Assignment 2 Part 1

Name: Haochen Zou, Student ID: 40158179

Question 1

a) **<u>Question:</u>**

Given an array of integers of any size, n ≥ 1, and an integer m, write an algorithm as a pseudo code that would find all the flipped pairs from the given array. A flipped pair is governed by the equation b = a + m. For instance, given [1, 3, -5, 9, -2, -4, 2, 7, 4, 6], and m=3 the algorithm will return [1, 4] and [3, 6], [-5, -2], and [9, 2]. You must only print pairs where index of a is less than that of b.

<u>Solution:</u>

For a given array A = [1, 2, 3, ……, n] with n elements, n ≥ 1, and an integer m.

```
FIND-FLIPPED-PAIRS (A, n, m)
x ← A[i];
for i ← 1 to n
    y ← A[j];
    for j ← 1 to n
        if y = x + m
            return x, y;
        else
            continue;
```

b) **<u>Question:</u>**

Find a simple solution with time complexity of $O(n^2)$

Solution:

```
x ← A[i];                  $O(1)$
for i ← 1 to n             $O(n)$
    y ← A[j];              $O(1)$
    for j ← 1 to n         $O(n^2)$
        if y = x + m
            return x, y;
```

Overall

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

c) **<u>Question:</u>**

Can this task be performed in O(nlogn) or O(logn)? If yes, please provide the pseudo code for your approach. If not, explain why it is not possible.

<u>Solution:</u>

We can find a method performed in $O(nlogn)$

For a given array A = [1, 2, 3, ……, n] with n elements, n ≥ 1, and an integer m.

```
FIND-FLIPPED-PAIRS-IN-NLOGN (A, n, m)
MERGE-SORT (A, 1, n)
if 1 < n
    mid ← (1 + n)/2;
    MERGE-SORT (A, 1, mid);
    MERGE-SORT (A, mid, n);
Merge (A, 1, mid, r)              $O(nlogn)$
```

```
x ← A[i];
for i ← 1 to n                          $O(n)$
    if exist x + n
        return x, x + n;                $O(1)$
    else
        continue;
```

Overall

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

We cannot find a method performed in $O(logn)$.

Because we need to at least scan for checking all the n elements in the given array which will cost $O(n)$. Therefore, we cannot solve this problem by a method less than $O(n)$.

d) <u>**Question:**</u>

Is a solution in O(n) even a possibility? If yes, please provide the pseudo code for your approach. If not, explain why it is not possible.

<u>**Solution:**</u>

We can find a solution in $O(n)$.

For a given array A = [1, 2, 3, ……, n] with n elements, n ≥ 1, and an integer m.

```
FIND-FLIPPED-PAIRS-IN-N (A, n, m)
COUNTING-SORT(A)
for i ← 0 to k do
    c[i] = 0;
for j ← 0 to n do
    c[A[j]] = c[A[j]] + 1;
for i ← 1 to k do
    c[i] = c[i] + c[i-1];
for j ← n-1 downto 0 do
    B[ c[A[j]]-1 ] = A[j];
    c[A[j]] = c[A[j]] – 1;              $O(n)$
x ← A[i];
for i ← 1 to n                          $O(n)$
    if exist x + n
        return x, x + n;                $O(1)$
    else
        continue;
```

Overall

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

Question 2

Given a collection of n numbers, write an algorithm, using pseudo code that will output all possible combinations of r numbers that sum up to a prime number. For instance, given {1,2,3,4,5}, algorithm must output (1,2), (1,4), (2,3), (2,5), and (3,4) for r = 2.

a) <u>**Question:**</u>

Write an iterative as well as a recursive solution for this problem.

<u>**Solution:**</u>

For a given array A = [1, 2, 3, ……, n] with n elements, n ≥ 1, and an integer r.

```
FIND-SUM-UP-PRIME-NUMBER(A, A', start, end, index, r)
    if index = r
        for j ← 1 to r
            A'[j] ← A[j];
    for i ← start to end and end – i + 1 ≥ r – index
        A'[index] = A[i]
        FIND-SUM-UP-PRIME-NUMBER(A, A', start+1, end, index+1, r);
for k ← A'[start] to A'[end]
     sum = sum + k;
if sum ≤ 3
     return sum > 1;
for p ← 2 to sum
     if sum mod i = 0
         continue;
return sum;
```

b) **<u>Question:</u>**

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

<u>Solution:</u>

Time complexity of the algorithm is $O(n^2)$.

c) **<u>Question:</u>**

What changes will you make to the algorithm to output all permutations instead. The output for the same input will now contain (1,2) as well as (2,1).

<u>Solution:</u>

First, pre-sort the array with any kind of sorting algorithm, then the algorithm starts to process the sorted array from the start element to the end element. Therefore, no repeated element pairs will appear since after processing the element, we will not analyze it again later. All the element after will be greater than the previous one.

Question 3

<u>Question:</u>

Develop a well-documented pseudo code that inserts a duplicate node as the left child of the node. Is it guaranteed to have the resulting tree as a balanced tree (BT) if the input was a BT?

<u>Solution:</u>

For a given node set N= [1, 2, ……, n] with n nodes, n ≥ 1. Node set data = N[i], where i is 1 to n. Each node has left node left and right node right. A binary tree node has data, a pointer to left child and a pointer to right child.

```
INSERT-NODE(N, root, node)
BUILD-TREE(node)
    if node = null
        return node;
    // build subtree
    BUILD-TREE(node.left)
    BUILD-TREE(node.right)
    // duplicate the node to its left
    node.left.left = node.left;
    node.left = N[data];
```

It is guaranteed to have the resulting tree as a balanced tree if the input was a balanced tree.