

Date : 25th - 09 - 2020

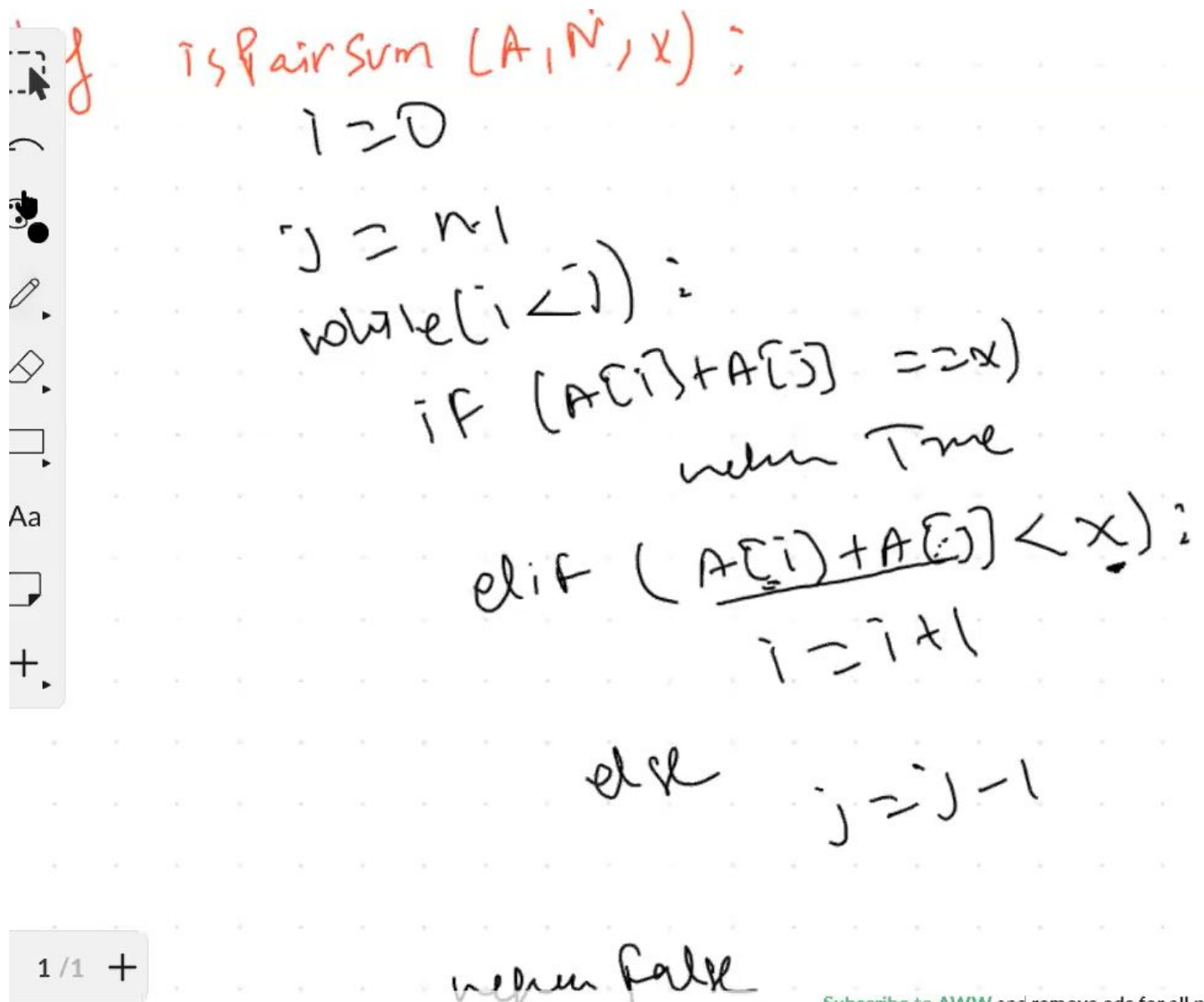
Morning Session : 9am - 11.00 PM

By ~ Rohan Kumar

Topics: two pointer Technique

Two Pointers Technique

The two pointer technique is a useful tool to utilize when searching for pairs in a sorted array. Although not it's only use case, when used this technique can save both time and space complexity.



```
bool isPairSum (A, N, x) :  
    i = 0  
    j = n-1  
    while (i < j) :  
        if (A[i] + A[j] == x)  
            return True  
        elif (A[i] + A[j] < x) :  
            i = i + 1  
        else  
            j = j - 1  
    return False
```

The time complexity of this solution is **$O(n^2)$** .

Now let's see how the two-pointer technique works. We take two pointers, one representing the first element and other representing the last element of the array, and then we add the values kept at both the pointers. If their sum is smaller than X then we shift the left pointer to right or if their sum is greater than X then we shift the right pointer to left, in order to get closer to the sum. We keep moving the pointers until we get the sum as X.

Find the closest pair

```
import sys

def printClosest(arr1,arr2,m,n,x):
    diff = sys.maxsize
    l = 0
    r = n - 1
    while l < m and r >= 0:
        if abs(arr1[l] + arr2[r] - x) < diff:
            l_index = l
            r_index = r
            diff = abs(arr1[l] + arr2[r] - x)
        if arr1[l] + arr2[r] > x:
            r = r - 1
        else:
            l = l + 1
    print(arr1[l_index],arr2[r_index])

printClosest([2,4,6,8],[3,5,7,9],4,4,12)
```

Find all triplets with zero sum

- 1) Sort in ascending order
- 2) Traverse from start to end

```
: def findTriplets(arr):  
    arr.sort()  
    for i in range(len(arr)):  
        l = i + 1  
        r = len(arr) - 1  
        x = arr[i]  
        while l < r:  
            if x + arr[l] + arr[r] == 0:  
                print(x, arr[l], arr[r])  
                l = l + 1  
                r = r - 1  
            elif x + arr[l] + arr[r] < 0:  
                l = l + 1  
            else:  
                r = r - 1  
findTriplets([0, -1, 2, -3, 1])    #O(n^2)
```

```
-3 1 2  
-1 0 1
```

MCQ 1 :

1. Why do we use the two pointer technique?
 - a. For better time complexity
 - b. For better space complexity

Answer: A, for Better time Complexity

MCQ 2 :

2. Selection Sort time complexity

- a. $O(n*n)$
- b. $O(\log n)$
- c. $O(n \log n)$

Answer: 2, $O(n*n)$

MCQ 3 :

3. Insertion Sort time complexity

- a. $O(n*n)$
- b. $O(n \log n)$
- c. $O(\log n)$

Answer: A, $O(n * n)$

MCQ 4 :

4. Quicksort best case time complexity

- a. $O(\log n)$
- b. $O(n*n)$
- c. $O(n \log n)$

Answer: C, $O(n \log n)$