Master Class - 3

Two Pointer Technique && Sliding Window Technique

- Venu Gopal

Two - Pointer Technique

Model - 1

2ptr's moves in opposite direction

Model - 2

2-ptr's, moves in same direction

Two Pointer [Model-1 : Moves in Opposite Direction]

$l = 0$

n : # of elements in array.

$r = n - 1$

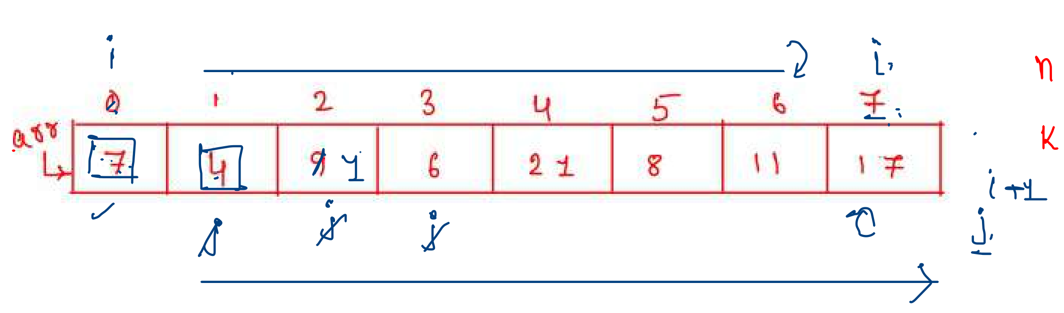
1) Find a pair whose sum is equal to k [a+b=k]

↳ 2-ele's

T/F·

Yes/No

i/p



n=8

K=16

Ap1:-

Brute Force

arr[i] + arr[j]

7 +

7,_9

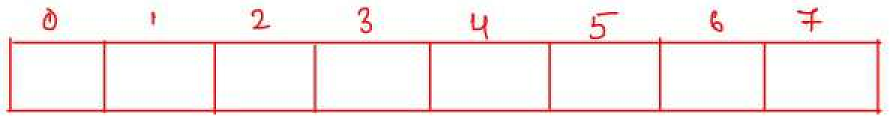
i : 0 →

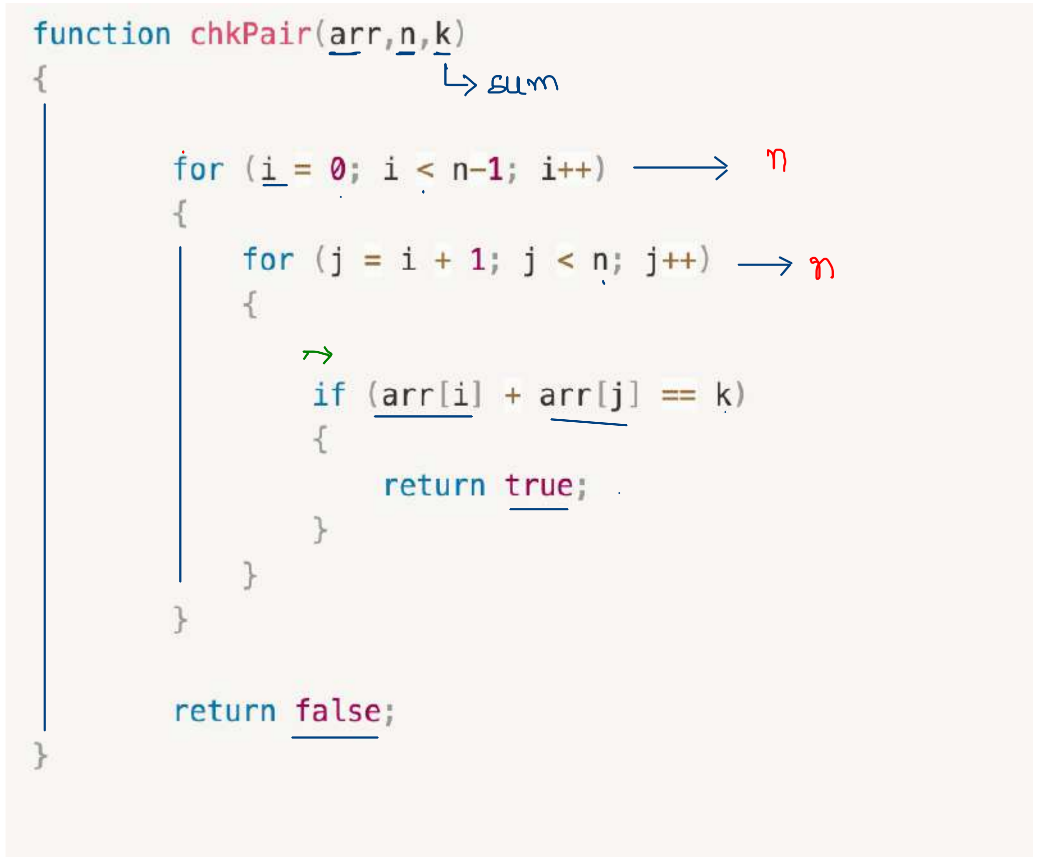
j : i+1

ATOBE

UD

7+7+2 = 16


0
1
2
3
4
5
6
7



```
function chkPair(arr,n,k)
{

        for (i = 0; i < n-1; i++)
        {
            for (j = i + 1; j < n; j++)
            {

                if (arr[i] + arr[j] == k)
                {
                    return true;
                }
            }
        }

        return false;
}
```

$${}^{n}C_{r} = \frac{n!}{(n-r)! \times r!}$$

$\Rightarrow O(n^2)$ T.C

How to Prove Time Complexity Mathematically :-

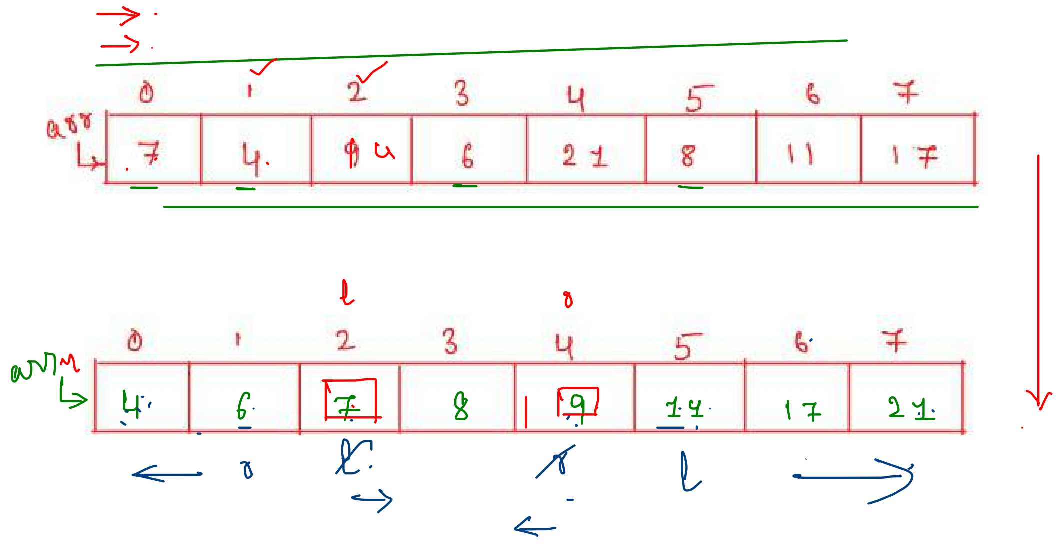
Any Two elements, whose sum is K

out of n elements, choose any 2 elements

$${}^{n}C_{\underset{\uparrow r}{2}} = \frac{n!}{(n-2)! \times 2!}$$

$$= \frac{n \times n-1 \times \cancel{(n-2)}!}{\cancel{(n-2)}! \times 2}$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} \Rightarrow O(n^2)$$



n = 8

k = 16

15

4 + 21 = 25 ✓

sum > k

25 v/s 16 ✓

sort()

l++

l = 1, r = 7

sum = 27

r-- ✓

l = 0, r = 6

sum = 21

sum = arr[l] + arr[r] v/s k

1. sum == k

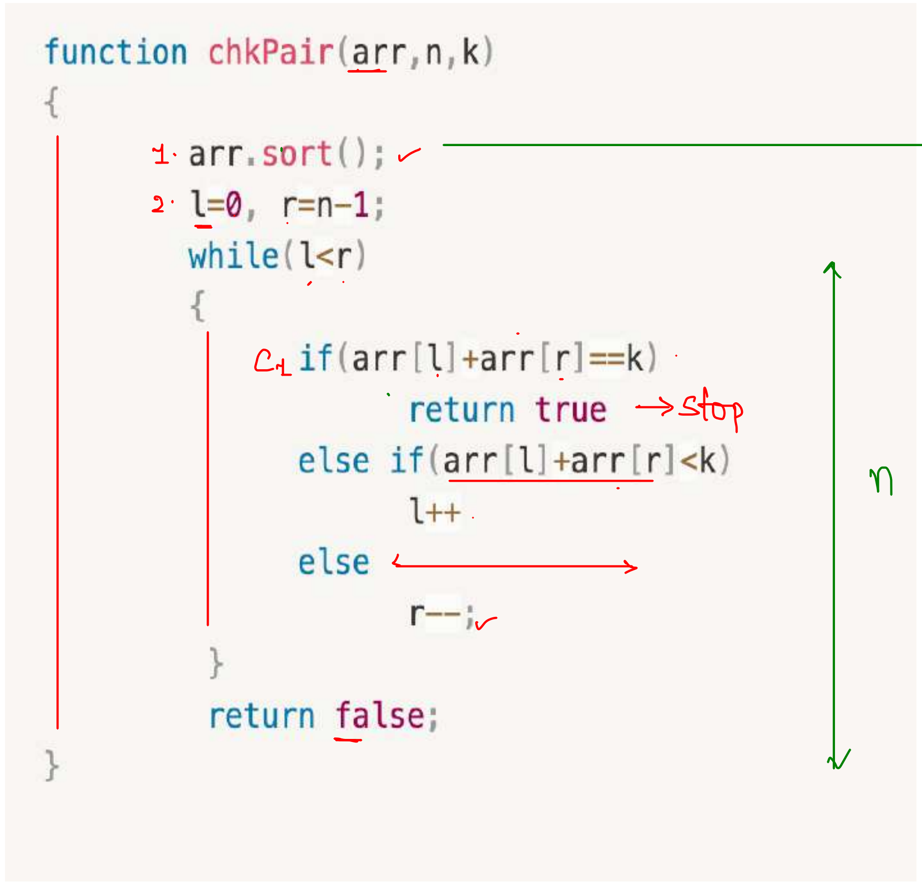
stop ✓

2. sum < k

l++ ✓

3. sum > k

r--



```
function chkPair(arr,n,k)
{
      1. arr.sort();
      2. l=0, r=n-1;
         while(l<r)
         {
           C1 if(arr[l]+arr[r]==k)
                   return true  → stop
              else if(arr[l]+arr[r]<k)
                   l++
              else
                   r--;
         }
         return false;                          n
}
```

randomised q.s ($O(n \cdot \log n)$)
↳ quick sort

$\Rightarrow \cancel{n} + n \cdot \log_2^n$

$O(n \cdot \log_2^n)$

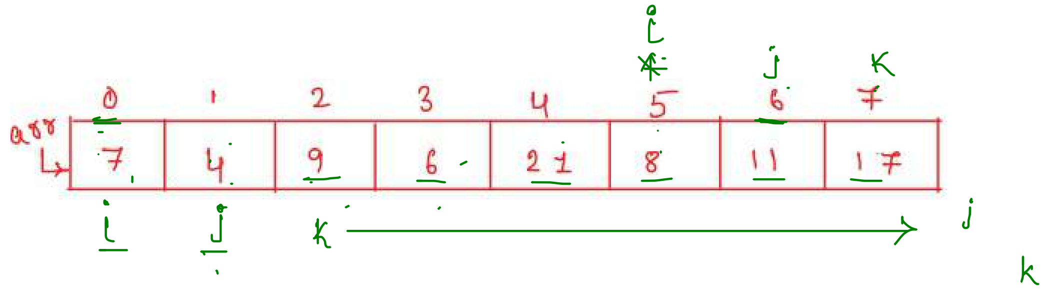
AP_1	✓ AP_2
$O(n^2)$	$O(n \cdot \log_2^n)$

if i/p is sorted then T.C : ?

$O(n)$

$\rightarrow$ 3 - ele's.

2) Find a triplet whose sum is equal to k [a+b+c=k]



$n = 8$

$K = 33$

$i \le n-3 \Leftrightarrow i < n-2$ $\quad$ $k: 2$ to 7

$j \le n-2 \Leftrightarrow j < n-1$ $\quad$ $j: 1$ to 6

$K \le n-1 \Leftrightarrow K < n$ $\quad$ $i: 0$ to 5

$a+b=K$ ✓

n_1

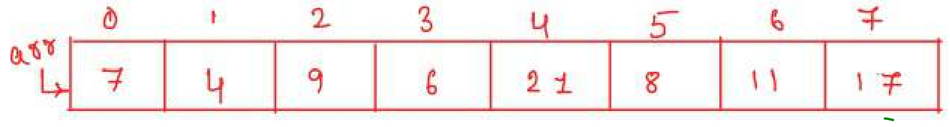
$${}^nC_r = \frac{n!}{(n-r)! \times r!}$$

$${}^nC_3 = \frac{n!}{(n-3)! \times 3!}$$

$$= \frac{n \times (n-1) \times (n-2) \times \cancel{(n-3)!}}{\cancel{(n-3)!} \times 6}$$

$$= \frac{n(n-1)(n-2)}{6} \Rightarrow n^3$$

$\therefore O(n^3)$



$n = 8$

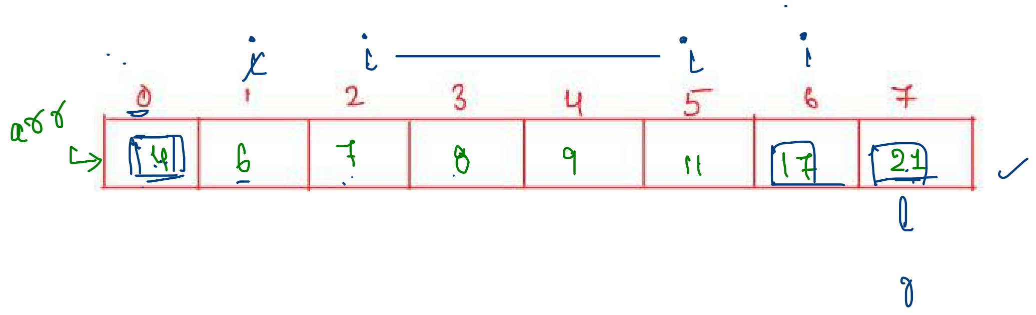
$\underline{a} + \underline{b} + \underline{c} = K$
✓ ? ?

$K = \underline{33}$ ✓

1. sorted.

$arr[i] + arr[l] + arr[r]$
17 21 21

✗



$4 + (\underbrace{6 + 21}) = \underline{31}$

$28 + 4 = 32$

$i \leq \underline{n-3} \Rightarrow i < n-2$

$\hookrightarrow n\left(\overset{2\text{-}ptr}{\longleftrightarrow}\, n\right)$

while loop

for ~ loop

chktriplet

```
function chkPair(arr,n,k)
{
     1.arr.sort();
      for(i=0;i<n-2;i++)
      {
          l=i+1
          r=n-1
          while(l<r)
          {
              if(arr[i]+arr[l]+arr[r]==k)
                     return true
              else if(arr[i]+arr[l]+arr[r]<k)
                     l++
              else
                     r--;
          }
      }
      return false;
}
```

$\rightarrow n \cdot \log_2^n$

$\rightarrow \cong n$

$\rightarrow n^2$

$\rightarrow n$

$\cancel{n\log_2^n} + n^2$

$\therefore \underline{O(n^2)}$ T.C

**

$\underline{AP_1}$ $O(n^3)$

$\underline{AP_2}$ $O(n^2)$

$\underline{obj}$ Hash_map

3) Seperate 0's and 1's

$M_1 \rightarrow$ 2ptr (opposite - dir)

arr ↳

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	0	0	0	1	1	0	0	1	0	0	0

$n = 13$

o/p arr ↳

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	1	1	1	1	1

$\longleftrightarrow$ $\longleftrightarrow$

Ap1 :-

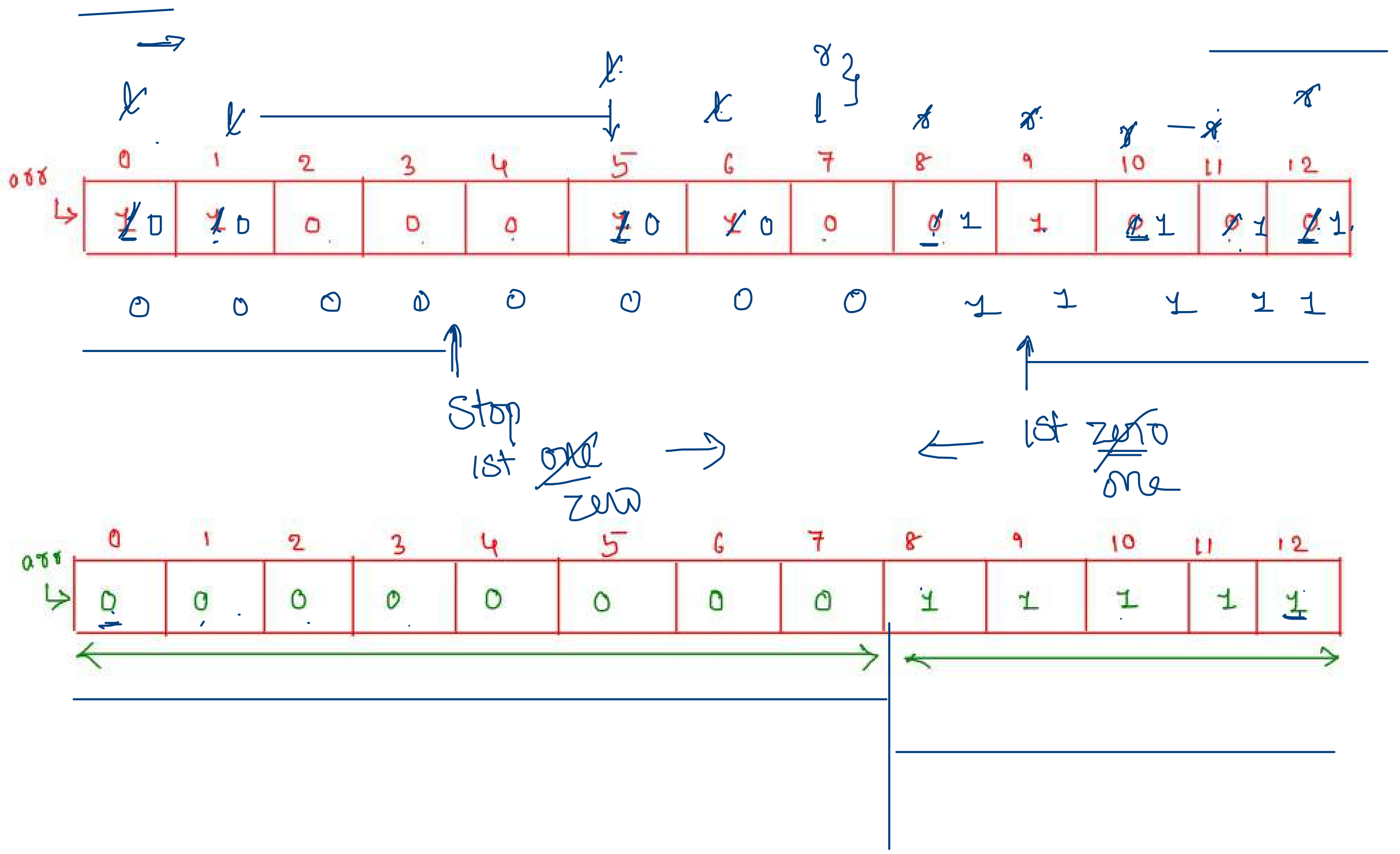
n ↳ zero[ ] ✓
n one [ ] ✓ } combine.

↳ T.C : O(n) ✓.
S.C : O(n) ✗

Ap2 : -

arr.sort( )

↳ T.C : $O(n \cdot \log_2^n)$.



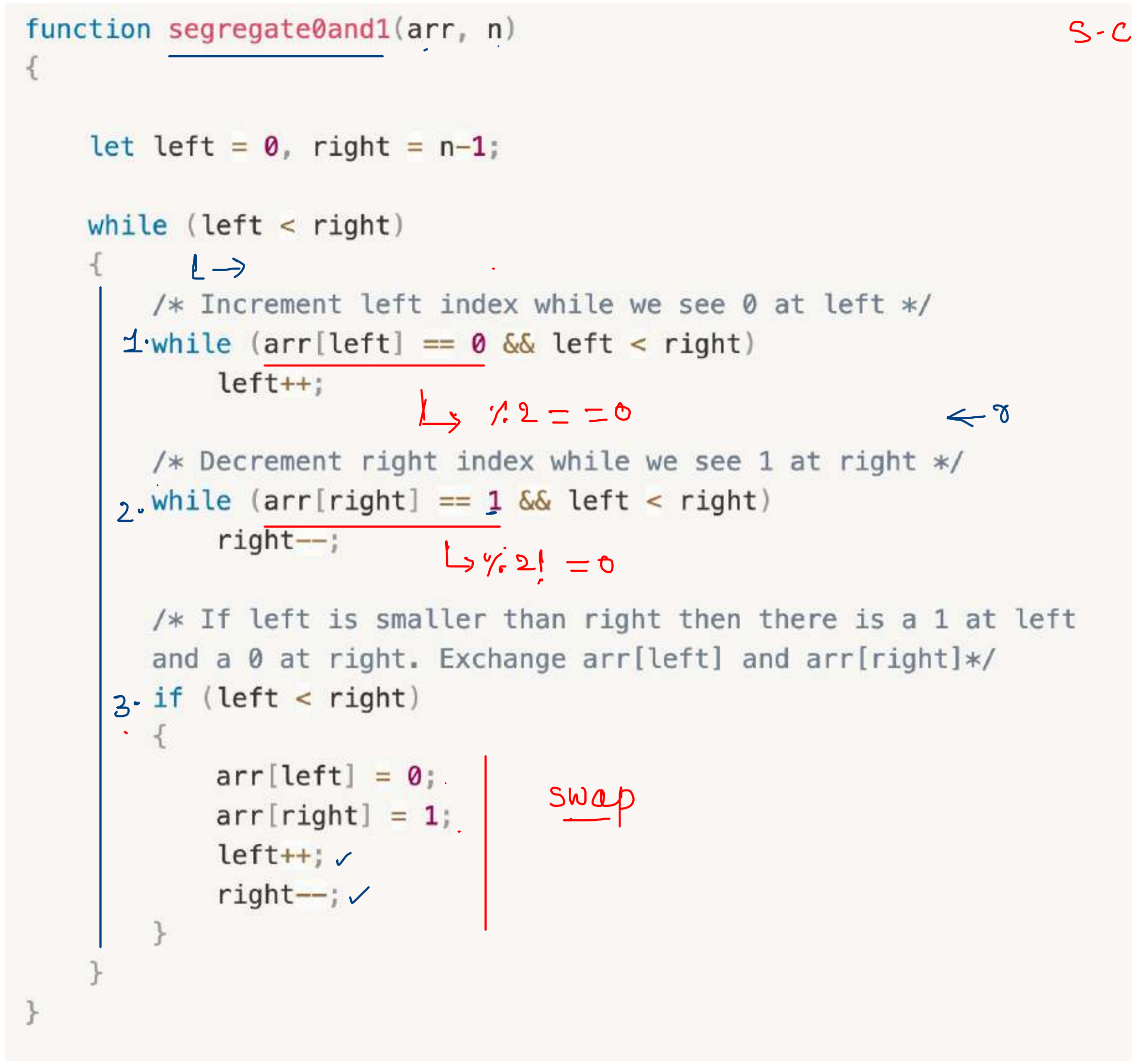
arr
0 1 2 3 4 5 6 7 8 9 10 11 12
0 0 0 0 0 0 0 0 1 1 1 1 1
Stop
1st ~~one~~ zero
1st ~~zero~~ one
arr
0 1 2 3 4 5 6 7 8 9 10 11 12
0 0 0 0 0 0 0 0 1 1 1 1 1

arr[l] = 0 ✓

arr[r] = 1 ✓

T-C : O(n)

S-C : O(1)



```
function segregate0and1(arr, n)
{

    let left = 0, right = n-1;

    while (left < right)
    {       l →
        /* Increment left index while we see 0 at left */
      1.while (arr[left] == 0 && left < right)
            left++;
                            ↳ %2 == 0                  ← r

        /* Decrement right index while we see 1 at right */
      2.while (arr[right] == 1 && left < right)
            right--;
                              ↳ %2! = 0

        /* If left is smaller than right then there is a 1 at left
        and a 0 at right. Exchange arr[left] and arr[right]*/
      3.if (left < right)
        {
            arr[left] = 0;
            arr[right] = 1;        swap
            left++; ✓
            right--; ✓
        }
    }
}
```

⇒ seperate all even and odd.

0	1	2	3	4	5	6	7	8
12	13	14	16	19	21	18	12	16

↓

0	1	2	3	4	5	6	7	8
12	14	16	18	16	13	19	21	19

even odd

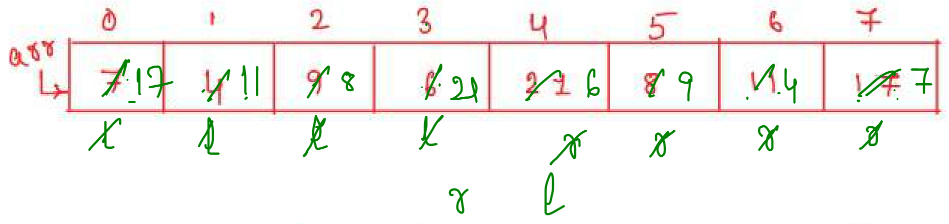
order is no problem

4. Reverse the array [in-place]

↳ S.C : O(1)

M1 - 2ptr

Yes



↳

0	1	2	3	4	5	6	7
17	11	8	21	6	9	4	7

2ptr + swap

⇒ T.C : O(n)

S.C : O(1)

① pair

② triplet

③ 0's and 1's

④ Rev arr.

⑤ palindrome.

M1 - 2ptr

opposite

Two Pointer [Model-2 : Same Direction]

$$\left.\begin{array}{l} i = 0 \\ j = 0 \end{array}\right\} \rightarrow \text{move same Dir}$$

5) Merge Two Sorted Arrays

arr3 = [ ]

$\uparrow$ arr1 $\uparrow$ arr2

final

$\Rightarrow$ sort ✓

↳ $O(n \cdot \log_2 n)$

i/p

$n_1 = 5$

sorted

arr1 ↳

0	1	2	3	4
1	3	5	7	9

✓

$n_2 = 5$

sorted

arr2 ↳

0	1	2	3	4
2	4	6	8	10

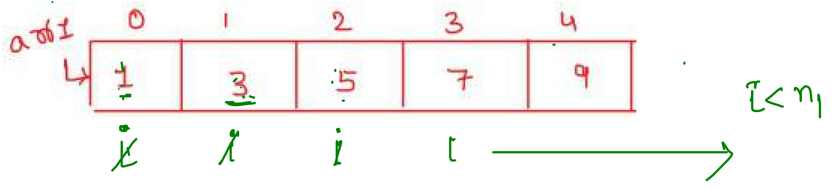
✓

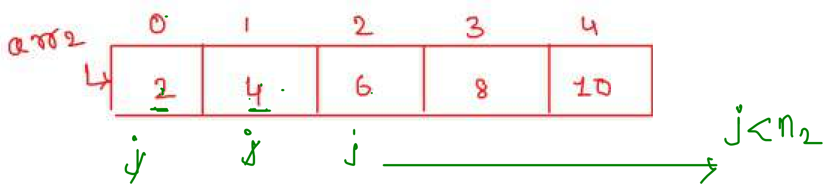
arr3 ↳

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

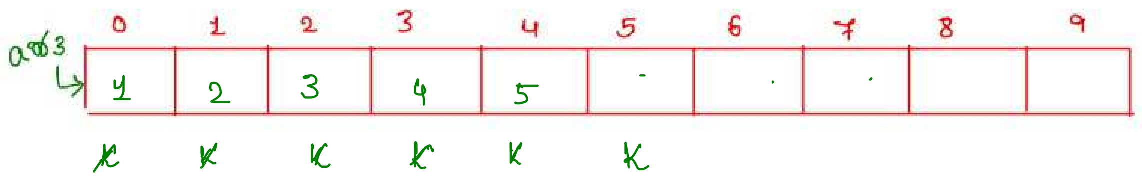
$\rightarrow$ final sorted

$n_1 = 5$ $n_2 = 5$





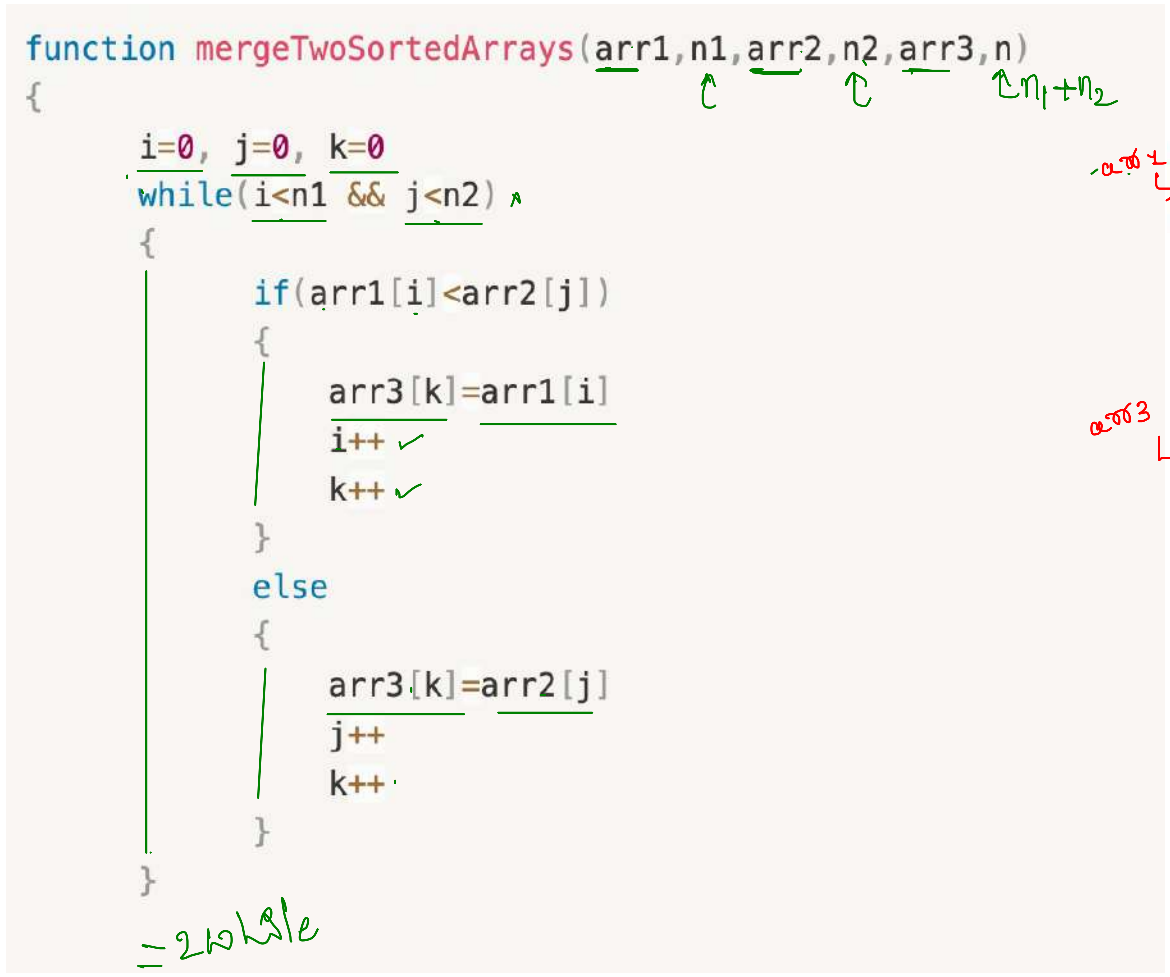
$M_2 \to$ 2ptr
same



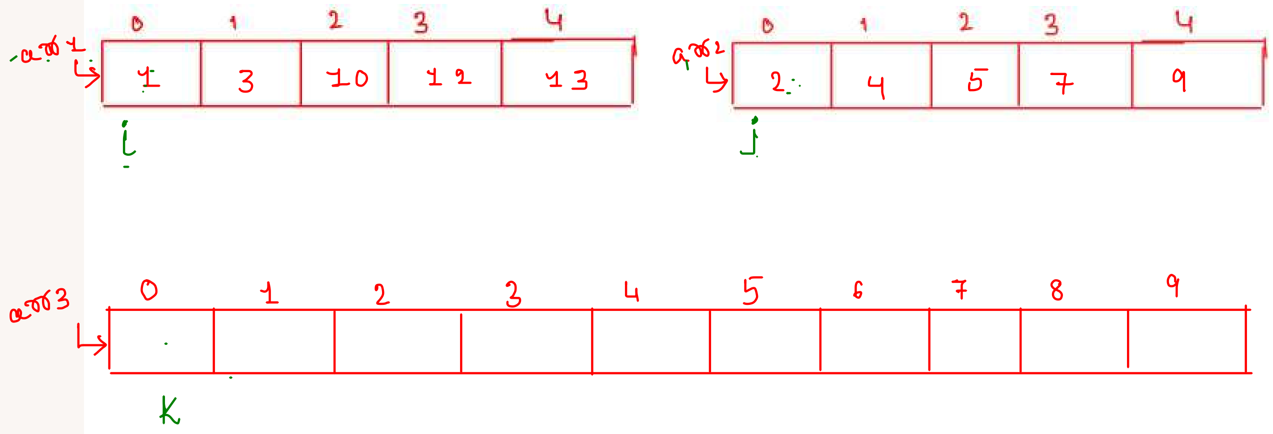
$\underline{n} = n_1 + n_2 = \underline{10}$

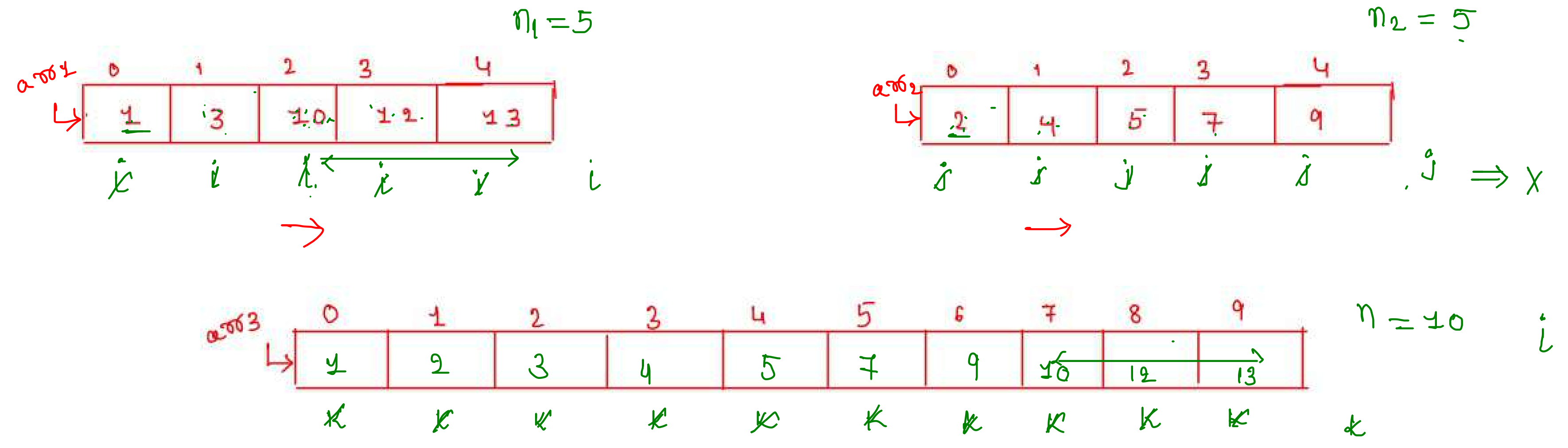
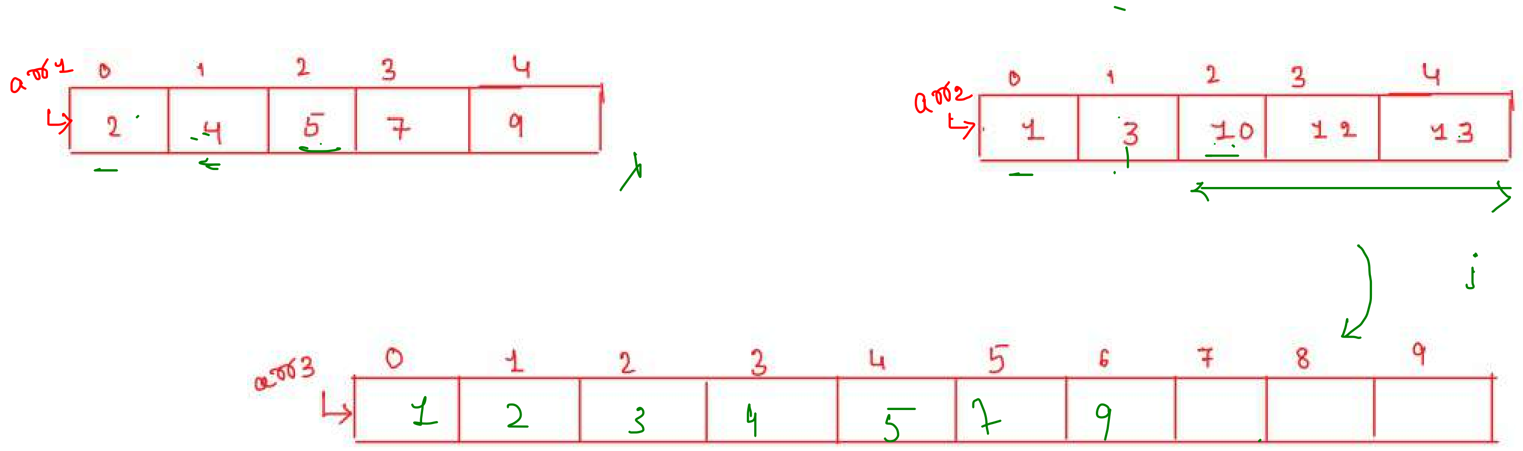
$k < n_1 + n_2$

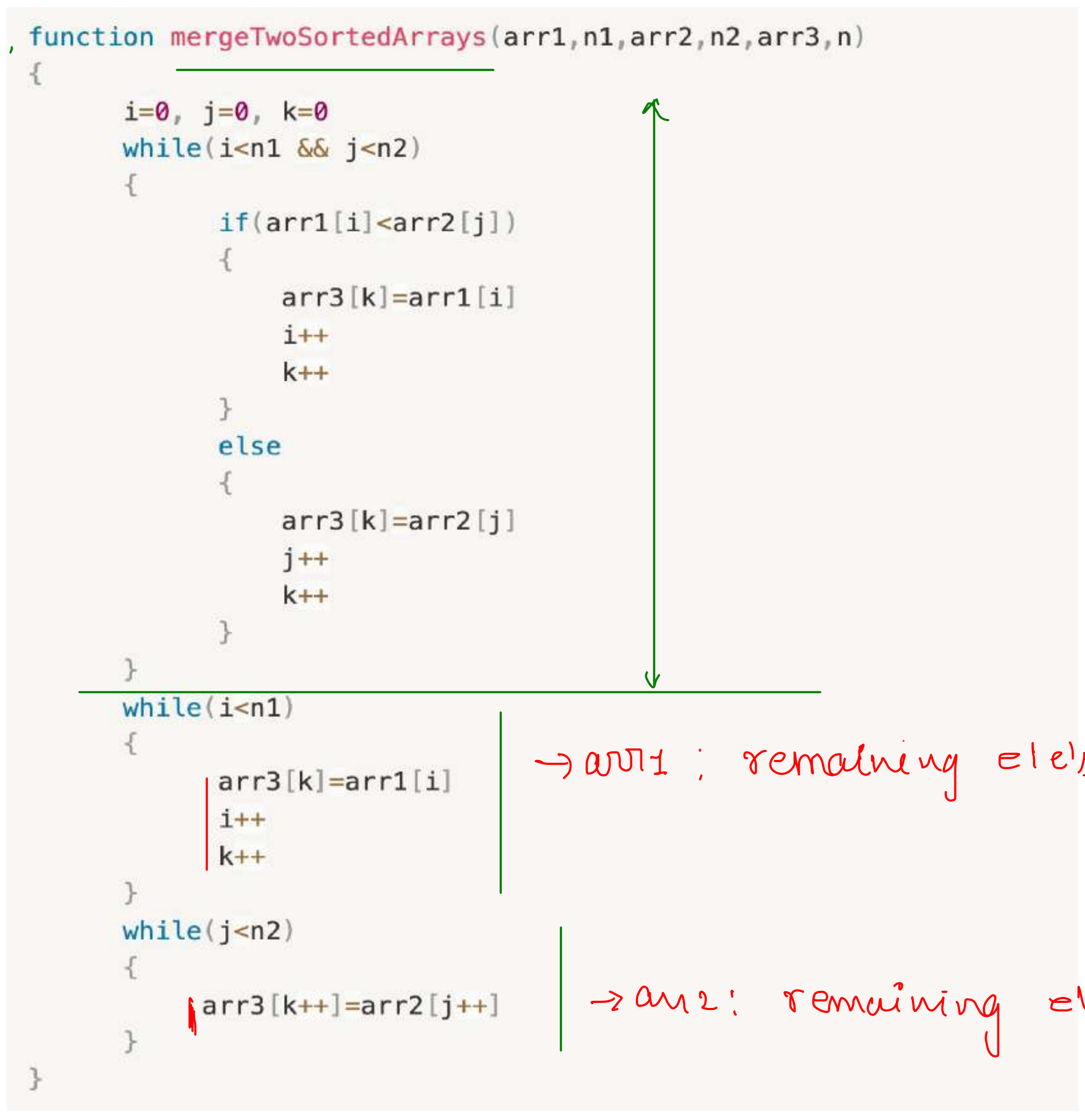
$\underline{arr1}[i]$ v/s $\underline{arr2}[j]$



```
function mergeTwoSortedArrays(arr1,n1,arr2,n2,arr3,n)
{
        i=0, j=0, k=0
        while(i<n1 && j<n2)
        {
                if(arr1[i]<arr2[j])
                {
                        arr3[k]=arr1[i]
                        i++ ✓
                        k++ ✓
                }
                else
                {
                        arr3[k]=arr2[j]
                        j++
                        k++
                }
        }
```




$n_1 = 5$
$n_2 = 5$
arr1
0 1 2 3 4
1 3 10 12 13
i i i i i i
arr2
0 1 2 3 4
2 4 5 7 9
j j j j j j ⇒ X
arr3
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 7 9 10 12 13
k k k k k k k k k k k
$n = 10$
i

arr1
0 1 2 3 4
2 4 5 7 9
arr2
0 1 2 3 4
1 3 10 12 13
j
arr3
0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 7 9



```
function mergeTwoSortedArrays(arr1,n1,arr2,n2,arr3,n)
{
        i=0, j=0, k=0
        while(i<n1 && j<n2)
        {
                if(arr1[i]<arr2[j])
                {
                     arr3[k]=arr1[i]
                     i++
                     k++
                }
                else
                {
                     arr3[k]=arr2[j]
                     j++
                     k++
                }
        }
        while(i<n1)
        {
                arr3[k]=arr1[i]
                i++
                k++
        }
        while(j<n2)
        {
              arr3[k++]=arr2[j++]
        }
}
```

→ arr1 : remaining ele's

→ arr2 : remaining ele's

$O(n_1+n_2) \Rightarrow O(n)$

⇒ Merge_sort

→ Merge_procedure

$O(n)$

Post-inc.

arr3[k++] = arr2[j++] ⇒

1: arr3[k] = arr2[j]

2. j++ } k++

3. k++ } j++

M2 - 2ptr.

6) Remove Duplicates from Sorted array

n = 9

arr ↳

0	1	2	3	4	5	6	7	8
1	1	1	2	2	3	4	4	4

⇒ o/p : 1, 2, 3, 4

n = 9

arr ↳

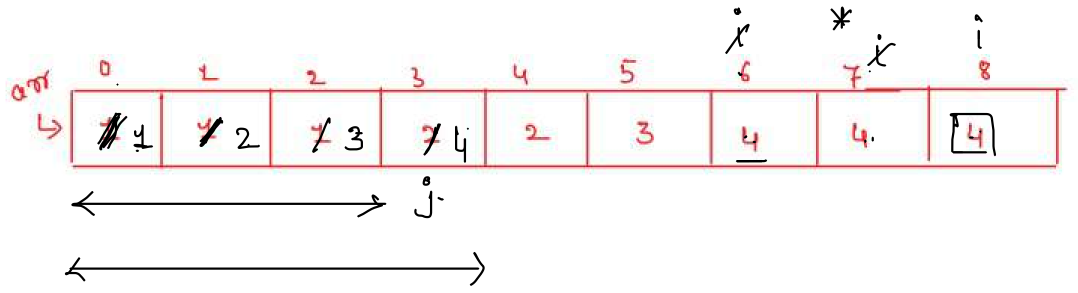
0	1	2	3	4	5	6	7	8
1	1	1	2	3	4	4	4	5

⇒ o/p : 1, 2, 3, 4, 5

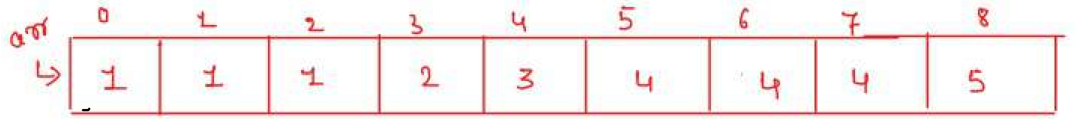
M2 - 2ptr

```
function removeDupSortedArray(arr, n)
{
        j=0
      x for(i=0;i<=n-2;i++)
        {
            ✓if(arr[i]!=arr[i+1])
            {
                1·arr[j]=arr[i]
                2·j++
            }
        }
      → arr[j]=arr[n-1] ✓

        for(i=0;i<=j;i++)
        {
            print(arr[i])            => 1, 2, 3, 4
        }
}
```

n=9



Assignment



$\underline{M_2}$: 2ptr ($\rightarrow$)

1. Merge - 2 sorted array.
2. Remove dup. from sorted array.
3. Find middle node LL $\uparrow$
4. Find cycle in LL $\downarrow$

Sliding Window

(S.W)

11: 25

Model-1

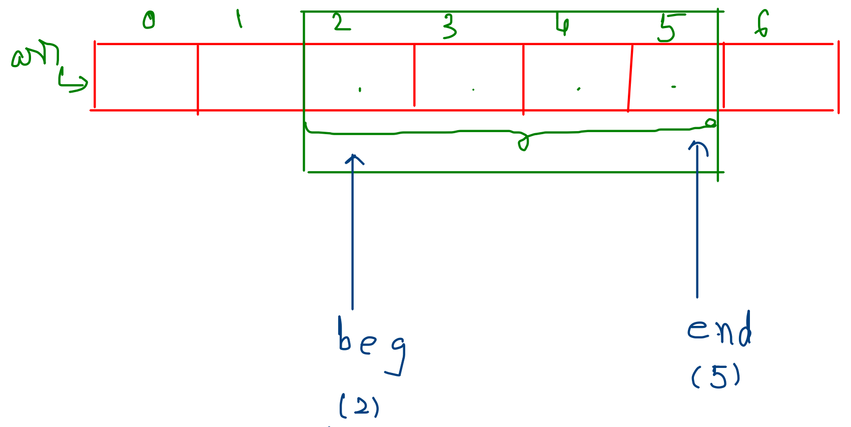
→ size of (S.W) fixed
K

→ given in question

* Model-2 [objects/ hashmap/ key→value]

→ size of (S.W), variable.

→ not given in question.



size = 4 = 5 - 2 + 1

= end - beg + 1 ✓

Where we can Apply ?

Arrays / Strings + Sub-Array / Sub-string + Largest sum

smallest $\Rightarrow$ S.W

min

max

Window Size : $\boxed{k}$

$M_1 \rightarrow$ Size is given (k)

$\leftarrow$

$M_2 \rightarrow$ Size is not given (x)

Example :-

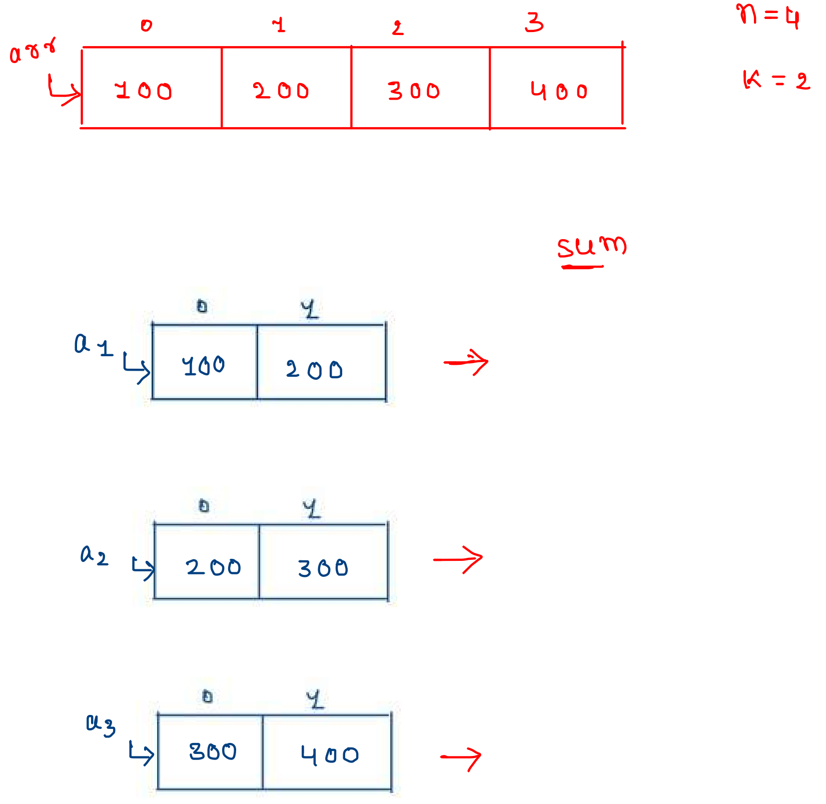
Given input Array, Find the maximum sum of all subarays of size k ✓

$\hookrightarrow M_1$

Model - 1 [Fixed Size SW]

7) Given input Array, Find the maximum sum of all subarays of size k

```
Input  : arr[] = {100, 200, 300, 400}
         k = 2
Output : 700

Input  : arr[] = {1, 4, 2, 10, 23, 3, 1, 0, 20}
         k = 4
Output : 39
We get maximum sum by adding subarray {4, 2, 10, 23}
of size 4.

Input  : arr[] = {2, 3}
         k = 3
Output : Invalid
There is no subarray of size 3 as size of whole
array is 2.
```



arr ↳

0	1	2	3	4	5	6	7	8
1	4	2	10	23	3	1	0	20

$n = 9$

$k = 4$

```
function fun(arr,n,k) // fixed size
{
  max_sum=-Infinity
  for(i=0;i<=n-k;i++)
  {
    sum=0
    for(j=i; j<=i+k-1; j++)
    {
      sum=sum+arr[j]
    }
    if(sum>max_sum)
    {
      max_sum=sum)
    }
  }
  return max_sum
}
```

arr

0	1	2	3	4	5	6	7	8
1	4	2	10	23	3	1	0	20

$n = 9$

$k = 4$

```
function fun(arr,n,k) // fixed size
{

  max_sum=0
  for(i=0;i<=k-1;i++)
  {
    max_sum=max_sum+arr[i]
  }

  curr_sum=max_sum
  for(i=k; i<n; i++)
  {
    curr_sum=curr_sum-arr[i-k]+arr[i]
    max_sum=Math.max(curr_sum,max_sum)
  }
  return max_sum

}
```

AP1

```
function fun(arr,n,k) // fixed size
{
  max_sum=-Infinity
  for(i=0;i<=n-k;i++)
  {
    sum=0
    for(j=i; j<=i+k-1; j++)
    {
      sum=sum+arr[j]
    }
    if(sum>max_sum)
    {
      max_sum=sum)
    }
  }
  return max_sum
}
```

AP2

```
function fun(arr,n,k) // fixed size
{

  max_sum=0
  for(i=0;i<=k-1;i++)
  {
    max_sum=max_sum+arr[i]
  }

  curr_sum=max_sum
  for(i=k; i<n; i++)
  {
    curr_sum=curr_sum-arr[i-k]+arr[i]
    max_sum=Math.max(curr_sum,max_sum)
  }
  return max_sum

}
```

Model-2 [Variable Size SW]

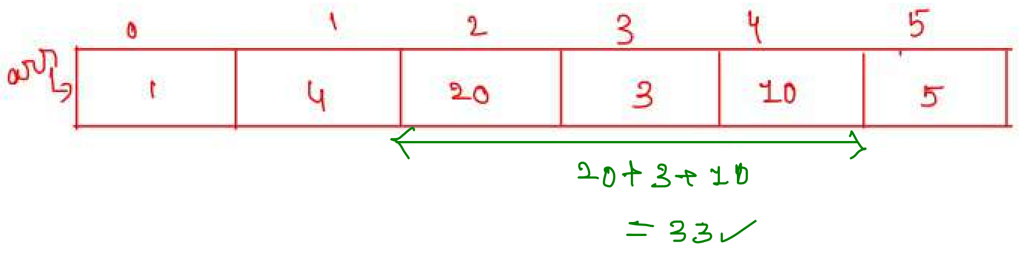
Size is not given

$(\underline{end} - \underline{beg} + 1)$

$j - i + 1$

$end - start + 1$

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



$n = 6$

sum = 33 ✓

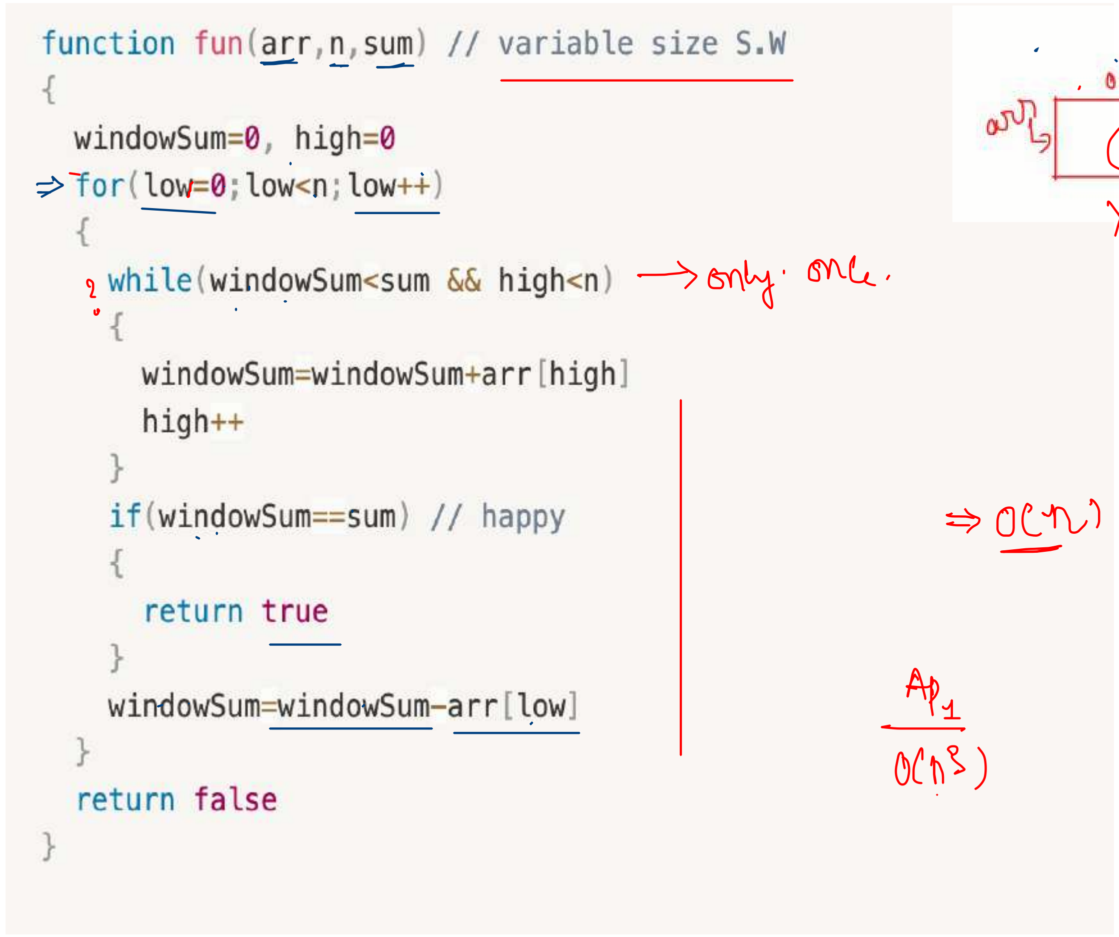
Ap1:-

n^2 ⇒ → Find all sub-Arra's

n → check sum. of each and every S.A

$\underset{C}{n^2} \times n = n^3 \Rightarrow \underset{T.C}{O(n^3)}$ ✓

sub-Array → cont.



```
function fun(arr,n,sum) // variable size S.W
{
  windowSum=0, high=0
  for(low=0;low<n;low++)
  {
    while(windowSum<sum && high<n)
    {
      windowSum=windowSum+arr[high]
      high++
    }
    if(windowSum==sum) // happy
    {
      return true
    }
    windowSum=windowSum-arr[low]
  }
  return false
}
```

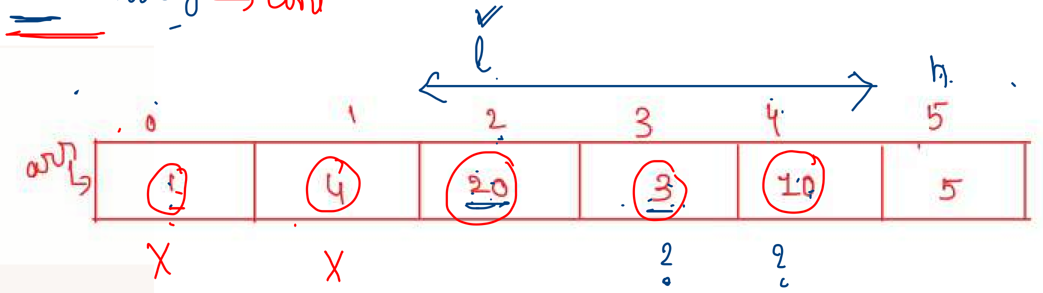
n ⇒

→ only once.

⇒ O(n)

Ap_1 $O(n^3)$

Ap_2 $O(n)$



n = 6

sum = 33

WS = 0 + 1 = 1 + 4 = 5

25

28

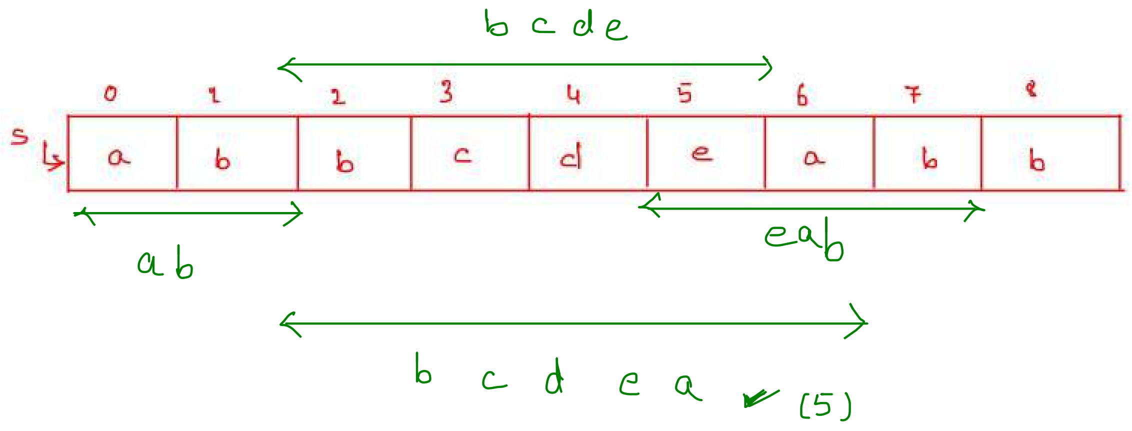
38 − 1 = 37 − 4

= 33 ✓

start : l

end : h−1

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



$n \Rightarrow \frac{n(n+1)}{2}$

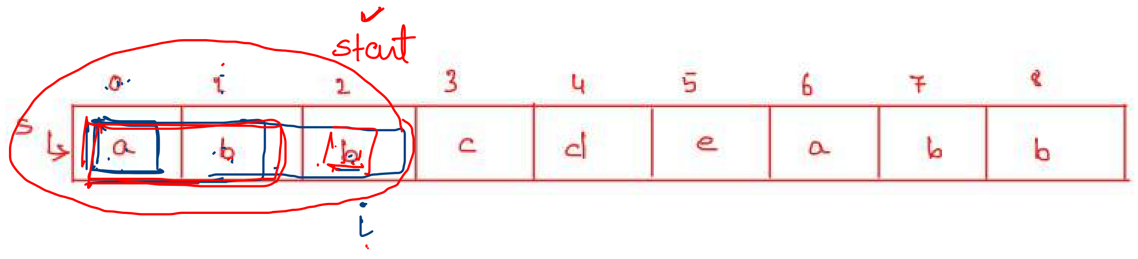
$\Rightarrow O(n^2)$

$n = 9$

V. ⇒ Ite:- 2-var

F.

length



$(i - start + 1)$

$\frac{0 - 0 + 1}{1}$ max_len = ~~0~~ ~~1~~ 2 → $\boxed{a|b}$

$\boxed{a|b}$ ✓

$\underbrace{\boxed{a|b}}\boxed{b} \Rightarrow \boxed{b} + (\quad)$

$1 - 0 + 1 = 2$

start = Max(start, $\underline{hm.get(s[i]) + 1}$)

$1 \quad + \quad 1$

start = max (0 , 2)

= 2

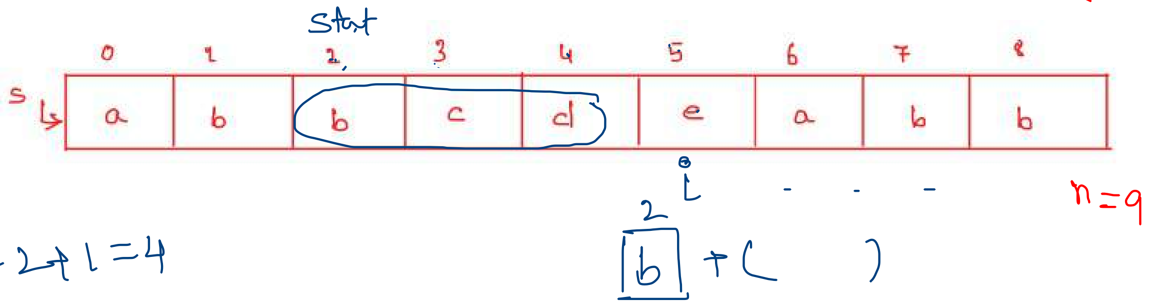
~~obj~~ hm

Key	Value
a	0
b	~~1~~ 2

obj

array indices.

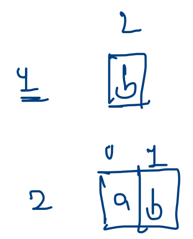
```
function longestUniqueSubstr(s,n)
  {

  ① let hm be a hashmap/ object  ⇒ ✓

    maximum_length = 0;


    start = 0;

    for(i= 0; i < n; i++)
    {

   1. if(hm.containsKey(s[i]))
      {
        start = Math.max(start, hm.get(s[i]) + 1);
      }
  2. hm.put(s[i], i);
  3. maximum_length = Math.max(maximum_length, i-start + 1);

    }
    return maximum_length;
  }
```



$5-2+1=4$

hm ↳

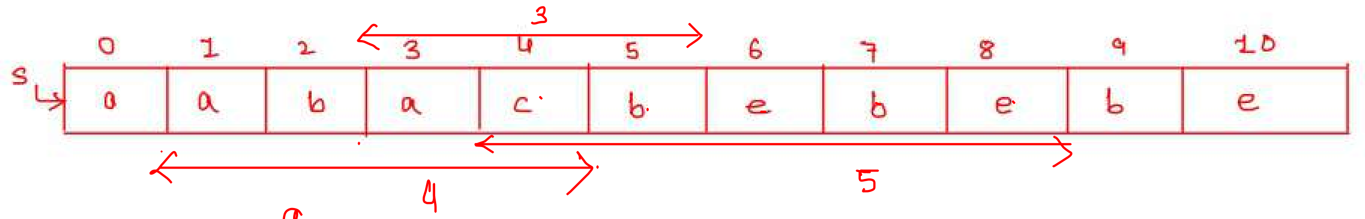
Key	Value
a	0
b	2
c	3
d	4
e	5



$4-2+1=3$

max_len = ~~0~~ ~~1~~
~~2~~
~~3~~
4

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



n = 11

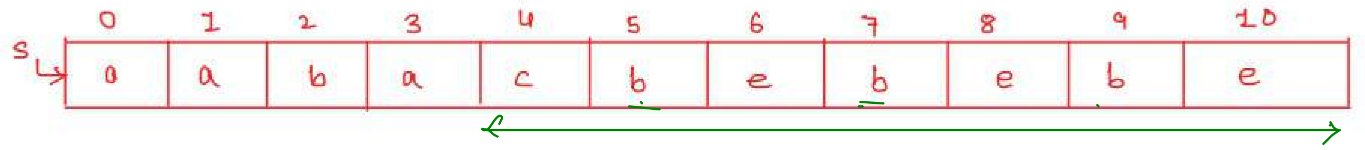
k = 3 ✓

a

b

c

a c b ⇒ 3



n = 11

k = 3 (distinct char's)

c ✓

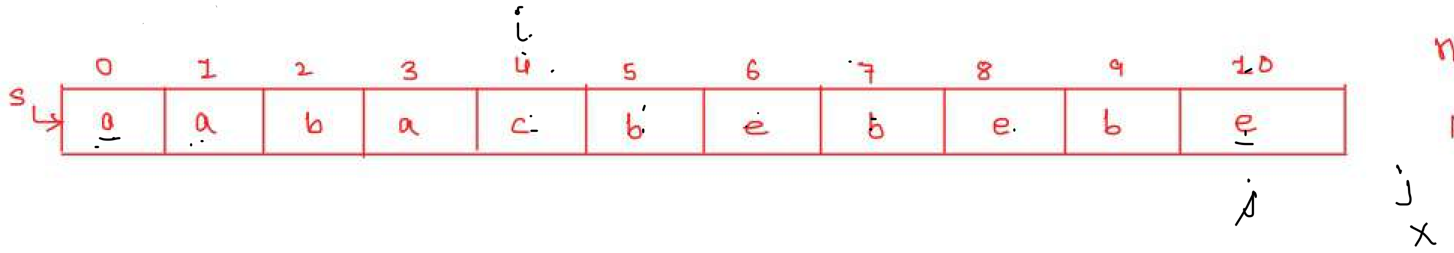
e ✓

b ✓

$10 - 4 + 1 = \underline{7}$ ✓

sub-string

len : j-i+1



n = 11

k = 3

start }
end } => end-start+1

~~max_len = -1~~
~~3~~
~~7~~
~~8~~
7

hm → frequency

Key	Value
b	~~1~~ ~~2~~ ~~1~~ ~~2~~ 3
c	1
e	~~1~~ ~~2~~ 3

4

$\textcircled{3}$

# of destinct char's = hm.size()

hm.len()

```
function longestStringWithKdistinctChar(s,n, k)
{
      i=0, j=0, maxLen=-1
      let hm be a HashMap / object
      while(j<n)
      {
          if(hm.containsKey(s[j]))
          {
              hm.put(s[j], hm.get(s[j]) + 1 )   ⇒
          }
          else
          {
              hm.put(s[j],1)
          }

    (1)   if(hm.sze()<k)
          {
              j++
          }
    (2)   else if(hm.size()==k)
          {
              maxLen=Math.max(maxLen, j-i+1)
              j++
          }
```

```
    (3)   else *
          {
              while(hm.size()>k && i<n )
         2    {
                  if(hm.containsKey(s[i]))
                  {
                      hm.put(s[i], hm.get(s[i])-1 )
                  }
                  if(hm.get(s[i])==0)
                  {
                      hm.remove(s[i])
                  }
                  i++
              }
          }
          j++
      }
      return maxLen;
}
```

j

	0	1	2	3	4	5	6	7	8	9	10
s ↳	o	a	b	a	c	b	e	b	e	b	e

i →

n = 11

k = 3

Key	value
~~a~~	~~2 1 0~~
b	~~1~~ ~~2~~ 1
c	1
e	1

~~4~~ 3

max-len = ~~-1~~ ~~5~~ 6

THNK-YOU