ECS529U Algorithms and Data Structures **Mock exam**

Student name:

Student ID:

Answer all 4 Questions.

Unless stated otherwise, **you are not allowed to use** built-in Python functions apart from `len` (for arrays). Also, unless stated otherwise, no other data structures can be used apart from arrays. In particular, you cannot use built-in list operations for appending to a list.

You can use subarray-creating constructs like `A[lo:hi]`.

You can use helper functions.

If you find a specific part too challenging, remember you can **move on and come back** to it later – try not to get stuck.

Question 1

This question is about algorithms on integers and on arrays.

a) Write a Python function

```
def allEqual(x,y,z)
```

which takes as inputs three integers and returns `True` if they are all equal, and `False` otherwise.

[3 marks]

b) Write a Python function

```
def tri(x,y,z)
```

which takes as inputs three integers and returns `True` if the sum of any two of them is greater or equal to the remaining one, and `False` otherwise. For example, `tri(3,5,3)` should return `True`, as 3+5 ≥ 3, 3+3 ≥ 5 and 5+3 ≥ 3. On the other hand, `tri(3,5,1)` should return `False`, as 3+1 is smaller than 5.

[4 marks]

c) Write a Python function

```
def isSuperSorted(A)
```

which takes as input an array of integers `A` and returns `True` if each element of the array is greater or equal than the square of the element before it, and `False` otherwise. For example, `isSuperSorted([1,2,4,17])` should return `True` (1*1 ≤ 2, 2*2 ≤ 4, 4*4 ≤ 17), while `isSuperSorted([1,4,2,17])` should return `False`, and `isSuperSorted([1,2,4,10])` should also return `False` (4*4 = 16 > 10).

[4 marks]

d) Write a Python function

```
def summify(A)
```

that takes as input an array `A` and returns a new array with the same length as `A` that contains all initial sums of elements of `A`, that is the result should be:

```
[A[0], A[0]+A[1], A[0]+A[1]+A[2], ..., A[0]+A[1]+...+A[len(A)-1]]
```

For example, `summify([1,21,4,10])` should return `[1,22,26,36]`.

[5 marks]

e) Using recursion, write a Python function

```
def applyI(f,x,i)
```

that applies `i`-many times `f` to `x`. For example, `applyI(f,42,5)` should compute and return the result of `f(f(f(f(f(42)))))`.
Note that `applyI(f,x,0)` should simply return `x`.

[4 marks]

f) Using recursion, write a Python function

```
def fold(A, m, f)
```

which takes as input an array of integers `A`, an integer `m`, and a function `f` that takes as input two integers and returns an integer. The function `fold` should apply `f` to `m` and `A[0]`, then apply `f` to the result of the latter and `A[1]`, and so on, and return the final result. In particular, `fold([],m,f)` should return `m`.

For example, `fold([10,20,30,40],5,f)` should return:

```
f(f(f(f(5,10),20),30),40)
```

[5 marks]

Question 2

This question is about complexity classes and array lists.

a) Complexity questions (you do not need to justify your answers).

   i. For each of the following expressions, find if they are $\Theta(1)$, $\Theta(\log n)$, $\Theta(n)$, $\Theta(n^{50})$ or $\Theta(2^n)$:

      1. $500 + 5\log n$
      2. 5000
      3. $500 + n + 5\log n + 50n$
      4. $5n\log n + 2^n + 300n^{50}$

   ii. Find the complexity, in terms of big-Θ, of the following expression:

$$5(\log n)^{13} + 300n^6 + 30n^5\log n + 100$$

[6 marks]

b) Consider array lists, implemented using array and count. What is the worst-case time complexity, in terms of big-Θ, of each of these operations:

   i. add an element in the array list (if no resizing is required) :
   ii. search for an element in the array list :
   iii. remove an element from the array list :
   iv. add an element in the array list (when resizing is required)

   Notes: Express the time complexity with respect to the size n of the array list (i.e. the number of elements that it contains). Assume the array list is not sorted. You do not need to justify you answers.

[4 marks]

b) Explain in your own words, and in 2-3 sentences, what is the purpose of the resizing function in an array list implementation that uses array and count.

[2 marks]

The next 4 sub-questions ask you to write functions for the `ArrayList` class, the code of which is appended at the end of this test.

c) Write a function

```
def removeAll(self, A)
```

that removes from the array list (the one represented by `self`) all the elements of the array `A`.

[2 marks]

d) Write a function

```
def isTooEmpty(self)
```

that returns `True` if the size of the array list is less than half the size of its internal array, and otherwise `False`.

[3 marks]

e) In the same way that we resize-up our internal array when it gets too full, we can also resize it down when it is too empty. Write a function

```
def _resizeDown(self)
```

that uses the function `isTooEmpty` defined above to check if the size of the array list is less than half than that of its internal array and, if so, it resizes the internal array down to half its original size. Otherwise, the array list is left as it is.

Notes: The order of the elements in the array list should remain the same, and no elements should be lost.

[4 marks]

f) Write a function

```
def duplicate(self)
```

that changes the array list by adding after each element a copy of itself. For example, if the array list `ls` is `[2,4,1,5,1,4]`, then `ls.duplicate()` should change it to `[2,2,4,4,1,1,5,5,1,1,4,4]`.

[4 marks]

Question 3

This question is about linked lists, trees and heaps.

```
class Node:
      def __init__(self, d, n):
            self.data = d
            self.next = n
```

In the first two sub-questions we consider linked lists which are sequences of connected nodes, taken from the class above.

a) You are asked to solve the following pointer game. Start from this initial configuration:

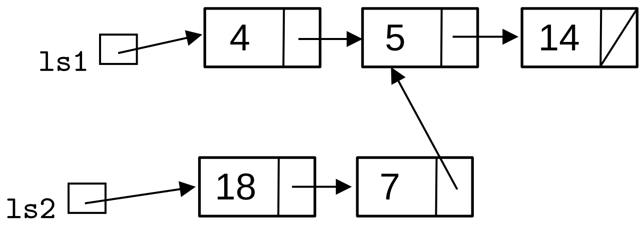

Figure 1: initial pointer configuration

Perform the following sequences of operations and draw the final configuration in each case. Each time you should start from the configuration in Figure 1 above.

i.
```
ls2.next = ls2.next.next
```

ii.
```
ptr = ls2
while ptr != None:
   ptr.data = 42
   ptr = ptr.next
```

iii.
```
ptr = ls1
while ptr.next != None:
   ptr = ptr.next
ptr.next = Node(42,ls2)
```

[6 marks]

b) This sub-question is about binary search trees. Draw the binary search tree we obtain if we start from the empty tree and add consecutively the numbers:

40, 6, 12, 45, 0, 54, 0, 42, 13, 44, 99, 4

[4 marks]

c) Starting each time from the binary search tree you constructed in part b, and using the remove functions that we saw in the lectures, remove from the tree the nodes containing the following numbers and draw the new binary search tree we obtain:

i. 4

ii. 42

iii. 40

[4 marks]

d) This sub-question is about heaps. Draw the heap we obtain if we start from the empty heap and add consecutively the numbers:

4, 6, 12, 45, 0, 54, 0, 42, 13, 44, 99, 4

[4 marks]

e) Write a function

```
def third(A)
```

that takes as input an array `A` of integers that represents a heap and returns the third largest element in `A`. For example, if the three largest elements in `A` are 42, 30 and 27, it should return 27. If `A` has length less than 3, `third(A)` should return `None`.

[4 marks]

f) Write a Python function

```
def filter(ls, t)
```

that takes as input a linked list `ls` (i.e. an object of the class `Node`), and a test function `t`, i.e. such that `t(d)` returns a boolean for any value `d` in the list. The `filter` function returns a new list that contains all elements `d` of `ls` that pass the test, i.e. such that `t(d)` is `True`. The original list `ls` should not be changed.
For example, if `ls` is `[4, 42, 3, 2, 1]`, and `even` is the function:

```
def even(d):
   if d%2 == 0: return True
   return False
```

then `filter(ls,even)` should return a new linked list `[4, 42, 2]`.

[5 marks]

Question 4

This question is greedy algorithms, dynamic programming and hash tables.

a) For each of the following statements, say whether they are correct or not:

  i. Every problem has an optimal greedy solution.

  ii. Greedy algorithms are preferred to simple recursive ones because they are faster.

  iii. Dynamic programming algorithms are preferred to simple recursive ones because they are faster.

  iv. Dynamic programming means that, at each step of an algorithm, we decide what next step to make based on which step gives us the best immediate outcome.

  v. In general, dynamic programming algorithms tend to solve problems faster than greedy ones.

[5 marks]

b) We define the Cool sequence of numbers by the following function $cool$:

  - $cool(0) = 0$
  - $cool(1) = 1$
  - $cool(n) = 2 \cdot cool(n\text{-}1) + 3 \cdot cool(n\text{-}2)$, if $n > 1$

  Write a <u>recursive</u> Python function

```
def cool(n)
```

  that, on input `n`, returns $cool(\mathtt{n})$. [3 marks]

e) Change the function from part 2 into a dynamic programming one:

```
def coolDP(n)
```

  using memoisation. [5 marks]

f) Change the function from part 3 into a dynamic programming bottom-up one:

```
def coolDPBU(n)
```

  using iteration (i.e. a for-loop). [5 marks]

g) Starting from an empty hash table of size 15, and assuming a hash function:

$$\text{hash}(i) = i \,\%\, 15$$

  which returns the remainder of dividing i by 15, draw the hash table we obtain after inserting consecutively the elements:

  42, 45, 10, 56, 90, 99, 10, 2, 2, 2103

[4 marks]

h) Explain what is the worst-case complexity of each of the hash table operations (add, search, remove) if we assume simple uniform hashing and do not use resizing-up.

[3 marks]

Appendix: ArrayList code

```
class ArrayList:
      def __init__(self):
            self.inArray = [0 for i in range(10)]
            self.count = 0

      def get(self, i):
            return self.inArray[i]

      def set(self, i, e):
            self.inArray[i] = e

      def length(self):
            return self.count

      def append(self, e):
            self.inArray[self.count] = e
            self.count += 1
            if len(self.inArray) == self.count:
                  self._resizeUp()

    def insert(self, i, e):
       for j in range(self.count,i,-1):
           self.inArray[j] = self.inArray[j-1]
       self.inArray[i] = e
       self.count += 1
       if len(self.inArray) == self.count:
           self._resizeUp()

   def remove(self, i):
       self.count -= 1
       val = self.inArray[i]
       for j in range(i,self.count):
           self.inArray[j] = self.inArray[j+1]
       return val

   def _resizeUp(self):
       newArray = [0 for i in range(2*len(self.inArray))]
       for j in range(len(self.inArray)):
           newArray[j] = self.inArray[j]
       self.inArray = newArray
```