Model answers to revision questions

In [ ]:
```
## Question 1

# part a
def allEqual(x,y,z):
    return x == y and y == z

# part b
def tri(x,y,z):
    return x + y >= z and x + z >= y and y + z >= x

# part c
def isSuperSorted(A):
    for i in range(len(A)-1):
        if A[i]*A[i] > A[i+1]: return False
    return True

# part d
def summify(A):
    B = [0 for i in range(len(A))]
    sumI = 0
    for i in range(len(A)):
        sumI += A[i]
        B[i] = sumI
    return B

# part e
def applyI(f,x,i):
    if i == 0: return x
    return f(applyI(f,x,i-1))

# part f
def fold(A, m, f):
    if A == []: return m
    return fold(A[1:],f(m,A[0]),f)

# for example:
# fold([10,20],5,f)
# -> fold([20],f(5,10),f)
# -> fold([],f(f(5,10),20),f)
# -> f(f(5,10),20)

# alternative implementation
def fold2(A, m, f):
    if A == []: return m
    return f(fold(A[:len(A)-1],m,f),A[len(A)-1])
```

In [ ]:

```
## Question 2

# part a
'''
i.   θ(1)
ii.  θ(n)
iii. θ(n)
iv.  θ(n)
'''

# part b
'''
Its purpose is to make the internal array larger whenever we reach its full capacity, that is,
whenever the length of the array list becomes the same as the length of the internal array.
This way, we can add more elements to the array list.
'''

# part c
def removeAll(self, A):
    for i in range(len(A)):
        self._removeVal(A[i])

def _removeVal(self, d):
    for i in range(self.count):
        if self.inArray[i] == d:
            self.remove(i)
            return
    # Note that the question does not clarify whether all occurrences of elements of A should
    # be removed or just one for each element. This solution removes just one (the first one).

# part d
def isTooEmpty(self):
    return self.count < len(self.inArray)/2

# part e
def _resizeDown(self):
    if self.isTooEmpty():
        newArray = [0 for i in range(len(self.inArray)//2)]
        for i in range(len(newArray)):
            newArray[i] = self.inArray[i]
        self.inArray = newArray

# part f
def duplicate(self):
    copy = [self.inArray[i] for i in range(self.count)]
    for i in range(len(copy)):
        self.insert(2*i,copy[i])

# alternative implenentation -- resizes up by default
def duplicate2(self):
    newArray = [0 for i in range(len(self.inArray)*2)]
    for i in range(len(self.inArray)):
        newArray[2*i] = self.inArray[i]
        newArray[2*i+1] = self.inArray[i]
    self.inArray = newArray
    self.count *= 2

# part g
'''
i.   Adding an element is θ(n) because when adding a new element we need to make sure that the
     array list remains sorted. so the element needs to be added in its ordered position in the
     array list. In the worst case, we need to add it in the first position and for that we need
     to move all the other elements one position to the right.

ii.  Seraching an element is θ(logn) in the worst case because we can do binary search.

iii. Removing an element is also θ(n) because e.g. when removing the first element of the array
     list, every other element needs to be moved one position to the left.
'''
```

In [ ]:

```
## Question 3

# part a

# i
'''
ls1 → 4 → 5 → 14 → None
          ↑
    ls2 → 18
'''
# ii
'''
ls1 → 4  → 42 → 42 → None
          ↑
ls2 → 42 → 42
'''
# iii
'''
ls1 → 4 →  5 → 14 → 42
          ↑         |
ls2 → 18 → 7        |
       ↖____________/
'''

# part b
'''
             40
           /    \
          /      \
         /        \
        6         45
      / \        /  \
     /   \      /    \
    0    12    42    54
     \    \     \     \
      0    13   44    99
       \
        4
'''

# part c

# i
'''
             40
           /    \
          /      \
         /        \
        6         45
      / \        /  \
     /   \      /    \
    0    12    42    54
     \    \     \     \
      0    13   44    99
'''

# ii
'''
             40
           /    \
          /      \
         /        \
        6         45
      / \        /  \
     /   \      /    \
    0    12    44    54
     \    \           \
      0    13         99
       \
        4
'''

# iii
'''
             42
           /    \
          /      \
         /        \
        6         45
      / \        /  \
     /   \      /    \
    0    12    44    54
     \    \           \
      0    13         99
       \
        4
'''
```

```
# part d
'''
            99
          /    \
         /      \
        /        \
       54         45
      /  \        / \
     /    \      /   \
    13    44     6    0
   / \    / \   /
  4  12  0  42  4
'''

# part e
def third(A):
    if len(A) < 3: return None
    if A[1] > A[2]:
        third = A[2]
        if len(A) >= 3:
            if A[3] > third: third = A[3]
        if len(A) >= 4:
            if A[4] > third: third = A[4]
    else:
        third = A[1]
        if len(A) >= 5:
            if A[5] > third: third = A[5]
        if len(A) >= 6:
            if A[6] > third: third = A[6]
    return third

# part f:
def filter(ls, t):
    ptr = ls
    while ptr != None and not t(ptr.data):
        ptr = ptr.next
    if ptr == None: return None
    ls2 = Node(ptr.data,None)
    ptr2 = ls2
    while ptr.next != None:
        ptr = ptr.next
        if t(ptr.data):
            ptr2.next = Node(ptr.data,None)
            ptr2 = ptr2.next
    return ls2

# alternative solution -- use an initial dummy node
def filter2(ls, t):
    ptr = ls
    ls2 = Node(42,None) # dummy node starting ls2
    ptr2 = ls2
    while ptr != None:
        if t(ptr.data):
            ptr2.next = Node(ptr.data,None)
            ptr2 = ptr2.next
        ptr = ptr.next
    return ls2.next # throws away dummy node
```

In [ ]:

```
# Appendix: Node and ArrayList code

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

    def __str__(self):
        if self == None: return "None"
        return str(self.data) + " -> " + str(self.next)

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()     # resize array if reached capacity

    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()     # resize array if reached capacity

    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 __str__(self):
        return str(self.inArray[:self.count])

    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
```