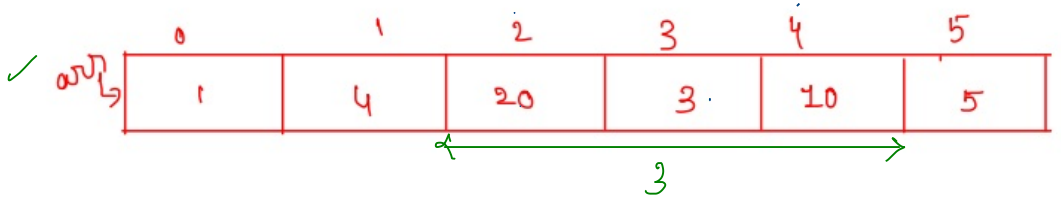
Model-2 [Variable Size SW]

1) Find is there any sub-array with the given sum k [return True/ False]

i/p



n = 6 ✓

k = 33 ✓

o/p

True ✓

API :-

↳ generate all possible sub arrays. ✓ ⟶ $\frac{n(n+1)}{2} = O(n^2)$

↳ check (O(1)) each subarray sum equals to k (or) not

↳ O(n) : finding sum

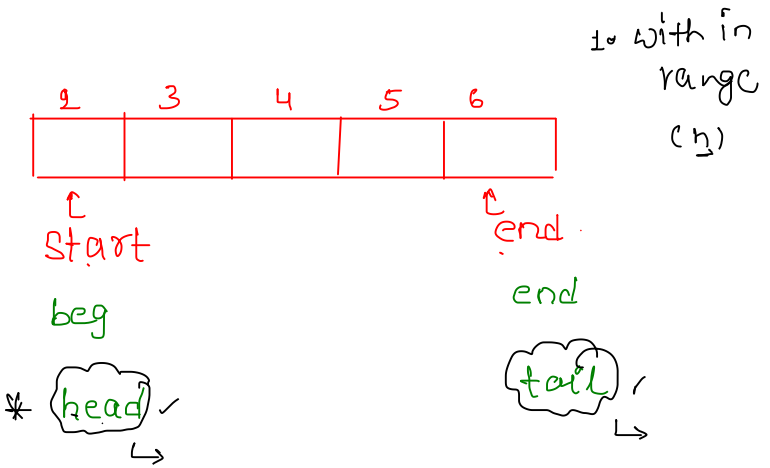
H/W

Total T.C : $O(n^2) \times O(n) = O(n^3)$

TLE

once, after identifying the problem belongs to variable size SW

then you need to use following structure:

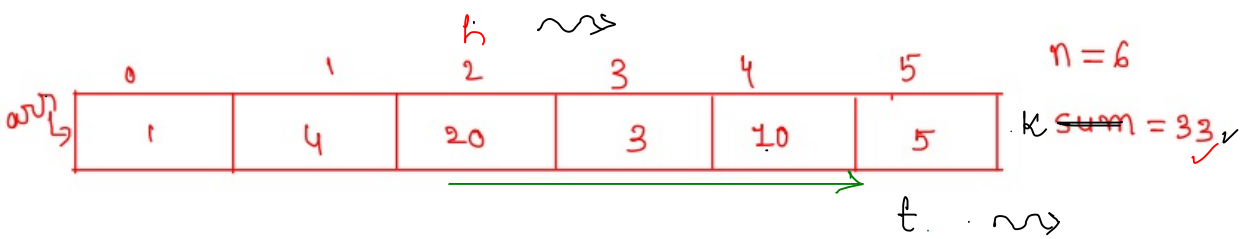



```
for(.....)
{
    while(.....)
    {
        ?

    }
}
```

-> some time we use simple variables
-> some times we need extra data strctures
like hashmap, hashset etc...

dict set

fixed



-> initially both head, tail points to first element
-> next tail will be moving, head will fixed at 1st element
-> after some time head will also moves
-> now, between head and tails the elements are considerd as part of window

↦ Window sum

WS = 0 ✓
= 0 + 1 = 1
= 1 + 4 = 5
= 5 + 20 = 25
= 25 + 3 = 28
= 28 + 10 = 38

38 − 1 = 37

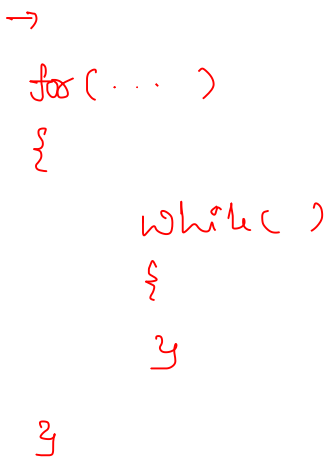
37 − 4 = 33

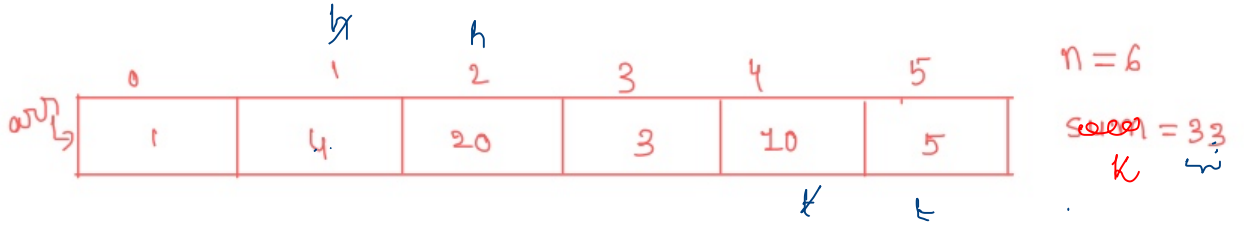
compare WS v/s k

c_1	c_2 ✓	c_3 ✓
WS == k ✓	WS < k	WS > k
Stop. return true	→ Add more eles to WS	→ Remove some eles from WS → order should be maintained

→ return false

```python
def has_subarray_with_sum(arr, k):
    tail = 0
    current_sum = 0

    for head in range(len(arr)):
        current_sum += arr[head]

        # Shrink the window until the current sum is less than or equal to k
        while current_sum > k and tail <= head:
            current_sum -= arr[tail]
            tail += 1

        # Check if we have found a subarray with sum equal to k
        if current_sum == k:
            return True

    return False
```



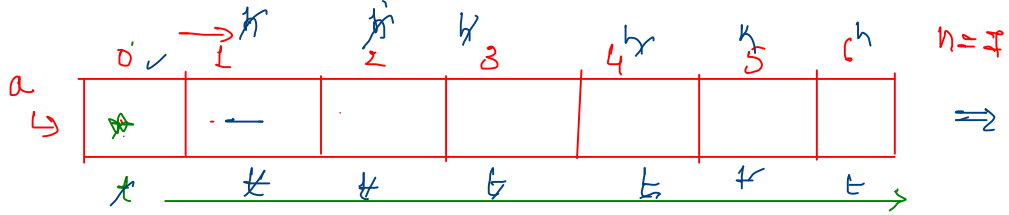



```
function fun(arr,k)
{
        head=0
        ws=0
    1.  for(tail=0;tail<n;tail++) →
        {
            1. ws=ws+arr[tail]
            2. while(ws>k and head<=tail) ✓
               {
                        ws=ws-arr[head] ✓
                        head++
               }
            → if(ws==k) ✓
                        return true ✓   → end
        }
        retrun false ✓
}
```

WS = 0 + 1 = 1 + 4 = 5 + 20 = 25 + 3 = 28 + 10 = 38 ✓

38 − 1 = 37

37 − 4 = 33



O(1) SC

```
function fun(arr,k)        → O(n) TC
{
        head=0
        ws=0
    L1  for(tail=0;tail<n;tail++)  →
        {
            ws=ws+arr[tail]
    X L2    while(ws>k and head<=tail)   → F
            {   T
                ws=ws-arr[head]
                head++
            }
            if(ws==k)
                return true
        }
        retrun false
}
```

O(n²)

Fail

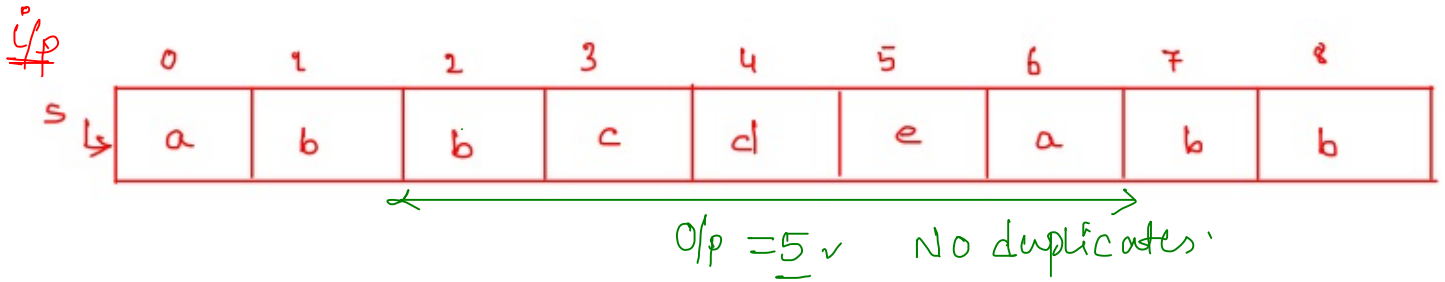
T T

L1 ws = ws + * ✓

L2 ✓

Always fails-

2) Find the size of largest sub-string which doesn't contains any repeated characters in given string



→ 1. variable sw.

2. for(. . .)
{
while(. . .)
{

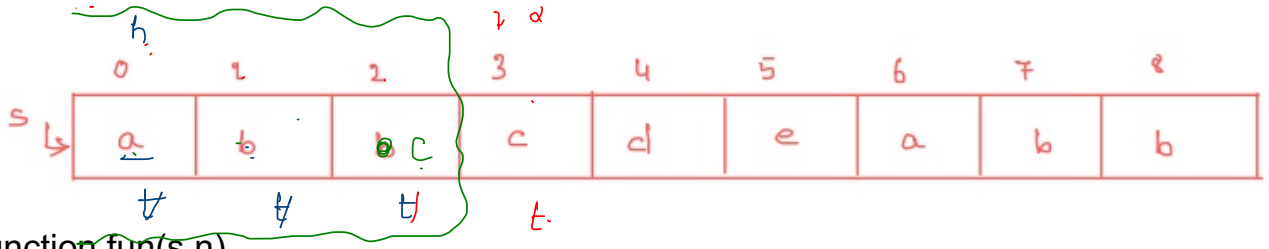
}
}

we should use map {}

3. Some how, we need to check characters should not repeat
∴ each character count at max-1

when

→ next class



n = 9

6:00

```
function fun(s,n)
{
        head=0
        map={}
        res=0
        for(tail=0;tail<n;tail++)
        {
                key=s[tail]
                while(key in map and map[key]>0)
                {
                        res=max(res,tail-head)
                        map[s[head]]=map[s[head]]-1
                        head++
                }
                if key in map
                {
                        map[key]++
                }
                else
                {
                        map[key]=1
                }
        }
        retrun max(res,tail-head)


}
```

key = abc

res = 0 ← o/p

3

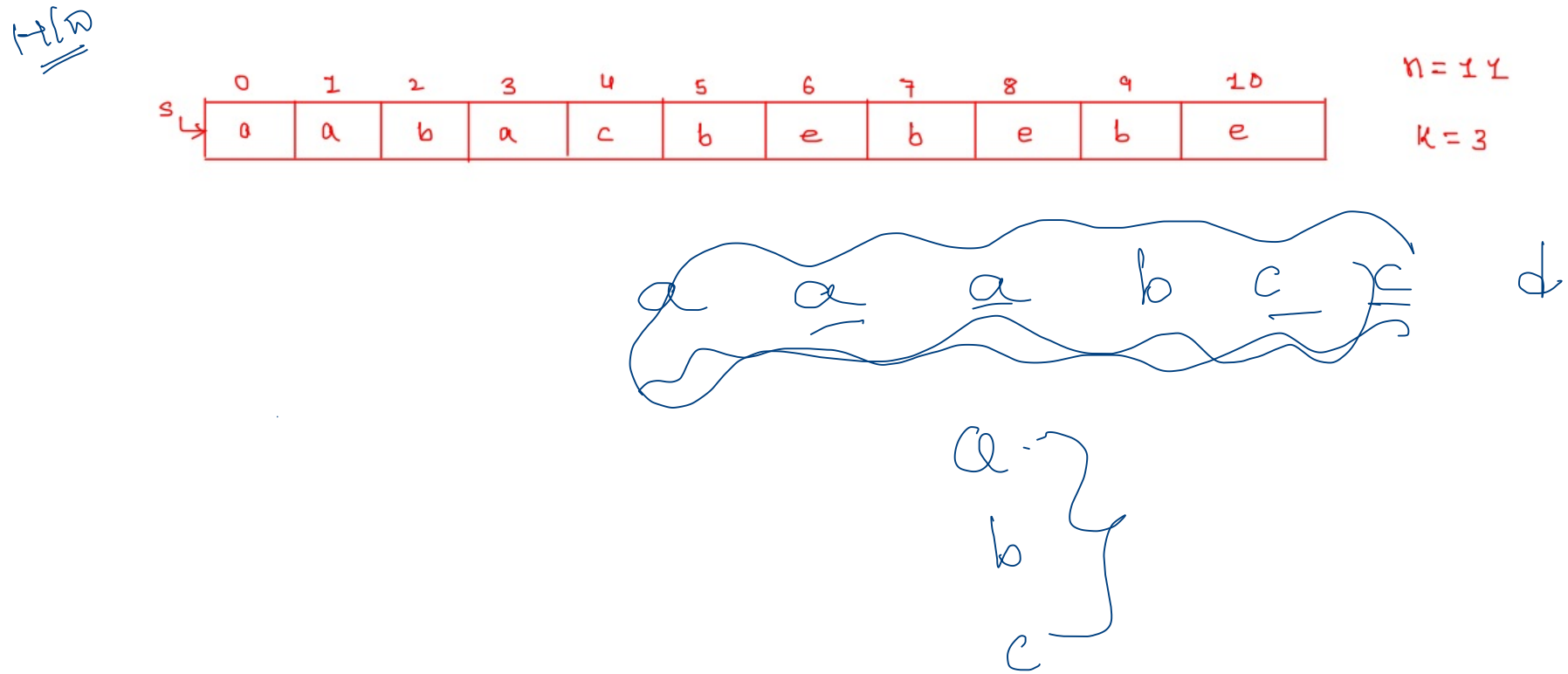
U1

map

key	value
a	1
b	1
c	1

```
function length_of_longest_substring(s):
{
    if len(s) == 0:
    {
        return 0
    }
    if len(s) == 1:
    {
        return 1
    }
    char_map = {}
    tail = 0
    res = 0
    for head in range(len(s)):
    {
        key = s[head]
        while key in char_map and char_map[key] > 0:
        {
            res = max(res, head - tail)
            char_map[s[tail]] -= 1
            tail += 1
        }
        char_map[key] = char_map.get(key, 0) + 1
    }
    return max(res, len(s) - tail)  # max to handle edge cases like "au"
}
```

3) Find the Longest Substring which contains K distinct / Unique characters



```
function length_of_longest_substring_k_distinct(s, k):
{
    if k == 0:
    {
        return -1
    }
    char_map = {}
    tail = 0
    res = -1
    for head in range(len(s)):
    {
        key = s[head]
        char_map[key] = char_map.get(key, 0) + 1
        while len(char_map) > k:
        {
            char_map[s[tail]] -= 1
            if char_map[s[tail]] == 0:
            {
                char_map.remove(s[tail])
            }
            tail += 1
        }
        if len(char_map) == k:
        {
            res = max(res, head - tail + 1)
        }
    }
    return res
}
```