Programming and Problem Solving
Assignment II : Part I

Rahul Patil
40166394

1.

a)

```
array <-  [ 1, 3, -5, 9, -2, -4, 2, 7, 4, 6 ]
m <- 3
a <- -inf
b <- -inf

for i <- 0 to length of array :
        a <- array[i]
        for j <- 0 to length of array :
                b <- array[j]
                if b != a :
                        if i < j and b = a + m :
                                print a b
```

b) $O(n^2)$ solution:

```
public class Q1 {

        public static void main(String[] args) {
                int[] arr = {1, 3, -5, 9, -2, -4, 2, 7, 4, 6};

                int m = 3;
                int a = -(int)1e9;
                int b = -(int)1e9;
                for(int i = 0; i< arr.length; i++) {
                        a = arr[i];
                        for(int j = 0; j < arr.length; j++) {
                                b = arr[j];
                                if(b != a) {
                                        if( i < j && b == a + m) {
                                                System.out.println(a+" "+b);
                                        }
                                }
                        }
                }
        }

}
```

c) No, As two loops will be required for calculating the possibility of each pair with the same array. Maybe it’s still possible with the same array sorting and inplace manipulation.

d) No.

2.

a. Iterative and Recursive pseudocode

Iterative Solution :

```
sum <- 0
flag <- false
index <- 0
array <- {1, 2, 3, 4, 5};
r <- 2;
n <- arr.length;
res <- array of length
for ( i <- 0 to length of array ) :
    add to res <- array[i]
    for ( j <- i + 1 to length of array ):
        index <- j
        add to res <- array[j]
        if (index is r - 1) :
            for (k <- 0 to r ) :
                sum <- sum + res at index k
            if ( sum == 0 or sum == 1) :
                flag <- true
            else :
                for ( k <- 2 to sum/2 ) :
                    if (sum % k == 0 ) :
                        flag <- true
                        break
                if (!flag) :
                    print res
    clear res array for next result
```

Recursive Solution :

```
checkIfPrime( number ) :
    flag  <- false
    if number == 0 or number == 1 :
        flag <- true
    else :
        for (k <- 2 ; k <= number / 2; k++) :
            if s % k == 0 :
                flag <- true
                break
    return flag

combinationSum ( int arr[], int data[], int start, int end, int index, int r ):
 sum <- 0;
 if (index == r) :
     for (j <- 0 to r) :
         sum = sum + data[j]

     checkIfPrime(sum);
     if ( !checkIfPrime(sum) ) :
            print data

         return;


     for ( i <- start to end ):
         data[index] <- arr[i]
         combinationSum(arr, data, i+1, end, index+1, r)


    main() :
            array <- {1, 2, 3, 4, 5}
            r <- 2
            n <- arr.length
            data <- array of length r
            combinationSum(array, data, 0, n-1, 0, r)
```

b. Time Complexity for the above algorithm runs in exponential
c. While getting any specific result, also adding the possible combinations of the result set with shuffling to the result and maintaining consistency for duplicates check.

3.

If the value for the incoming node is equal or less than the iterator node, then new node will be placed as a left child or right child in the other case

```
class Node {
    int value;
    Node left;
    Node right;

    Node(int value) {
        this.value = value;
        right = null;
        left = null;
    }
}
private Node addRecursive(Node current, int value) {
    if (current == null) {
        return new Node(value);
    }

    if (value <= current.value) {
        current.left = addRecursive(current.left, value);
    } else if (value > current.value) {
        current.right = addRecursive(current.right, value);
    } else {
        return current;
    }

    return current;
}
```