Question 1 – Short questions [15 marks]

In this part you are required to answer the following short questions. Your answer should be concise. As a guideline, it should require no more space than the space that is provided.

(1) In MIPS, how many bits are required to store an array of 9 elements, where each element fits in one byte?

(2) In MIPS, how many bits long is each of the temporary registers?

(3) Is it possible to translate all recursive solutions into iterative ones? Explain your answer.

(4) Is Quick-sort stable? Explain your answer.

(5) What is the role of the **PC** register in MIPS.

7.5

This page intentionally left blank, use if needed but it will not be marked.

(6) Consider a sorted linked list of integers, to which you are trying to add element X. What values of X will yield the best-case time complexity? And the worst?

(7) List 3 different ways covered in the lectures to implement a Queue ADT

(8) Tail recursion is ...

(9) What does get printed after executing the following code in Python?

```
a = [1, 2, 3]; b = a; b.append(4); print(a)
```

(10) The instance variables of a Node supporting a Binary Search Tree are:

7.5

This page intentionally left blank, use if needed but it will not be marked.

Question 2 – Array Containers [10 marks = 3 + 2 + 5]

Consider the partial implementation of an array-based stack given below:

```
1  class Stack:
2
3      def __init__():
4          self.array = 100 * [None]
5          self.top = -1
6
7      def __len__(self):
8          return self.top + 1
9
10     def is_empty(self):
11         return self.top == -1
12
13     def is_full(self):
14         return self.top == len(self.array)-1
15
16     def push(self, item):
17         if self.is_full():
18             self.resize()
19         self.array[self.top+1] = item
20         self.top += 1
```

(a) Implement the method `resize(self)` within the Stack class, which is called from the `push` method given above (line 18). This method should double the size of the underlying array, each time such array runs out of capacity. **Important:** `self.array` should be used like an array (as it has been done in the above code), not like a Python list (that is, do not use any additional operations like iterators, slices, etc).

3

This page intentionally left blank, use if needed but it will not be marked.

(b) What are the best-case and worst-case time complexity of a complete `push` implementation? Explain your answer.

(c) The following implements a mystery container based on two stacks.

```
1  class MysteryContainer:
2
3      def __init__(self):
4          self.stack_a = Stack()
5          self.stack_b = Stack()
6
7      def __len__(self):
8          return len(self.stack_a) + len(self.stack_b)
9
10     def shift(self):
11         if self.stack_b.is_empty():
12             while not self.stack_a.is_empty():
13                 self.stack_b.push(self.stack_a.pop())
14
15     def mystery_one(self, item):
16         self.stack_a.push(item)
17
18     def mystery_two(self):
19         if len(self) == 0:
20             raise ValueError('Cant do this when I am empty')
21         self.shift()
22         return self.stack_b.pop()
```

What are the two mystery methods doing? What kind of container Data Type, of the ones we have seen in class, is our `MysteryContainer` class trying to emulate? Explain your answer by means of an example.

This page intentionally left blank, use if needed but it will not be marked.

Question 3 – Binary Trees [10 marks = 3 + 5 + 2]

Consider the partial implementation of a binary tree given below, which uses the tree node class:

```
1  class TreeNode:
2
3      def __init__(self, item=None, left=None, right=None):
4          self.item = item
5          self.left = left
6          self.right = right
7
8      def __str__(self):
9          return str(self.item)
10
11
12 class BinaryTree:
13
14     def __init__(self):
15         self.root = None
16
17     def is_empty(self):
18         return self.root is None
19
20     def height(self):
21         return self.height_aux(self.root)
22
23     def height_aux(self, current):
24         if current is None:
25             return 0
26         else:
27             max_val = max(self.height_aux(current.left),
28                           self.height_aux(current.right))
29             return max_val + 1
```

(a) Implement the method `__len__(self)` within the `BinaryTree` class, which determines the number of nodes in the binary tree.

3

This page intentionally left blank, use if needed but it will not be marked.

(b) Implement the method `is_balanced(self)` within the `BinaryTree` class, which returns `False` if the binary tree is not balanced, and `True` otherwise. A binary tree is balanced if the difference between the height of the left subtree and the right subtree is not larger than 1. An empty tree is assumed to be balanced.

(c) What is the worst-case time complexity of printing a binary tree in pre-order? Explain your answer.

7

This page intentionally left blank, use if needed but it will not be marked.

Question 4 – MIPS [7 marks = 4 + 3]

Consider the the following MIPS program:

```
1  .data
2  a: .word 1071
3  c: .word 462
4  temp: .word 0
5
6  .text
7  loop:
8    lw $t0, c
9    beq $t0, $0, endloop
10   sw $t0, temp
11   lw $t1, a
12   lw $t2, c
13   div $t1, $t2
14   mfhi $t3
15   sw $t3, c
16   sw $t0, a
17   j loop
18
19 endloop:
20   lw $a0, a
21   addi $v0, $0, 1
22   syscall
```

(a) Translate the above MIPS code into python.

(b) Explain the purpose of the **fp** and **sp** registers.

7

This page intentionally left blank, use if needed but it will not be marked.

Question 5 – Iterators [9 marks = 6 + 3]

Consider an *iterable* `List` abstract data type, where the list iterator has the usual `next()` method to return the current item in the list (if any) and move the iterator to the next, and also method `has_next`, which returns `True` if the iterator has not yet reached the end of the list, and `False` otherwise.

Consider the following method:

```
1  def mystery(iterable_list):
2      it1 = iterable_list.iter()
3      it2 = iterable_list.iter()
4      while it2.has_next():
5          item1 = it1.next()
6          item2 = it2.next()
7          if it2.has_next():
8              it2.next()
9              /* HERE */
10     return item1 + item2
```

(a) Given an iterable list `my_list` with elements 9, 6, 7, 5. 3, 4, 2, 1, and 8, where 9 is the item at the head of the list, show the value of `item1` and `item2` every time the execution reaches the program point marked by `/** HERE **/` during the call to `mystery(my_list)`.

6

This page intentionally left blank, use if needed but it will not be marked.

(b) What are the best-case and worst-case time complexity for the method? Explain.

3

This page intentionally left blank, use if needed but it will not be marked.

Question 6 – Namespaces [9 marks = 2 + 2 + 2 + 3]

Consider the following class, which is defined in file (also called module) `point.py`:

```
1  class Point:
2      def __init__(self,x, y):
3          self.xCoord = x
4          self.yCoord = y
5
6      def shift(self, x, y):
7          self.xCoord += x
8          self.yCoord += y
```

(a) Do identifiers `__init__` and `shift` belong to the same namespace? Explain.

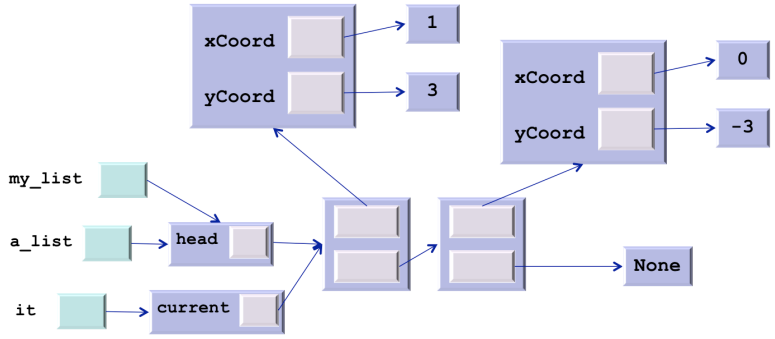
(b) Do the `self` identifiers appearing in lines 2 and 6 belong to the same namespace? Explain.

(c) Where is the `x` identifier appearing in line 7 bound? Explain.

6

This page intentionally left blank, use if needed but it will not be marked.

(d) Consider the following diagram:



which corresponds to the memory once the execution of call to `diagram(my_list)` with an iterable, linked list `my_list` that contains two point elements with coordinates (1,3) and (0,-3), has reached point `/*HERE*/` in the following code.

```
1  def diagram(a_list):
2      it = a_list.iter()
3      /*HERE*/
4      ...
```

Circle the objects corresponding to a list, to a point and to an iterator. Explain your choice.

3

This page intentionally left blank, use if needed but it will not be marked.

Question 7 – Linked Lists [10 marks = 7 + 3]

Consider the following linked `List` class, which you are in the process of defining:

```
1  class Node:
2
3      def __init__(self, item=None, link=None):
4          self.item = item
5          self.link = link
6
7  class List:
8
9      def __init__(self):
10         self.head = None
11
12     def is_empty(self):
13         return self.head is None
```

(a) Define a method `periodic_delete(self,n)` **within the** `List` **class** that deletes the first n nodes of the list, leaves the next n, erases the next n, and so on. For example, given `a_list` with elements 1,2,3,4,5,6,7,8,and 9 where 1 is the item at the head node, the call to `a_list.periodic_delete(2)` will leave `a_list` as 3,4,7,8.

7

This page intentionally left blank, use if needed but it will not be marked.

(b) How would the linked implementation compare in terms of efficiency and in terms of Big O complexity, to an implementation for an array-based lists? Explain.

3

This page intentionally left blank, use if needed but it will not be marked.

Question 8 – Recursive sorts [8 marks = 4 + 4]

This question is about recursive sorting algorithms. One day, while you are demonstrating your sorting algorithm on an array of integers, your friend decides to trick you and changes exactly one of the numbers to a value higher than any other value in the array. Use an array of size 6 to explain how would this affect the result of your sorting (that is, the order of the resulting array) if the change happened:

(a) To some element during the partition phase in mergesort. For example, to the first element in the list after the first call to partition has finished.

(b) To the pivot element, right after the pivot was chosen and you were about to perform a partition in quicksort. For example, the pivot was chosen as the last element, and that was the element changed before calling partition at some point during quicksort.

8

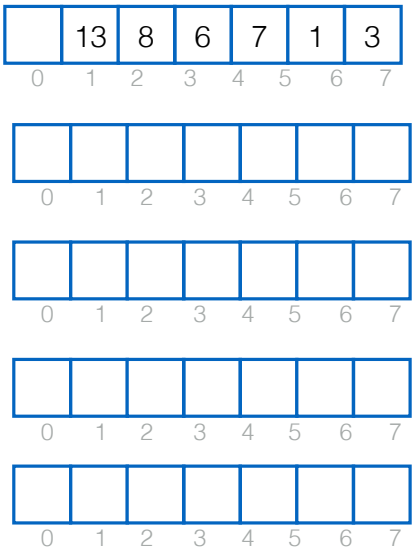
This page intentionally left blank, use if needed but it will not be marked.

Question 9 – Heaps [10 marks = 3 + 3 + 4]

(a) What is the worst-case time complexity of Heap Sort? Explain your answer.

(b) The elements in the list `[5, 8, 16, 1, 3, 7]` are added into a min-heap. If the min-heap is implemented using an array, draw the state of the array after all the elements have been added.

(c) A max-heap is implemented using an array, which contains `[None, 13, 8, 6, 7, 1, 3]`. Complete the diagram below showing how the array changes as the method `get_max` is executed. The diagram should depict what happens every time an element swaps positions in the array, and should finish with the state of the array once the maximum element has been retrieved.



10

This page intentionally left blank, use if needed but it will not be marked.

Question 10 – Hash Tables [12 marks = 4 + 2 + 2 + 4]

This question is about hash tables and dictionaries.

(a) List at least 3 data types covered in the lectures than can be used to implement a Dictionary ADT. In each case explain how insertions and deletions would perform in terms of complexity.

(b) A hash table is supported by an array of size 7. Draw the final state of the array after inserting the numbers, 45, 22, 57 and 33. Collisions are resolved using linear probing with hash function $h_1(k) = k\%7$.

8

This page intentionally left blank, use if needed but it will not be marked.

(c) Repeat the exercise above when collisions are resolved using double hashing. The first hash function is the same, i.e., $h_1(k) = k\%7$; the secondary hash function is $h_2(k) = 5 - (k\%5)$

(d) Consider the following code, where the class `HashTable` implements a hash table with a universal hash function.

```
1 list = [1, 2, 3, 4, 5, 1, 2]
2 table = HashTable(capacity=365)
3 for item in list:
4     table.insert(key=item, value=str(item))
```

After the above is executed, a user attempts `table.insert(6, '6')`. What is the chance of having a collision arising from this insertion? Explain your answer.

4

END OF EXAM.