Programming and Problem Solving
Assignment I : Part I

Rahul Patil
40166394

Q1.

a)

```
array <- [1, 3, 2, 7, 2, 4, 2]
p <- -1
itr <- 0
value <- 2
while( itr < length of array ) :
        if ( arr[itr] == value) :
                p <- p + 1
                swap ( arr[itr] , arr[p] )
                itr <- itr + 1
        else :
                itr <- itr + 1


function swap( array, i, j ) :
        temp <- array[i]
        array[i] <- array[j]
        array[j] <- temp
```

b) Time complexity of the algorithm is O(n) as the iterator within the while loop traverses the entire array once and swaps values with a pointer if a match with the desired value is found.

c) Space complexity of the algorithm is O(1) as the only space used by the algorithm is used by the temporary variable for the in place swap process into the array.

Q2.

a)

```
s <- “assignment1”
StringBuilder sb <- new StringBuilder(s)
for i from 1 to sb.length - 1 :
       swap (sb, i - 1, i + 1)

swap( sb, i, j ) :
       temp <- char at i of sb
       char at i of sb <- char at j of sb
       char at j of sb <- temp
```

b) Time complexity of the algorithm is O(n) as iterator i traverses the length of the string excluding first and last character swapping (i -1)th character with the (i+1)th character..

c) Space complexity of the algorithm is O(1) as the only space used by the algorithm is used by the temporary variable for the in place swap process into the StringBuilder string.

Q3.

i)

```
arr <- [ 20, 52, 400, 3, 30, 70, 72, 47, 28, 38, 41, 53, 20 ]
p <- 0
itr <- 0

lastP <- arr.length - 1
farthestRes <- “”
consecutiveRes <- “”

elementsInBetween <- MIN_VALUE
maxCons <- MIN_VALUE

while ( itr < arr.length and lastP >= 0 and p < arr.length - 1 ) {

    // getting farthest
    diff -> Math.abs(digitsSum(arr[itr]) - digitsSum(arr[lastP]));
    curr <- lastP - itr - 1;

       if ( diff == 1 and curr > elementsInBetwn ) :
         elementsInBetwn <- curr
         farthest <- arr[itr++]+" "+arr[lastP--]
```

```
        else :
            itr <- itr + 1;
             lastP <- lastP - 1;


        // getting consecutive
        cons <- digitsSum(arr[p + 1]) - digitsSum(arr[p]);
        if (cons > maxCons) :
            maxCons <- cons;
            consecutive <- arr[p]+" "+arr[p+1];

        p <- p + 1;
    }

    digitsSum( n ) :
           sum <- 0
           while( n != 0 ) :
                  sum <- sum + n / 10
                  n <- n + n / 10
           return sum
```

ii) The algorithm in terms of computing both the results is efficient as it provides both the results in a single iteration whereas two separate loops could have been used to traverse the array twice in order to get the results separately.

iii) Time complexity of the algorithm is O(n) as both the results are computed in a single array length iteration.

iv) Stack growth of the algorithm can be initially filled up by the temporary variable assignments. The while loop where the iterator traverses the length of the array internally to ensure that computations are done for the farthest elements with difference of digits sum as 1 and two consecutive elements with the largest difference of their digits sum in order. Additionally a function call to the digitsSum is occasionally called for the temporary results where the sum is returned. Space complexity of the algorithm is O(1) constant as the results are computed over iteration into two variables.

****

programs for the respective questions

1.

```
private static void swap(int[] arr, int i, int j) {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

public static void main(String[] args) {
    int[] arr = {1, 3, 2, 7, 2, 4, 2};
    int p = -1;
    int itr = 0;
    int m = 2;

    while(itr < arr.length) {
        if(arr[itr] == m) {
            swap(arr, itr++, ++p);
        }
        else{
            itr++;
        }
    }
    for(int i : arr) {
        System.out.println(i);
    }
}
```

2.

```
public static void swap(StringBuilder sb, int i, int j) {
    char temp = sb.charAt(i);
    sb.setCharAt(i, sb.charAt(j));
    sb.setCharAt(j, temp);
}

public static void main(String[] args) {
    String s = "assignment1";
    StringBuilder sb = new StringBuilder(s);
```

```
        for(int i = 1; i < sb.length() - 1; i++) {
            swap(sb, i - 1, i + 1);
        }
        System.out.println(sb);
    }
```

3.

```
    private static int digitsSum(int n) {
        // System.out.println(n);
        int sum = 0;
        while(n != 0) {
            sum += n % 10;
            n = n / 10;
        }
        return sum;
    }

       public static void main(String[] args) {
        int[] arr = {20, 52, 400, 3, 30, 70, 72, 47, 28, 38, 41, 53, 20};
        int p = 0;
        int itr = 0;

        int lastP = arr.length - 1;
        String farthest = "";
        String consecutive = "";
        int elementsInBetwn = -(int)1e9;
        int maxCons = -(int)1e9;
        while(itr < arr.length && lastP >= 0 && p < arr.length - 1){

            // getting farthest
            int diff = Math.abs(digitsSum(arr[itr]) - digitsSum(arr[lastP]));
            int curr = lastP - itr - 1;

            if(diff == 1 && curr > elementsInBetwn) {
                elementsInBetwn = curr;
                farthest = arr[itr++]+" "+arr[lastP--];
            }
            else {
                itr++; lastP--;
            }
        // getting consecutive
        int cons = digitsSum(arr[p + 1]) - digitsSum(arr[p]);
        if(cons > maxCons) {
            maxCons = cons;
            consecutive = arr[p]+" "+arr[p+1];
        }
        p++;
    }

    System.out.println(farthest);
    System.out.println(consecutive);
```