Longest Subarray with given sum k (positives) :-

eg:- [2, 3, 5] . k = 5.

o/p -> 2 ({2, 3})

Approach-1 :- Hashing.

-> Assign a presummap where sum of the elements to the index are stored in a map.

For example to arr -> 2, 3, 5.

Hash map -> {(0, 2), (1, 5), (2, 8)}.

-> Traverse the arr and add the element to a variable s (initialized to 0).

-> Check if k-s (remaining sum) is present in the hashmap. If present check for the max length.

Code:-

```
def getLongestSubarray(a, k):
    n = len(a)
    preSumMap = {}
    s = 0
    maxLen = 0
    for i in range(n):
        s += a[i] # Calculate the prefix sum till i
        if s==k:
            maxLen = max(maxLen, i+1)
```

2024.01.21 13:51

(positives)

the elements

(2, 8) }.

ent to a

present

the

ix sum

```
# Calculate the sum of remaining part i.e., x
rem = s - k
# Calculate the length and update maxLen
if rem in preSumMap:
    length = i - preSumMap[rem]
    maxLen = max(maxLen, length)
# update the map
if s not in preSumMap:
    preSumMap[s] = i
return maxLen.
```

T.C → O(N) or O(N log N) depending upon the map data structure.

S.C → O(N) → map data structure.

Approach - 2 :- Two Pointer

→ Start left, right pointers at 0 index.

→ Take a variable s = 0. Add a[right] to s for every increment of right.

→ If s > k (target) then remove a[left] from s and increment left. Continue the process until s <= k.

→ Follow above two steps until right > n.

Eg :- 1 2 3 1 1 1 1 3 3 , k = 6

i, j ; len = 0

s = 1 < k ✓ j++ (j = 1)

s = 3 < k ✓ j = 2

s = 6 == k ✓ len = max(len, j - i + 1)

= max (0, 2 - 0 + 1)

len = 3

j = 3 , s = 7 < 6 ✗ . i++

s = 7 - 1 = 6 == k ✓

len = 3 .

j = 4 , s = 7 < 6 ✗ i++ .

s = 7 - 2 = 5 < k ✓ . i = 2

j = 5 , s = 6 == k len = max(3, 5 - 2 + 1)

len = 4

j = 6 , s = 7 < k ✗ , i = 3 .

s = 7 - 3 = 4 < k ✓

j = 7 , s = 4 + 3 = 7 < k ✗ , i = 4 .

s = 7 - 1 = 6 == k ✓ len = max(4, 7 - 4 + 1)

len = 4 .

j = 8 , s = 9 < k ✗ , i = 5 .

s = 9 - 1 = 8 < k ✗ , i = 6 .

s = 8 - 1 = 7 < k ✗ i = 7 .

2024.01.21 13:51

Two pointer

Code :-

def longest

i = 0

j = 0

n = len

maxLen

s = a

while

T.C

s = 7 - 1 - 6. ✓ i = 7, j = 8.

len = max(4, 8 - 7 + 1)

∴ len = 4

Two pointer can only be used for positive elements.

Code:-

```
def longestSubarraywithSumk(a, k):
    i = 0
    j = 0
    n = len(a)
    maxLen = 0
    s = a[0]
    while j < n:
        while i <= j and s > k:
            s -= a[i]
            i += 1
        if s == k:
            maxLen = max(maxLen, j - i + 1)
        j += 1
        if j < n:
            s += a[j].
```

return maxLen.

S.C → ~~O(n)~~ O(1).

T.C → ~~O(N + N)~~ O(2 * N)

→ Outer j loop moves to N-1, inner i loop runs upto j times. So it becomes O(2 * N) instead of O(N²).