Assignment 1 Part I

Name: Haochen Zou Student ID: 40158179

Question 1

1. **Question**:

   Given an array of integers of any size, n≥1, and a number m, develop an algorithm as a pseudo code (not a program!) that would move all numbers with value m to beginning of the array keeping order of remaining elements in the array same. You must perform this operation in place i.e., without creating an additional array and keeping the number of operations as small as possible. For example, given an array [1, 3, 2, 7, 2, 4, 2], and a number 2, the algorithm will return [2, 2, 2, 1, 3, 7, 4]. Finally, your algorithm must not use any auxiliary/additional storage to perform what is needed.

   Solution:

   For a given array A = [1, 2, ……, n] with n elements, n ≥ 1, and a number m.

   MOVE-TO-BEGINNING (A, n, m)

   x ← A [i];

   for i ← 1 to n

   if x ≠ m

   then i ← i + 1;

   else if i = 1

   then i ← i + 1;

   else for i ← 2 to i

   exchange A [i - 1] ↔ A [i];

   return A [1, 2, ……, n];

2. **Question**:

   What is the time complexity of your algorithm, in terms of Big-O?

   Solution:

   x ← A [i]; $O(1)$

   for i ← 1 to n $O(n)$

   if x ≠ m

   then i ← i + 1; $O(1)$

   else if i = 1

   then i ← i + 1; $O(1)$

   else for i ← 2 to i

   exchange A [i - 1] ↔ A [i]; $O(n^2)$

   return A [1, 2, ……, n];

   Overall

   T (n) = $O(1) + O(n) + O(1) + O(1) + O(n^2) = O(n^2)$

3. **Question**:

   What is the space complexity of your algorithm, in terms of Big-O?

   Solution:

   O(n)

Question 2

1. **Question**:
   Given a string of random length and random contents of characters, that do not include special characters, write an algorithm, using pseudo code that will swap every (i+1)th character with the (i-1)th character, starting from the second character. For instance, given a string "assignment1" the algorithm should return the string: "signment1sa".
   Solution:
   For a given string S = $S_1 S_2 \dots \dots S_n$ with n characters, n ≥ 3.
   SWAP-CHARACTER (S)
   for i ← 2 to n – 1
   x ← S_{i-1};
   $S_{i-1} \leftarrow S_{i+1}$;
   $S_{i+1} \leftarrow S_{i-1}$;
   return S = $S_1 S_2 \dots \dots S_n$;
2. **Question**:
   What is the time complexity of your algorithm, in terms of Big-O?
   Solution:
   $O(n)$
3. **Question**:
   What is the space complexity of your algorithm, in terms of Big-O?
   Solution:
   $O(1)$

Question 3

1. **Question**:
   Develop a well-documented pseudo code that finds two farthest elements in the array with the difference of their digit sum equal to 1, and two consecutive elements with the largest difference in their digit sum. The code must display values and indices of these elements. For instance, given the following array [20, 52, 400, 3, 30, 70, 72, 47, 28, 38, 41, 53, 20] your code should find and display something like the following (notice that this is just an example. Your solution must not refer to this example.) Two farthest elements with difference of their digit sum equal to 1 are 52 and 53 which have 9 elements between them. Two consecutive elements with the largest difference in their digit sum are 20 and 52. In case of multiple occurrences of the pairs with farthest distance or largest difference, your code must display the first found pair.
   Solution:
   For a given array A = [1, 2, ……, n] with n elements, n ≥ 1.
   FARTHEST-CONSECUTIVE(A)
   //the minimum and maximum difference of the difference between 2 elements
   min ← 0;
   max ← 0;

for i ← 1 to n-1
    // count for the minimum difference
    if |A [i + 1] – A [i]| < |A [min + 1] – A [min]|
    then min ← i;
    // count for the maximum difference
    if |A [i+1] – Arr[i]| > |A [max + 1] – A [max]|
    then max ← i;
return A [min] and A [min + 1];
return A [max] and A [max + 1];

2. **Question**:
   Briefly justify the motive(s) behind your design.
   Solution:
   We iterate through all elements in the array and get the difference between the element and the next element in the array. If the difference is less than the current minimum difference, then the minimum difference is the new difference. If the difference is greater than the current maximum difference, the maximum difference is the new difference.
3. **Question**:
   What is the time complexity of your solution? You must specify such complexity using the Big-O notation. Explain clearly how you obtained such complexity.
   Solution:
   FARTHEST-CONSECUTIVE(A)
   //the minimum and maximum difference of the difference between 2 elements
   min ← 0; $O(1)$
   max ← 0; $O(1)$
   for i ← 1 to n-1 $O(n)$
       // count for the minimum difference
       if |A [i + 1] – A [i]| < |A [min + 1] – A [min]|
       then min ← i; $O(1)$
       // count for the maximum difference
       if |A [i+1] – Arr[i]| > |A [max + 1] – A [max]|
       then max ← i; $O(1)$
   return A [min] and A [min + 1];
   return A [max] and A [max + 1];

$$T(n) = O(1) + O(1) + O(n) + O(1) + O(1) = O(n)$$

4. **Question**:
   What is the maximum size of stack growth of your algorithm? Explain clearly
   Solution:
   Variable memory allocation occurs at the time of definition, the variable min and max definition is outside the for loop, so the space does not increase with the increasement of the number of elements in the array, the space complexity is $O(1)$, the maximum size of stack growth of algorithm is constant.