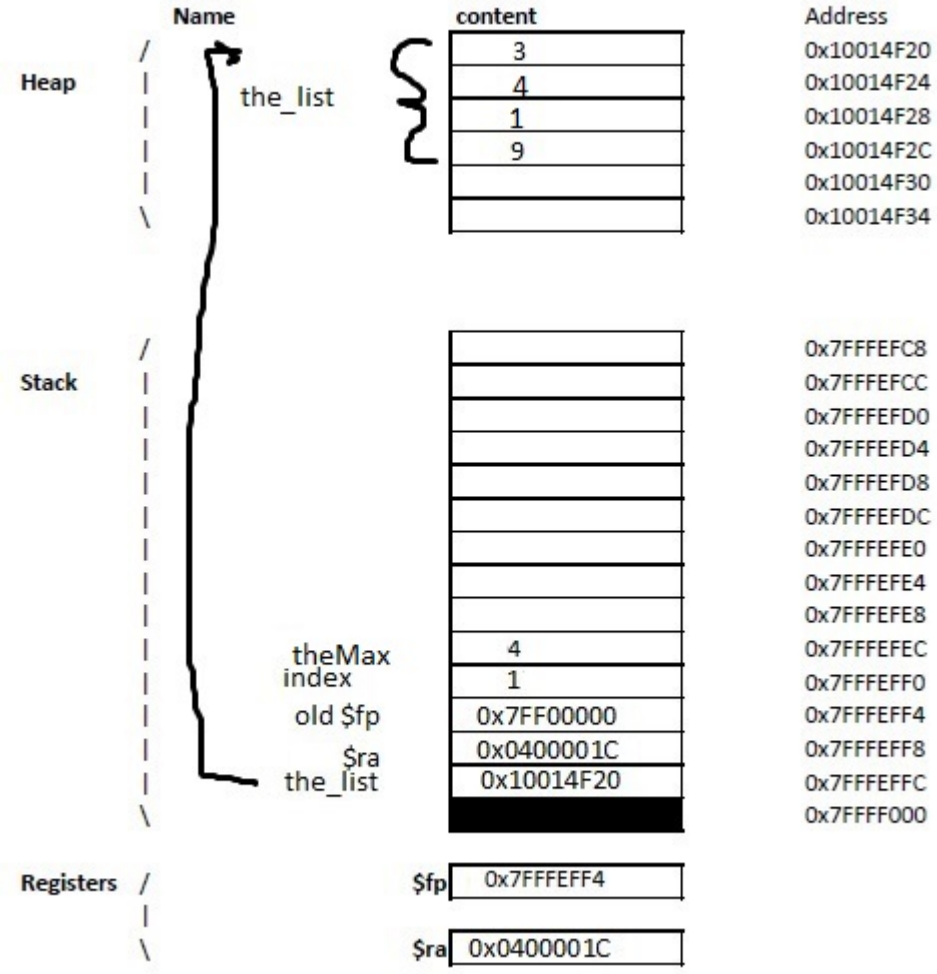
- 2 marks for correctly initialising variables at the start
- 2 marks for a+b
- 2 marks for c = a+b
- 2 marks for c%2
- 2 marks for integer dividing c by 2
- 4 marks for if statement
  - 2 marks only if they have an else as well
- 2 marks for correct if condition (could be c%2≠ 0)
- 2 marks for incrementing c by 1
- 2 marks for printing c

0

Question 3: [24 marks]

Consider the following python code, which you wish to translate into MIPS. Complete the memory diagram when execution reaches the line with the comment #HERE. Include names and contents. PC for the `jal getMax` instruction would be `0x04000018` and $fp was `0x7FF00000` before jumping. You will need to add to the stack and heap anything *getMax* will need access to.

```
def getMax(the_list):
        index = 0
        theMax = the_list[index]
        index+=1
        #HERE
        while index < len(the_list):
                if the_list[index]>theMax:
                        theMax = the_list[index]
                index+=1
        return theMax
getMax([4,1,9])
```



- 2 marks for putting the list in the Heap (rather than the stack)
- 2 marks for including the size with the list
- 1 mark for size above all items in the list
- 2 marks for list items given in correct order
- 1 mark for correct frame pointer register
- 1 mark for correct return address register
- 1 mark for correct old $ra in stack
- 1 mark for correct old $fp in stack
- for each of theMax, index and the_list respectively (9 marks total)...
  - 1 mark for clear name
  - 2 marks for correct contents
- 2 marks for stack contents in correct relative order
- 2 marks for beginning stack from the bottom

0

Question 4: [16 marks]

(a) (8 marks) Consider the 32bit instruction format for MIPS. List and justify one way that going to 64bit instructions can improve the performance or abilities of the system using those instructions.

You should use your knowledge of the instruction format to justify this.
Note: you do not need to know the precise number of bits used for each part of the instruction but you should have a general idea of what's involved.

the 32 bit instructions include some number of bits to identify which instruction it is as well as which registers and memory locations are used. Finally the maximum size of immediate values.

If we double the number of bits for the full instruction we can potentially double each of these. This could mean significantly more instructions available (allowing some previous groups of instructions to be done as a single instruction saving time)

This could also mean more registers (even a single additional bit for register number doubles the number available) which allows for more to be stored in registers simultaneously as needed.

This could also mean more memory locations meaning more RAM is available and more can be stored simultaneously persistently; this means at the very least more recursion depth is possible.
Finally there could be more bits for immediate values (up to the maximum possible to hold in a register); this last point may be difficult for students to appreciate as they won't know whether the register size will have changed or not.

- 4 marks for what it improves (e.g. more system memory)
  - 2 marks if vague
- 4 marks for why it is improved (e.g. longer memory addresses mean more memory can be addressed)
  - 2 marks if vague

(b) (8 marks) Consider the MIPS code shown below.

```
.data
        value:  .word 25
.text
1.      lw $t0, value
2.      addi $t1,$0,4

3.      div $t0, $t1
4.      mflo $t0
5.      sw $t0, value

6.      lw $t0, value
7.      div $t0, $t1
8.      mflo $t0
9.      mfhi $t1

10.     lw $a0, value
11.     addi $v0, $0, 1
12.     syscall
```

Using the table provided, show the effect of this on the ***HI, LO, $t0 and $t1*** registers and the contents of the label ***value*** in main memory.
Note: You may use '?' to represent an undefined value

1 mark per correct line (based on previous values) for each line from 2 to 9

Line number	**HI**	**LO**	**$t0**	**$t1**	**value:**	line run...
1	?	?	25	?	25	lw $t0, value
2	?	?	25	4	25	addi $t1, $0, 4
3	1	6	25	4	25	div $t0, $t1
4	1	6	6	4	25	mflo $t0
5	1	6	6	4	6	sw $t0, value
6	1	6	6	4	6	lw $t0, value
7	2	1	6	4	6	div $t0, $t1
8	2	1	1	4	6	mflo $t0
9	2	1	1	2	6	mfhi $t1
10	2	1	1	2	6	lw $a0, value
11	2	1	1	2	6	addi $v0, $0, 1
12	2	1	1	2	6	SYSCALL

End of Mid Semester Test