Merging

Tuesday, 31 August 2021 8:10 PM

TSP1 + TSP2

150 → 30 *

L2 →
- Recursion and backtracking
- LinkedList
- Trees

further seq:
- Arrays and Strings
- graphs
- Heap and HashMaps
- DP
- Bits
- Stacks and Ques
- searching Sorting
- tries.

- Recursion and backtracking
- LinkedList
- Trees.

LinkedList X
- Array

Trees.
- Graph.

BFS + DFS

5 = 6

TSP3 → logic → Remove.

1.5x → speed of PP Batch.

Class Schedule → 30 questions
- 7-8 Questions
- 5 ques.

Tuesday → 8:00 - 12:00

Thursday → 8:00 - 12:00

Saturday → 10 - 2 → 6:30 - 10:30 (7-11)

Sunday → 10 - 2 → 6:30 - 10:30

Pace → target complete

understanding → logic Development

Faulty Keyboard / Long Pressed Name

Tuesday, 31 August 2021 8:42 PM

Name → a b c c d

typed → a a b c c d

output → TRUE

Name → l e e l e e

typed → l l e e e l e e

output → TRUE

Name → s a e e d

typed → s s a a e d d

output → False

Name → s h r e e

type → s s h r r e

Oputi → FALSE

Name → s h r e e r e

typed → s s h r r e r e e

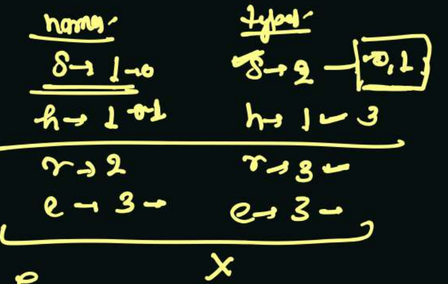
Return → False.

Brute Force (1)

Name → s h r e e r e

typed → s s h r r e e r e e e

Running condition -



HashMap✗

Name → s h r e e r e

i ↗ [i is valid && j is valid]

typed → s s h r r e e r e

Name → s h r e e] Index out of bound

typed → h r e e

Return → False

Character vs Array List < Wrapper Class >

```
while( i<length of name && j<length of typed ){
    if( name[i] == typed[j] ){
        i++;
        j++;
    } else if( name[i-1] == typed[j] ){
        j++;
    } else {
        return false;
    }
}
```

name → a a b b c c d d e e f → length 10

typed → a a b c c d d e f → 8

name.length() > typed.length()

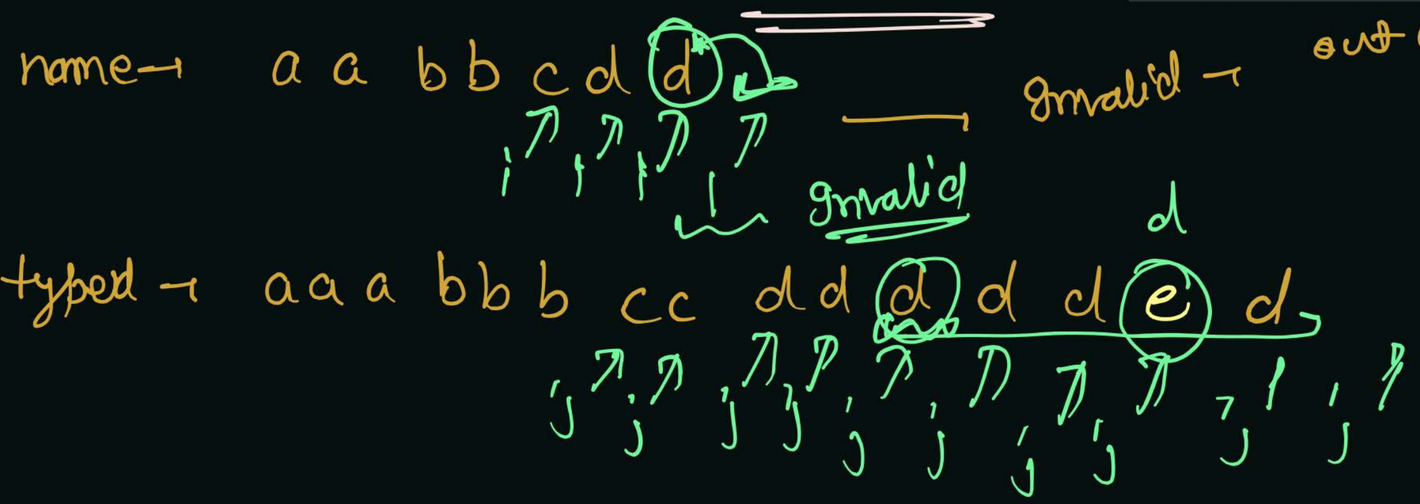
Impossible to Extract String of name from typed. → return FALSE

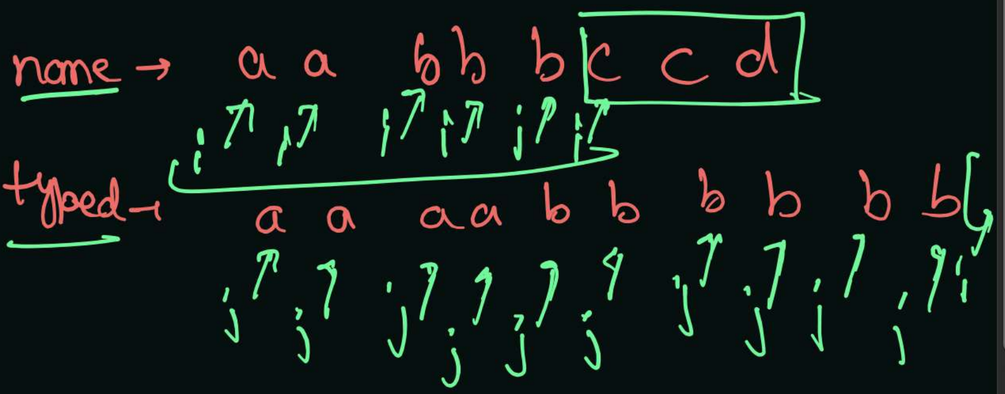
```java
// leetcode 925 https://leetcode.com/problems/long-pressed-name/
public boolean isLongPressedName(String name, String typed) {
    int i = 0; // name string
    int j = 0; // typed string

    if(name.length() > typed.length()) return false;

    while(i < name.length() && j < typed.length()) {
        if(name.charAt(i) == typed.charAt(j)) {
            i++;
            j++;
        } else if(0 <= i - 1 && name.charAt(i - 1) == typed.charAt(j)) {
            j++;
        } else {
            return false;
        }
    }
    while(j < typed.length()) {
        if(typed.charAt(j) != name.charAt(i - 1)) {
            return false;
        }
        j++;
    }
    return true;
}
```

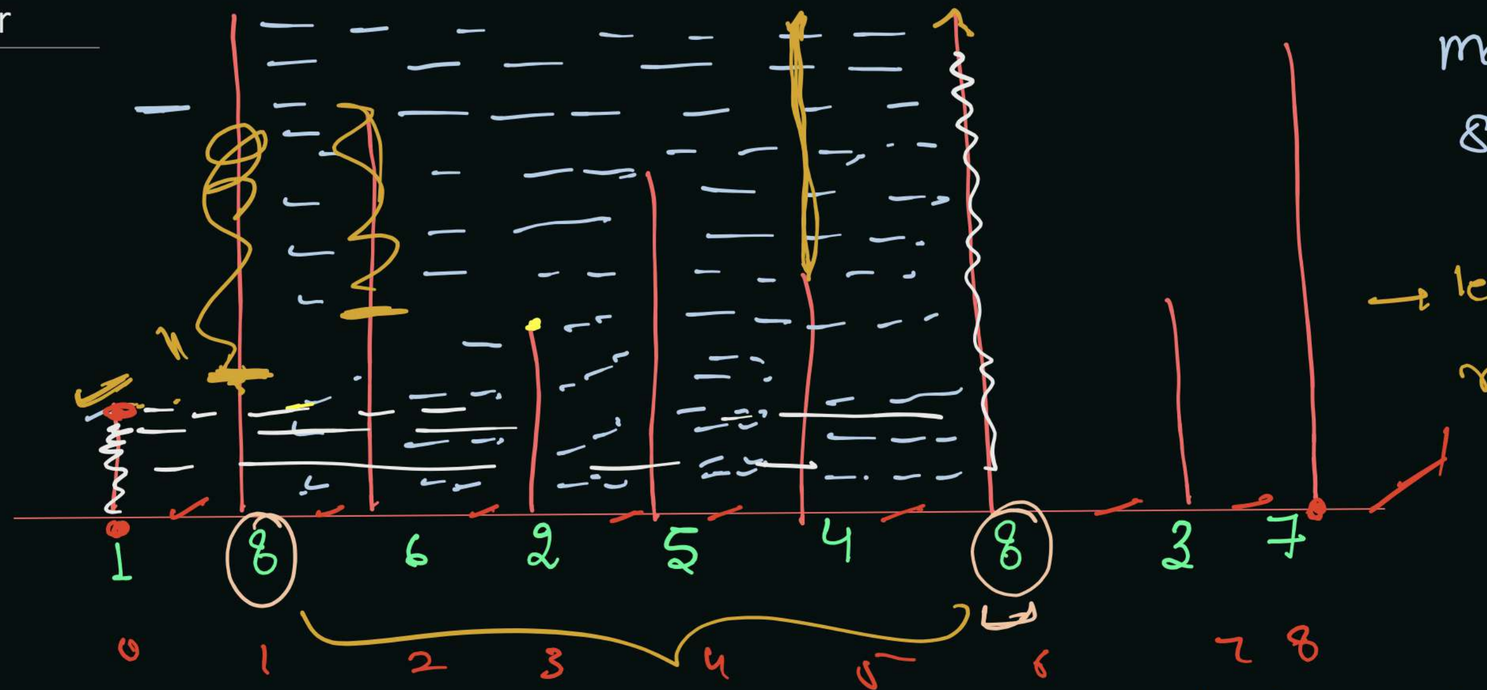
out of loop





Container With Most Water

Tuesday, 31 August 2021 8:42 PM



max water store??

→ left max, right max → width depends

↳ may be result pair with skip

Brute — Method-0 Try to solve all pairs and maximise result.

(i, j) pair

0-1 1-2 2-... ... 6-7 7-8
0-2 1-3 ... 1 6-8
0-3 1-4 1
0-4 1-5 1 1
0-5 1-... 1
0-6 ⋮ 1
0-7 1-8
0-8

Time → $O(n^2)$

area = l × b × h. [volume of water]

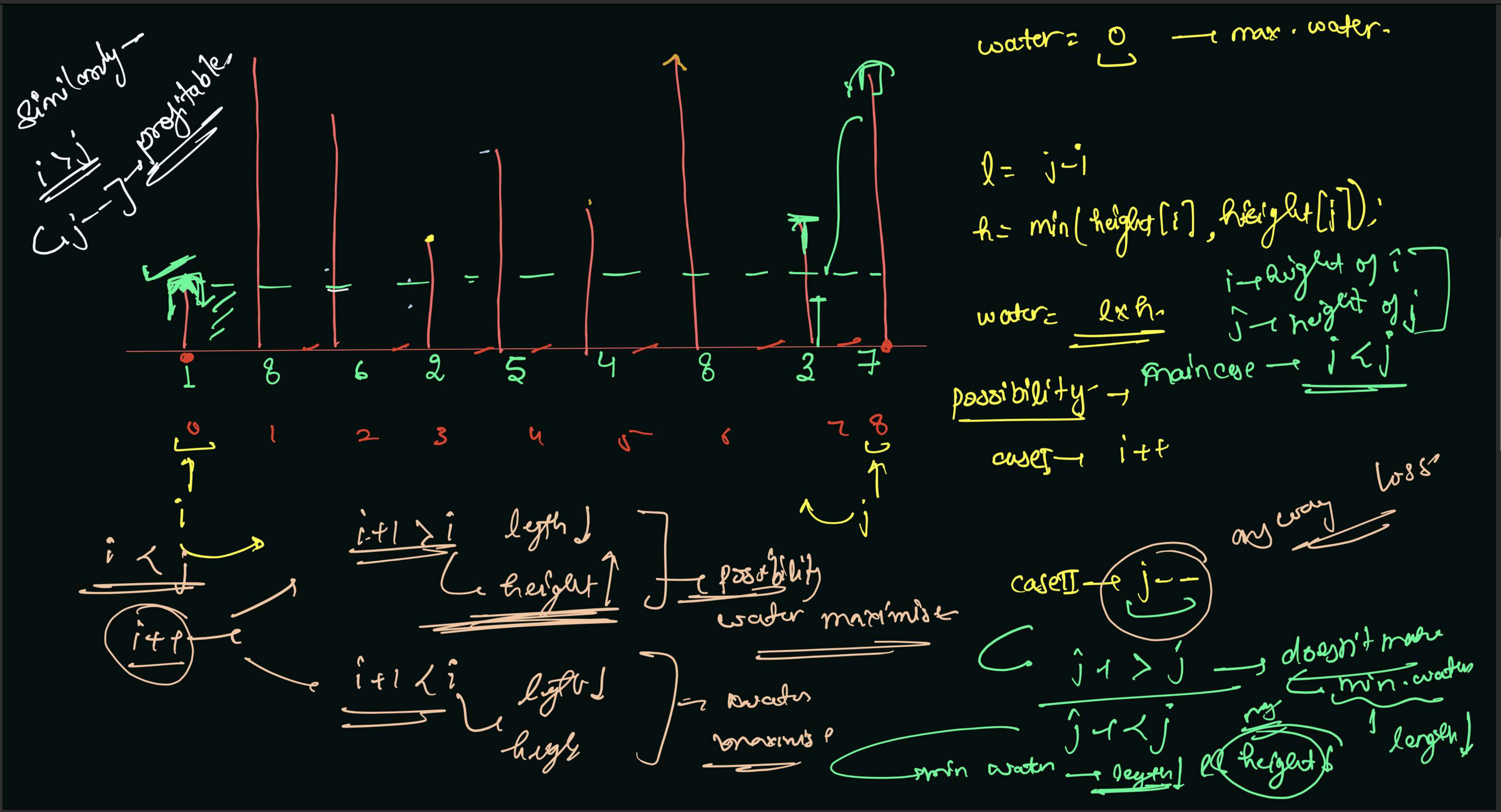
b = 1

area = l × h

= (j − i) × min(,)

l = j − i

height = min(height[i], height[j])


Similarly
i < j
j--, j → profitable
1 8 6 2 5 4 8 3 7
0 1 2 3 4 5 6 7 8
i
j
i < j
i+1
i+1 > i
length ↓
height ↑
possibility
water maximise
i+1 < i
length ↓
height ↓
water minimise
water = 0 → max. water.
l = j - i
h = min(height[i], height[j]);
water = l × h
i → height of i
j → height of j
possibility → increase → i < j
case I → i++
any way loss
case II → j--
j-1 > j → doesn't make min. water
j-1 < j
min water → decrease
may height ↓
length ↓

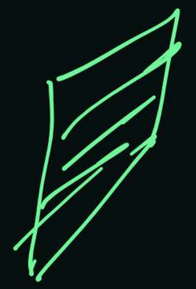
ht →	1	8	6	2	5	4	8	3	7
indx →	0	1	2	3	4	5	6	7	8

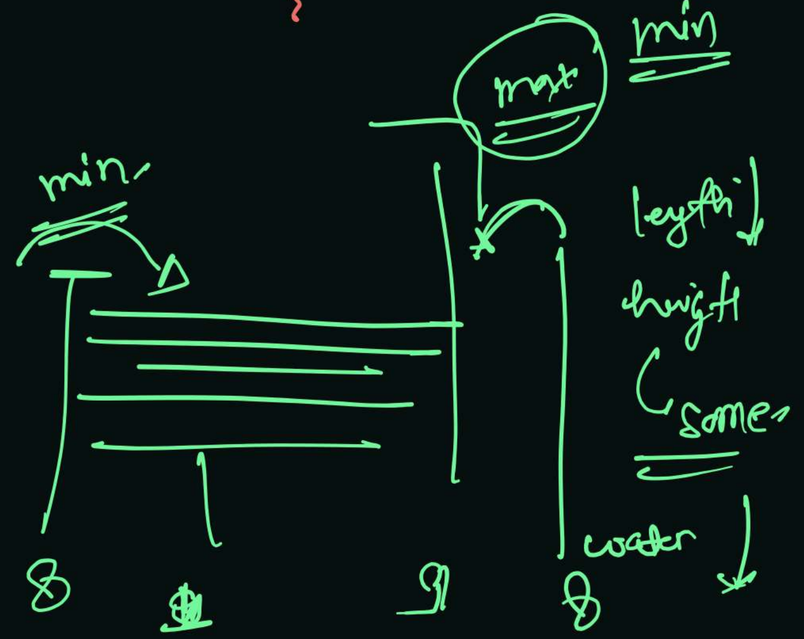
i, j

```
if(ht[i] < ht[j]){
    i++;
} else {
    j--;
}
```

l = j − i =	~~8~~	~~7~~	~~6~~	~~5~~	~~4~~	~~3~~	~~2~~	1
h = min(ht[i], ht[j]) =	~~1~~	~~7~~	~~3~~	~~8~~	~~4~~	~~5~~	~~2~~	6
water = l × h =	~~8~~	~~49~~	~~18~~	~~40~~	~~16~~	~~15~~	~~4~~	6

max water = ~~0~~ ~~8~~ 49 → Result





Square of Sorted Array

Tuesday, 31 August 2021 8:42 PM

Sorted array →

0	2	4	7	9
0	1	2	3	4 ← Index

+ve Number
↳ Single Iteration
↳ O(1) Space
↳ O(n) Time

O/P → Sorted Squared array →

0	4	16	49	81

Square of Every Element → Sorted

arr →

-4	-3	-2	0	2	3	5
16	9	4	0	4	9	25

Square →

O(n) + O(nlogn) → O(nlogn)

↳ Sorted ✗

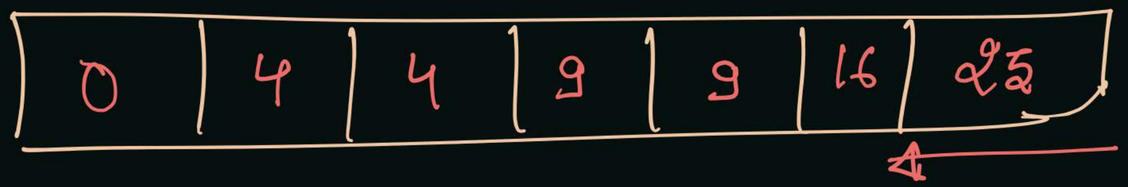
↳ Sort → nlogn

Sort → nlogn

Time → O(nlogn)

Allowed Time → O(n) ×

arr → -4 -3 -2 0 2 3 5



left

Square

most +ve val

k

max Square

↳ most -ve Element

↳ most +ve Elem

val1 = arr[i];

val2 = arr[j];

for(int k=length-1; k>=0; k--) {

if(val1 * val1 > val2 * val2) {

i++; place → arr[k] = val2 * val2;

} else {

j--; place → arr[k] = val2 * val2;

}

}

Majority Element - I

Tuesday, 31 August 2021 8:42 PM

array → 2 3 4 3 3 5 3 3 2

Method → 0 Travel on array, assume every element as valid candidate for majority and check its occurence in array,

Time → $O(n^2)$

Space → $O(1)$

Method → 1 – Maintain a frequency map, and check if freq. is greater than $n/2$ then element is majority Element.

Time → $O(n)$

Space → $O(n)$

hashMap → <Integer, Integer>

Key: Element, Value: freq.

Proof of Every time complexity –

$O(n^2)$

– $O(\log)$

→ $O(n \log n)$

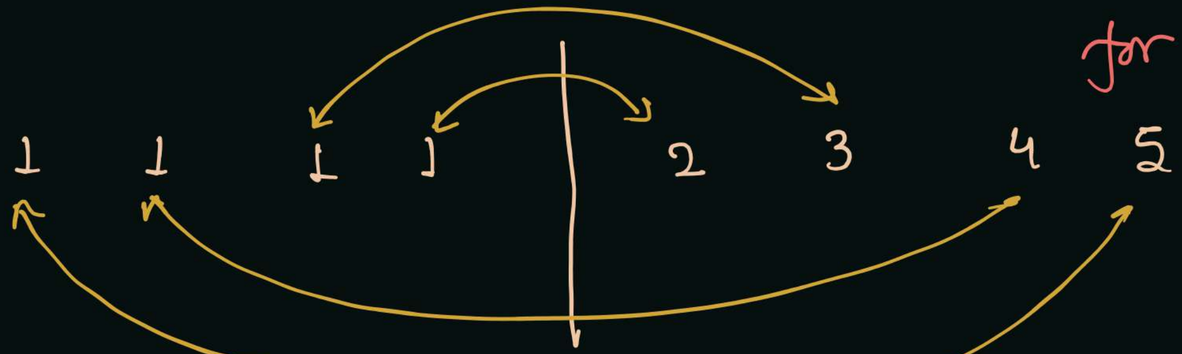
Method 2 (optimise) → pair up distinct Elements and find unpaired valid candidate for majority Elements.

array → size = 10 Majority → 6 space occupied by an Element

pairing technique

unpaired → valid candidate for majority Element

array → [1 1 1 1 1 1 2 3 4 5]



Unpaired → valid candidate for majority

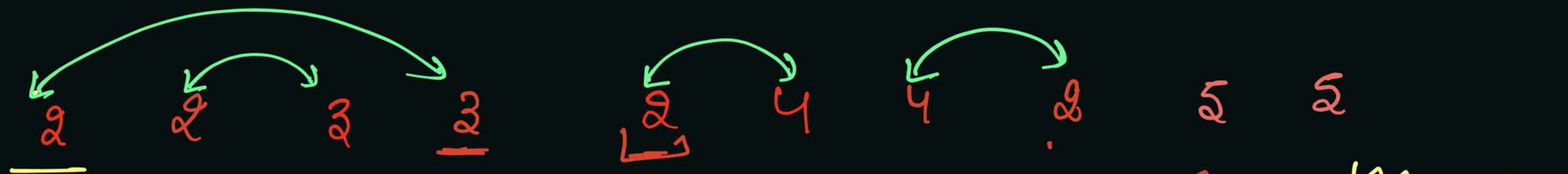
MOORE'S VOTING Algorithm →

Steps → (simple)

① pair up distinct Elements and find a valid cand.

② check if candidate have freq. more than $n/2$ times it is Majority Ele.

MOORE'S VOTING Algorithm



val = 2 4 2 val

count = 1 2 1 0 1 0 1 0 1 0 2

count → some Element Count

val → 5
count = 2

```
if( arr[i] == val) {
    Count++;
}else {
    if(count > 0) {          // pair up.
        Count--;             → count = 0
    }else {
        val = arr[i];
        count = 1;
    }
}i++;
```

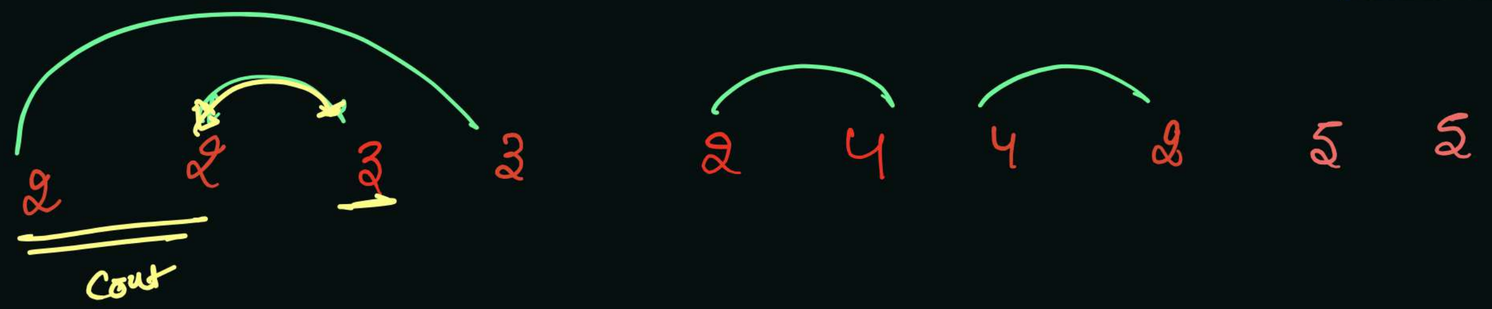
NOTE: If majority is Exist in array then it is 'val'.
Otherwise no majority Elemt Exist

val → A valid candidate for majority Elemt

After that →

check if freq of val in arr is more than $n/2$ or not

val is majority → doesn't Exist



Count = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ ~~0~~ ~~1~~ (2)

val = ~~1~~ ~~2~~ ~~2~~ ~~1~~ 5

```
if(count==0) {
  count=1;
  val=arr[i];
} else {
  if (arr[i] == val){
    count++;
  } else {
    count--;
  }
}
```

→ valid candidate is (5)

↳ Freq. of 5 is not more than N/2

→ so majority doesn't exist

Majority - I

$n/2$

majority - II

$n/3$

majority - III

n/k

$K > 3$

Final → ↳ HashMap