Assignment 2

Written by: Mukesh Kumar Angrish (Student ID - 40203596)

Part I.

1ab. Given an array of integers of any size, n ≥ 1, and an integer m, write an algorithm as a pseudo code (not a program!) 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. Find a simple solution with time complexity of O(n2).

A: And array of integers
n: length of the array A
m: integer which defines the relation of a flipped pair (a, b), where b = a + m.
S: set of all flipped pair. Initially S <- Ø

```
function findPairs(A, m):
        for i = 0 -> n do
                for j = i+1 -> n do
                        if A[i] is equal to A[i] + m do
                                S <- S ∪ (A[i], A[j])
                        endif
                endfor
        endfor
        return S
```

1c. 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.

This task cannot be performed in O(n log n) or O (log n) time. This is because we cannot sort/search the array in logarithmic time since for a flipped pair (a, b), the index of a needs to come before the index of b.

1d. 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.

This task cannot be performed in O(n) time. This is because for every pair (a, b), the numbers could exist anywhere in the Array and hence we need to traverse the array from index of a to the last index for finding such a pair. This will be repeated for every index of a.

2a. 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. Write an iterative as well as a recursive solution for this problem.

```
x: a number
function isPrime(x):
    if x <= 1 do
        return false
    for i <- 2 to √x do
        if  i % x == 0 do
            return false
        endif
    endfor
    return true

A: an array of numbers
r: a positive integer
function findPrimeCombinationsIterative(A, r):
    prevSolns = {}
    primeCombinations = {}
    foreach number in A do
        newSolns = {}
        soln = {number}
        newSolns = newSolns ∪ soln
        foreach prevSoln in prevSolns do
            if size(prevSoln) == r-1 do
                sum = 0
                foreach toAdd in prevSoln do
                    sum = sum + toAdd
                endfor
                sum = sum + number
                if isPrime(sum) do
                    soln = prevSoln ∪ number
                    primeCombinations = primeCombinations ∪ soln
                    newSolns = newSolns ∪ soln
                endif
            else if size(prevSoln) < r-1 do
                soln = prevSoln ∪ number
                if soln prevSoln do
                    newSolns = newSolns ∪ soln
                endif
            endif
        endfor
        prevSolns = prevSolns ∪ newSolns
    endfor
    return primeCombinations
```

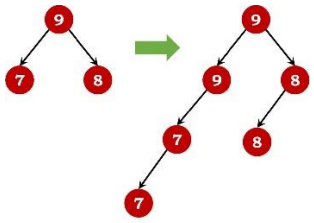
2b. What is the time complexity of your algorithm, in terms of Big-O?

The time complexity is of order $O(n^{r+1})$

2c. 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).

While generating new solutions from previous solution, we can store an inverse of the array (i.e. values from end to the start) and that way we can generate all possible permutations.

3. Develop a well-documented pseudo code that inserts a duplicate node as the left child of the node. A sample tree transformation is provided below.



Is it guaranteed to have the resulting tree as a balanced tree (BT) if the input was a BT?

```
T: a tree
root: the root node of the tree
function copyNodeToLeft(root):
        if root is null:
                return null
        endif
        copyNodeToLeft(left child of root)
        copyNodeToLeft(right child of root)
        newNode <- makeNode()
        newNode.data <- root.data
        newNode.left <- root.left
        root.left <- newNode
        return root
```

Part II.

The Sales.java program defines a Sales class with different attributes corresponding to the information regarding a sale. The Sales class also has multiple accessors and mutators in order to provide flexibility during object creation and modification. There are also methods to convert object to string and to compare different Sales cine objects

The SalesDatabase.java class allows filtering unique records from within each file in a directory tree and writes the filtered records to an output file. This is done by use of many custom methods made with respect to the Sales Database.

InvalidFileException.java implements a custom exception class that is thrown when a file is invalid.

EmptyFolderException.java implements a custom exception class that is thrown when a folder is empty.

DuplicateRecordException.java implements a custom exception class that is thrown when a duplicate record is found.