Double Helix (SPOJ)

In this problem we have 2 sorted arrays and can move through them in zig-zag manner using common elements as junction points to maximize the sum. That's why the name double helix.

Steps →

1) We will maintain 2 pointers to traverse both arrays.
2) Maintain 2 variables sum1 and sum2
3) When common element is found:
   - Pick the maximum of 2 sums and add it to the result.
   - Reset the sum counters
4) Continue until the end of arrays

Code →

```
def maxSumPath(arr1, arr2):
    i,j = 0,0
    sum1, sum2 = 0,0
    result=0
    while i<len(arr1) and j<len(arr2):
        if arr[i] < arr2[j]:
            sum1 += arr[i]
            i += 1
```

```
elif arr1[i] > arr2[j]:
    sum2 += arr2[j]
    j += 1
else
    result += max(sum1, sum2) + arr[i]
    sum1, sum2 = 0, 0
    i += 1
    j += 1
while i < len(arr1):
    sum1 += arr1[i]
    i += 1
while j < len(arr2):
    sum2 += arr2[j]
    j += 1
result += max(sum1, sum2)
return result
```

arr1 = [3, 5, 7, 9, 20, 25, 30, 40, 55, 56, 57, 60, 62]

arr2 = [1, 4, 7, 11, 14, 25, 44, 47, 55, 57, 100]

Step	i (arr1)	j (arr2)	sum1	sum2	Common?	Result
1	3	1	3	1	No	0
2	5	4	8	5	No	0
3	7	7	15	12	Yes	15 (`max(15,12) + 7`)
4	9	11	9	11	No	15
5	20	14	29	25	No	15
6	25	25	54	50	Yes	54 (`max(54,50) + 25`)
7	30	44	30	44	No	79
8	40	47	70	91	No	79
9	55	55	125	193	Yes	272 (`max(125,193) + 55`)
10	56	57	56	57	No	327
11	57	57	113	114	Yes	384 (`max(113,114) + 57`)
12	60	100	60	100	No	384
13	62	End	122	100	No	450 (`max(122,100)`)

Time complexity

Best Case(O(N+M))

If both sequences have no common elements, we just traverse both array once & take the sum of the larger one

* No switching between paths

Worst Case (O(N+M))

* If there are many common elements, we ~~test~~ keep resetting sum1 and sum2 at every common number

* Still, since we only scan each list one, the time complexity remains O(N+M)