JOSEPH STEVEN MARTINEZ

3816842

Problem #1: (15 points)

You have two sorted lists, L_1 and L_2. You know the lengths of each list, L_1 has length N_1 and L_2 has length N_2.
(a) Design an efficient algorithm (only pseudocode) to output a sorted list $L_1 \cap L_2$ (the intersection of L_1 and L_2).

```
Function list_1, list_2
    i , j equal 0
    while i < length of list_1 and j < length of list_2
        if list_1[i] < list_2[j]
            increment i by 1
        else if list_2[j] < list_1[i]
            increment j by 1
        else
            print list_2[j]
            increment j by 1
    end
done
```

//Also review Java implementation

(b) If you know that $N_2 > N_1$. What is the running time complexity of your algorithm? Justify.

If n2 > n1 then the running time complexity is O(n) for the worst case.

If n1 or n2 is size 1 then the best case is O(1).

Problem #2: (40 points)

(a) Given an array of integers numbers, write a program in **Java** to find the stability index in it.
For an array A consisting n integers elements, index i is a stability index in A if
A[0] + A[1] + ... + A[i-1] = A[i+1] + A[i+2] + ... + A[n-1]; where 0 < i < n-1.
Similarly, 0 is an stability index if (A[1] + A[2] + ... + A[n-1]) = 0 and n-1 is an stability index
if (A[0] + A[1] + ... + A[n-2]) = 0
Example: Consider example, consider the array A = {0, -3, 5, -4, -2, 3, 1, 0}. The stability index found at index 0, 3 and 7.

Please review Java implementation

(b) What is the running time complexity of your program? Justify.

The running time complexity for this function worst case is O(n).

N1 + N2 = O(n)

Where the N1 is the process for summing the left stability index, and N2 is the running time for summing right stability index while reducing the left stability index and comparing the current right/left side sum NOT including the current index position.

Problem #3: (45 points: 25 + 15 + 5)

(a) Write a program in **Java** to implement a **recursive** search function
public static int terSearch(int A[], int l, int r, int x)
that returns location of x in a given **sorted array** A[l…r] is present, otherwise -1.
The **terSearch** search function, unlike the binary search, must consider two dividing points
int d1 = l + (r - l)/3
int d2 = d1 + (r - l)/3

Please review Java implementation

Running complexity time for this function is O(log n) where log has a base of 3.

(b.1) Write a program in **Java** to implement an efficient function
public static long exponentiation(long x, int n)
to calculate x
n
.
Please note that in your function you can use **only** the basic arithmetic operators (+, -, *, %, and /).
(b.2) What is the running time complexity of your function? Justify.

Pease review recursive implementation.

The running complexity time using the "recursive implementation" is O(n).

NOTE: There may be another implementation that may have a more efficient solution, this in one possible solution for this problem.