Name: Anurag AGARWAL

Student ID: 40232644

Part 1 – Question 1

A) PSEUDOCODE:

```
{ //Swap function
Initialize n to length of array.
Repeat steps for i = 0 to (n-1)-1
if (first element of array == number)
    first element of array  = second element of array
    second element of array  = number;
    i += 1}
{   //In the main function
   print prompt "input an array of integers"
        Take an array of integers from the user.
   print prompt "input a number to swap"
        Take a number from the user.
   Send an array and a number to the swap function
   Print the result array to the user.
}
```

B) Time Complexity: O(N) - As it uses only one loop.

C) Space Complexity: O(1) - The variable used is one.

Part 1 – Question 2

A) PSEUDOCODE:

```
Prompt user for Input a String
    Take and store a string
Repeat steps for i = 0 to less than length of string
Create a hashmap countmap (character, integer)
Initialise currentchar to i'th character of string
Initialise currentcount to 0
Initialise countmap with currentchar and current count
Repeat steps for i = 0 to less than length of string
Check If i’th character of string is vowel
        Append it to vowelbuilder string
Else check if i’th character of string is consonant and countmap currentchar has value 1
        Append it to consonantbuilder string
Else
        Check if hashset appendedcharset does not contain currentchar
        Append it to repeatcharbuilder string and hashset appendedcharset
Concat consonantbuilder, repeatcharbuilder and vowelbuilder and store to result
print result String
```

B) Time Complexity: O(N)

C) Space Complexity: O(N)

<u>Part 1 – Question 3</u>

A) PSEUDOCODE:

```
Create method to check if elements are tetradic checkTetradic() which returns true or false.
Take in element as input, convert it to string and check if its reverse is same, and also check if it contains 0,1,8.
To find consecutive tetradic elements with largest difference, we iterate through the array.
Initialize max_differnece = -1.
While iterating through elements keep checking if element is tetradic or not using checkTetradic().
Repeat For index =0 to arr.length
if  checkTetradic (arr[index])==false or  is_tetradic(arr[index + 1]==false)
if arr[index + 1] - arr[index] > max_difference then max_difference= arr[index + 1] - arr[index];
answer = (arr[index], arr[index + 1])
 end of for loop
return answer
create an empty hashset which stores tetradic as a key, and it's the smallest index as value
Set<Integer, Integer> number_index = new HashSet<>
Iterate through array
instantiate two values, max_difference = arr[i] + difference, min_difference = arr[i] – difference
instantiate smallest_index = Double.POSITIVE_INFINITY;


if(max_difference in number_index) then smallest_index=
min(smallest_index, number_index.getKey(max_difference)

if(min_difference in number_index) then smallest_index =
min(smallest_index, number_index .getKey(min_difference)

if(smallest_index != Double.POSITIVE_INFINITY) then ans = (smallest_index, index)

return ans
```

B) Motive: Trying to keep the complexity as least as possible and find the required answers.

C) step 3 to 10 -> O(n) as we are iterating through loop once

step 11 to 18, we iterate through array and also while iteration we also check in our hashset if values are present or not at step 15 and 16, hence O(n^2)

Hence max time complexity = O(n^2)

D) Since hash look ups are constant time operations and we are iterating array once and performs only hash look ups, Also consecutive functions iterates once and performs constant time operations.