ECS529U Algorithms and Data Structures Mid-term test

Answer all 3 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 (e.g. you cannot use hashtables).

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

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

Question 1

This question is about algorithms on integers and on arrays.

1. Write a Python function

    `def areInOrder(x,y,z)`

    which takes as inputs three integers and returns `True` if they are in order (i.e. if `x` is less or equal to `y`, which in turn is less or equal to `z`), and `False` otherwise.

    [3 marks]

```
def areInOrder(x,y,z):
    return x <= y and y <= z
```

2. Write a Python function

    `def notAllDifferent(x,y,z)`

    which takes as inputs three integers and returns `True` if two or more of them are equal, and `False` otherwise.

    [3 marks]

```
def notAllDifferent(x,y,z):
    return x == y or y == z or x == z
```

3. Write a Python function

```
def isSorted(A)
```

which takes as input an array of integers `A` and returns `True` if the array is sorted (in increasing order), and `False` otherwise. For example, `isSorted([1,1,4,10])` should return `True`, while `isSorted([1,4,1,10])` should return `False`.

[4 marks]

```
def isSorted(A):
    for i in range(len(A)-1):
        if A[i] > A[i+1]:
            return False
    return True
```

4. Write a Python function

```
def reverse(A)
```

that takes as input an array `A` and returns a new array with the same elements as `A` but in reverse order.

[4 marks]

```
def reverse(A):
    B = [0  for i in range(len(A))]
    for i in range(len(A)):
        B[i] = A[len(A)-i-1]
    return B
```

5. Write a Python function

```
def binSearchCube(A,k)
```

which takes as input a sorted array of integers `A` and an integer `k`, and returns the position in `A` of the number `k*k*k` (i.e. the cube of `k`). If `k` is not in `A` then it should return `-1`. Your function should use binary search.

[5 marks]

```
def binSearchCube(A,k):
    lo = 0
    hi = len(A)-1
    k3 == k*k*k
    while lo <= hi:
        mid = (lo+hi)//2
        if A[mid] == k3:
            return mid
        if A[mid] < k3:
            lo = mid+1
        else:
            hi = mid-1
    return -1
```

6. Write a Python function

    def countDuplicates(A)

which takes as input an (unsorted) array of integers A and returns the number of duplicate elements contained in A. For example:

- on input [4,5,2,5,2,4,4] it should return 4: two of the elements with value 4 are duplicates, one 5 is duplicate, and one 2 is a duplicate.
- on input [4,5,2,3] it should return 0: there are no duplicates in this array.

For full marks, your solution should be optimised and work in $\Theta(n \log n)$, where n is the length of A. The array A should remain unchanged.

You can use the function mergesort(A) that sorts A in increasing order.

[4 + 2 marks]

```
def countDuplicates(A):
    B = [A[i] for i in range(len(A))]
    mergeSort(B)
    count = 0
    for i in range(len(B)-1):
        if B[i] == B[i+1]:
            count += 1
    return count
```

Question 2

This question is about time complexity, recursion and sorting.

1. What is the worst-case time complexity, in terms of big-Θ, of each of these algorithms:

i.	insertion sort	$\Theta(n^2)$
ii.	quicksort	$\Theta(n^2)$
iii.	merge sort	$\Theta(n \log n)$

[2 marks]

2. Write a Python function

    def findLeast(A)

which takes as input an array A of integers and returns its least element. If the array is empty, it should return None. For example, on input [2,42] it should return 2.

What is the worst-case time complexity of your function, in terms of big-Θ, with respect to the length of the array A? (you do not need to justify your answer).

[3 + 1 marks]

```
def findLeast(A):
    if len(A) == 0:  return None
    least = A[0]
    for i in range(1,len(A)):
        if A[i] < least:
            least = A[i]
    return least
```

Worst case is $\Theta(n)$

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

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

i. $500 + 5\log n$ $\boxed{\Theta(\log n)}$

ii. 5000 $\boxed{\Theta(1)}$

iii. $500 + n + 5\log n + 50n$ $\boxed{\Theta(n)}$

iv. $5\,n\log n + 2^n + 300\,n^{50}$ $\boxed{\Theta(2^n)}$

b) Find the complexity, in terms of big-Θ, of the following expression:

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

[5 marks]

4. Using recursion, write a Python function

```
def sumArray(A)
```

which takes as input an array `A` of integers and returns the sum of its elements.

[4 marks]

```
def sumArray(A):
    if len(A) == 0:
        return 0
    else :
        return A[0] + sumArray(A[1:])
```

5. Using recursion, write a Python function

```
def square(A)
```

which takes as input an array `A` of integers and replaces each of its elements by its square. For example, on input `A = [1,12,4,10]` it should set `A` to `[1,144,16,100]`.

[5 marks]

```
def square(A):
    return squareRec(A,0)

def squareRec(A, lo):
    if lo == len(A):
        return
    A[lo] = A[lo] * A[lo]
    return squareRec(A,lo+1)
```

6. Using recursion, write a Python function

```
def solve(f,n)
```

which takes a function `f` (from integers to integers) and a positive integer `n`, and returns the least number `x` in the range from `0` to `n` (inclusive) such that `f(x)` evaluates to `0`. If no such number exists, the function `solve` should return `None`.

For example, the following code should return `2`:

```
def fun1(x):
  return (x**2 -7*x + 10) # so, fun1(x) == 0 if x == 2 or x ==5

solve(fun1,10)
```

whereas this code should return `None`:

```
def fun2(x):
  return (x**2 - 10*x - 11) # so, fun2(x) == 0 if x == -1 or x == 11

solve(fun2,10)
```

[5 marks]

```
def solve(f,n):
    return solveRec(f,n,0)

def solveRec(f,n,x):
    if x > n:
        return None
    elif f(x) == 0:
        return x
    else :
        return solveRec(f,n,x+1)
```

Question 3

This question is about Greedy and Dynamic Programming algorithms.

1. For each of the following statements, say whether they are correct or not:
   i. Every problem has an optimal greedy solution.

      Incorrect

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

      Correct

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

      Correct

   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.

      Incorrect

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

      Incorrect

[5 marks]

2. 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*(`n`).

[3 marks]

```
def cool(n):
    if n <= 1:
        return n
    else :
        return 2*cool(n-1) + 3*cool(n-2)
```

3. Change the function from part 2 into a dynamic programming one:

```
def coolDP(n)
```

using memoisation.

[5 marks]

```
def coolDP(n):
    memo = [-1 for i in range(n+1)]
    return coolMem(n,memo)

def coolMem(n, memo):
    if memo[n] != -1:
        return memo[n]
    if n <= 1:
        memo[n] = n
    else:
        memo[n] = 2*coolMem(n-1,memo)+3*coolMem(n-2,memo)
    return memo[n]
```

4. 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]

```
def coolDPBU(n):
    memo = [-1 for i in range(n+1)]
    memo[0] = 0
    memo[1] = 1
    for i in range(2,n+1):
        memo[i] = 2*memo[i-1]+3*memo[i-2]
    return memo[n]
```

5. Write a <u>greedy</u> Python function

    def coinSplitGD2(m, avail)

   that is a variant of the greedy coin splitting function that we saw in the lectures and which works as follows:

   - it takes as input an integer m and an array avail which stores how many coins of each type we have available (e.g. avail[0] = 5 means we have 5 coins of value 200 available),
   - and returns the least number of coins that we can split m into using available coins.

   Your function should use as given the array: coin = [200,100,50,20,10,5,2,1].

   For example, coinSplitGD2(400,[1,4,0,0,0,0,0,0]) should return 3 because the best split we can do is: 1·200 + 2·100. On the other hand, coinSplitGD2(400, [2,4,0,0,0,0,0,0]) should return 2 because we can do the split: 2·200.

[5 marks]

```
def coinSplitGD2(m, avail):
    if m == 0:
        return 0
    for i in range(len(coin)):
        if coin[i] <= m and avail[i] > 0:
            avail[i] -= 1
            return 1 + coinSplitGD2(m-coin[i], avail)
```

6. Compute the time complexity of your function coinSplitGD2 from Part 5 with respect to the input m.

[2 marks]

In the worst case, we will need to make m recursive calls, as each call reduces the value m by at least 1. Moreover, each call takes constant time, as the for number of repetitions of the for loop is bounded by the length of coin, which is a fixed number (8). Thus, in the worst case we need to make $\Theta(m)$ many steps.