Lecture 15 - Hashing II

(58) Given N array elements, count distinct elements in every window of size K

N [1 2 1 3 2 1 2 1 1 2 3] K=4

⇒ [3, 3, 3, 3, 2, 2, 2, 3]

N-K+1

Brute Force

```
ans = [ ]
for i in range(n-k+1):
    l = [ ]
    for j in range(k)
        if A[j+i] not in l:
            l.append(A[j+i])
    ans.append(len(l))
```

TC: O(N)

Using frequency mapping

```
for i in range(n-k+1):
    freq = {}
    for j in range(k):
        if A[j+i] not in freq:
            freq[A[j+i]] = 1
        else:
            freq[A[j+i]] += 1
    ans.append(len(freq))
```

TC : O(N)

Optimize space complexity in frequency mapping

```
freq = {}
n = len(arr)

# 0, 1, 2, ... k-1
for i in range(k):
    if arr[i] in freq:
        freq[arr[i]] += 1
    else:
        freq[arr[i]] = 1


ans = []
ans.append(len(freq))

# a1 .... ak ; remove a0 ; add ak ; so on
for end in range(k, n):
    if arr[end] in freq:
        freq[arr[end]] += 1
    else:
        freq[arr[end]] = 1

    element_to_remove = arr[end - k]
    freq[element_to_remove] -= 1

    if freq[element_to_remove] == 0:

        del freq[element_to_remove]
    ans.append(len(freq))
return ans
```

59 Given N array elements, check if there exists a pair (i, j) s.t. arr[i] + arr[j] == k

[1 2 5, 7, 3, 2, 1]

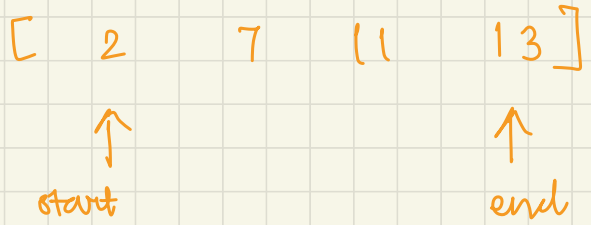
Brute Force

K = 12

```
for i in range(n)
    for j in range(i+1, n):
        if A[i] + A[j] == k:
            return True
return False
```

TC: $O(N^2)$

Using two-pointer approach ON THE SORTED ARRAY



(1) K = 9

start + end (15) > K ✗

end -= 1

start + end (13) > K → end -= 1

start + end (9) == K ✓

(2) K = 18

start + end (15) < K → start += 1

start + end (20) > K → end -= 1

start + end (18) == K ✓

TC : O(N log N)

Optimized

```
freq = {}
for i in range(n):
    if k - arr[i] in freq:
        return True
    else:
        if arr[i] not in freq:
            freq[arr[i]] = 1
return False
```

(60) given an array of elements A and a non-negative integer B. Check if there exists i and j s.t. $A[i] - A[j] = B$ $i \neq j$

```
hashmap = {}
for i in A:
    if A[i] in hashmap:
        hashmap[A[i]] += 1
    else:
        hashmap[A[i]] = 1

if len(A) < 2
    return 0

for i in range(N):
    a = A[i]
    b = A[i] + B

    if a != b:
        if b in hashmap:
            return 1
    else:
        if hashmap[b] > 1:
            return 1

return 0
```

case when B = 0 so num - num = 0

num's freq should be > 1

(61) Given N elements of an array, find the longest sequence which can be rearranged to form a sequence of increasing consecutive numbers

[100, 4, 200, 1, 2, 3] ⇒ [1, 2, 3, 4]

Brute Force

```
l.sort()
i = 0
while i < N:
    count = 1
    prev = arr[i]
    i += 1
    while arr[i] == prev + 1:
        prev += 1
        i += 1
        count += 1
    ans = max(ans, count)
```

TC: O(NlogN)

Using dict

0 1 2 3 4 5 6 7 8

[98 99 7 6 5 4 32 100 97]

{ 2 : 1
3 : 1
4 : 1
5 : 1
6 : 1
7 : 1
98 : 1
: }

```
freq = {}
for i in range(n):
    if A[i] in freq:
        freq[A[i]] += 1
    else:
        freq[A[i]] = 1
ans = 0
for i in range(n):
    if A[i]-1 in freq:
        continue
    else:
        val = arr[i]
        count = 0
        while val in freq:
            count += 1
            val += 1
        ans = max(ans, count)
```

TC: O(N)

(62) Given an array of integers A and target B. Find out the first continuous subarray which adds to B, otherwise return -1

[1, 2, 3, 4, 5]

ps [1 3 6 10 15] ⟹ [2, 3]

B = 5

Solution 1:

freq { 1 : 0

3 : 1

i = 2

6 - 5 = 1 in freq

A[0+1 : 2+1]

A[1 : 3]

[1 2 3 4 5]

```
N = len(A)
ps = [0]*N
ps[0] = A[0]

for i in range(N):
    ps[i] = ps[i-1] + A[i]

freq = {}
for i in range(N):
    if ps[i]-B in freq:
        return A[freq[ps[i]-B]+1 : i+1]

    elif ps[i]-B == 0:
        return A[: i+1]
    elif ps[i] not in freq:
        freq[ps[i]] = i

return [-1]
```

Solution 2 :

```
start_index = 0
end_index = 0
curr_sum = 0

while (start_index < N) and (end_index < N):

        if start_index > end_index:
            end_index = start_index
            curr_sum = A[end_index]

        if curr_sum < B:
            curr_sum += A[end_index]
            end_index += 1

        if curr_sum > B:
            curr_sum -= A[start_index]
            start_index += 1

        if B == curr_sum:
            return A[start_index : end_index]

return [-1]
```

(63) Given an array of N strings A. B contains a string of size 26 denoting the order. Return 1 iff given words are sorted lexicographically in order defined in B.

A = ["hello", "scaler", "interview"]

B = "adbhqrstijklmvw...."

h < s < i

same order as in A ⟹ return 1

```
hb = {}
for i in range(len(B)):
    hb[ B[i] ] = i

for i in range(1, len(A))
    prev = A[i-1]
    curr = A[i]

    j = 0
    flag = False
    while j < len(prev) and j < len(curr):
        if hb[prev[j]] < hb[curr[j]]:
            flag = True
            break
        elif hb[prev[j]] > hb[curr[j]]:
            return 0
        else:
            j += 1
    if not flag and len(prev) > len(curr):
        return 0
return 1
```