Question 1

You are given an integer m (1 ≤ m ≥ 1 000 000) and two non-empty, zero-indexed arrays A and B of n integers, a0, a1, ... , an-1 and b0, b1, ... , bn−1 respectively (0 ≤ ai, bi ≤ m). The goal is to check whether there is a swap operation which can be performed on these arrays in such a way that the sum of elements in array A equals the sum of elements in array B after the swap. By swap operation we mean picking one element from array A and one element from array B and exchanging them.

Expectations

1. Solution to the problem

Implementation in python programming :

def swap\_pairs(A, B):

sum\_A = sum(A)

sum\_B = sum(B)

diff = sum\_A - sum\_B

if diff % 2 != 0: # If the difference is odd, no swap can make the sums equal

return False

target\_diff = diff // 2

set\_B = set(B) #to extract unique elements

for a in A:

b\_candidate = a +target\_diff

if b\_candidate in set\_B:

# Swap found

return True

# No swap found

return False

2. Complexity of the algorithm both time and space

Time Complexity: The sum of both arrays is calculated in O(n) time. The subsequent loop to check for a valid swap also takes O(n) time. Therefore, the overall time complexity is O(n).

Space Complexity: The additional space used is only for the set\_B, which stores the elements of array B. In the worst case, the space complexity is O(n) as it may need to store all elements of array B.

So, the solution has a time complexity of O(n) and a space complexity of O(n).