Array Problems - II & Time/Space Complexity

Foundation Course on Data Structures & Algorithm - Part I

Love Babbar • Lesson 8 • Apr 28, 2022

Questions:

T.C

- Linear Search in an array → O(n)
- Reverse an Array → O(n)
- find max and min element in an array → O(n)
- Swap Alternates in an array
- Sort an Array of 0, 1 and 2 → O(n)
- Move all negative number to one side of array
- Find union and intersection of 2 sorted arrays
- Program to cyclically rotate an array by one
- find duplicate in an array of N+1 integers
- find the pair that sum to a given value →
- find the triplet that sum to a given value
- Check whether an array is palindrome or not
- Minimum swaps required bring elements less equal k together → SW
- Unique Number of Occurrences - Leetcode
- Kadane's Algo

Questions:

Find common elements in 3 sorted arrays

Find first repeating element in array

Find first non-repeating element in an array

Find subarrays with equal 0s and 1s

Find subarray with 0 Sum in an array

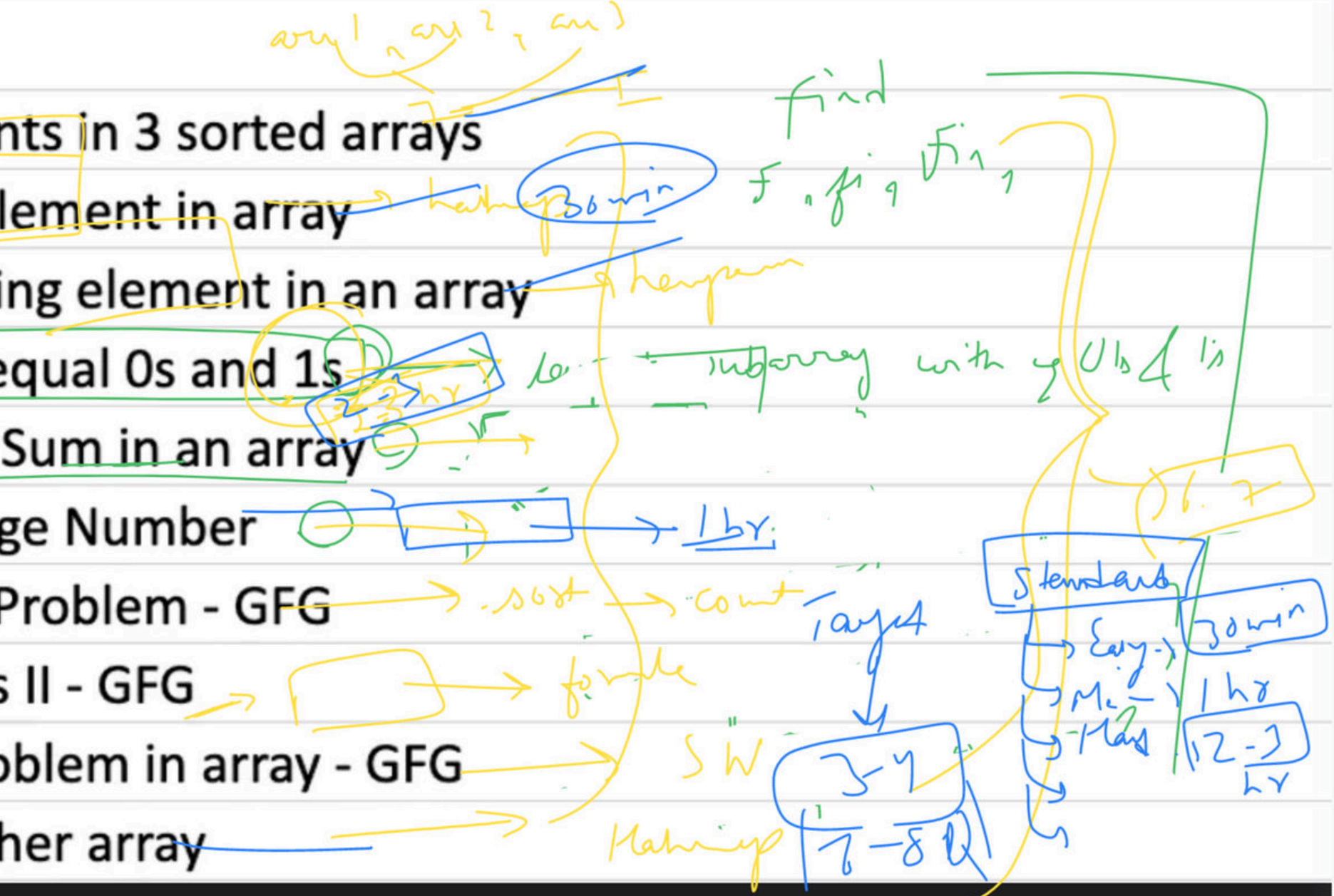
find factorial of a Large Number

Minimum Platforms Problem - GFG

Minimise the Heights II - GFG

Majority Element Problem in array - GFG

Array SubSet of another array

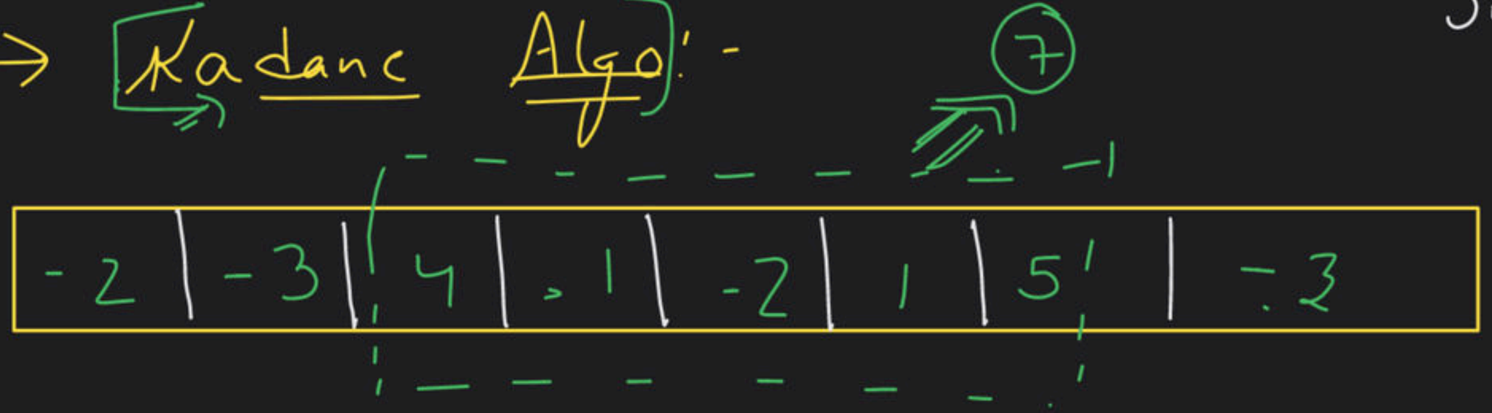


Array Problems:-

→ Largest Sum Continous Subarray

group

Kadane Algo:-



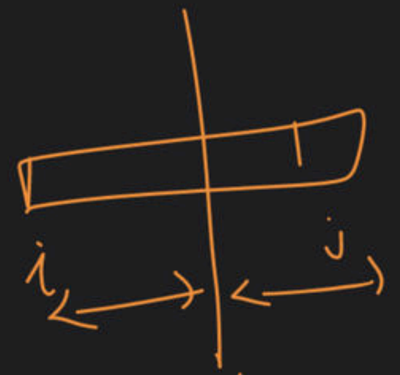
→ Brute force

find every subarray

sum 1
sum 2
sum 3

max → any

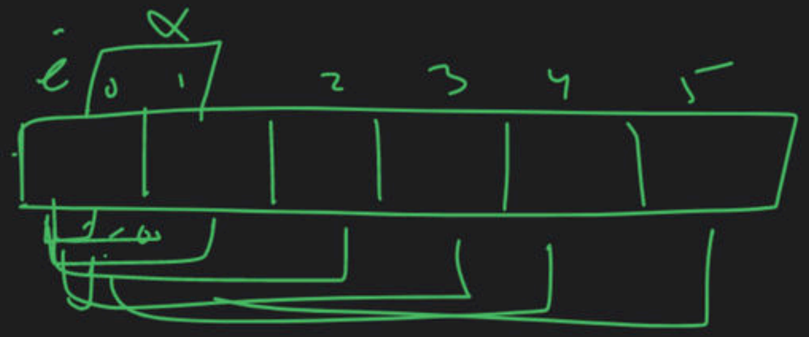
min swap → Ques



n/2

O(n/2)

O(n)



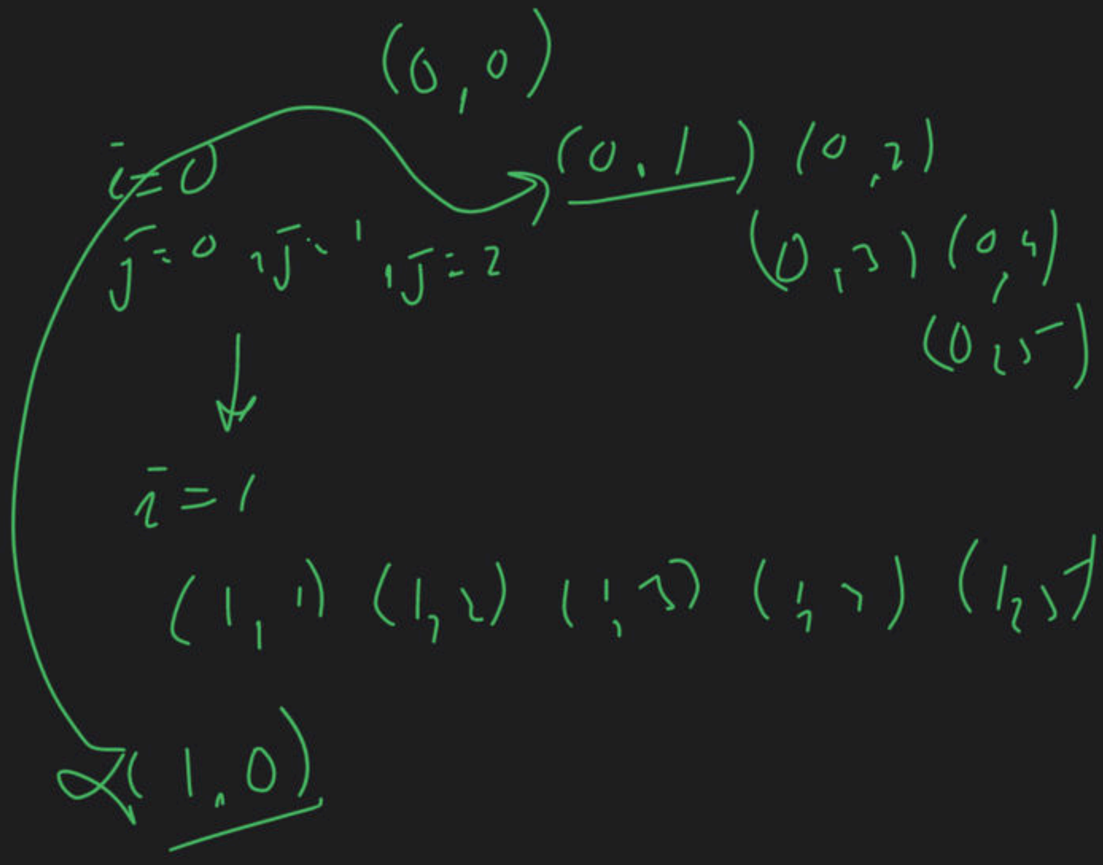
i → start

j → end

T.C / S.C

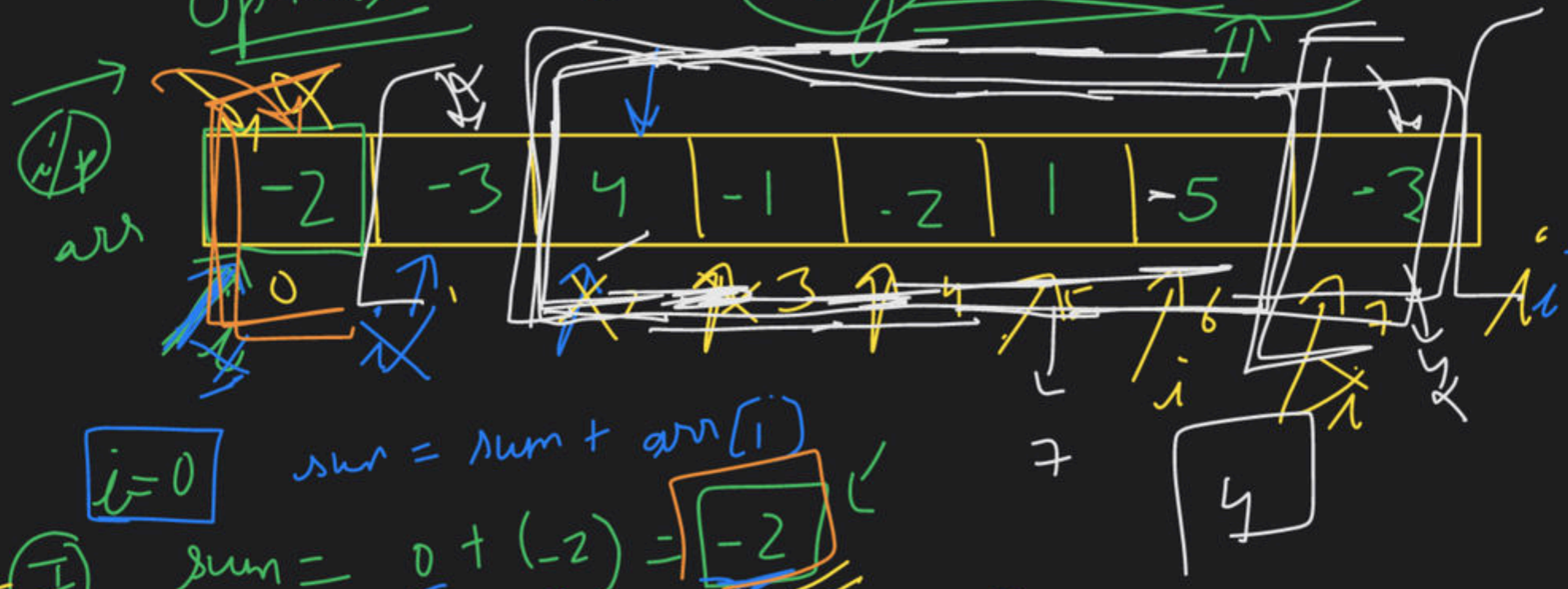
```
for (i -> 0 ---< n)
{
    sum = 0
    for (j = i ---> < n)
    {
        sum += arr[j]
    }
}
```

① Brute force



Optimise :-> Single traversal

Kadane Algorithm



sum = 0

maxi = INT_MIN

ans

i = 0

sum = sum + arr(i)

sum = 0 + (-2) = -2

naya answer aaya kya ?

ans = max(INT_MIN, -2) = -2

(sum < 0)

sum = 0

sum = -2 < 0

sum = 0

why ?

why

i = 1

sum = 0 + (-3) = -3

ans = max(-2, -3) = -2

sum = 0

i = 2

sum = 0 + 4 = 4

ans = max(-2, 4) = 4

sum = 4 > 0

I

II

start end

III

i = 3

sum = 4 + (-1) = 3

ans = max(4, 3) = 4

sum < 0 ✗

i = 4

sum = (3) + (-2) = 1

ans = max(4, 1) = 4

sum = 1 > 0

i = 5

sum = 1 + 1 = 2

ans = max(4, 2) = 4

sum = 2 > 0

Dry run

sum = 4 , ans = 4

i = 6

sum = 2 + 5 = 7

ans = max(4, 7) → 7

sum = 7 > 0

i = 7

sum = 7 + (-3) = 4

ans = max(7, 4) → 7

sum = 4 > 0

ans = 7 → sum

ans = 4 max(7,4) = 7

sum = 3 1 2 7 4

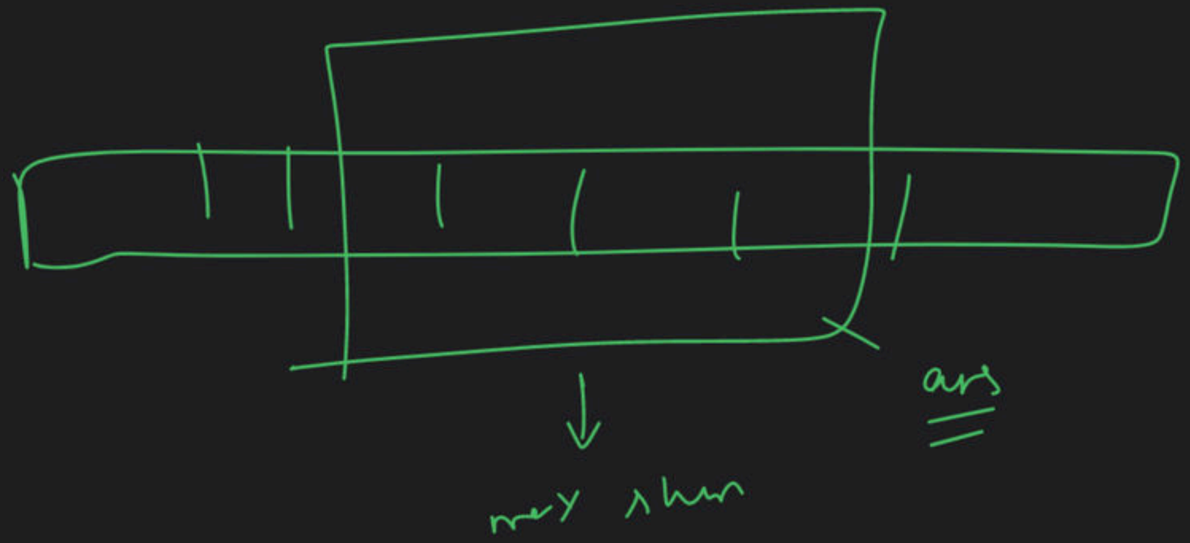
→ subset

→ How ?

→ why ?

→ largest sum contiguous subarray

→ largest sum contiguous subarray

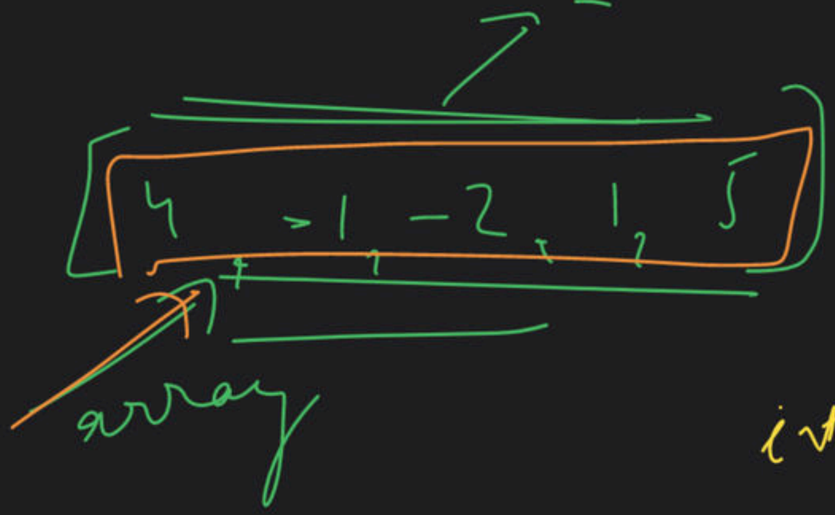


→ largest sum

sum

subarray

extension



→ Array → starting index

ending index

→ window

int start
end

start 2
end :

→ Time Complexity

Amount of time taken

taken by algo / code / program

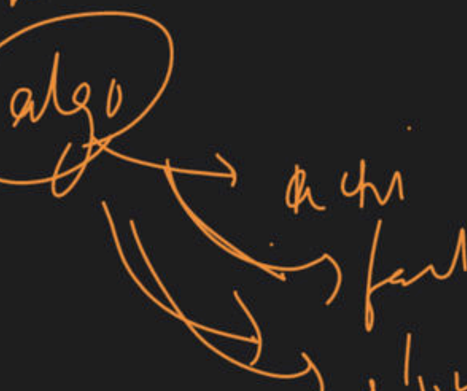
as a function of input

Notation

Big O → O(n) → Upper bound

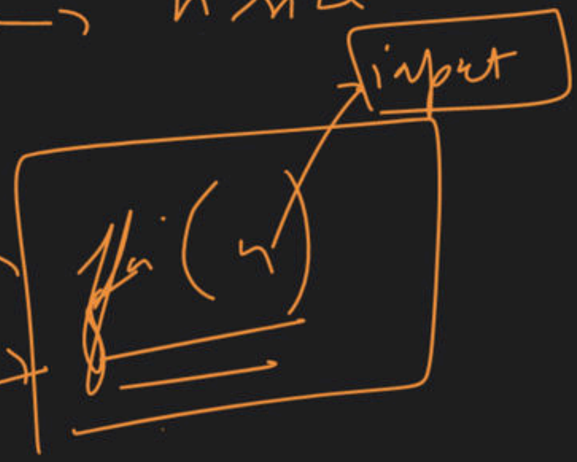
theta → θ(n) → avg - case complexity

omega → Ω → lower bound



T.C →

arr → n size



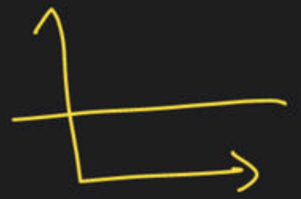
fast → T.C

Distance

→ Km

Big-O notation:-

Ex -> O(n) -> linear time

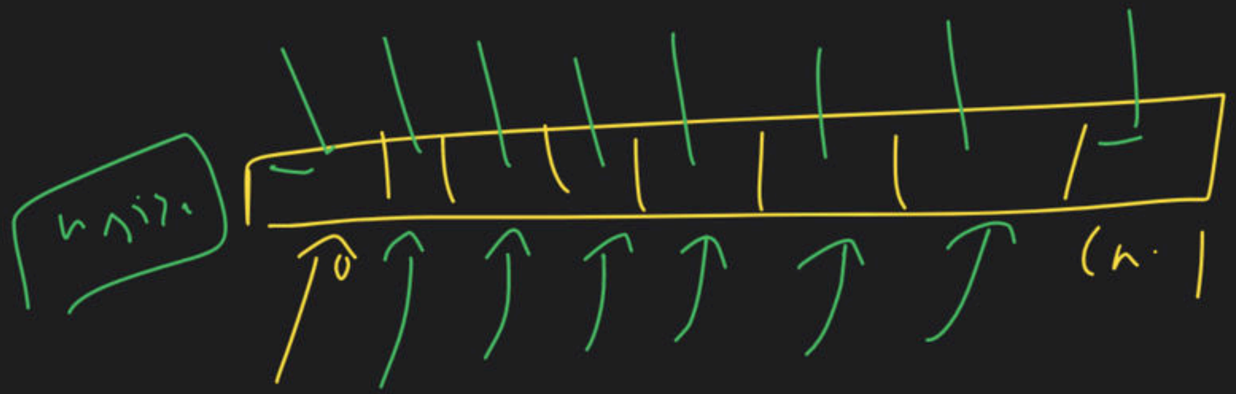


- O(1) -> constant time
- $O(n^2)$ -> quadratic
- $O(n^3)$ -> cubic
- O(log n) -> logarithmic
- :
- :
- :

→

```
for (int i=0 ; i<n; i++)
{
    cout << arr[i];
}
```

T.C → O(n)



total no of operation = n

→ for (int i = 0 ; i < 10; i++)

{ → ——

}

10

20

2000

10^14

1001

10000

constant

T.C -> O(1)

O(10)

→ int i = 101;

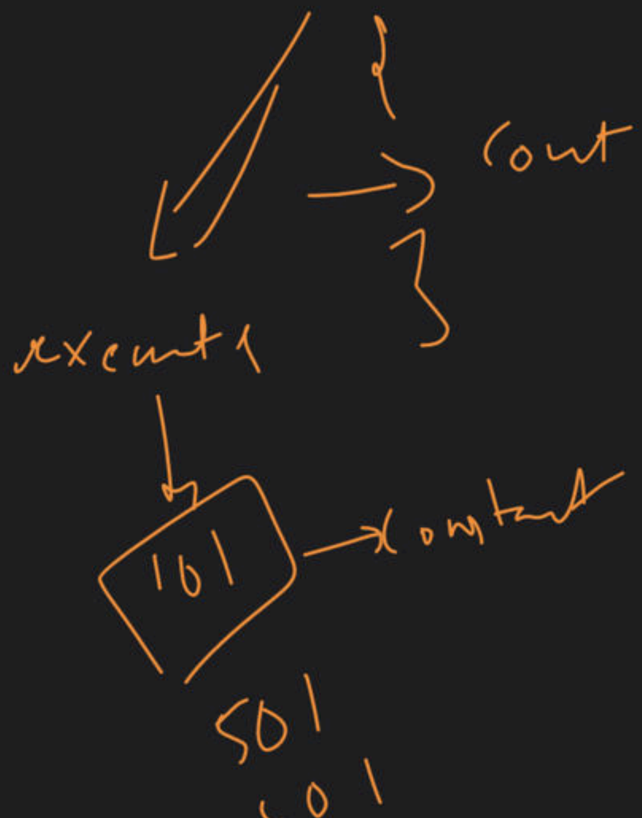
while (i--)

→ T :(→ O(1) ≈

{

cout << "mera bhai";

}



→

for (int i → 0 → n)

{

for (j → 0 → n)

{

cout << "bachaun";

}

}

↳ T :(→

→ for (i=0 → n)

{ for (j=0 → n)

{ for (k=0 → n

}

}

}

↳ T.C → $O(n^2)$

$O(1)$ →

n

i=0 → j = 0, 1, 2, 3 — (n-1)

i=1 → j = 0, 1, 2, 3 — (n-1)

⋮

i=(n-1) → j = 0, 1, 2 — (n-1)

n × n = n^2

n × n × n → n^3

$O(n^3)$

→ why we need T.C ?

↳ comparison b/w algo

H/W

```
for (int i → 0 -><n)
{
    for (int j = i ⟶ <n)
    {
        code
    }
}
```

i=0	j → 0 --- (n-1)
i=1	j = 1 — --(n-1)
i=2	j = 2 — — —(n-1)
i=3	j = 3 — — —(n-1)
⋮	
i=(n-1)	j = (n-1)

→ Kadane

↳ T.C → O(n) , n = size of array

$f(n) = n + 3n \rightarrow 4n \rightarrow O(n) \rightarrow O(n)$

$f(n) = \frac{n^2}{4} \longrightarrow O(\frac{n^2}{1}) = O(n^2)$

$f(n) = n^3 + 2n \rightarrow n^3 \rightarrow O(n^3)$

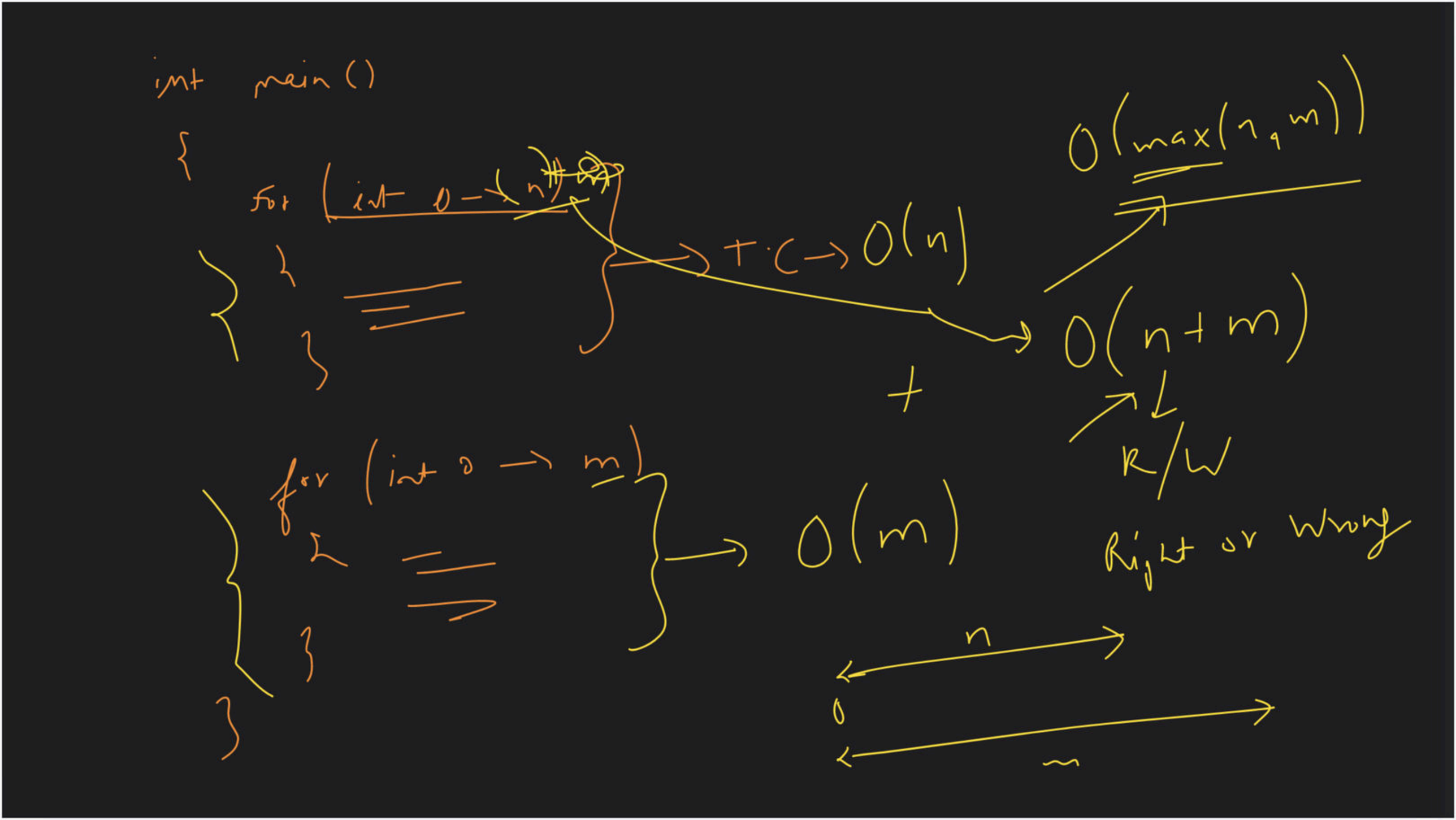
$f(n) = 311 \longrightarrow O(1)$

$f(n) \rightarrow n^3 + \frac{n^3}{5} \longrightarrow n^3\left(1 + \frac{1}{5}\right) = O(n^3)$

$f(n) \rightarrow n^3 + n^6 + n^{2/3} \longrightarrow O(n^6)$

$f(n) \rightarrow n^2 + 1 \longrightarrow O(n^2)$

$\longrightarrow 2n \longrightarrow O(2n) = O(n)$


int main()
{
for (int 0 -> n)
}
}
T.C -> O(n)
O(max(n, m))
O(n+m)
+
for (int 0 -> m)
}
O(m)
L
R/W
Right or Wrong
n
0
m

$k = n + m$

```
for (int i=0 ; i<n+m ; i++)
{
}
```

$O(k)$

$O(n+m)$

O $n + n^2$

Space Complexity

space taken by p/g


```
int i=0
for ( ; i<n ; i++)
{
    cout << i;
}
```

→ S.C → O(1)

constant space

t/w pattern

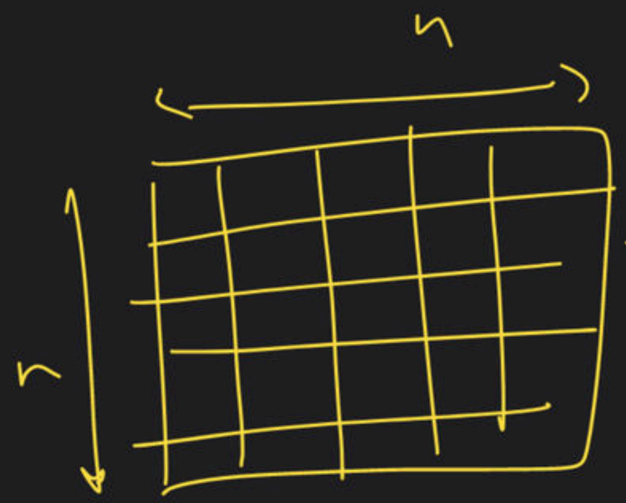
↓ → (2)

T.C → ?

→ {

n

int arr [n] → S.C → O(n)



3 → Total → n * n

S.C → O(n²)

{

int arr[n]

int arr[m]

}

$S.C \rightarrow O(n+m)$

{

int arr[n+m];

int arr2[n+m];

}

$\cancel{O(n+m)}$

$O(\cancel{O}(m+n))$

$O(m+n)$

→ { func ( )

int arr[115]; → S.C → O(1)

(constant)

}

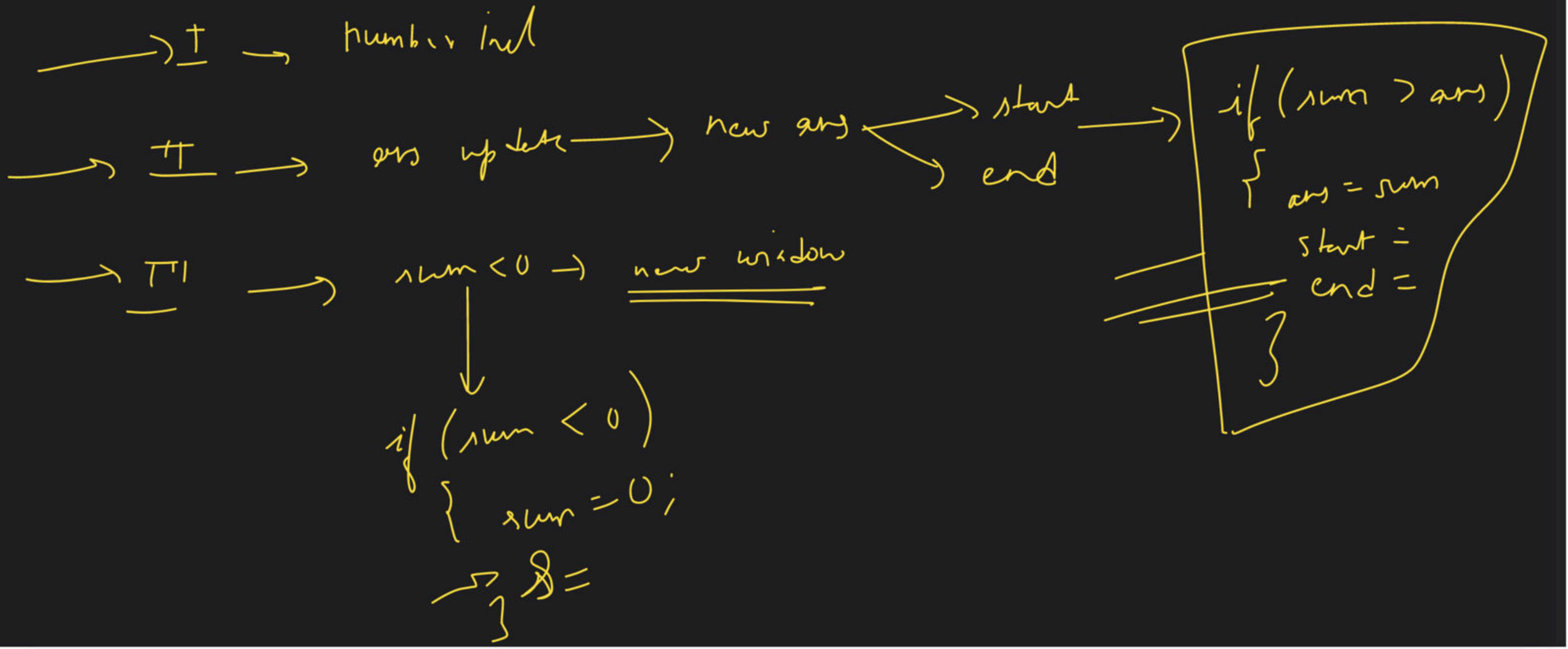
→ { ( )

int v1, v2, v3, v4 → O(1)

10000

an
vector
v
this
LL

}


i
number inc
ans update
new ans
start
end
if (sum > ans)
{
ans = sum
start =
end =
}
sum < 0 → new window
if (sum < 0)
{ sum = 0;
}

H/W → T.C / S.C

→ GFG → Mock

→ Codestudio

→ Interviewbit → 100 MCQ

→ O/P

→ T.C

→ S.C

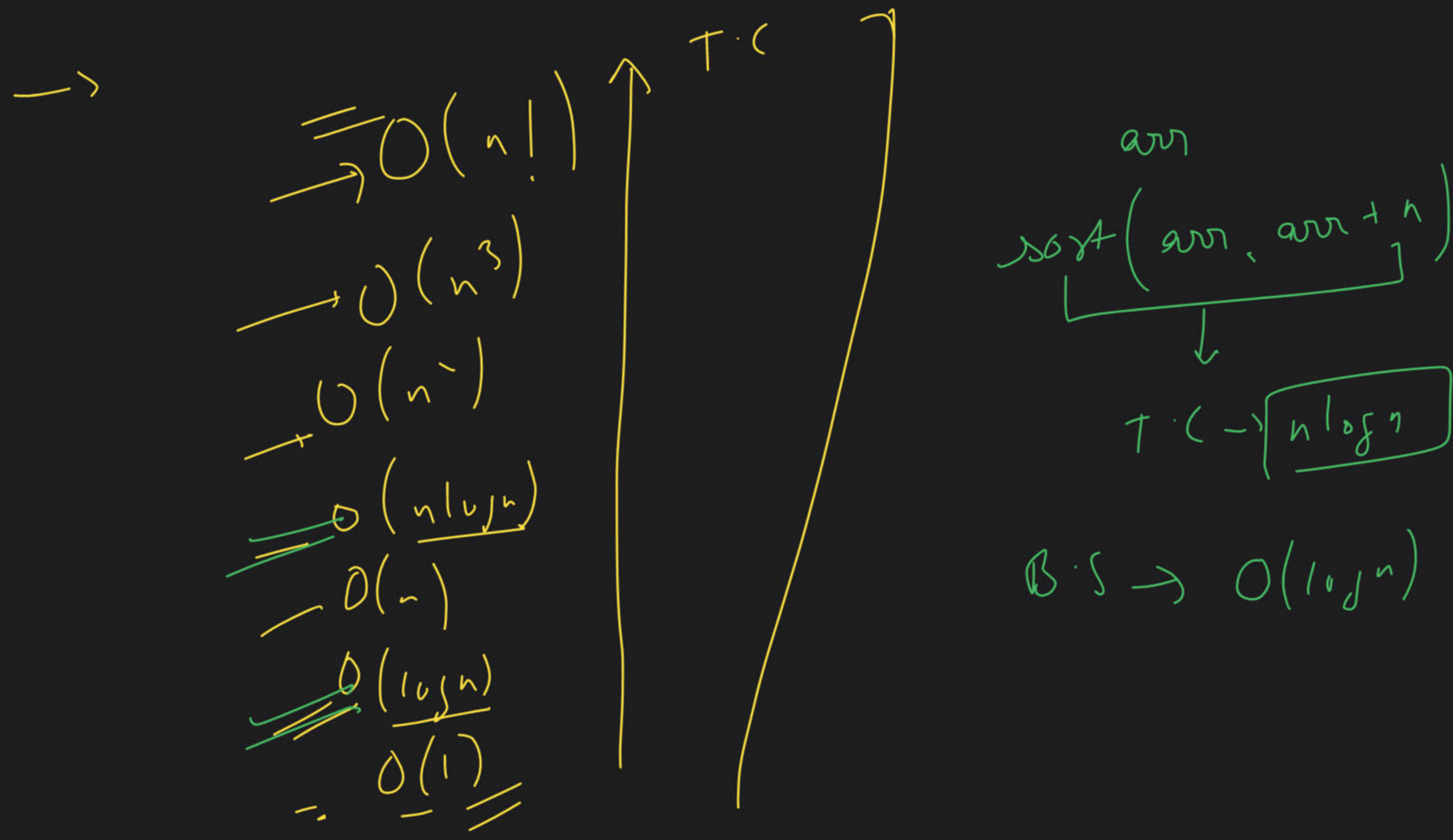
30 min

→ code → T.C / S.C

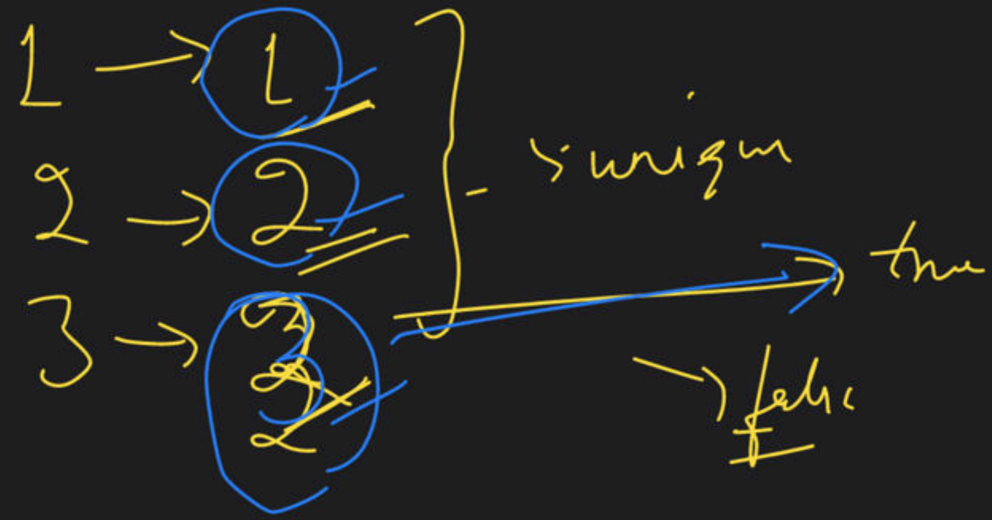
$O(n^2)$ — optimize → $O(n)$

→ $O(\log n)$

→ $O(n \log n)$


O(n!)
O(n^3)
O(n^2)
O(n log n)
O(n)
O(log n)
O(1)
T.C
arr
sort(arr, arr + n)
T.C -> n log n
B.S -> O(log n)

→ i/p → arr → { 3, 2, 2, 1, 3, 3 }



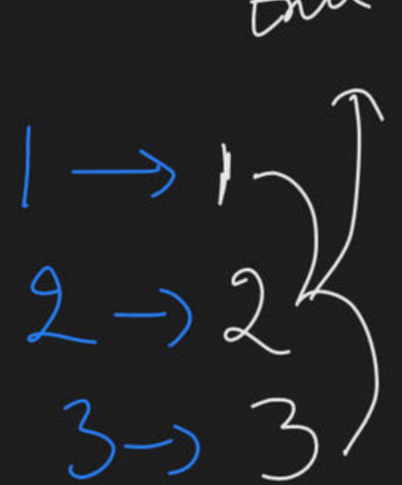
→ approach :-

→ sort { 1, 2, 2, 3, 3, 3 }

→ count occurrence

→ check occurrence

map



D.S

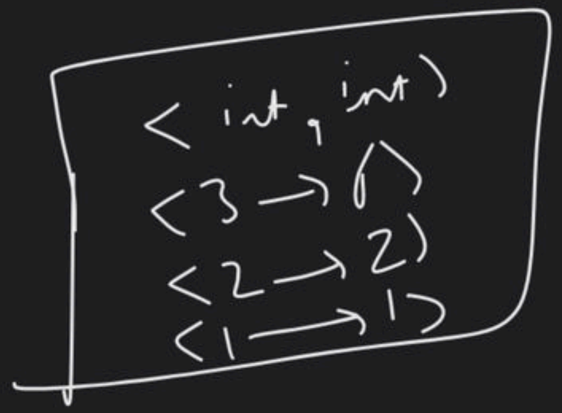
↳ map

unordered

ordered

map < int , int > m;

O(1)



→ sort (arr , arr + n);

→ arr = { 3, 1, 4, 5, 9, 13, 11 } 3+11=17

target → 14

o/p → pair

<1, 13>

pair →

<5, 9>

11+4

13

(1, 13)

⇒ brute force

search pair

check

print

for(i → 0 < 4)

for(j → i+1 < 4)

}

O(n²)

nlogn

⇒ approach :-

sort

sort(arr, arr + n)

{ 1, 3, 4, 5, 9, 11, 13 }

i i i i i

j j j j j

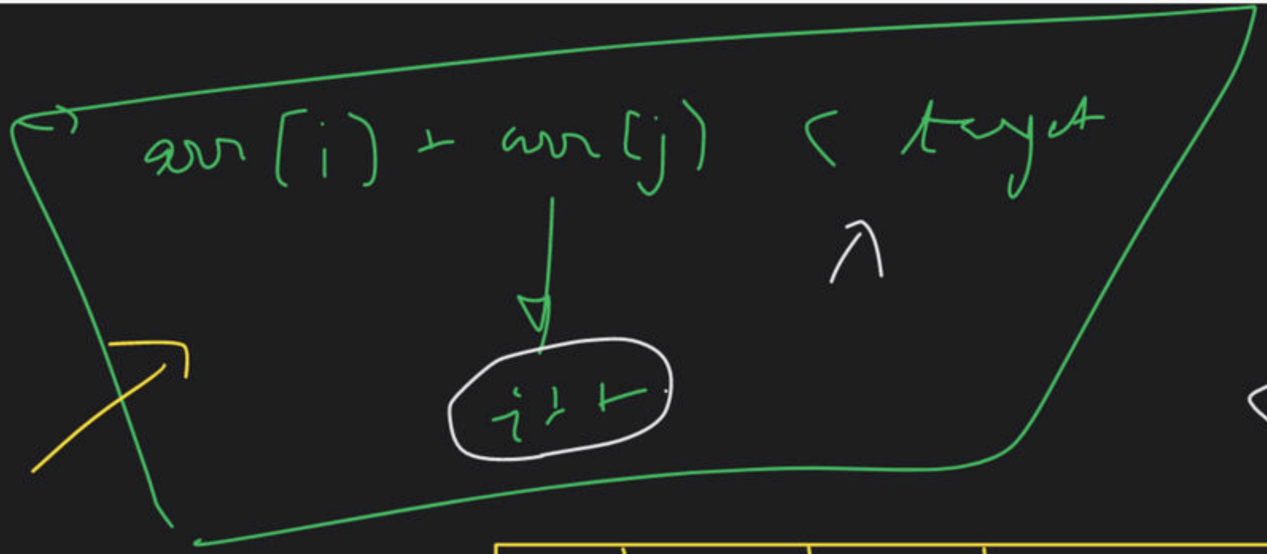
3+13 = 16

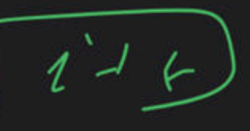
O(nlogn)

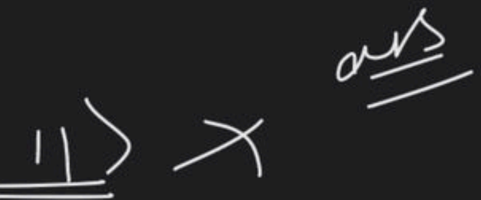
→ arr[i] + arr[j] == target

break → print pair

→ arr(i) + arr(j) > target → j--

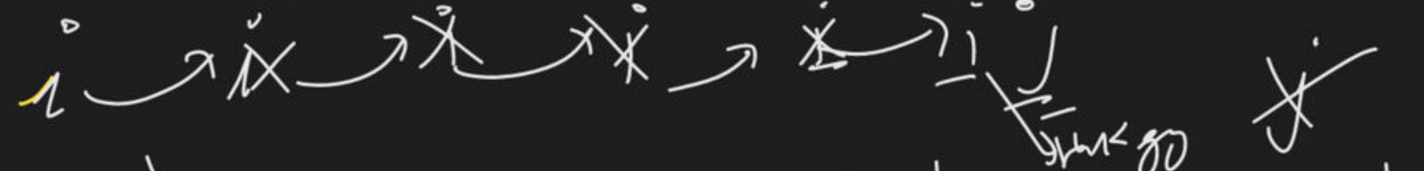

arr[i] + arr[j] < target
i++
i--
j--

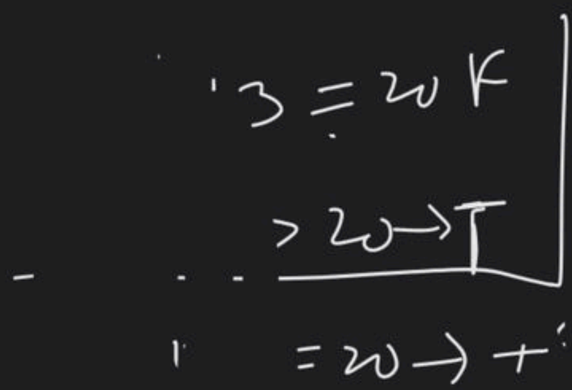

i--
j--

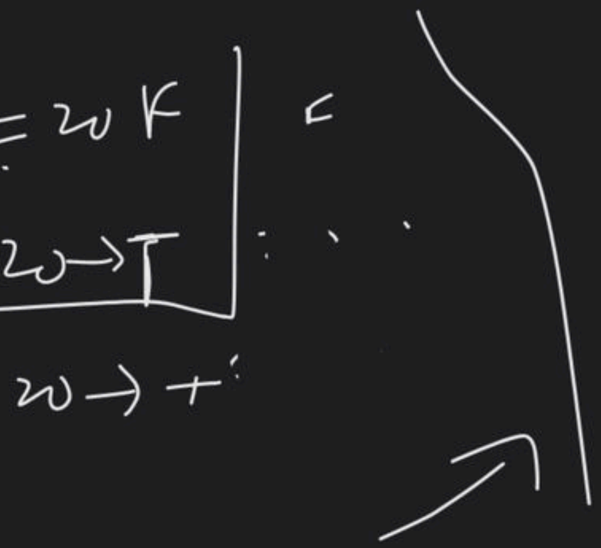

arr
<5, 11>

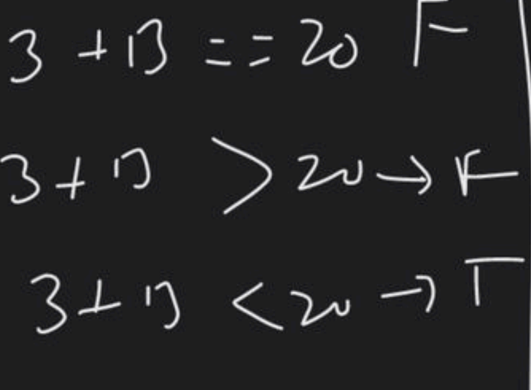

target = 20

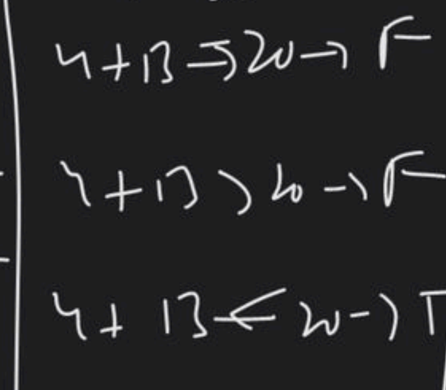

1
3
4
5
9
11
13

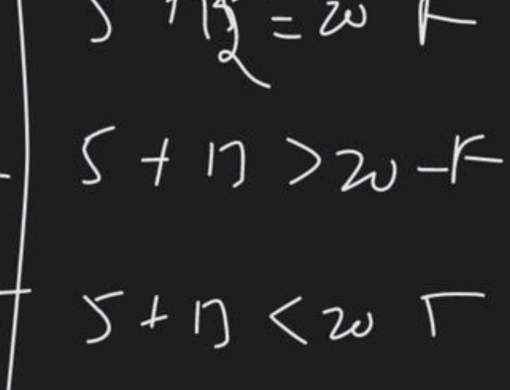



3 == 20 F
> 20 → F
= 20 → F




3 + 13 == 20 F
3 + 13 > 20 → F
3 + 13 < 20 → T


4 + 13 == 20 → F
4 + 13 > 20 → F
4 + 13 < 20 → T

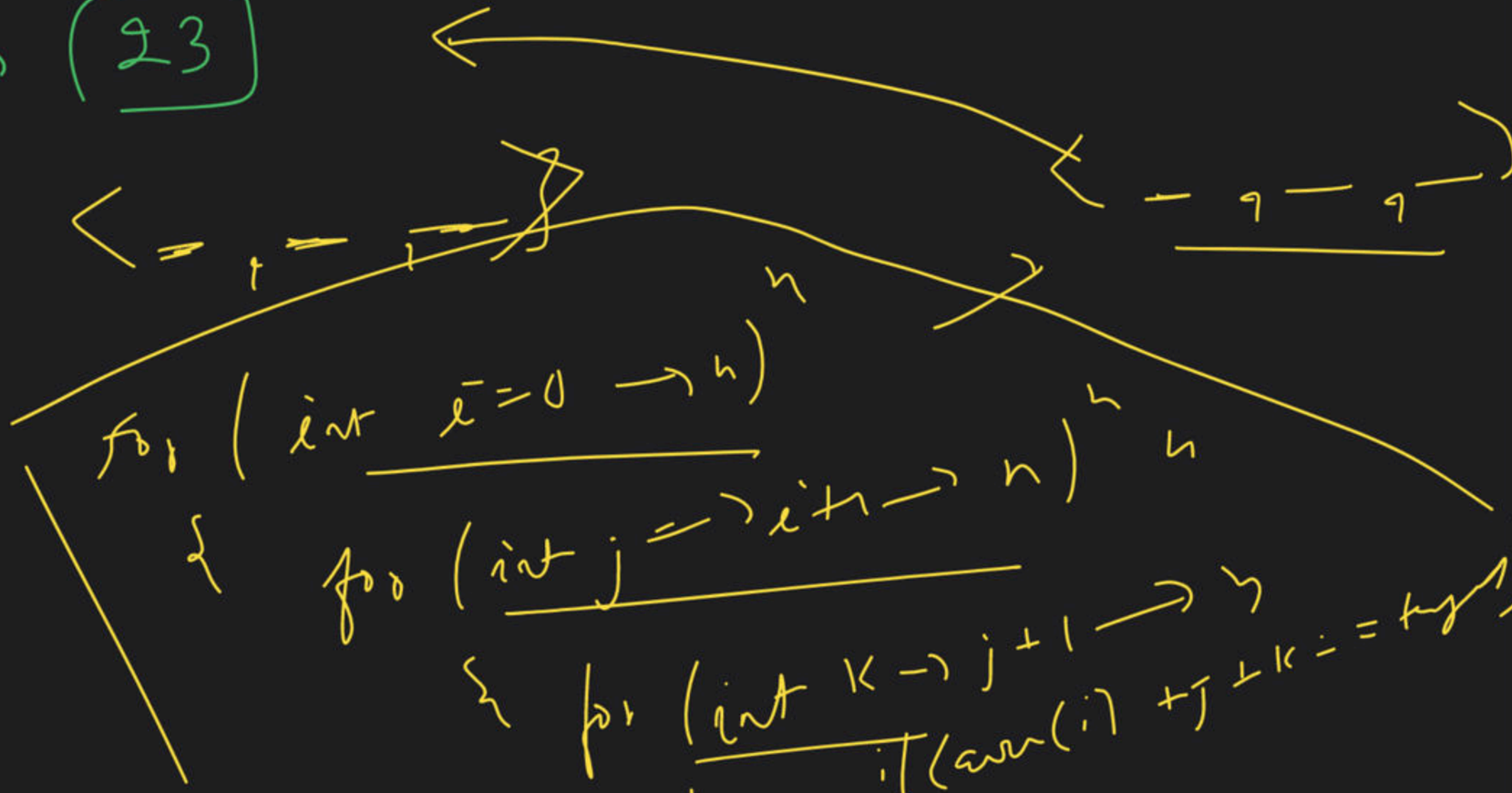

5 + 13 == 20 F
5 + 13 > 20 F
5 + 13 < 20 T

→ triplet

arr → {1, 3, 11, 6, 13, 4}

target ⇒ 23

o/p → <_, _, _>



→ Brute force

O(n³)

optimise

target

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

{

int first = i;

int newTarget = target - first;

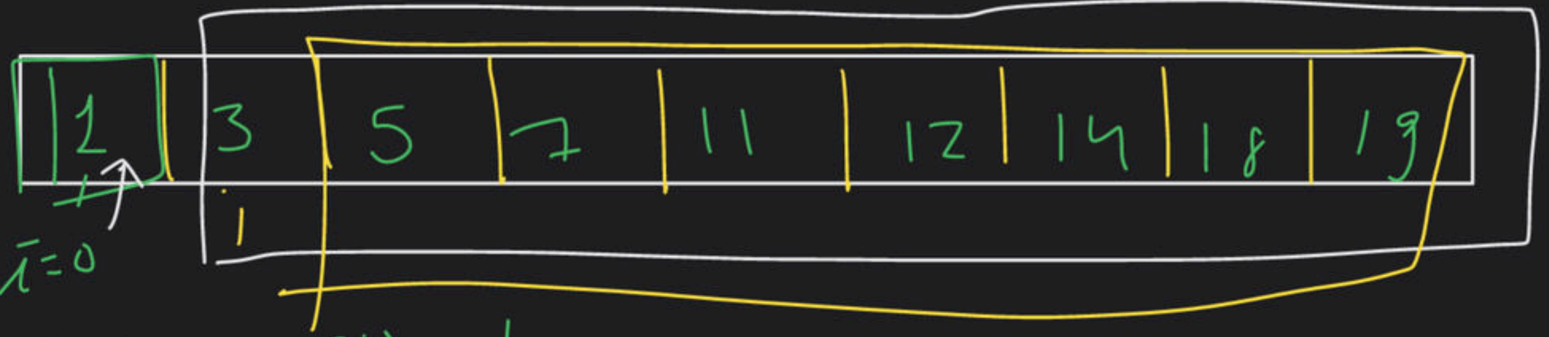
n log n

}

$h^2 \log n$

$n^2 \log n$ $\geq$ n^3

accha



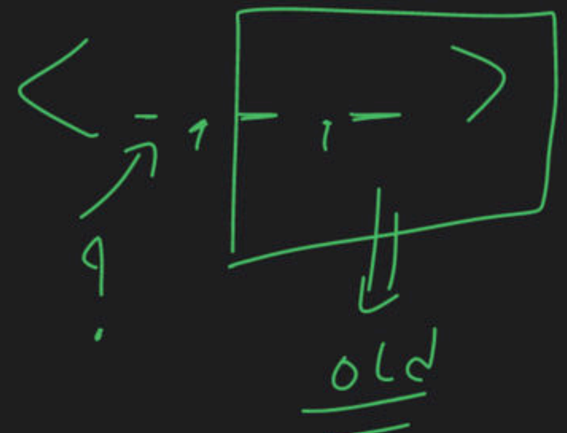
target = 37

$i = 0$, arr[i] = 1

newTarget = 37 − 1 = 36 <_, _>

$i = 1$, arr[i] = 3

newTarget = 37 − 3 = 34 <_, _>



Code, by run

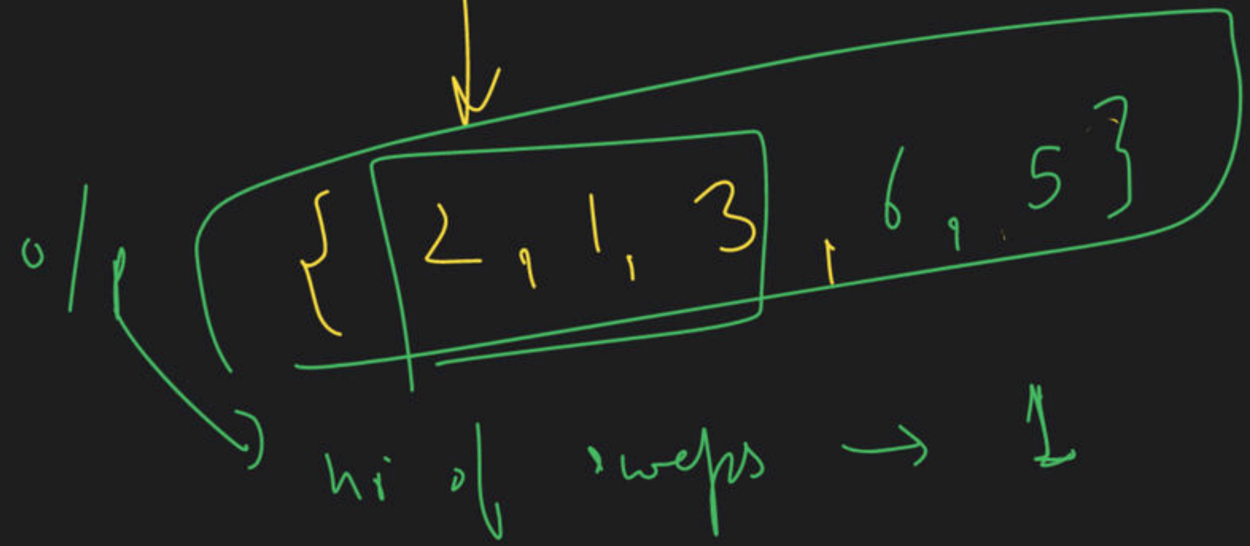
Optimise

↳ explore

Min Swaps

K = 3

<= K



o/p

No of swaps → 1

So find minimum no of swaps req to bring elements <= K together

Count = 3

for(int i=0; i<(bcA

→ 100! → ∝

↳ 158 digits

arrays → [1]

= [5]

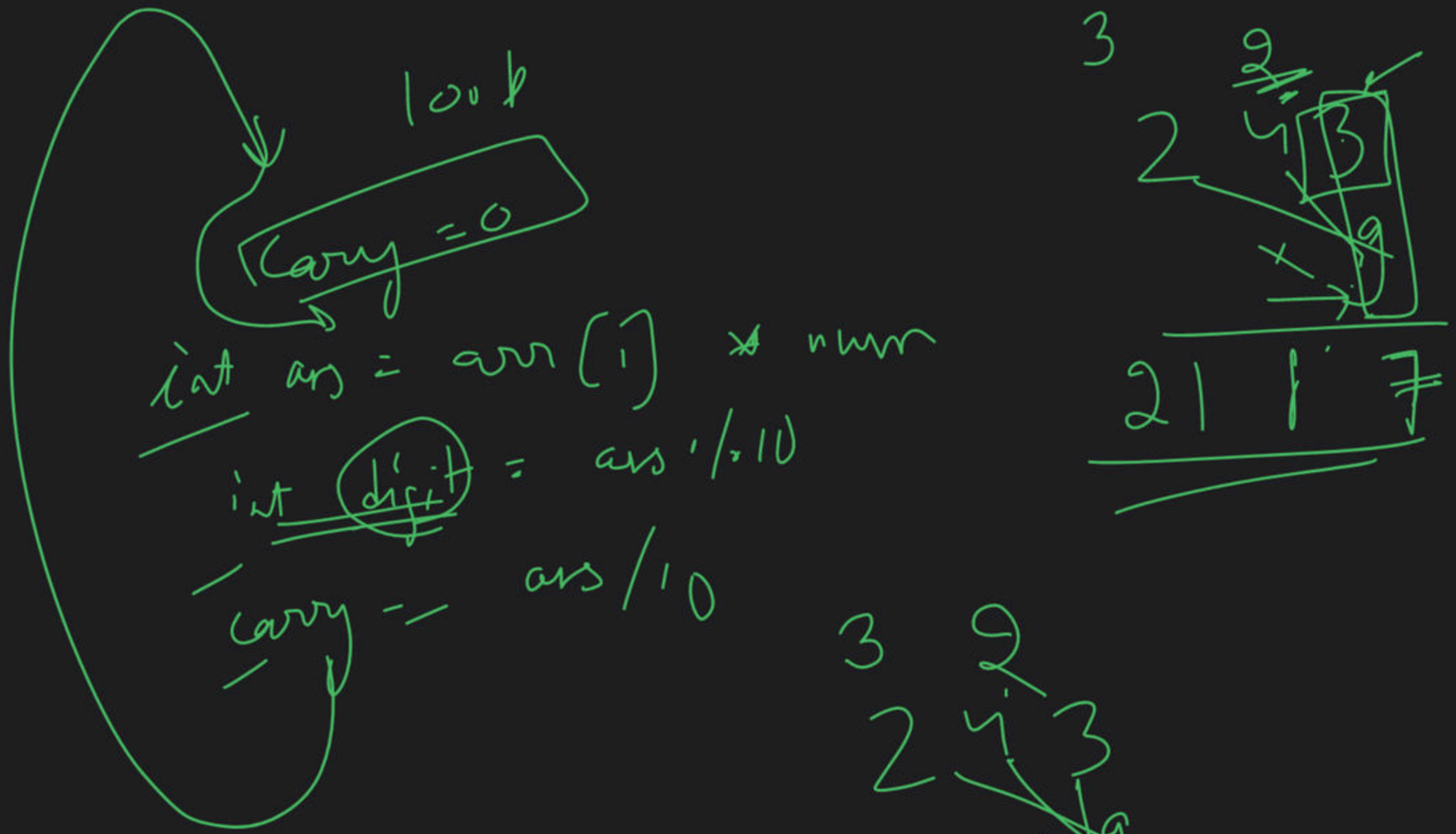
51 = [5] × [7] × 3 × 2 × 1 = [0|2]

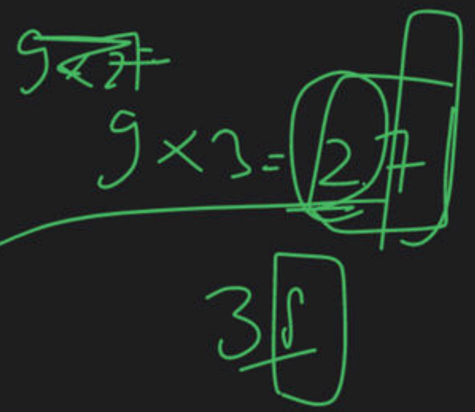
int carry = 0

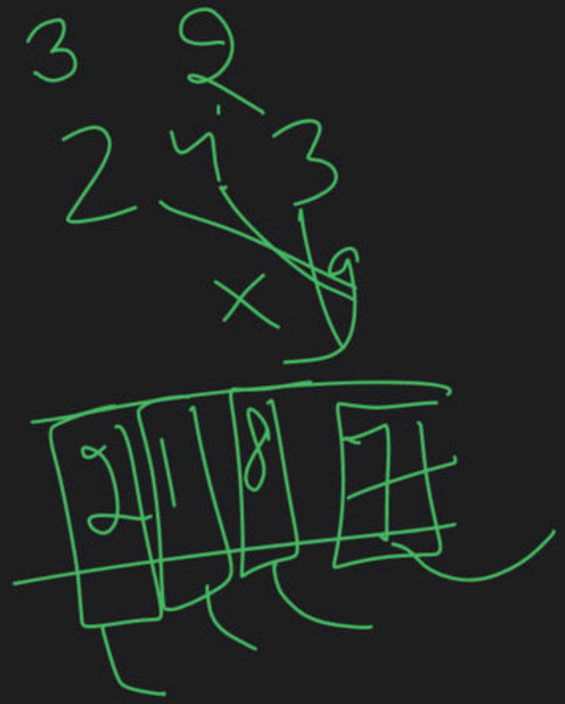
int ans = arr[i] × num + carry = [0|6]

int digit = ans % 10; = [0|2|1]

carry = ans / 10 = [0|2|1]

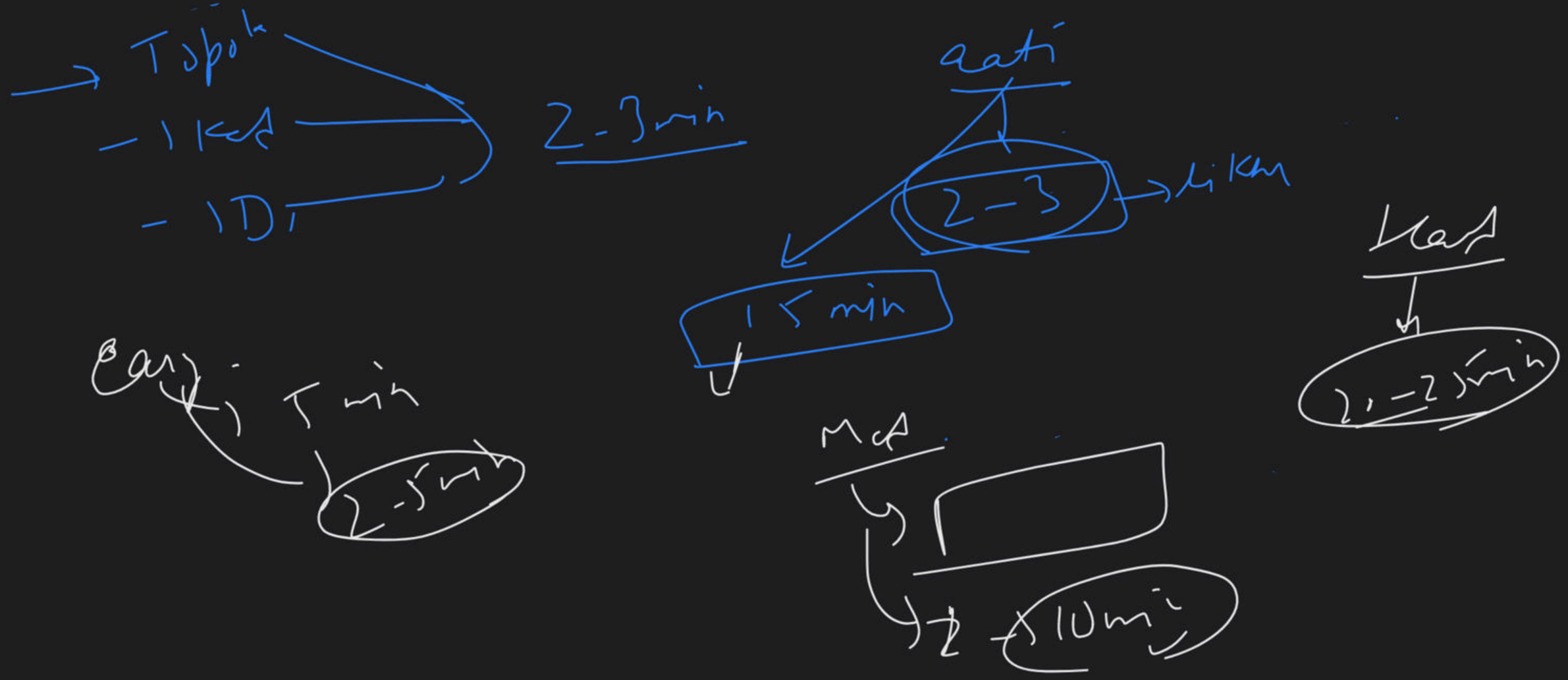


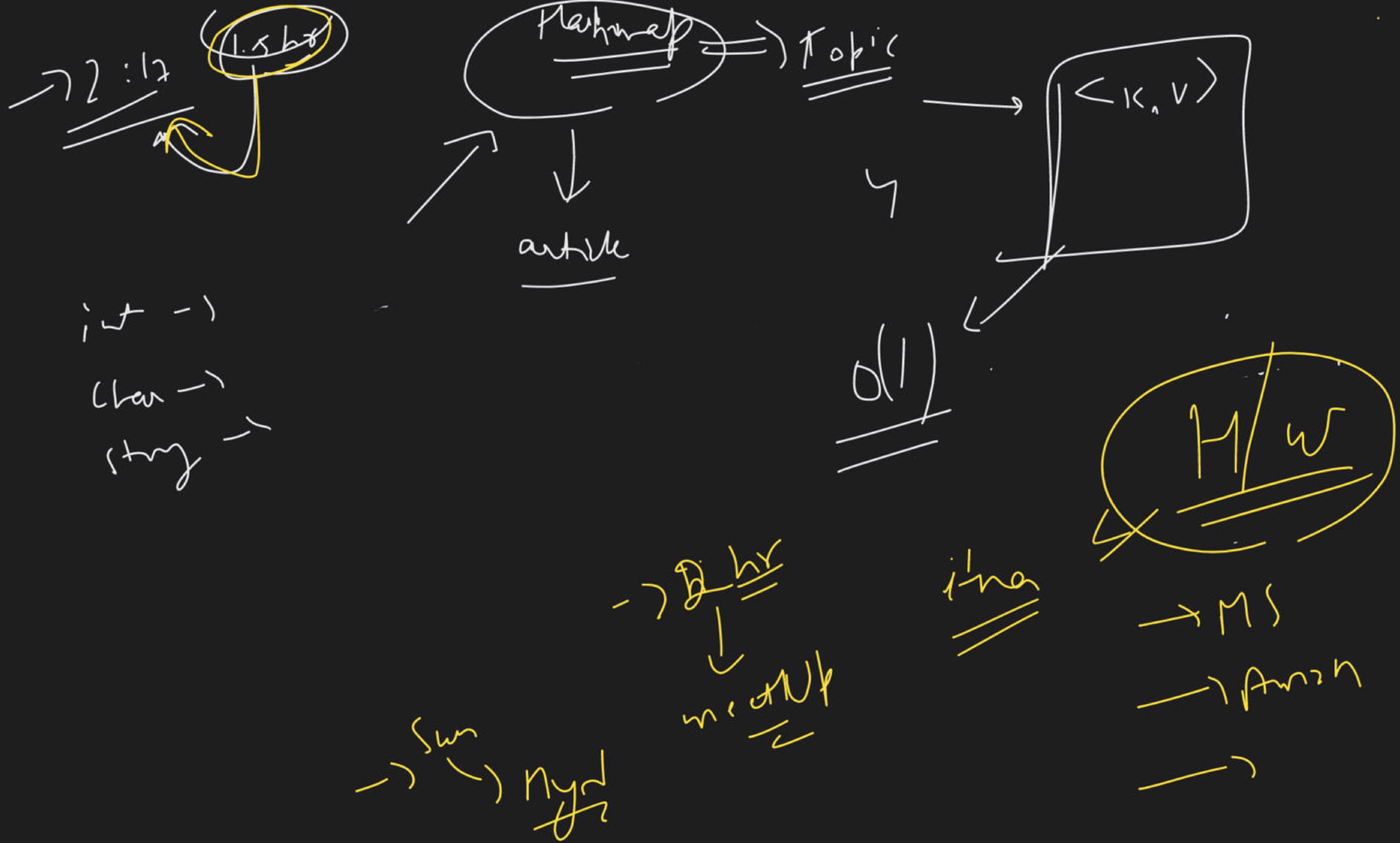




$\frac{38}{10} = 3$ $\frac{27}{10} = 2$

int arr[10000];


2-3 min
2-3
15 min
2, -2 5 min
2-5 min
10 min


2:17
1.5 hr
Hashmap
Topic
<K, V>
article
O(1)
int
char
string
H/W
2 hr
meetup
MS
Amazon
Sun
hyd