FIT1008-2085 S1 2019 exam solutions

Q1 - Python to MIPS translation

```
func:
  # save the $fp and $
  addi $sp, $sp -8   # r                k for the two registers
  sw $ra, 4($sp)     # s
  sw $fp, 0($sp)     # s

  addi $fp, $sp, 0  # cc

  addi $sp, $sp, -4 # m                r (result)

   # if n< = 0
  lw $t0, 8($fp)     # load argument n
  slt $t0, $0, $t0   # if 0 < n then $t0 = 1
  bne $t0, $0, else  # if $t0 = 1 (i.e., n > 0) go to else
  sw $0, -4($fp) # result = 0

  j endif           # jump over else branch

else:

  # compute n-1 and store it in $t0
  lw $t0, 8($fp)
  addi $t0, $t0, -1

  # save the argument (n-1) in the stack
  addi $sp, $sp, -4
  sw $t0, 0($sp)

  jal func # call func with n-1 as argument

  addi $sp, $sp, 4 # remove argument

  # result = 4*n + func(n-1)
  lw $t0, 8($fp)   # load n into $t0
  sll $t0, $t0, 2   # 4*n shifting by 2
  addi $t0, $t0, $v0, # 4*n + func(n-1)
  sw $t0, -4($fp)            # store it in result

endif:

  lw $v0, -4($fp) # put result in $v0

  addi $sp, $sp, 4  # remove local

  lw $fp, 0($sp) # restore $fp and $ra
  lw $ra 4($sp)
  addi $sp, $sp, 8

  jr $ra #go back to the callee
```

Part 1.f

The iterative version will require exactly the same number of bytes (N) as the recursive version, since the number of dynamic objects created during their executions (which are the only ones stored by [illegible]p) will not change.

For the MIPS code [illegible]nothing is created in the Heap.

Note that, in practic[illegible]te objects for integers and will indeed use the Heap.

Part 1.h

The Stack for the iterative version of **func(n)** will contain the argument **n** (4 bytes), the saved **$ra** and **$fp** (4+4 bytes), and the local variable **result** (4 bytes). This means a total of N = 16 bytes for the iterative version.

In the recursive version, the callee will call **func(n)** which will then call itself n} times. And each time it will take N (16 as we shown above) bytes. That means a total of (**n+1**)*N bytes.

Q2 Solutions:

1

2

2

3

6

No output, code produ

7

No output, code produces an error

None

0

Q3 - CS saves the world

Write the output [illegible] mystery for the input values:

1
1
2
3
1
4

What does the function mystery compute?

It computes the sum of the digits of x in base 2.

What is the time complexity of mystery, using the O() notation? Prove your answer.

O(log x).

Write the output of the function enigma for the input …

1
1
3
6
1
5

What does the function enigma compute?

It computes mystery(x) + mystery(mystery(x)) + …

What is the time complexity of enigma, using the O() …

The output of mystery(x) has size log(x).
Since enigma computes mystery(x) + mystery(mystery(x)) + ..., it requires T(x) = log(x) + log(log(x)) + ... operations.
Since log(x) <=x/2, we have \log\log(x) <= log (x/2)<= x/4$.
Hence T(x) <= x/2 + x/4 + ... = x.
Computing enigma(x) is thus in O(x).

What does enig ? Justify your answer.

Observe that 4095 = = 2^{11} + 2^{10} + ... + 2^0, hence mystery(4095) returns 12.

Therefore enigma(4 igma(12).

mystery(12) returns turns 1.

Hence enigma(4095

Q4 - Natural merging

Write a function [illegible]hich, given a list as input, returns the list of indices betwe[illegible]put list is already sorted.

```
def find_interva[illegible]
    separators = [illegible]
    for i in range(1, len(l)):
        if l[i-1] > l[i]:
            separators.append(i)
    separators.append(i+1)
    return separators
```


What is the worst-case time complexity of the …

It should be O(n), where n is the length of the list. (Direct analysis from code above.)

Write a function natural_merge which takes the list to …


```
def natural_merge(l):
    separators = findintervals(l)
    while len(separators) > 2:
        merge(l, separators[0], separators[1], separators[2])
        separators.pop(1)
```


In the function above it is important not to call find_intervals multiple times.

What is the worst-case time complexity of the merge function we have provided?

O(end-start), i.e. the sum of the length of the two sublists. This is visible in the range used in the two for loops.

What is the best-case time complexity of the algorithm natural_merge?

In the best case, the input list is already sorted, and there are only 2 items in the separators list, which means that the only work to do was to call find_intervals. Complexity O(n), where n is the length of the input list.

What is the worst-case time complexity of the algorithm ...?

The worst-case occu[illegible]ted in the opposite order. In this case, all sublists have size 1, and we [illegible]ything iteratively. There are O(n) merges to do, the first one with 2 elem[illegible]e with 3, then 4, 5, ..., n. Since the merge function has complexity O(e[illegible]time is 1+2+3+4+...+n = O(n^2).

How could a so[illegible]ith better time complexity be designed using the ideas presented in this question?

Even though we use merging operations, we may end up doing many inefficient "mergings" if the two lists being merged are not balanced in size. Instead of merging from left to right, we could merge smaller lists first.

Q5 - Resolving collisions

22, 23, 33, 2, 37, [illegible]ne, 21