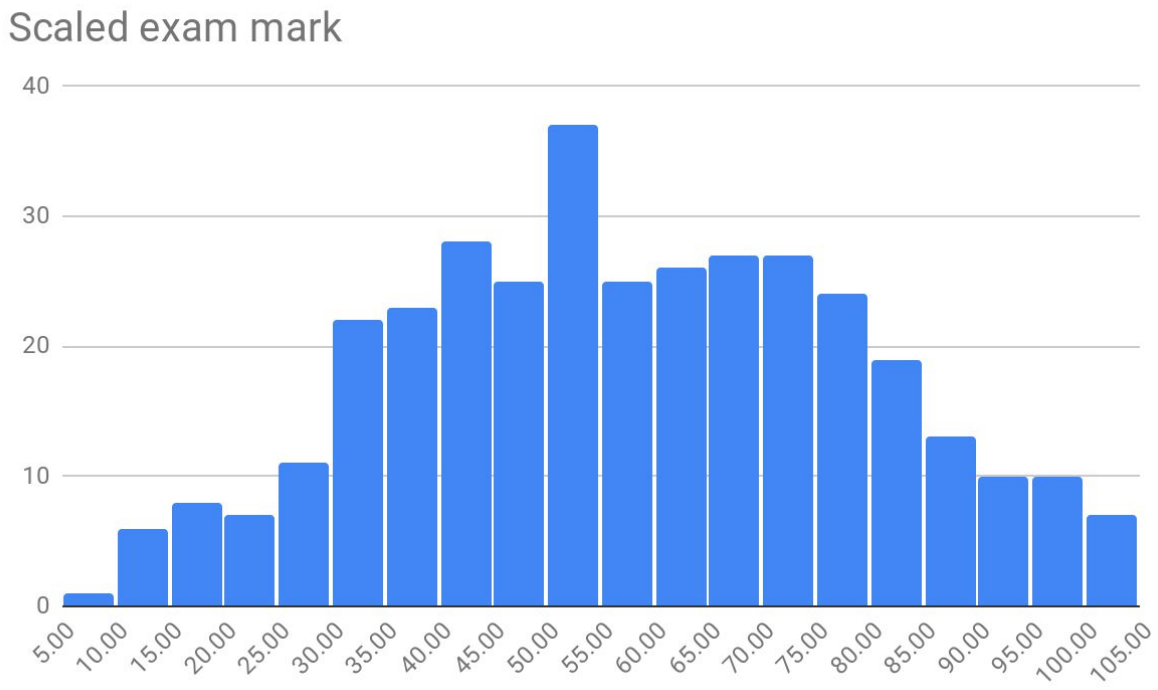
FIT1008-2085 S1 2019 exam feedback

Final exam mark

As indicated (in blue) throughout the document, a total of 26 marks have been removed from the exam. Questions that have been removed are marked normally and the marks of a question that is removed count just the same as the others. This means that if out of 100, your mark was X, then your new, scaled, exam mark is X * 100 / 74, capped at 100.

Throughout the document we provide histograms of mark distributions for FIT1008 and 2085 for both Malaysia and Clayton cohorts. The histograms for each question are given on the unscaled marks. The scaled exam marks are given below.



Exam Viewing

Should you request to view your exam, a printed version of your exam will be made available. See this link to see how to apply for exam viewing. Please do not reach out to your teaching team for that. We don't have your scripts!

Q1 - Python to MIPS translation

```
func:
  # save the $fp and $ra into the stack
  addi $sp, $sp -8   # make space in the stack for the two registers
  sw $ra, 4($sp)     # save $ra onto stack
  sw $fp, 0($sp)     # save $fp onto stack

  addi $fp, $sp, 0  # copy $sp into $fp

  addi $sp, $sp, -4 # make space for local var (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 functions in the Heap) will not change.

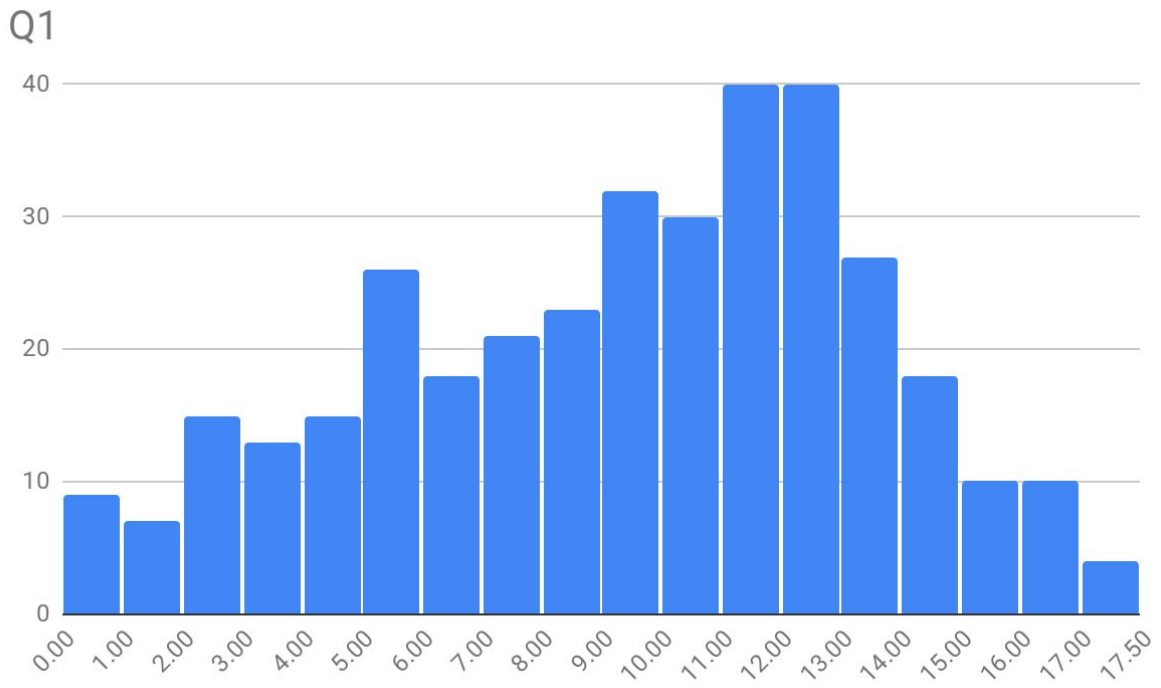
For the MIPS code provided, N is 0, as nothing is created in the Heap.

Note that, in practice, Python would create objects for integers and will indeed use the Heap.

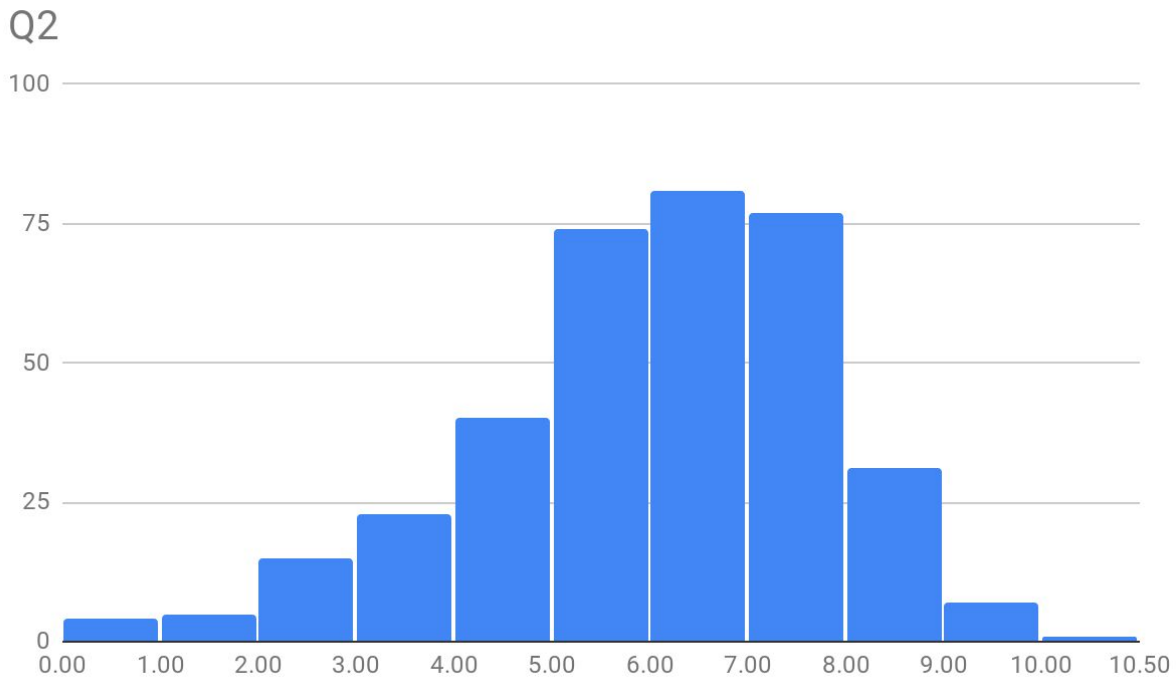
Part 1.h

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 - Scoping with classes in Python



Q3 - ADT and linear structures

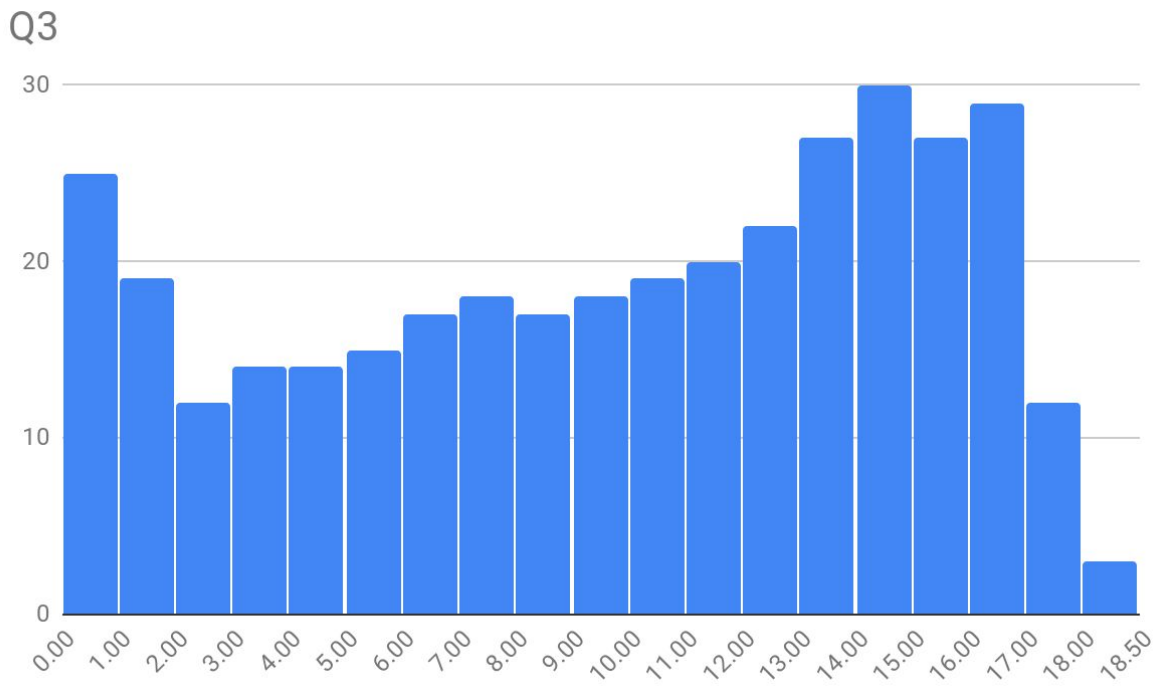
Define the class Node (with all its methods).

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None
        self.previous = None

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next

    def get_previous(self):
        return self.previous

    def set_data(self,newdata):
        self.data = newdata

    def set_next(self,newnext):
        if self.next is not None:
            self.next.previous = None
        newnext.previous = self
        self.next = newnext
```

Define the class Deque below (with all its methods).

```
class Deque:
    def __init__(self):
        self.head = Node(None)
        self.tail = Node(None)
        self.head.set_next(self.tail)
        self.nodes = 0

    def size(self):
        return self.nodes

    def remove_front(self):
        frontnode = self.head.get_next()
        newfrontnode = frontnode.get_next()
```

```
    self.head.set_next(newfrontnode)
    self.nodes -= 1
    return frontnode.get_data()

def remove_rear(self):
    rearnode = self.tail.get_previous()
    newrearnode = rearnode.get_previous()
    newrearnode.set_next(self.tail)
    self.nodes -= 1
    return rearnode.get_data()

def add_front(self, data):
    newfrontnode = Node(data)
    oldfrontnode = self.head.get_next()
    self.head.set_next(newfrontnode)
    newfrontnode.set_next(oldfrontnode)
    self.nodes += 1

def add_rear(self, data):
    newrearnode = Node(data)
    oldrearnode = self.tail.get_previous()
    oldrearnode.set_next(newrearnode)
    newrearnode.set_next(self.tail)
    self.nodes += 1
```



Q4 - CS saves the world

Write the output of the function 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). See solutions of Exercise 2 of tute 5.

Write the output of the function enigma for the input …

1
1
3
6
1
5

The subquestions below this line have been removed the exam (11 out of 20 marks). It was of the same difficulty of the digital root question (Exercise 6 of tute 5), which is a non-starred exercise. This was hard but doable, as the histogram shows. However, since there were other hard questions, this one was removed.

What does the function enigma compute?

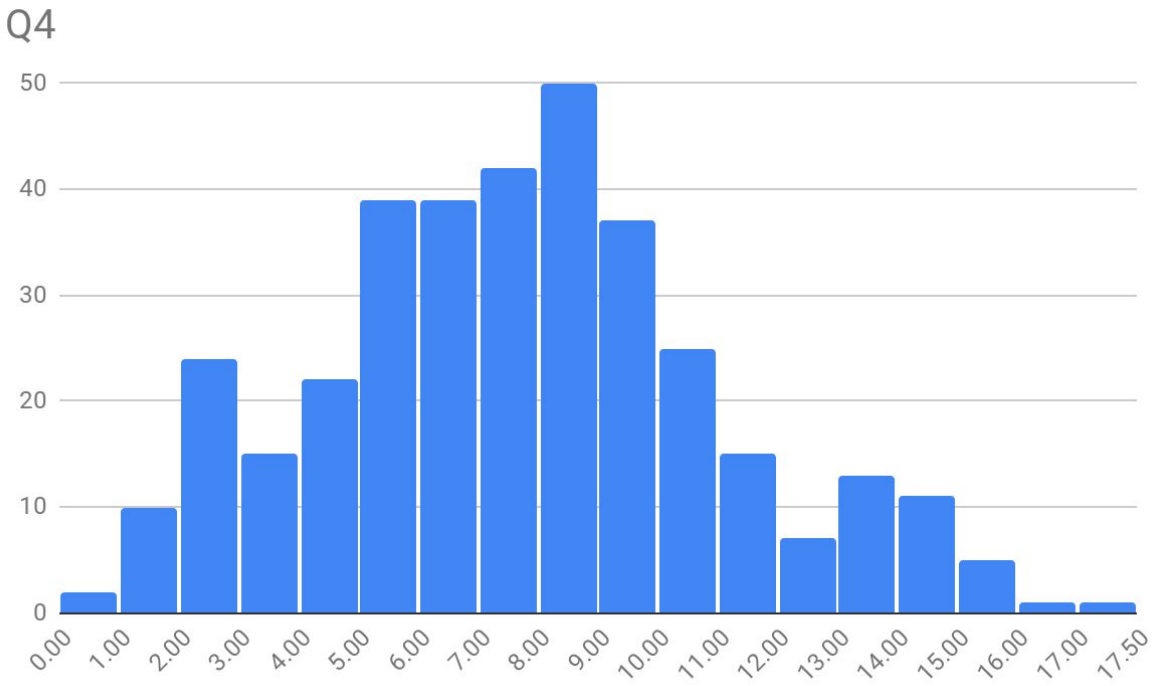
It computes mystery(x) + mystery(mystery(x)) + …

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).
See solutions of Exercise 6 of tute 5.

What does enigma(4095) return? Justify your answer.

Observe that 4095 = 4096 - 1 = 2^{12} -1 = 2^{11} + 2^{10} + ... + 2^0, hence mystery(4095) returns 12.
Therefore enigma(4095) returns 12 + enigma(12).
mystery(12) returns 2, and mystery(2) returns 1.
Hence enigma(4095) returns 15



Q5 - Natural merging (Jenica & Ammar ft. Ben)

Write a function find_intervals which, given a list as input, returns the list of indices between which the input list is already sorted.

```
def find_intervals(l):
    separators = [0]
    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?

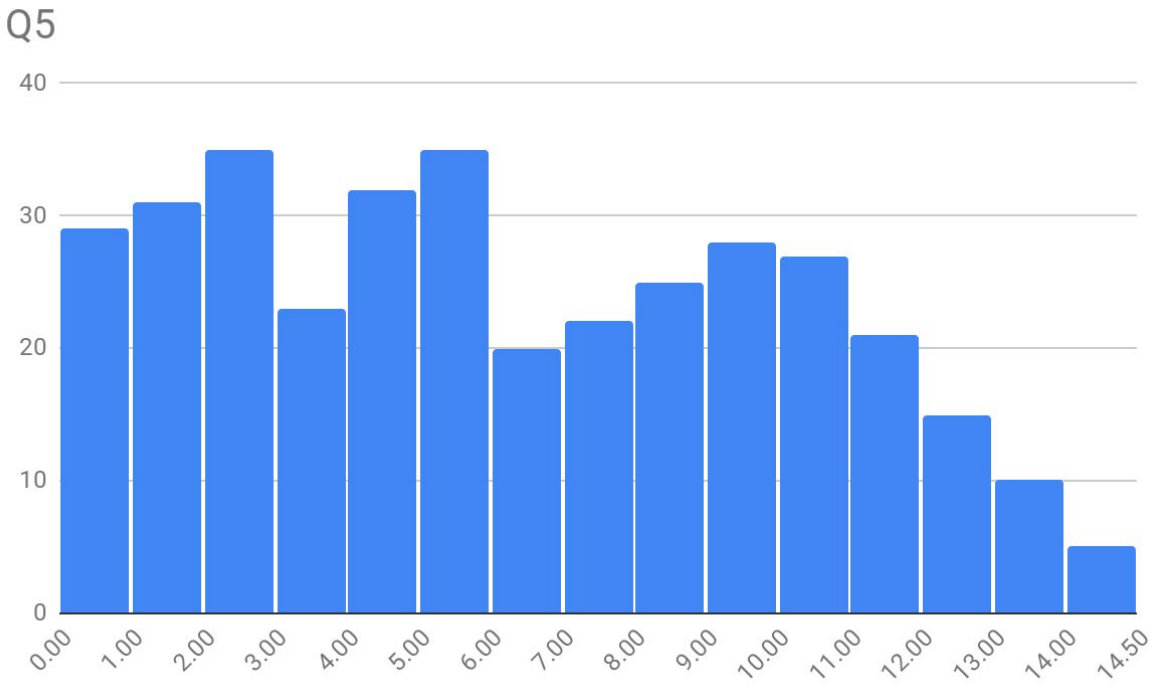
In the best case, the input list is already sorted, and there are only 2 items in the separators list, which means that the only work to do was to call find_intervals. Complexity O(n), where n is the length of the input list.

What is the worst-case time complexity of the algorithm …

The worst-case occurs when a list is sorted in the opposite order. In this case, all sublists have size 1, and we need to merge everything iteratively. There are O(n) merges to do, the first one with 2 elements, the second one with 3, then 4, 5, ..., n. Since the merge function has complexity O(end-start), the total runtime is 1+2+3+4+...+n = O(n^2).

How could a sorting algorithm with 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.



Q6 - Two coin problems

Write a function non_adjacent_coins which takes as input a list of coin values and outputs the maximum value of a valid subset of coins.

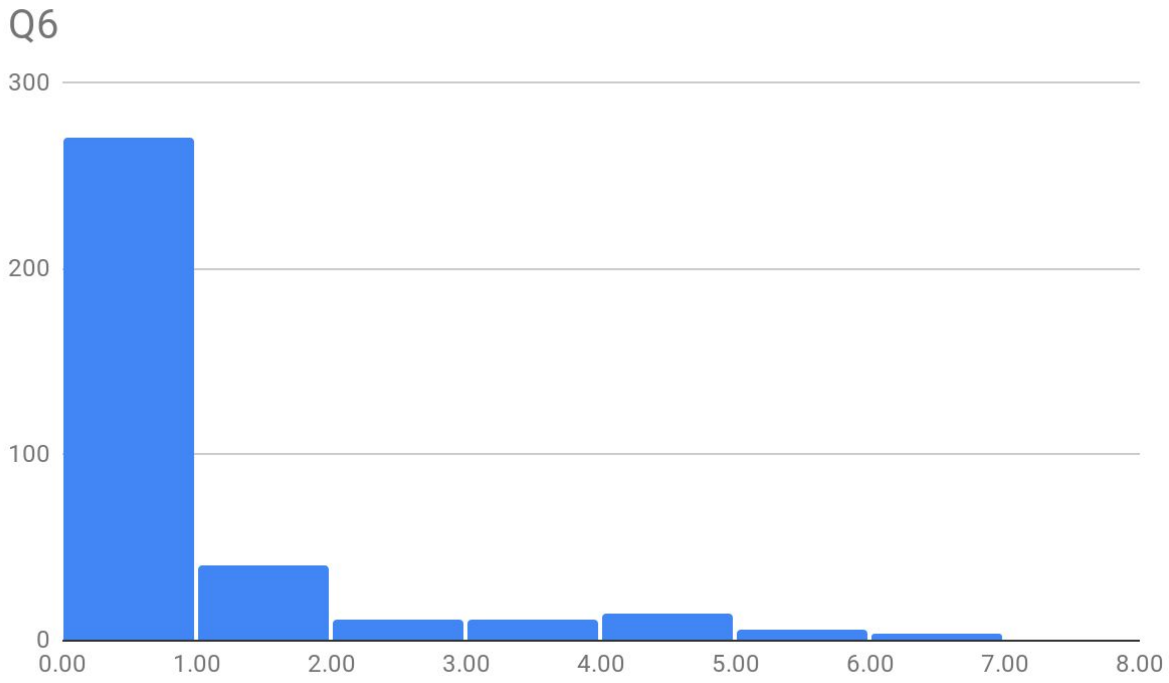
```
def non_adjacent_coins(l):
    n = len(l)
    mem = [None] * (n+1)
    mem[0] = 0
    mem[1] = l[0]
    for i in range (2, n+1):
            mem[i] = max(mem[i-1], l[i-1]+mem[i-2])
    return mem[n]
```

Here we expected a DP solution, but other methods have been given marks if they were efficient.

The subquestion below this line has been removed the exam (4 out of 8 marks). Even though there was very little to write, the reasoning behind the formulas went beyond what had been covered in other DP questions during the semester.

Write a function adjacent_coins which takes as input a list of coin values and outputs the maximum value of a valid subset of coins.

```
def adjacent_coins(l):
    n = len(l)
    #We compute the max achievable with the first i coins
    #in three different cases:
    zero_taken = [None] * (n+1)#if i is not picked
    one_taken = [None] * (n+1)#if i is picked but not i-1
    two_plus_taken = [None] * (n+1)#if i and i-1 are picked
    #we initialise the arrays
    zero_taken[0] = 0
    one_taken[0] = - math.inf
    two_plus_taken[0] = - math.inf
    #we iterate to compute all other values
    for i in range(1, n+1):
        zero_taken[i] = max(zero_taken[i-1], two_plus_taken[i-1])
        one_taken[i] = l[i-1] + zero_taken[i-1]
        two_plus_taken[i] = l[i-1] + max(one_taken[i-1], two_plus_taken[i-1])
    #we return the maximum value
    return max(zero_taken[n], two_plus_taken[n])
```


Q6
300
200
100
0
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00

Q7 - k smallest elements

The entire question has been removed from the exam (9 out of 9 marks). The question relied on students being able to measure the memory footprint of an algorithm, a concept that was not formally covered this semester.

Explain how the k smallest elements can be found in time O(n log n) and O(n) auxiliary memory.

We can sort using merge sort for a worst-case of O(n log n) and O(n) extra memory.

Explain how the k smallest elements can be found in time O(kn) and O(1) auxiliary memory.

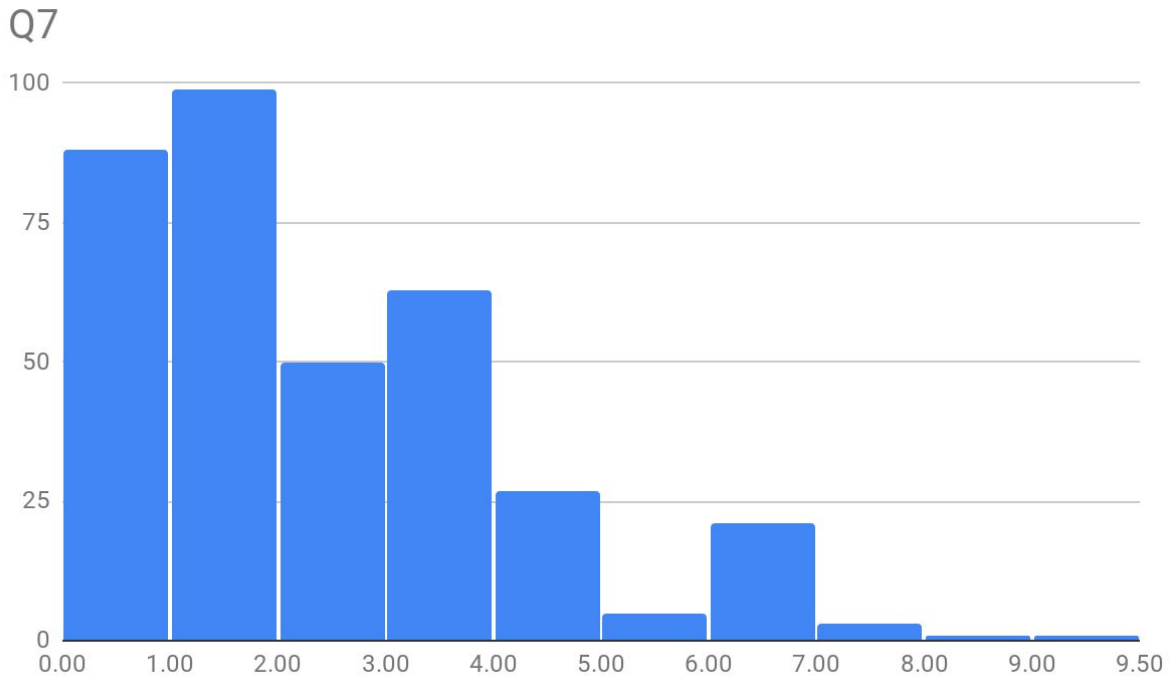
Iterate through the list to find the min. remember the index.
Continue iterating, and look for the smallest element which is larger than the previous element found or with higher index than the previous one. That's O(1) memory.
We iterate k times throughout the entire list, so O(kn) time.

Explain how the k smallest elements can be found in time O(n + k log n) and O(n) auxiliary memory.

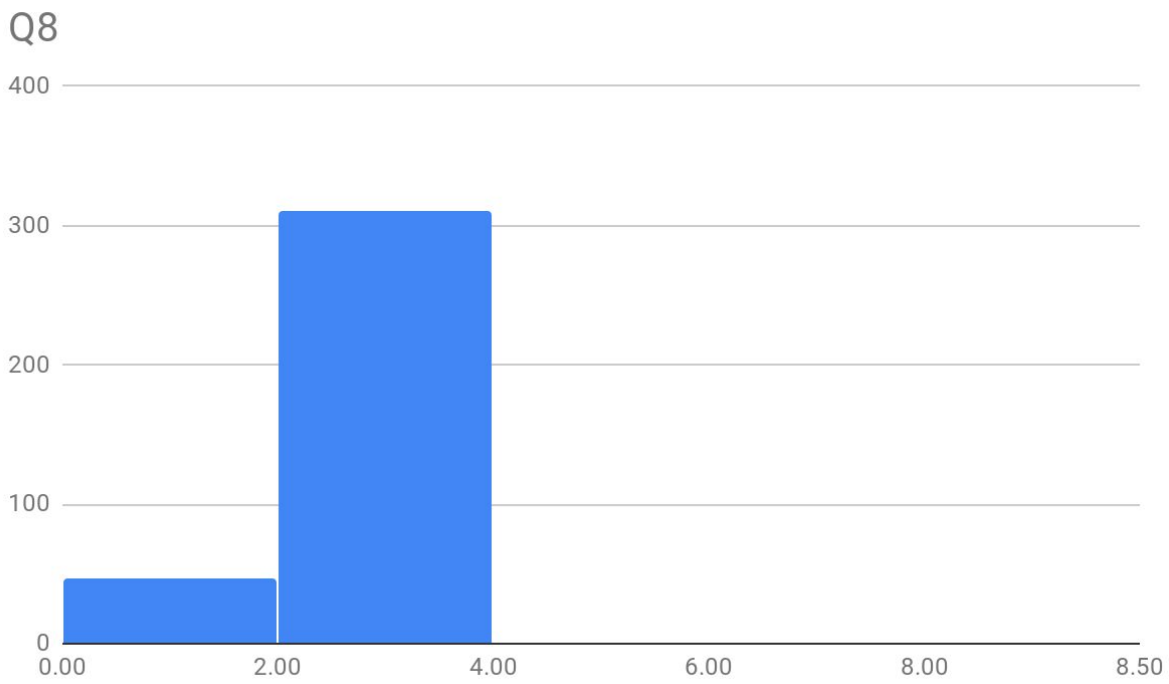
Build a min-heap of these elements in O(n) and pop the first k ones, each taking O(log n).

Explain how the k smallest elements can be found in time O(n log k) and O(k) auxiliary memory.

Build a max-heap with the first k elements of the list. O(k)
Iterate through the rest of the list. If the item in the list is smaller than the top of the heap, swap and re-heapify. O(log k) for each iteration. O(n log k) total.


Q7
100
75
50
25
0
0.00
1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
9.50

Q8 - Resolving collisions



Q9 - Resolving collisions

The entire question has been removed from the exam (2 out of 2 marks). Q9 resulted in an error due to cycling. While this could be discovered on a few iterations by hand, proving that the cycling continues indefinitely is beyond the scope of the unit.