Today's agenda

- first missing Positive
- Majority element
- majority element - 2
- Next greater element 3

leetcode 41: First missing Positive → {Amazon, microsoft, APPle}

↳ Given arr[N], find first missing natural number.

{1, 2, 3, 4, 5 ...}

Ex: arr[5]: 3 -2 1 2 7 → 4

arr[7]: -8 2 6 4 -7 1 3 → 5

arr[6]: 2 1 6 4 3 5 → 7

arr[5]: -4 8 3 -1 0 → 1

arr[4]: 2 1 4 3 → 5

arr[3]: 1 2 1 → 3

arr[4]: -2 -3 -8 -7 → 1

//Ideal

↳ Sort the array

arr[5]: 3 -2 1 2 7

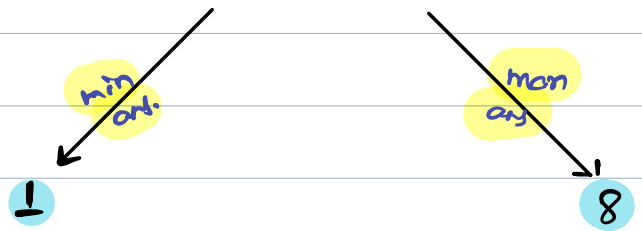
⇓

{-2 1 2 3 7} → 4 (i)

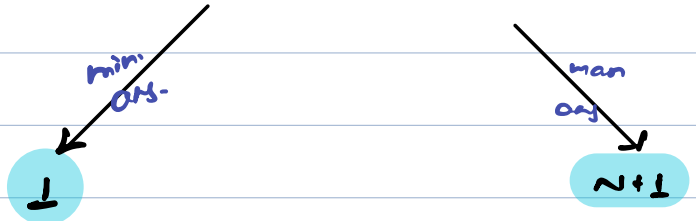
T.C: O(NlogN) + O(N) ≈ O(NlogN)

// idea2

obs1: arr[i]: 1 2 3 4 5 6 7

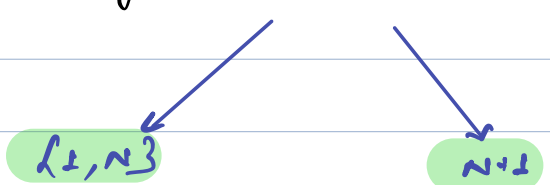


arr[N]: 1 2 3 _ _ _ _ N



→ ans range → {1, N+1}

obs2: Answer range → {1, N+1}



arr[N]: 1 2 ~~3~~ -3 _ _ _ _ N-1 N

Ex: arr[8]:

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

answer range: {1, 9}

myanswerange: {1, 8}

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

↳ How to do this mapping → Swapping

{1, 8}

↳ arr[8]:

0	1	2	3	4	5	6	7
~~4~~ ~~6~~ 1	2	~~-7~~ 3	~~6~~ 4	9	~~1~~ 6	-8	~~8~~ -7

i

↳ any: 5

arr[0] = 4 ⇒ index = 3, Swap(0, 3)

arr[0] = 6 ⇒ index = 5, Swap(0, 5)

arr[0] = 1 ⇒ index = 0, increment i

arr[1] = 2 ⇒ index = 1, increment i

arr[2] = -7 ⇒ irrelevant, increment i

arr[3] = 4 ⇒ index = 3, increment i

arr[4] = 9 ⇒ irrelevant, increment i

⋮

arr[7] = 3 ⇒ index = 2, Swap(7, 2)

// Pseudo code

→ {1, N+1} → {0, N}

```
int missingPositiveInteger(int arr[N]){
    int i=0;
    while (i < N){                                   // irrelevant | already solved
        if(arr[i]<1 || arr[i]>N || i==arr[i]-1)
            i++;
        }
        else{
            int idx = arr[i]-1;
            if(arr[i]==arr[idx]){i++;}
            else{ swap(i, idx); }
        }
    }

    for(int i=0; i<N; i++){
        if (i != arr[i]-1){return i+1;}
    }

    return N+1;
}
```

T.C: O(N)

S.C: O(1)

edge case: → {3, 5}

arr[5] = { 4 (i=0), 1, 3, 5 (i=3), 2 (i=4) }

→ {1, 8}

```
int missingPositiveInteger(int arr[N]){
    int i=0;

    while (i<N){
        if(arr[i]<1 || arr[i]>N || i==arr[i]-1)
            i++;
        }
        else{
            int idx= arr[i]-1;
            swap(i, idx);
        }
    }

    for (int i=0; i<N; i++){
        if (i != arr[i]-1){return i+1;}
    }
    return N+1;
}
```

(arr[i]<1 || arr[i]>N → irrelevant; i==arr[i]-1 → already solved)

arr[8]: ~~4~~ ~~6~~ 1, 2, ~~-7~~ 3, ~~8~~ 4, 9, ~~1~~ 6, -8, ~~3~~ -7

∴ ans=5

max no. swapping : N

edge case: arr[5] = { ~~4~~ ~~3~~ 3, 1, ~~2~~ ~~5~~ 3, ~~3~~ 4, 2 }

idx=2

i=0

leetcode 169: Majority Element → { Amazon, Apple }

↳ Given arr[N], find majority element

↳ An ele with freq > N/2

↳ majority element is definitely present

Ex1: arr[6] = { 1 2 1 6 1 1 } ⇒ ans: 1

↳ 6/2 → >3 occ.

Ex2: arr[9] = { 3 4 4 8 4 9 4 3 4 } ⇒ ans: 4

↳ 9/2 = >4 occ.

Ex3: arr[12] = { 3 3 4 6 1 3 2 5 3 3 3 3 }

↳ 12/2 = >6 occ. ↳ ans: 3

Ex4: arr[10] = { 4 6 5 3 4 5 4 4 4 8 }

↳ 10/2 = >5 occ.

↙ invalid input

↳ no majority element

Note: majority element is present in the array.

//idea1

↳ use nested loop → T.C: $O(N^2)$

//idea2

↳ Sort the array → T.C: $O(N \log N)$

//idea3

↳ use Hashmap to count → T.C: $O(N)$, S.C: $O(N)$

//idea 4 → expected T.C: O(N), S.C: O(1)

↳ Boyer moore voting algo

Obs 1 :

↳ How many majority elements can be Present in one array? → Atmost 1

$freq(maj1) > N/2$

$freq(maj2) > N/2$

$> \frac{N}{2} + > N/2 = > N$ → In total you have only N elements in the array.

Comb! → 15 People to Participate → >7

Obs 2 :

Subhesh :

Nishant :

Abhishek :

Comb 2

→ 15 People to Participate → >7

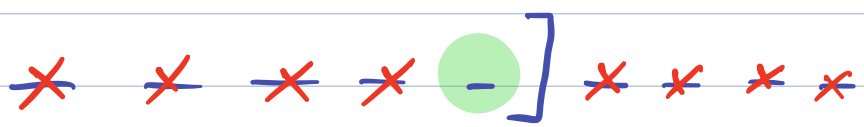
Subhesh: 

Nishant: 

Abhishek: 

>4 occ

→ arr[9] = { 6 6 3 8 8 6 9 6 6 }
0 1 2 3 4 5 6 7 8



↳ if we cancel out 1 majority element no. with 1 non-majority element number, the non-cancelled number at the end is going to be majority elem.

Obs 3:

Is you don't need to know majority element.

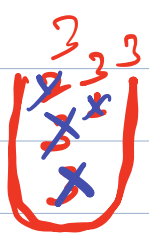
⇓

you Cancel out 2 distinct elements,

(majority, non-majority element)

Summary: Cancel out 2 distinct elements one by one, the uncancelled element is going to be majority element.

Ex: arr[11]: { 3 3 4 6 1 3 2 5 3 3 3 }



i	ele	freq	
0	3	1	
1	3	2	⇒ ans: 3
2	3	1	
3	3	0	
4	1	1	
5	1	0	
6	2	1	
7	2	0	
8	3	1	
9	3	2	

10 3 3

//Pseudo Code

```
int majorityElement(int arr[N]){

    int val = arr[0];
    int count = 1;

    for(int i=1; i<N; i++){
        if(arr[i] == val){
            count++;
        }
        else{
            if(count==0){
                val = arr[i];
                count = 1;
            }
            else{
                count--;
            }
        }
    }

    return val;
}
```

T.C : O(N)

S.C : O(1)

Q) **majority element-2**

↳ Given an **Integer array** of size **n**, find all **elements** that **appear** more than $\lfloor n/3 \rfloor$ times.

(not necessarily majority element is present.)

Ex: 0 1 2 3 4 5 6 7 8 9 10
2 3 2 4 2 3 5 3 3 2 4 → {2, 3}

Ex: 0 1 2 3 4 5 6 7 8 9 10
2 3 4 5 6 2 4 6 5 4 2 → { }

// Boyer moore voting idea

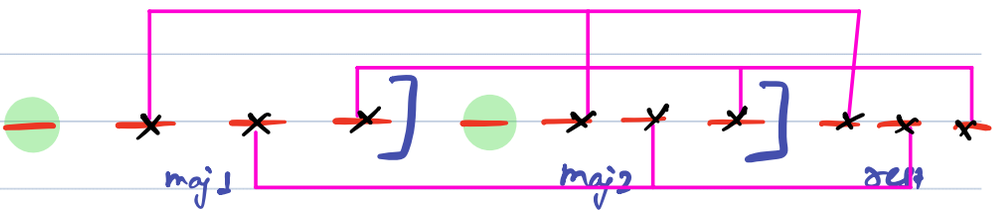
Obs 1: How many **majority elements** can be **present in one array?** → **Atmax 2**

total length = n

$$> \frac{n}{3} + > \frac{n}{3} + > \frac{n}{3}$$

$$= > n \quad \text{XX}$$

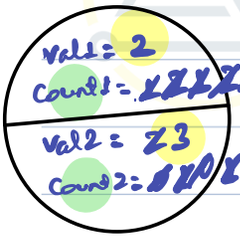
	0	1	2	3	4	5	6	7	8	9	10
Ex:	2	3	2	4	2	3	5	3	3	2	4



→ Cancelling distinct 3 elements at once.
↳ create distinct triplets.

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

v_i



{2, 3, 4} {2, 3, 4}
{2, 3, 5}

```
ArrayList<Integer> ls = new ArrayList<>();
List<Integer> ls = new ArrayList<>();
```

//Pseudo code

```
List<Integer> majorityElement(int[] nums){
    List<Integer> ans = new ArrayList<>();

    int val1 = nums[0];
    int count1 = 1;
    int val2 = -10;
    int count2 = 0;

T.c: O(N)
S.c: O(1)
    for(int i=1; i<nums.length; i++){
        if(nums[i]==val1){
            count1++;
        }
        else if(nums[i]==val2){
            count2++;
        }
        else if(count1==0){
            val1 = nums[i];
            count1 = 1;
        }
        else if(count2==0){
            val2 = nums[i];
            count2 = 1;
        }
        else{
            count1--;
            count2--;
        }
    }
```

```
int c1 = 0;
for (int i=0; i<n; i++) {
    if (nums[i] == val1) { c1++; }
}
if (c1 > n/3) { ans.add(val1); }

int c2 = 0;
for (int i=0; i<n; i++) {
    if (nums[i] == val2) { c2++; }
}
if (c2 > n/3) { ans.add(val2); }

return ans;
}
```


Tracing

```
int val1 = nums[0];
int count1 = 1;
int val2 = -10;
int count2 = 0;

for (int i = 1; i < nums.length; i++) {
    if (nums[i] == val1) {
        count1++;
    }
    else if (nums[i] == val2) {
        count2++;
    }
    else if (count1 == 0) {
        val1 = nums[i];
        count1 = 1;
    }
    else if (count2 == 0) {
        val2 = nums[i];
        count2 = 1;
    }
    else {
        count1--;
        count2--;
    }
}
```

	0	1	2	3	4	5	6	7 (i)	8	9	10
arr:	2	3	2	4	2	3	5	3	3	2	4

val1 = 2

count1 = ~~0~~ ~~1~~ ~~2~~ 1

val2 = ~~-10~~ 3

count2 = ~~0~~ ~~1~~ ~~0~~ 0

{2, 3, 4}

{2, 3, 5}

Q) Next greater element -3

↳ Given a character array with digits of a number at indexes. Find the smallest integer which has exactly the same digit and the value is greater than number in ch[].

Ex: 1 2 → 21

Ex: 9 7 5 3 2 → {-1}

//idea1 → int n

↳ iterate from n+1 till you get the answer & check for every integer if occ are same or not.

//idea 2

ch[]: 1 4 8 4 9 7 5 3 2

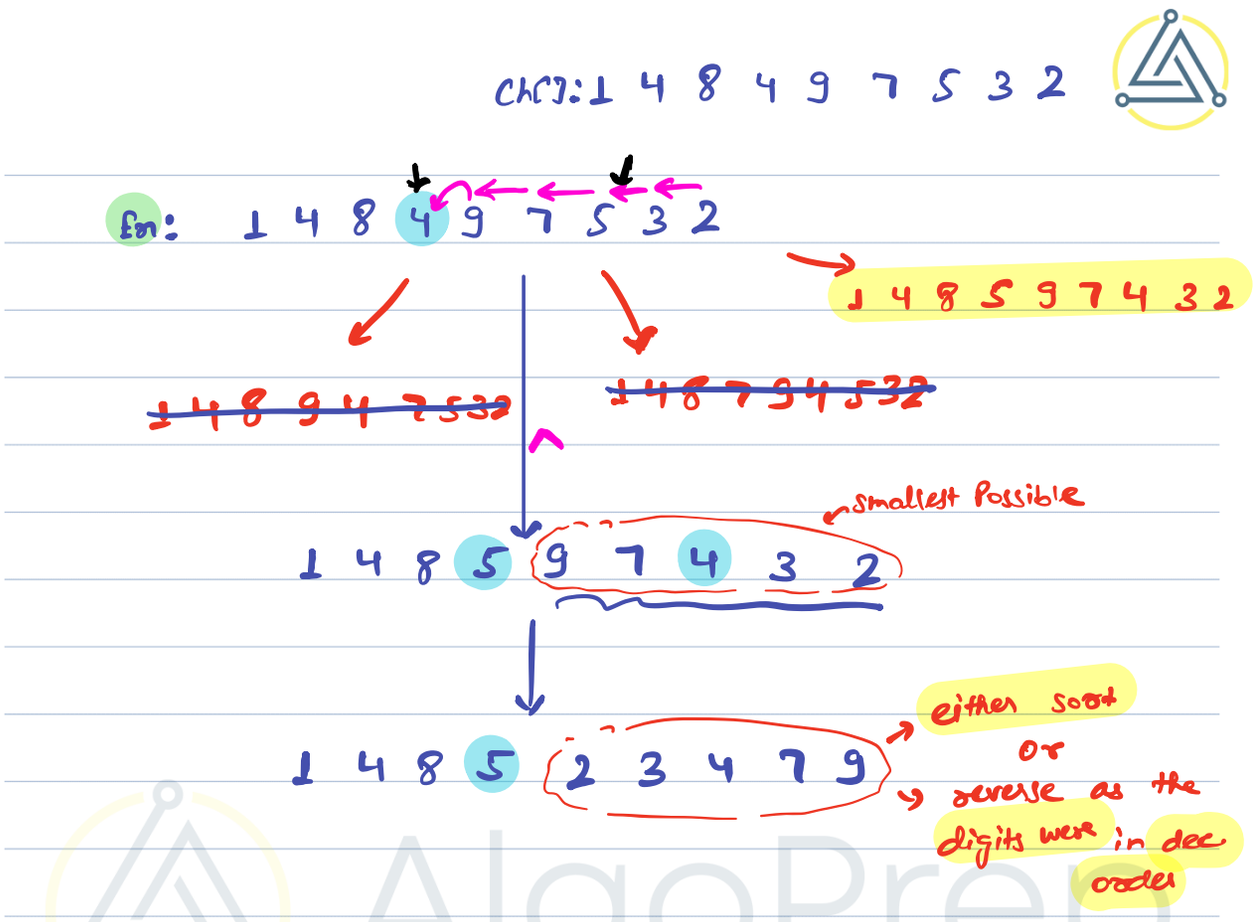
Ex: 1 4 8 4 9 7 5 3 2

2 4 7 1 1 3

2 4 7 2 1 3 > 2 4 7 1 9 2

min(00) max(90)

→ As we want to do minimum increment according to Problem Statement. we target Smaller Place value first using Swapping.


arr[]: 1 4 8 4 9 7 5 3 2
Ex: 1 4 8 4 9 7 5 3 2
1 4 8 5 9 7 4 3 2
1 4 8 9 4 7 5 3 2
1 4 8 7 9 4 5 3 2
smallest possible
1 4 8 5 9 7 4 3 2
1 4 8 5 2 3 4 7 9
either sort
or
reverse as the
digits were in dec
order

//Pseudo code

```
public int nextgreaterElement (int n){
    Char [] number = (n+ "").toCharArray();
    int idx=-1;
    for(int i= number.length-2; i>=0; i--){
        if (number[i] < number[i+1]){
            idx = i;
            break;
        }
    }

    if (idx == -1){
        return -1;
    }

    char val = number[idx];
    int Smallest idx = idx+1;

    for (int i= idx+1; i< number.length; i++){
        if ( number[i]>val && number[i]<=
                                number[Smallest idx])
            Smallestidx = i;
        }
    }

    Char temp= number[idx];
    number[idx] = number[smallestidx];
    number[smallestidx] = temp;
```

T.C: O(n) ↙ O(n) O(nlogn)

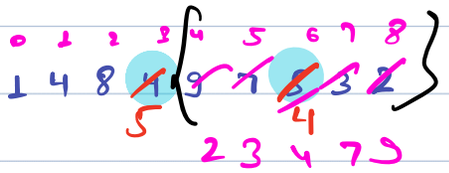
S.C: O(1)

```
Arrays.sort(numbers, idx+1, numbers.length);

long ans = Long.parseLong(new String(numbers));

    if (ans > Integer.MAX_VALUE){
        return -1;
    }
    else {
        return (int) ans;
    }
}
```

idx = ~~-1~~ 3 val = 4



Smallestidx = ~~4~~ 4 ~~5~~ 6

1 4 8 5 2 3 4 7 9

```
int idx=-1;
for(int i= number.length-2; i>=0; i--){
    if (number[i] < number[i+1]){
        idx = i;
        break;
    }
}

if (idx == -1){
    return -1;
}
char val = number[idx];
int Smallestidx = idx+1;

for (int i= idx+1; i< number.length; i++){
    if (number[i]>val && number[i]<=
                    number[Smallestidx])
        Smallestidx = i;
    }
}
```

```
char temp= number[idx];
number[idx] = number[Smallestidx];
number[Smallestidx] = temp;

Arrays.Sort (number, idx+1, number.length);
```