Part1
Question1
a)

```
Algorithm swapValueM(A, m)
  Input A: array of n integers, m: an integer
  Output: A
  for i <- 0 to n-2 do
    if A[i] == m then
      A[i] = A[i] + A[i+1]
      A[i+1] = A[i] - A[i+1]
      A[i] = A[i] - A[i+1]
  return A
```

b) Time complexity O(N)
c) Space complexity O(1)

Question2

```
Algorithm swapString(S)
  Input: S, a string with n characters
  Output: S_res, a string
  n <- length of S
  consonant <- string
  repeat <- string
  vowel <- string
  for i <- 0 to n - 1 do
    if S[i] is consonant:
      repeated <- false
      for j <- i + 1 to n - 1 do
        if S[j] == S[i] do
          repeat += S[i]
          repeated = true
          break
      if repeated == true:
        repeat += S[i]
      else:
        consonant += S[i]
    else if S[i] is vowel:
      vowel += S[i]
  return consonant+repeat+vowel
```

b) time complexity O(N^2)
c) space complexity O(N)

Question3

```
Algorithm tetradicNumber(Numbers):
  Input: Numbers, an array of integers
  Output: success1, success2, farthest1, farthest2
  success1 <- integer, first successive number with largest difference
  success2 <- integer, second sucessive number with largest difference
  largest_difference <- integer
  farthest1 <- integer, first number of two farthest numbers whose difference is ten
  farthest2 <- integer, second number of two farthest numbers whose difference is ten
  farthest <- integer

  n <- length of array Numbers

  for i <- 0 to n - 2:
    if absolute value of Numbers[i] - Numbers[i+1] > largest_difference:
      largest_difference = Numbers[i] - Numbers[i+1]
      success1 = i
      success2 = i+1
  for i <- 0 to n-1:
    for j <- i+1 to n-1:
      if absolute value of Numbers[j] - Numbers[i] == 10 && j - 1 > farthest:
        farthest = j - i
        farthest1 = i
        farthest2 = j

  return [success1, success2, farthest1, farthest2]
```

b) This is the brute-force solution based on iterations to find out the two groups, first iteration finds the two successive number with the largest difference, and the second iteration finds the two farthest numbers with their difference being 10

iii) time complexity is the O(N^2). The first loop takes the O(N), the second loop takes the O(N^2). Therefore the overall runtime is O(N^2)

iv) The stack growth is constant since this algorithm is not recursive and uses a constant number of variables, the stack usage is constant, and does not increase with respect to the input size.