# Task 1

Write pseudocode for each of the three methods for the suffix-prefix identification problem described above. Analyze the running time of each of the three methods.

## Method One

Compare every number in the first list with every number in the second list.

### Pseudocode

methodOne(suffix[1...n], prefix[1...m]):  
   smallest = inf  
   suffixIndex = inf  
   prefixIndex = inf  
   for n to 1:  
       for 1 to m:  
           sum = suffix[n] + prefix[m]  
           if sum < smallest:  
               smallest = sum  
               suffixIndex = n  
               prefixIndex = m  
   return [smallest, suffixIndex, prefixIndex]

### Runtime Analysis

This code should run in O(nm) time:

## Method Two

Sort the first list and sort the second list and iterate the two lists carefully to identify the two numbers you are looking for.

This method uses a recursive routine similar to binary search to find the closest fit. That subroutine is called findClosestSumToZero().

### Pseudocode

methodTwo(prefix(1..n), suffix(1..m)):  
 bestValue <- infinity  
 prefixIndex <- -1  
 suffixIndex <- -1  
  
 for every value in prefix:  
 current <- findClosestSumToZero(suffix, 0, (m-1), value)  
 if abs(current) < abs(bestValue):  
 prefixIndex <- prefix.index(value)  
 suffixIndex <- suffix.index(current)  
 bestValue <- abs(current + value)  
 return bestValue, prefix[prefixIndex], suffix[suffixIndex]

findClosestSumToZero(A[1..a], low, high, k):  
 find = -1 \* k  
 if high – low == 1:  
 if abs(A[low] + k) > abs(A[high] + k):  
 return A[high]  
 else:  
 return A[low]  
 if high – low == 0:  
 return A[high]  
 else:  
 mid = floor((high+low)/2)  
 if A[mid] < find:  
 return findClosestSumToZero(A, mid, high, k)  
 else if A[mid] > find:  
 return findClosestSumToZero(A, low, mid, k)  
 else:  
 return A[mid]

### Runtime Analysis

This summation translates to Θ(nlogm) time.

## Method Three

Combine the first list with the negative of the second list, and sort this combined list, keeping track of which list the numbers come from, and noticing that the two numbers we want are adjacent to each other.

### Pseudocode

methodThree(prefix[1...n], suffix[1...m])

   smallest = inf

   first = inf

   second = inf

   int x = n+m

   temp = []

   for 1 to m

   temp.append([suffix[m]\*-1,2] )

   for 1 to n

   temp.append([prefix[n],1])

   temp.sort(key=lamda x:x[0])

   for 1 to x

   if temp[x][1] != temp[x+1][1]

   if(temp[x] < 0 && temp[x+1] > 0 || temp[x] > 0 && temp[x+1] < 0)

   temp[x] + temp[x+1] < abs(smallest)

   smallest = abs(temp[x] + temp[x+1])

   first = temp[x][0]

   second = temp[x+1][0]

   else

   temp[x] - temp[x+1] < abs(smallest)

   smallest = abs(temp[x] - temp[x+1])

   first = temp[x][0]

   second = temp[x+1][0]

   return [smallest, first, second]

### Runtime Analysis

This results in a runtime of

# Task Two