[DOCUMENT TITLE]

[Document subtitle]

Q3. (20 marks)

Question A

Answer:

```
Function find_most_frequent(k):
      most_frequency = 0                       # O(n)
      for i in k:                              # O(1)
            count_for_now = 0                  # O(n)
            for j in k:                        # O(1)
                  if j == i:                   # O(1)
                     times += i                # O(1)
            if times larger than most_frequency:# O(1)
                  most_frequency = times       # O(1)
      return most_frequency
```

Use two for loop. Suppose k = an array of integers and most_frequency = 0. for i in k, using count_for_now = 0 to count the number i occurring in the array of integers. And then for j in k, if the number i is equal to the number j, then count_for_now += 1. After finish the loop of k, comparing the number count_for_now with the number most_frequency, and chose the number most_frequency is equal to the biggest one.
Return the most_frequency

Time complexity:

Two for loop, the first one: for i in k, is equal to O(n), the second one: for j in k, is equal to O(n). Therefore, the algorithm has O(n^2) time complexity

Space complexity:

only need count_for_now and most_frequency has fixed length 2, so it is O(1)

Question B

```
Function find_most_frequent(k):
      Set an empty Dictionary, called frequent_one         # O(1)
      For i in k:                                          # O(n)
            If i in frequent_one:                          # O(1)
                  Value of key i in frequent_one + 1       # O(1)
            else:
                  value of key i in frequent_one = 1       # O(1)
      sort the largest integer in frequent_one values      # O(1)
```

Create a dictionary, called frequent_one = {}. Suppose k = an array of

integers, for i in k if i exist in the frequent_one dictionary, then the value of that key plus one. Else, create an new key with starting value 1.
Sort the largest integer in frequent_one dictionary values and find the biggest value and return the keys of that value

Time complexity:
for i in range(length(k)) is equal to O(n). Therefore, T(n) = O(n)

Space complexity:
The frequent_one dictionary is fixed, therefore, so it is O(1)

Q4. (10 Marks)

Consider the following Pseudo-code that uses the Credit ADT from Q1.

```
function has_desc(accounts,desc):
    for account in accounts:
        for transaction in transactions(account):
            if transaction matches desc:
                return true
    return false
```

Let n be the number of cards in the Credit object, and let m be the number of cards in the accounts list.
Give the worst-case time complexity for has_desc, under the following assumptions. You must justify

your answer.
a) Assuming m is in O(1), and the cost of transactions(cc) is in O(log n)

Answer:
T = (O(log n) + O(1)*10)* O(1) = O(log n)
The cost of transactions(cc) is in O(log n), meaning it doesn’ t run in a constant time. Thus the inner loop is in O(n)

b) Assuming m is in O(n) and the cost of transactions(cc) is in O(n)
Answer:
T = (O(n) + O(1) * 10)* O(n) = O(nˆ2)