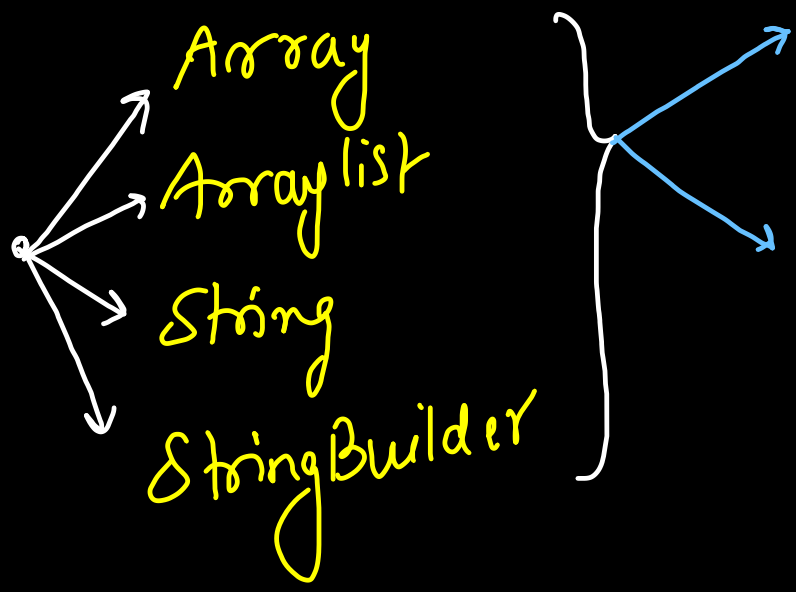
Arrays & Strings

primitive datatype ⇝ single unit of data



derived datatype

⇓

datastructure

(collection of data)

Array

⇓

⇒ fixed size

⇒ contiguous memory allocation

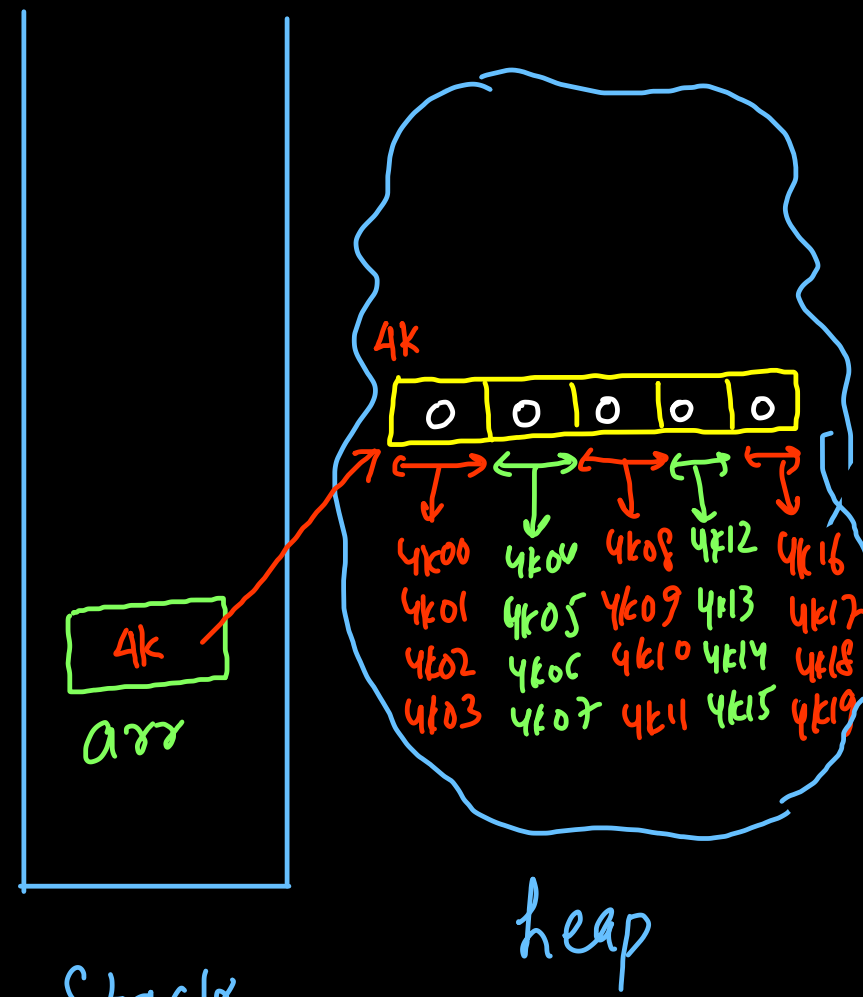
↳ random access

⇒ similar datatype

Array Syntax

eg: `int [] arr = new int[5];`

- `int []` → datatype
- `arr` → reference variable → declaration → Stack
- `new int[5]` → heap → array (instance/object)
- `5` → size
- initialization (array object creation)



default value

→ int/short/byte/long ⇒ 0

→ float/double ⇒ 0.0

→ boolean ⇒ false

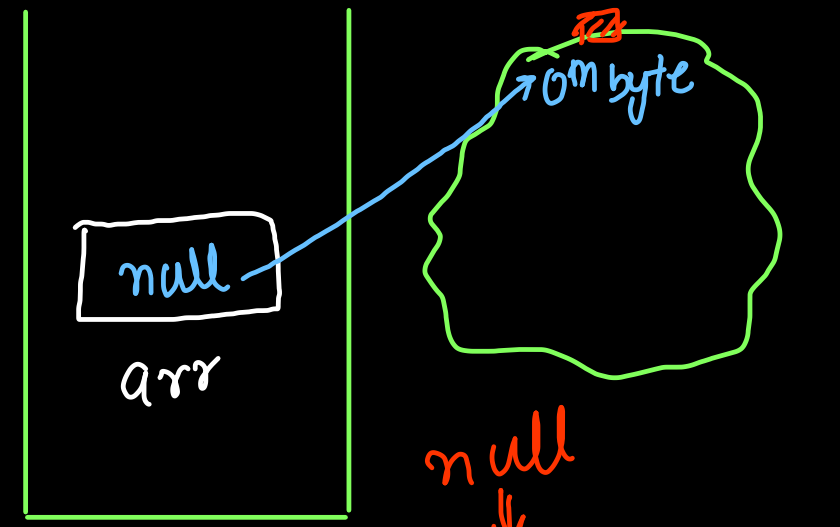
→ char ⇒ '\0' null character

Java is a fool-proof language

abstraction → java ↗ address ✗
↘ garbage ✗
reference variable ✗

array declaration only and no initialization

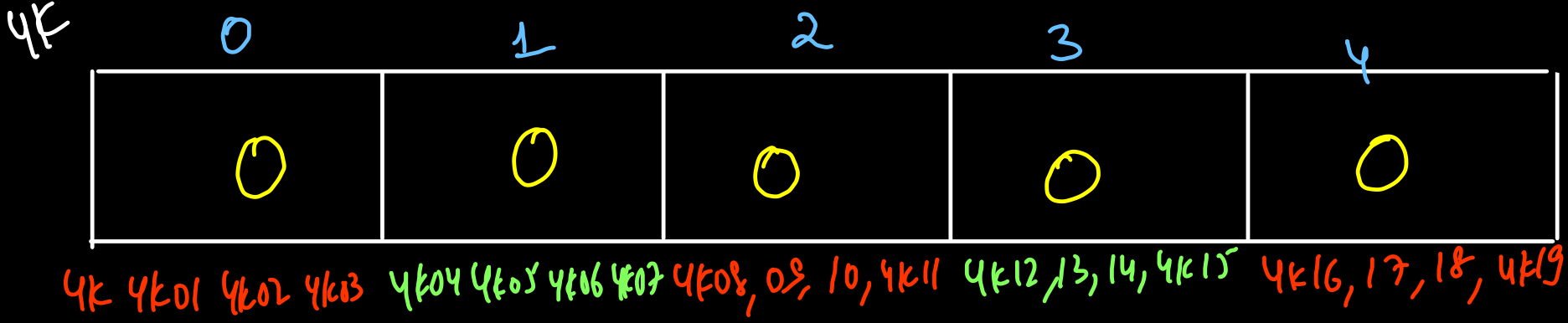
int [] arr; (or) int [] arr = null;



null ⇓ arr is not pointing to any valid memory address

random access (0-based indexing)

Base address = 4k



BA size*index

arr[0] = 4k + 4*0 = 4k

arr[1] = 4k + 4*1 = 4k04

arr[2] = 4k + 4*2 = 4k08

arr[3] = 4k + 4*3 = 4k12

arr[4] = 4k + 4*4 = 4k16

block address =

base address + datatypesize * index

size ⇒ 5 ⇒ N

indices ⇒ [0,4] ⇒ [0, N-1]

```java
// 1. Declaration
// int[] arr;
int[] arr = null;
System.out.println(arr);


// 2. Declaration & Initialization
int[] nums = new int[5];
System.out.println(nums);
```

```
Finished in 73 ms
null

Finished in 84 ms
null
[I@6d03e736
```

toString()
↳ hashcode
↳ hexadecimal string

this is not address!

```java
// 2. Declaration & Initialization
int[] nums = new int[5];
System.out.println(nums);
System.out.println(Arrays.toString(nums));

// 3. Random Access (0 based indexing)
for(int idx = 0; idx < 5; idx++){
    nums[idx] = idx + 1;
}

for(int idx = 0; idx < 5; idx++){
    System.out.print(nums[idx] + " ");
}
```

```
Finished in 94 ms
null
[I@6d03e736
[0, 0, 0, 0, 0]
1 2 3 4 5
```

```java
// 2. Declaration & Initialization
Scanner scn = new Scanner(System.in);
int n = scn.nextInt();

int[] nums = new int[n];
System.out.println(nums);
System.out.println(Arrays.toString(nums));

// 3. Random Access (0 based indexing)
for(int idx = 0; idx < nums.length; idx++){
    nums[idx] = idx + 1;
}

for(int idx = 0; idx < nums.length; idx++){
    System.out.print(nums[idx] + " ");
}
```

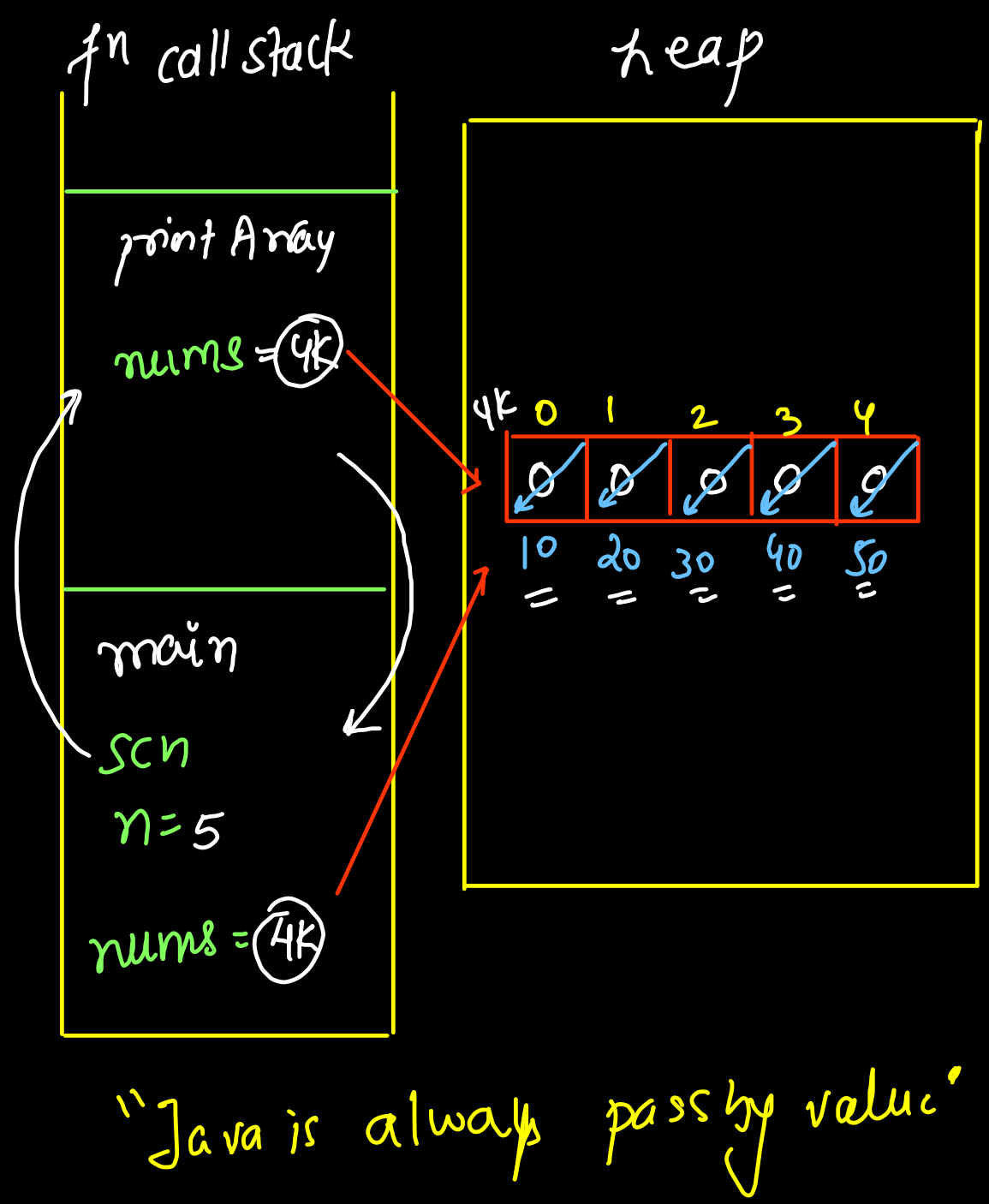
```
Finished in 131 ms
null
[I@2ef1e4fa
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1 2 3 4 5 6 7 8 9 10
```

stdin

```
10
```

public static void main(String[] args)

```java
        // 2. Declaration & Initialization
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        int[] nums = new int[n];
        System.out.println(nums);
        System.out.println(Arrays.toString(nums));

        // 3. Random Access (0 based indexing)
        for(int idx = 0; idx < nums.length; idx++){
            nums[idx] = scn.nextInt();
        }

        // 4. Arrays & Functions
        printArray(nums);
    }

    public static void printArray(int[] nums){
        for(int idx = 0; idx < nums.length; idx++){
            System.out.print(nums[idx] + " ");
        }
    }
```



"Java is always pass by value"

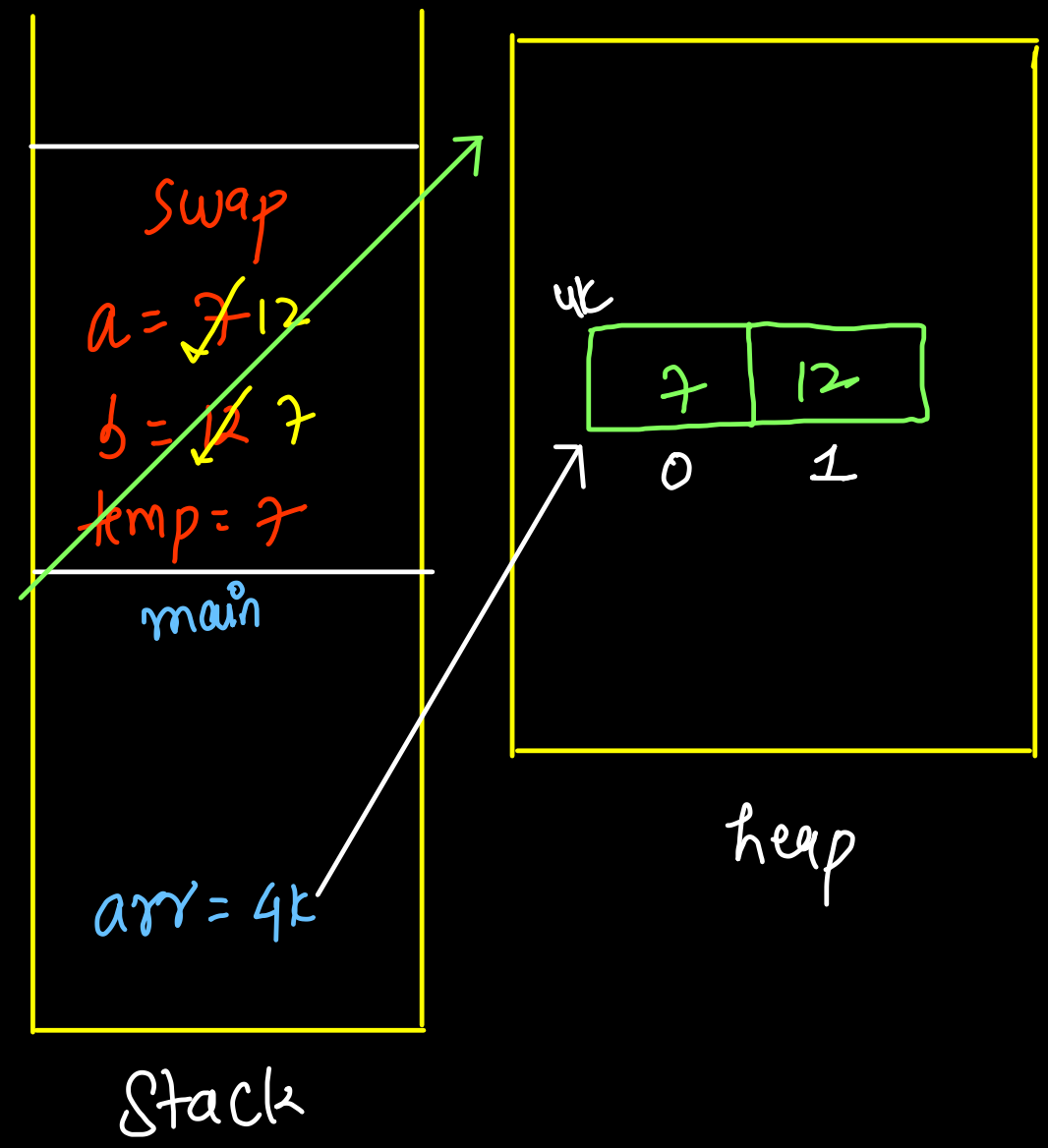
```java
public static void main(String[] args) {

    // This Array will also be in heap
    int[] arr = {7, 12};
    System.out.println(Arrays.toString(arr));
    swap(arr[0], arr[1]);
    System.out.println(Arrays.toString(arr));
}

public static void swap(int a, int b){
    System.out.println(a + " " + b);
    int temp = a;
    a = b;
    b = temp;
    System.out.println(a + " " + b);
}
```

7,12

7,12

7 12

12 7



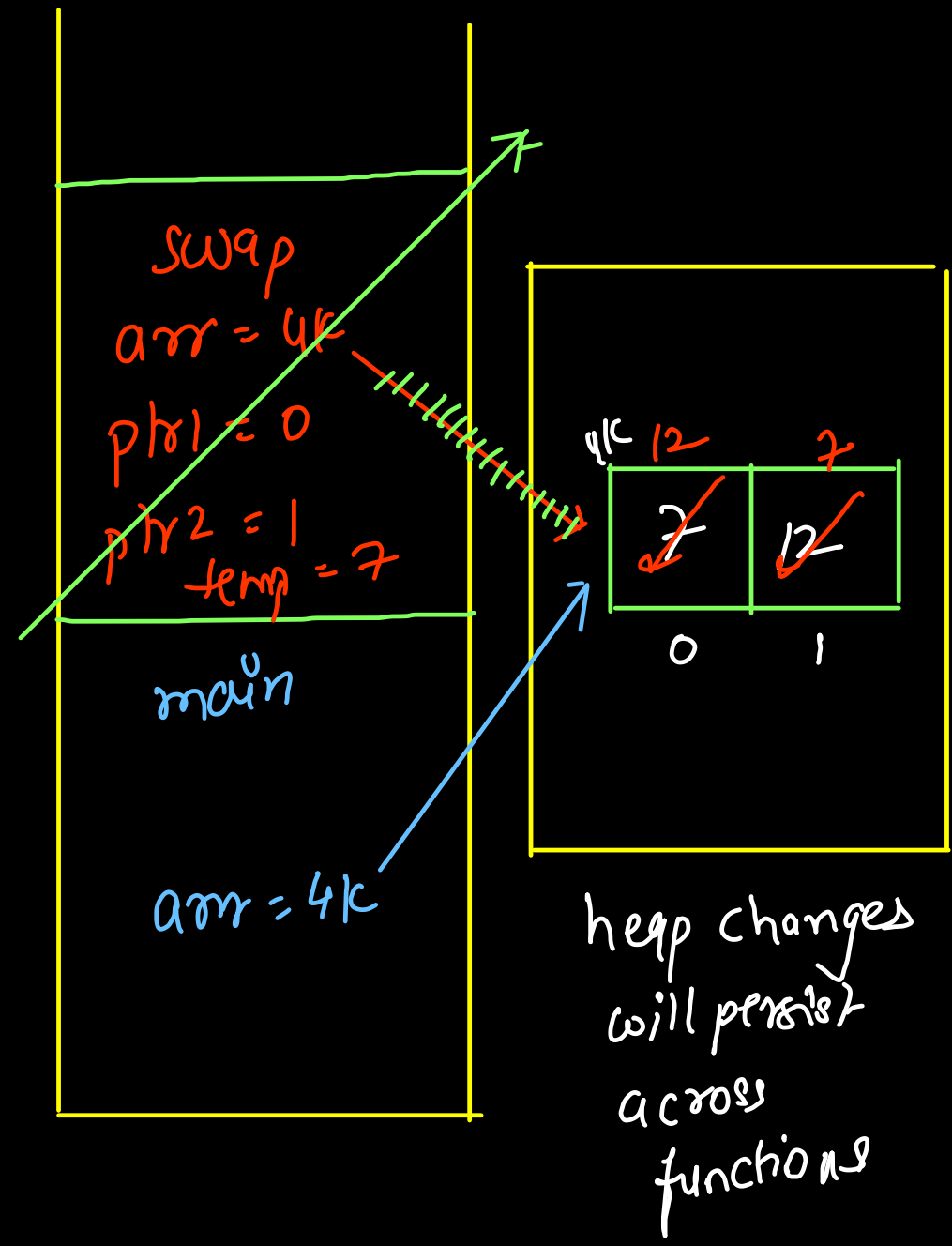
```java
public static void main(String[] args) {
    // This Array will also be in heap
    int[] arr = {7, 12};
    // System.out.println(Arrays.toString(arr));
    // swap(arr[0], arr[1]);
    // System.out.println(Arrays.toString(arr));

    System.out.println(Arrays.toString(arr));
    swap(arr, 0, 1);
    System.out.println(Arrays.toString(arr));
}

public static void swap(int[] arr, int ptr1, int ptr2){
    System.out.println(Arrays.toString(arr));
    int temp = arr[ptr1];
    arr[ptr1] = arr[ptr2];
    arr[ptr2] = temp;
    System.out.println(Arrays.toString(arr));
}
```

7,12

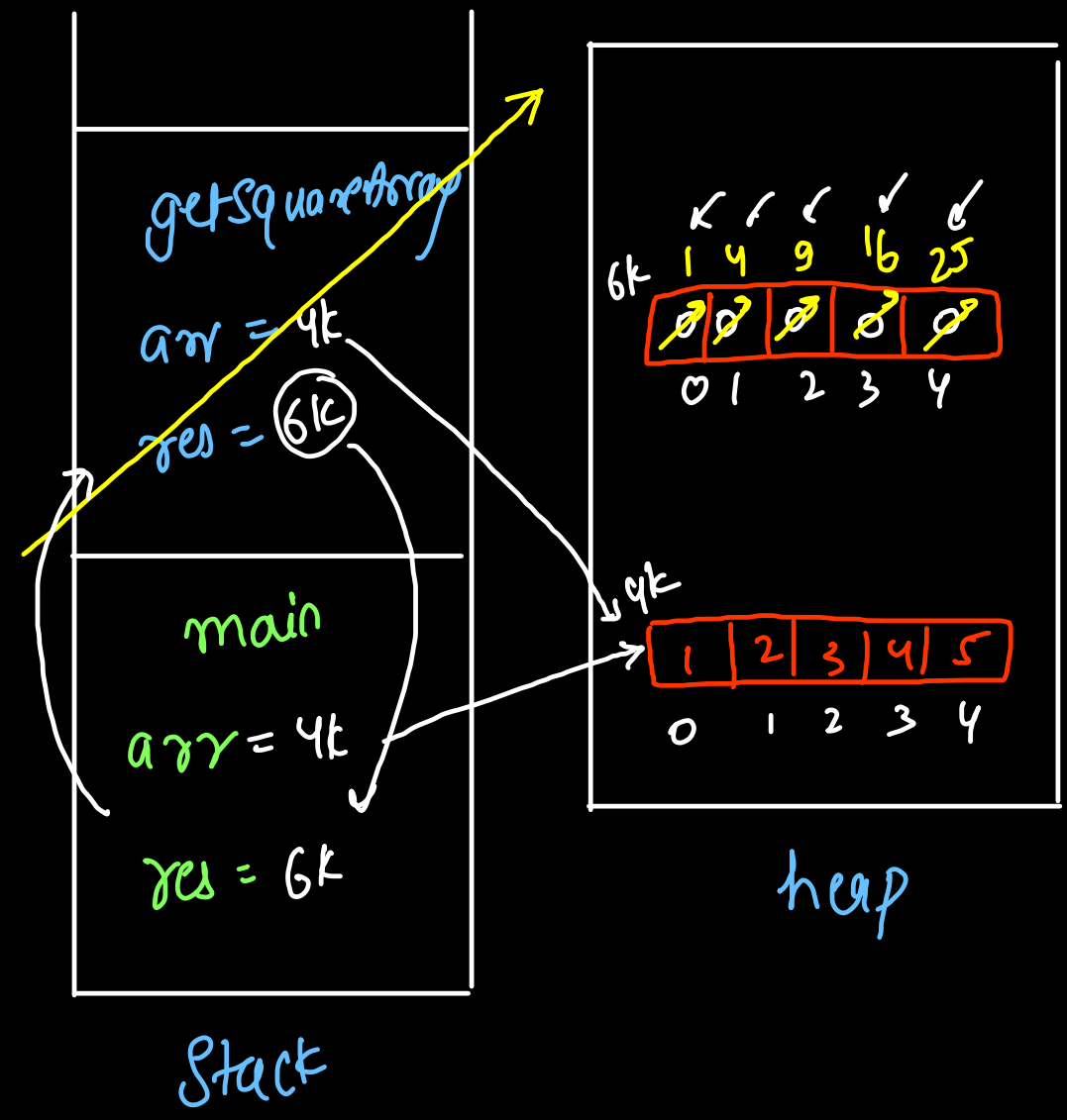
12,7

7,12

12,7



```java
public static void main(String[] args) {

    int[] arr = {1, 2, 3, 4, 5};
    int[] res = getSquareArray(arr);
    for(int idx = 0; idx < res.length; idx++){
        System.out.print(res[idx] + " ");
    }
}

public static int[] getSquareArray(int[] arr){
    int[] res = new int[arr.length];
    for(int idx = 0; idx < arr.length; idx++){
        res[idx] = arr[idx] * arr[idx];
    }
    return res;
}
```



Shallow vs deep copy

fake copy
(copying reference variable but instance is same)

actual copy
(creating a new instance as well as new reference variable)

```
int [] arr = {10,20,30};

int [] temp = arr;
```

```java
// Shallow Copy
int[] arr = {10, 20, 30};
int[] temp = arr; // or pass by value in function

temp[0] = 100; temp[1] = 200; temp[2] = 300;
System.out.println(Arrays.toString(arr)); → 100, 200, 300
System.out.println(Arrays.toString(temp)); → 100, 200, 300
```

Shallow copy is very efficient (fast);

$n = 10$ or $n = 10^6$ ⇒ Same time

But it modifies the same array using different reference variable { we should avoid shallow copy during updation }

```java
    int[] arr = {10, 20, 30};

    // deep copy
    int[] deep = new int[arr.length];
    for(int idx = 0; idx < arr.length; idx++){
        deep[idx] = arr[idx];
    }

    // shallow copy
    int[] shallow = arr; // or pass by value in function

    arr[0] = 100; arr[1] = 200; arr[2] = 300;

    System.out.println(Arrays.toString(arr));
    System.out.println(Arrays.toString(shallow));
    System.out.println(Arrays.toString(deep));

}
```

1 → different array objects

updates in one array does not effect deep copied array

```
Finished in 80 ms
[100, 200, 300]
[100, 200, 300]
[10, 20, 30]
```

Deep copy of bigger array will take more time than deep copy of smaller array

Deep copy will take more time than shallow copy

Arraylist → Collection Framework

import java.util.*;

array → offline algorithm

arraylist → online algorithm

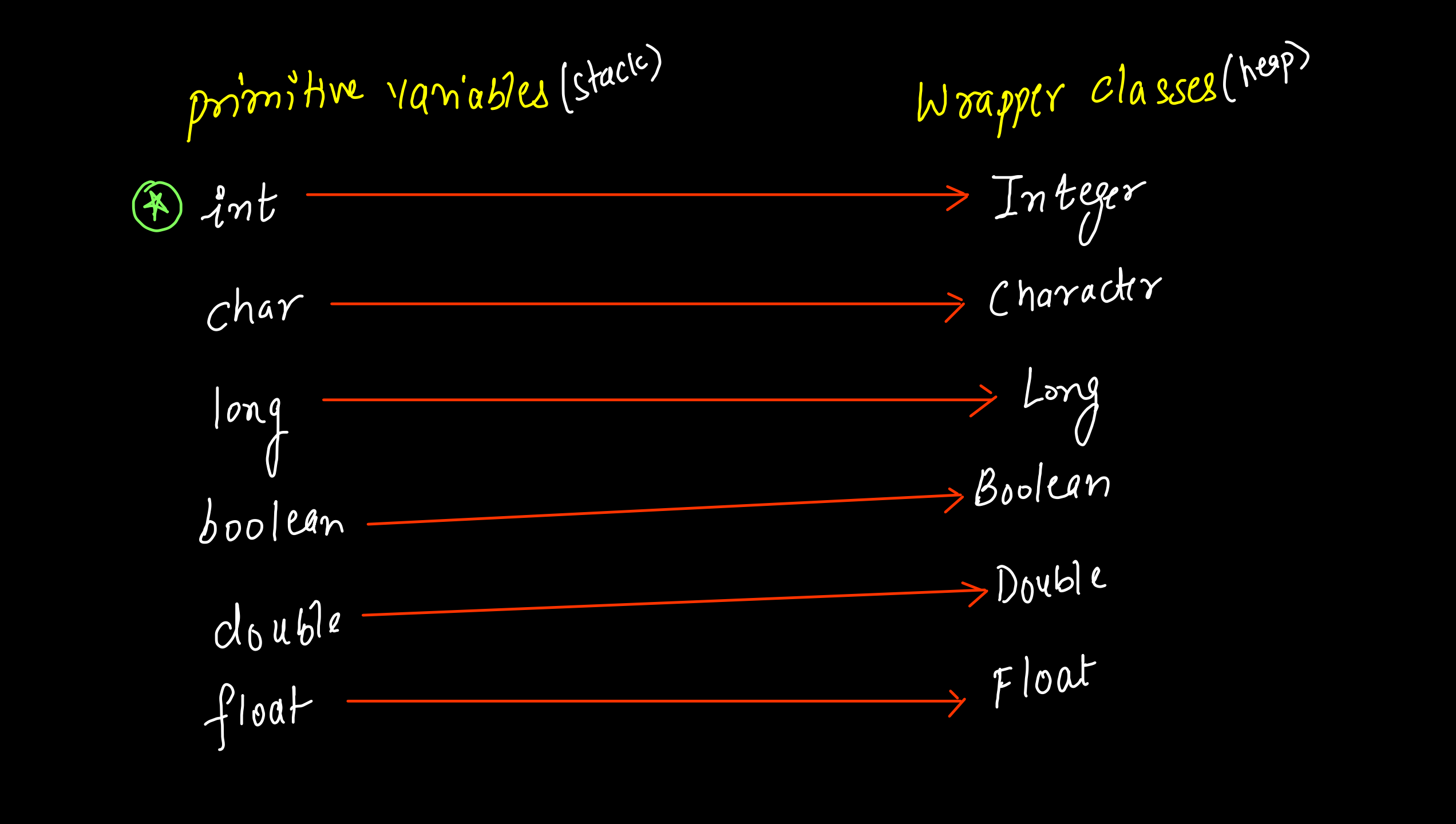
→ dynamic array

- array grow in size
- array shrink in size
- similar datatype
- contiguous memory allocation.
- no or minimal space wastage (size & capacity)

Internally arraylist is designed using array

Arraylist cannot be containing primitives.

~~int~~ ~~char~~ ~~boolean~~ ~~long~~


primitive variables (static)
Wrapper classes (heap)
int
Integer
char
Character
long
Long
boolean
Boolean
double
Double
float
Float

ArrayList<Integer> nums = new ArrayList<>();

↳ empty arr ⇒ size = 0 []

Creation (Insertn)

add ✓

push

enqueue

put

Read (Get)

get ✓

charAt

peek

[]

Update (Set)

set ✓

setCharAt

put

Deletion (Removal)

remove ✓

delete

pop

dequeue

addlast

nums.add(10); [ ] → [10] (0)

nums.add(20); [10] (0) → [10, 20] (0, 1)

nums.add(30); [10, 20] (0, 1) → [10, 20, 30] (0, 1, 2)

get

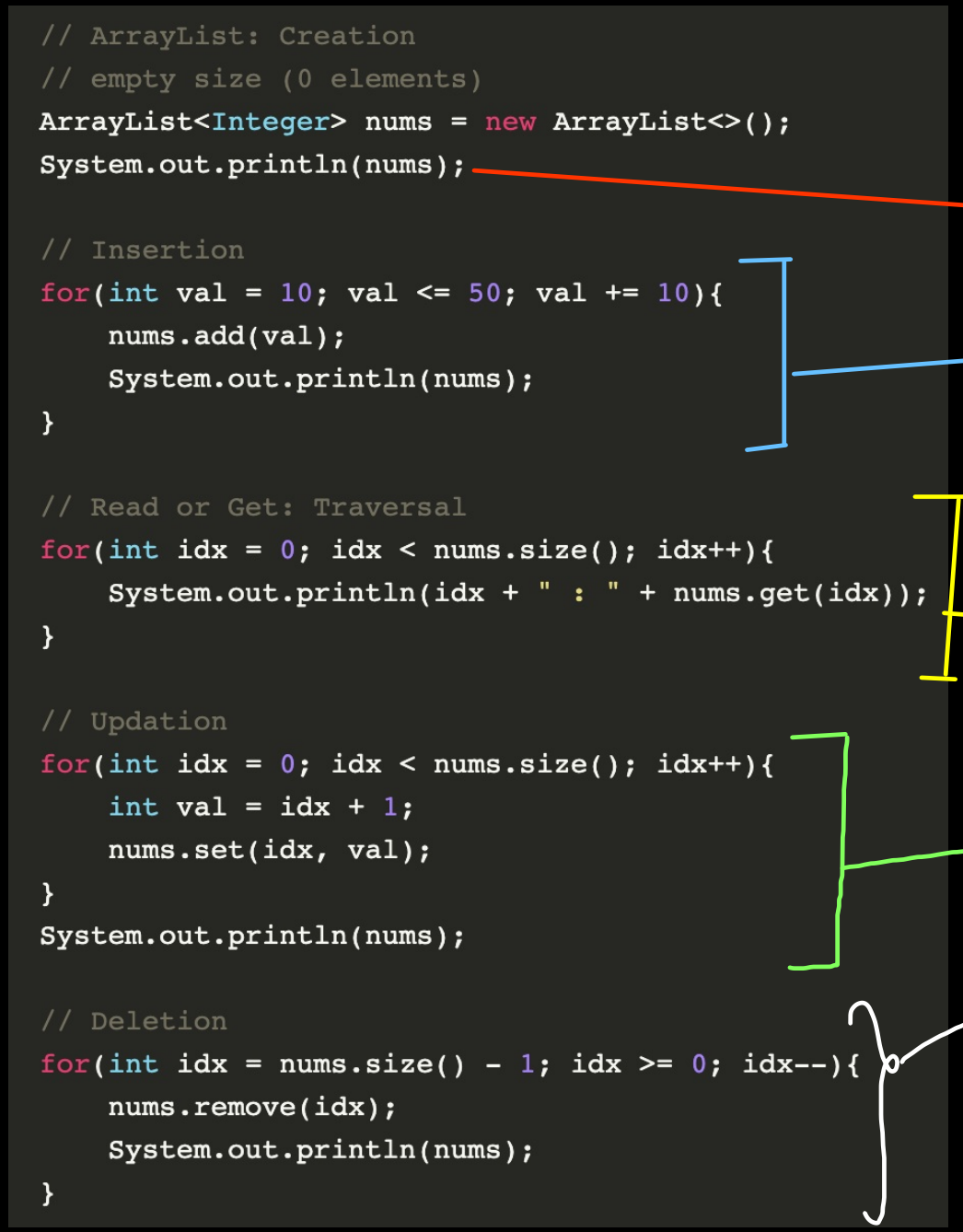
nums.get(0) ⇒ 10

nums.get(1) ⇒ 20

nums.get(2) ⇒ 30

nums.get(-1) → Index out of bound exception

nums.get(3) → Index out of bound exception



```java
// ArrayList: Creation
// empty size (0 elements)
ArrayList<Integer> nums = new ArrayList<>();
System.out.println(nums);

// Insertion
for(int val = 10; val <= 50; val += 10){
    nums.add(val);
    System.out.println(nums);
}

// Read or Get: Traversal
for(int idx = 0; idx < nums.size(); idx++){
    System.out.println(idx + " : " + nums.get(idx));
}

// Updation
for(int idx = 0; idx < nums.size(); idx++){
    int val = idx + 1;
    nums.set(idx, val);
}
System.out.println(nums);

// Deletion
for(int idx = nums.size() - 1; idx >= 0; idx--){
    nums.remove(idx);
    System.out.println(nums);
}
```

Linear Search Algorithm

steps / procedure ⟹ input → processing → output

{ 20, 10, 50, 30, 40, 30, 30 }

0 1 2 3 4 5 6

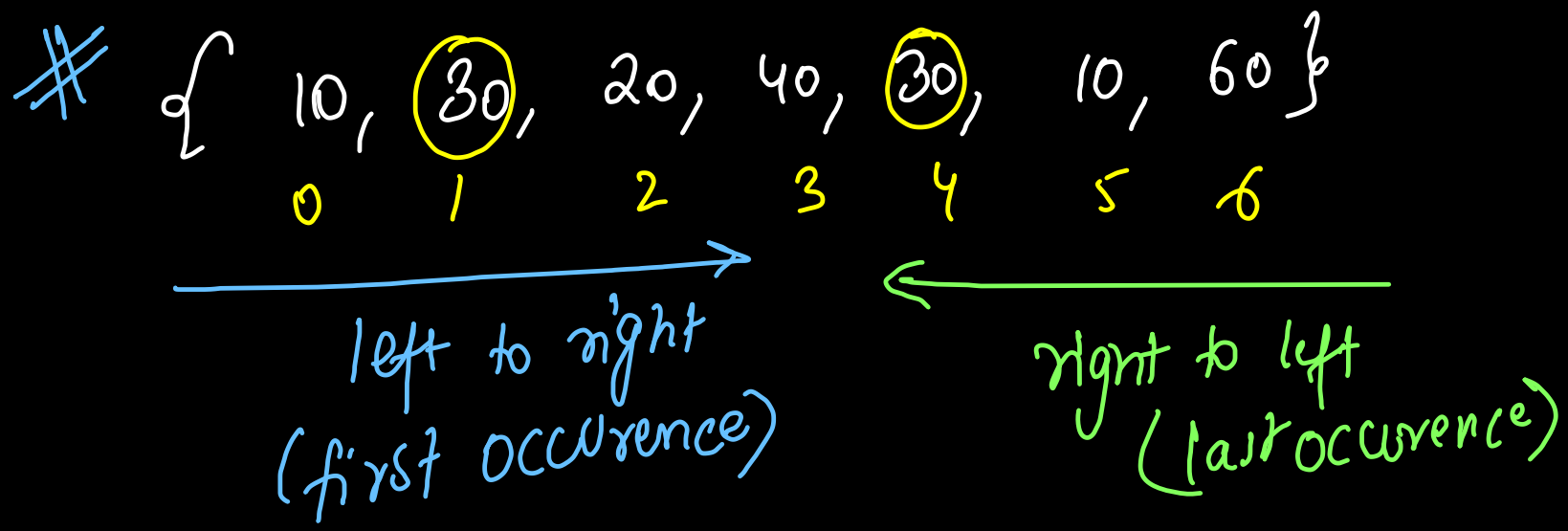
target = 30 ⇒ 3

target = 25 ⇒ -1

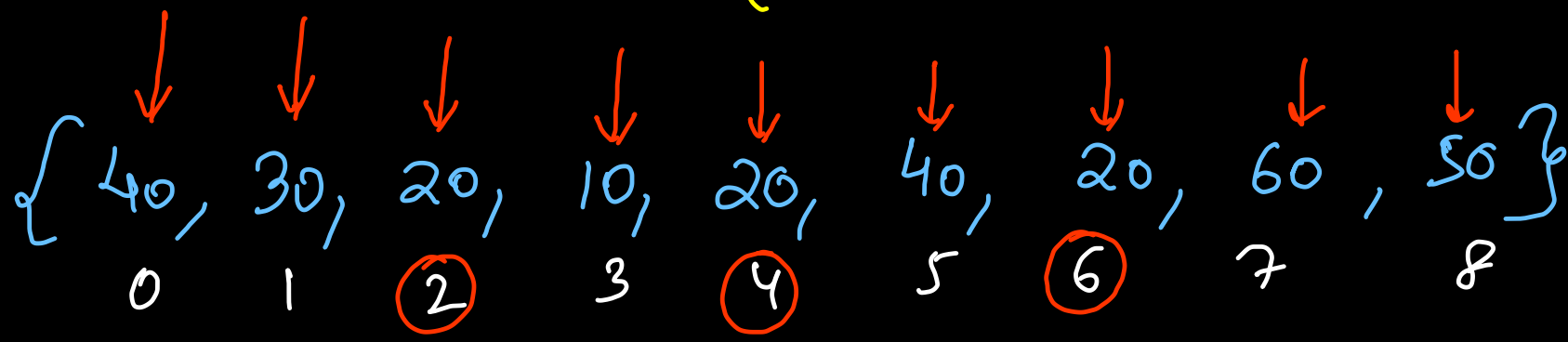
```
for(int idx=0; idx<arr.length; idx++){
    if(arr[idx]==target) return idx;
                         //first occurence
}
return -1; // unsuccessful search
```

Linear Search Algorithm

```java
static int search(int[] arr, int N, int target)
{
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == target){
            return idx; // first occurence
        }
    }
    return -1; // unsuccessul search
}
```



First & Last Occurence



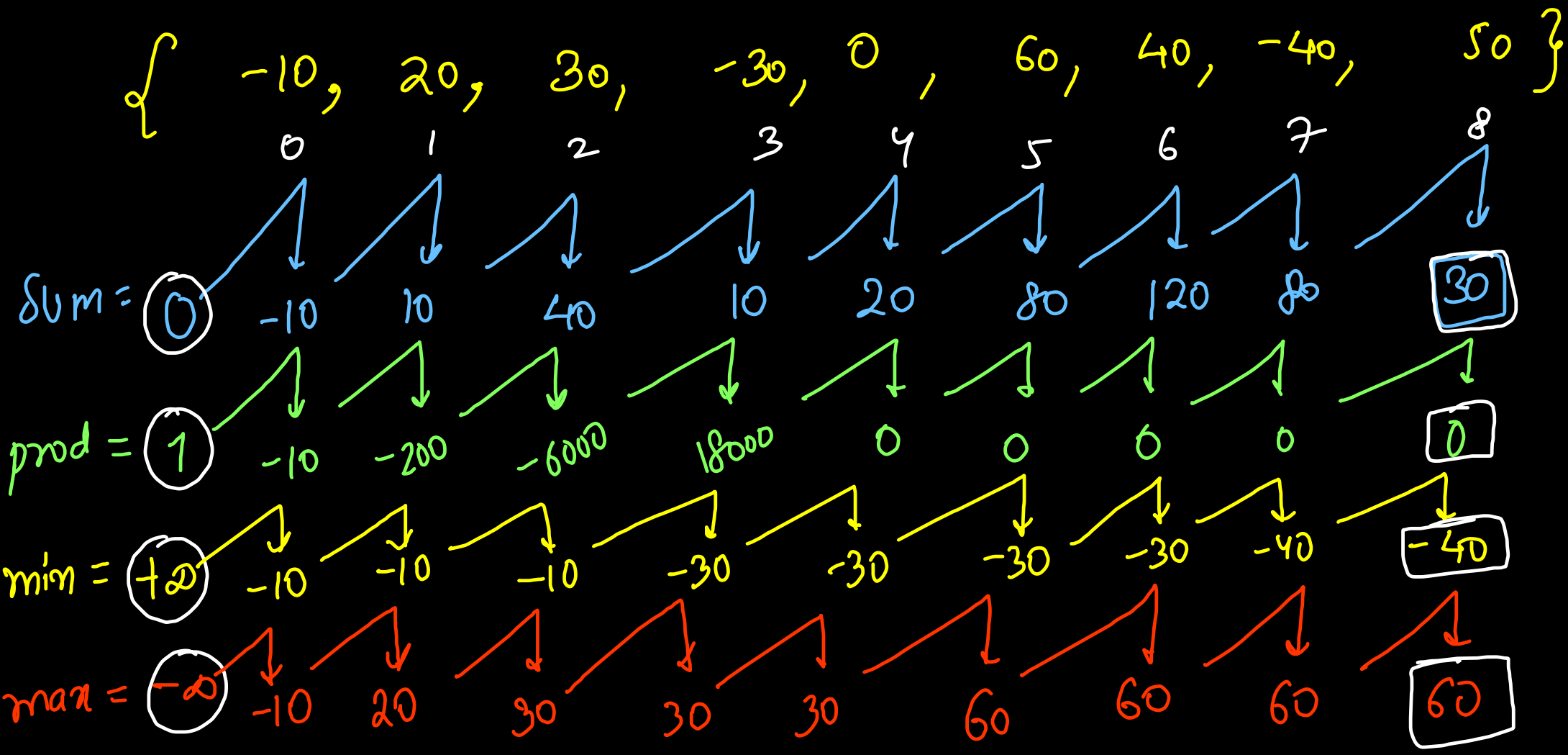
LC 34

```
int firstIndex = -1 2
int lastIndex = -1 2 4 6
```

	fi li
target = 20	[2, 6]
target = 60	[7, 7]
target = 35	[-1, -1]

```java
public int[] searchRange(int[] arr, int target) {
    int firstIndex = -1, lastIndex = -1;

    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == target) {
            lastIndex = idx;
            if(firstIndex == -1){
                firstIndex = idx;
            }
        }
    }

    int[] ans = {firstIndex, lastIndex};
    return ans;
}
```

min^m & max^m


{ -10, 20, 30, -30, 0, 60, 40, -40, 50 }
0 1 2 3 4 5 6 7 8
sum = 0
-10 10 40 10 20 80 120 80 30
prod = 1
-10 -200 -6000 18000 0 0 0 0 0
min = +∞
-10 -10 -10 -30 -30 -30 -30 -40 -40
max = -∞
-10 20 30 30 30 60 60 60 60

Approach ①

```java
public int solve(int[] arr) {
    int minimum = Integer.MAX_VALUE; // + infinity
    int maximum = Integer.MIN_VALUE; // - infinity

    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] < minimum){
            minimum = arr[idx];
        }
        if(arr[idx] > maximum){
            maximum = arr[idx];
        }
    }

    return (minimum + maximum);
}
```

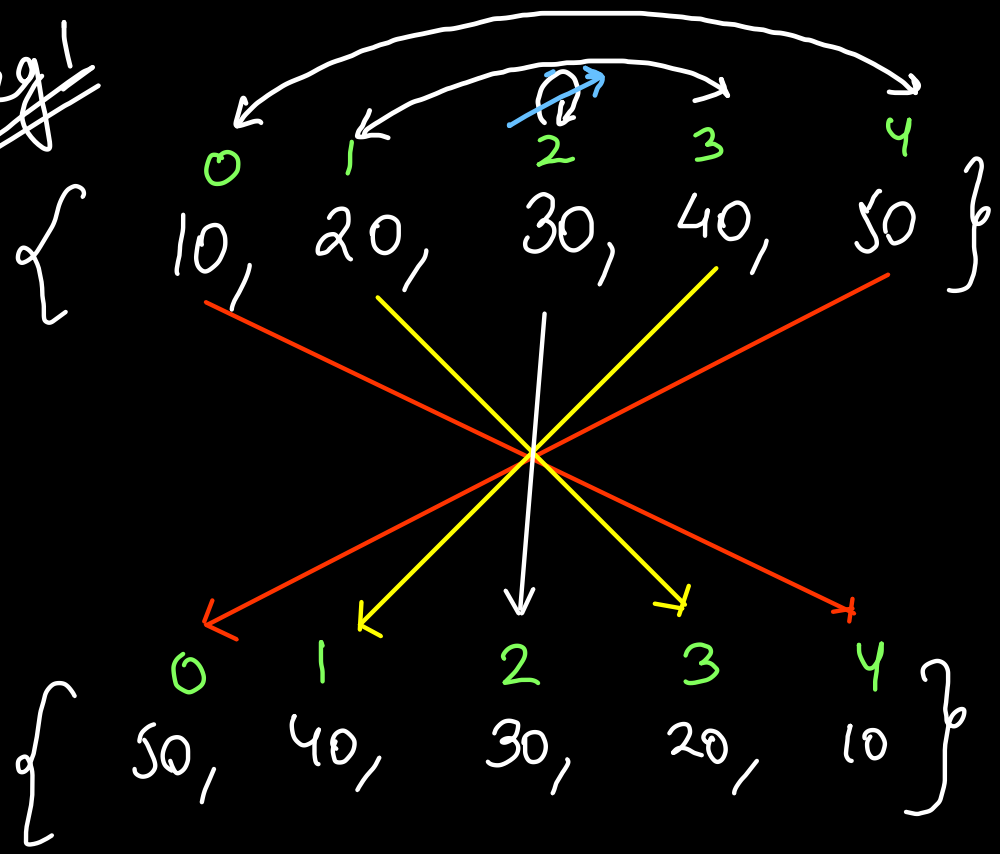
Approach ②

```java
public int solve(int[] arr) {
    int minimum = Integer.MAX_VALUE; // + infinity
    int maximum = Integer.MIN_VALUE; // - infinity

    for(int idx = 0; idx < arr.length; idx++){
        minimum = Math.min(minimum, arr[idx]);
        maximum = Math.max(maximum, arr[idx]);
    }

    return (minimum + maximum);
}
```

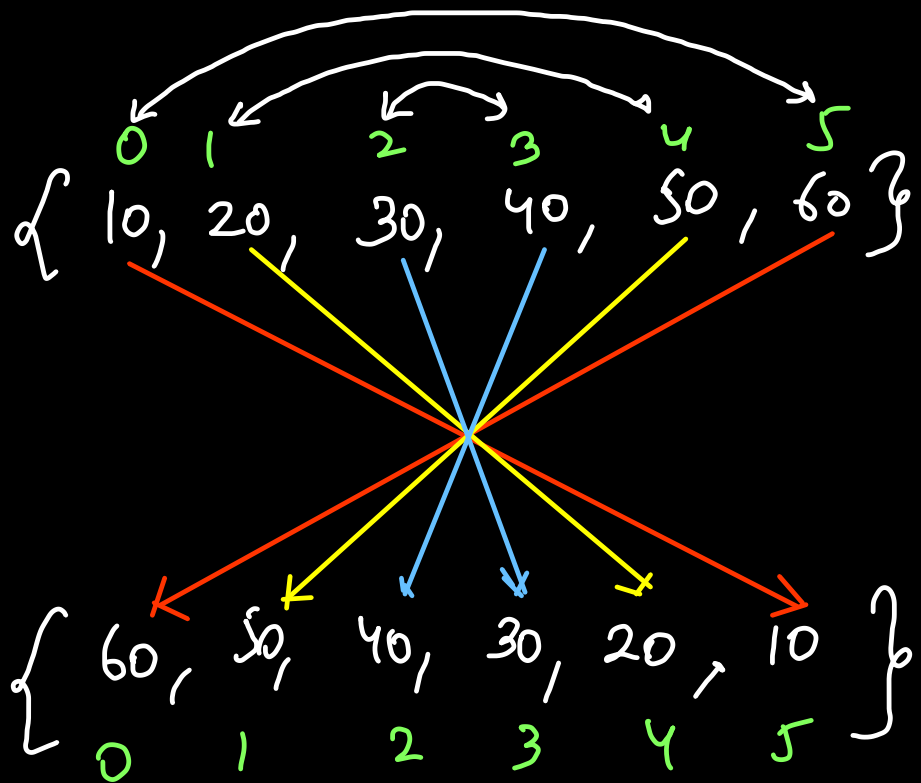
Reverse Array

{ inplace algorithm → change the input array }

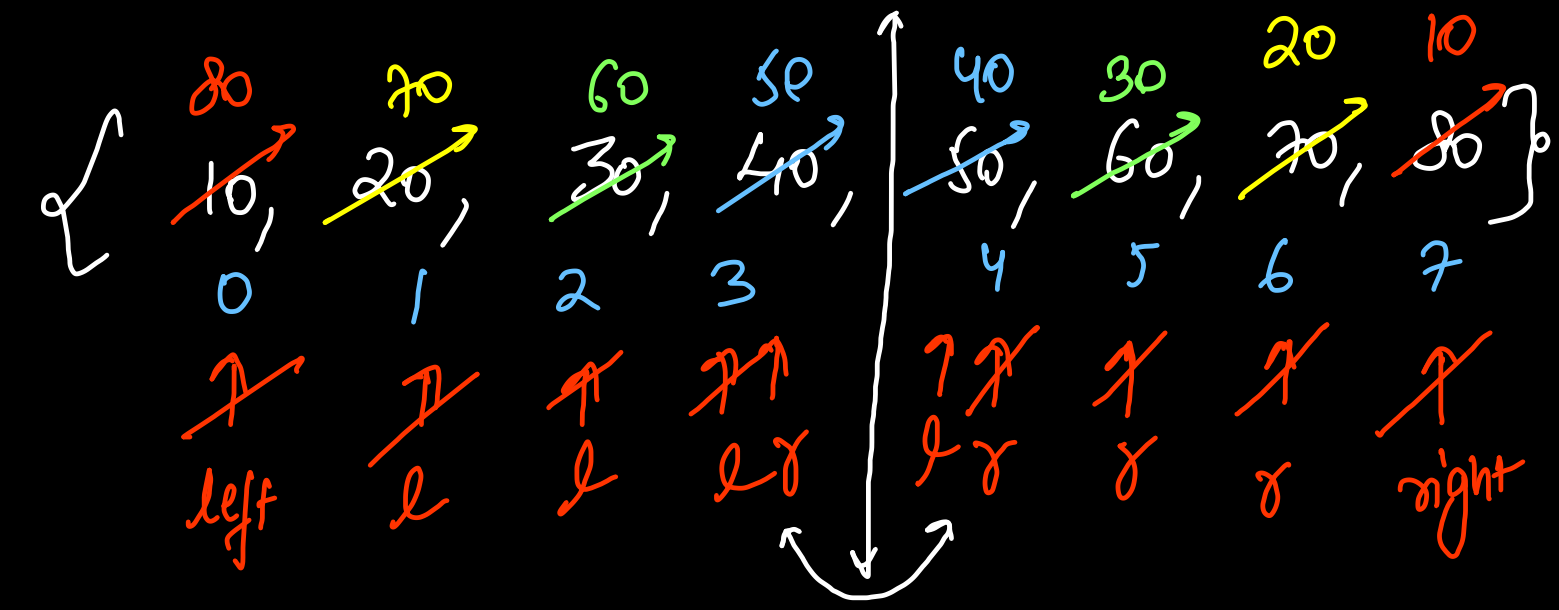
eg 1



eg 2



Two pointer Technique



Left to right

right to left

```
int left=0;
int right = n-1;
while (l<r){
swap(arr, left, right);
left ++;
right --;
}
```

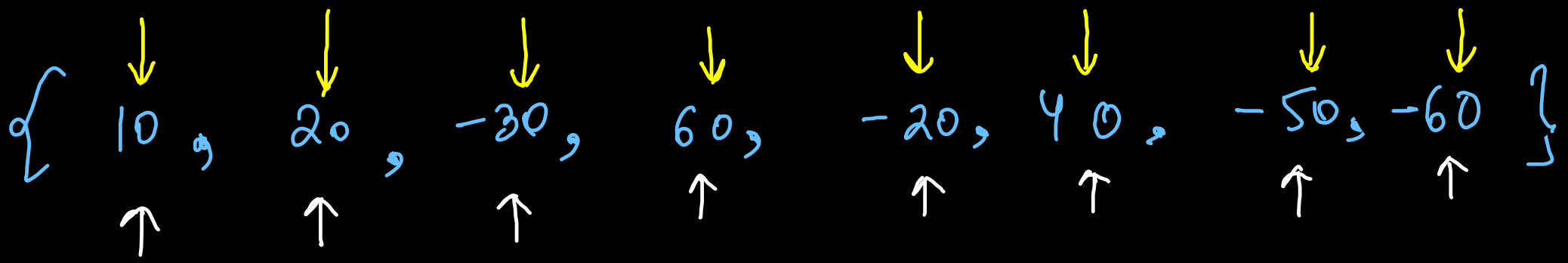
```java
public void swap(char[] arr, int left, int right){
    char temp = arr[left];
    arr[left] = arr[right];
    arr[right] = temp;
}

public void reverseString(char[] arr) {
    int left = 0, right = arr.length - 1;

    while(left <= right){
        swap(arr, left, right);
        left++; right--;
    }
}
```

Maxm prod of 2 elements

10	20	-30	40	-20	60	-50	-60
0	1	2	3	4	5	6	7

maxm product

- 2nd largest, largest: 40 * 60 = 240
- 2nd smallest, smallest: -50 * -60 = 300 ✓



largest = ~~-∞~~ ~~10~~ ~~20~~ 60

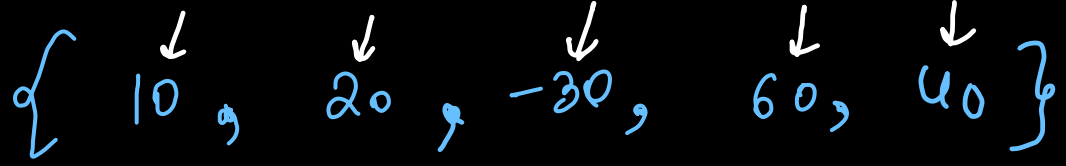
second largest = ~~-∞~~ ~~10~~ ~~20~~ 40

60 * 40 = 240

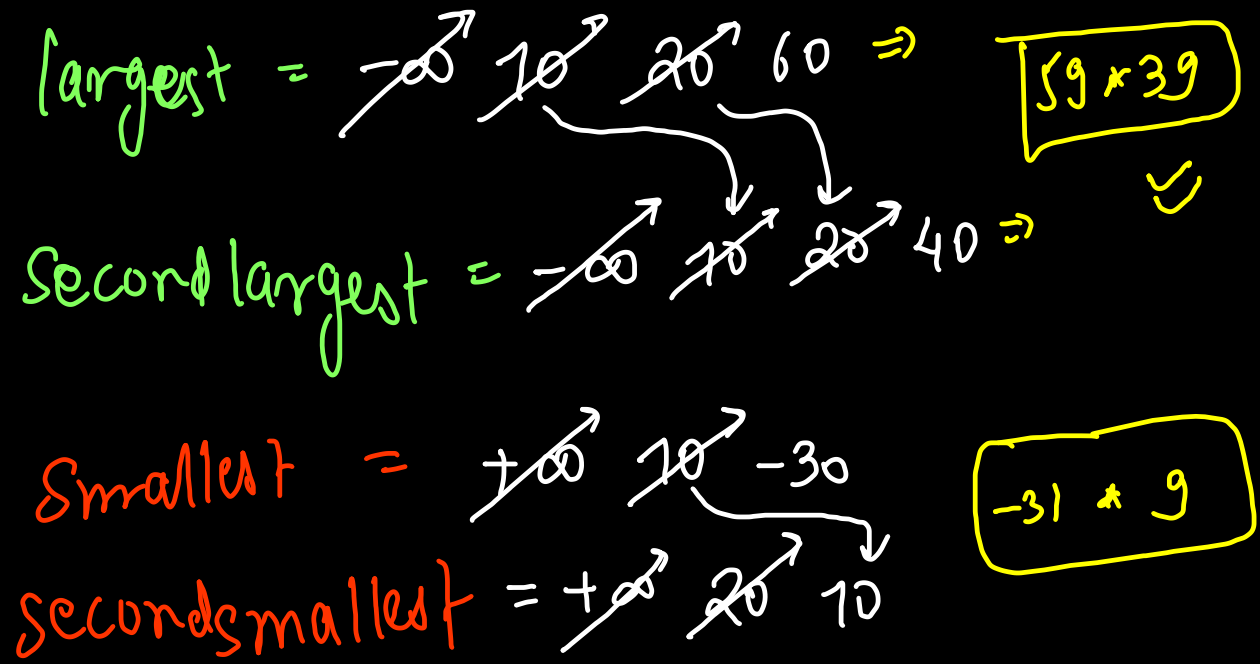
Smallest = ~~+∞~~ ~~10~~ ~~-30~~ ~~-50~~ -60

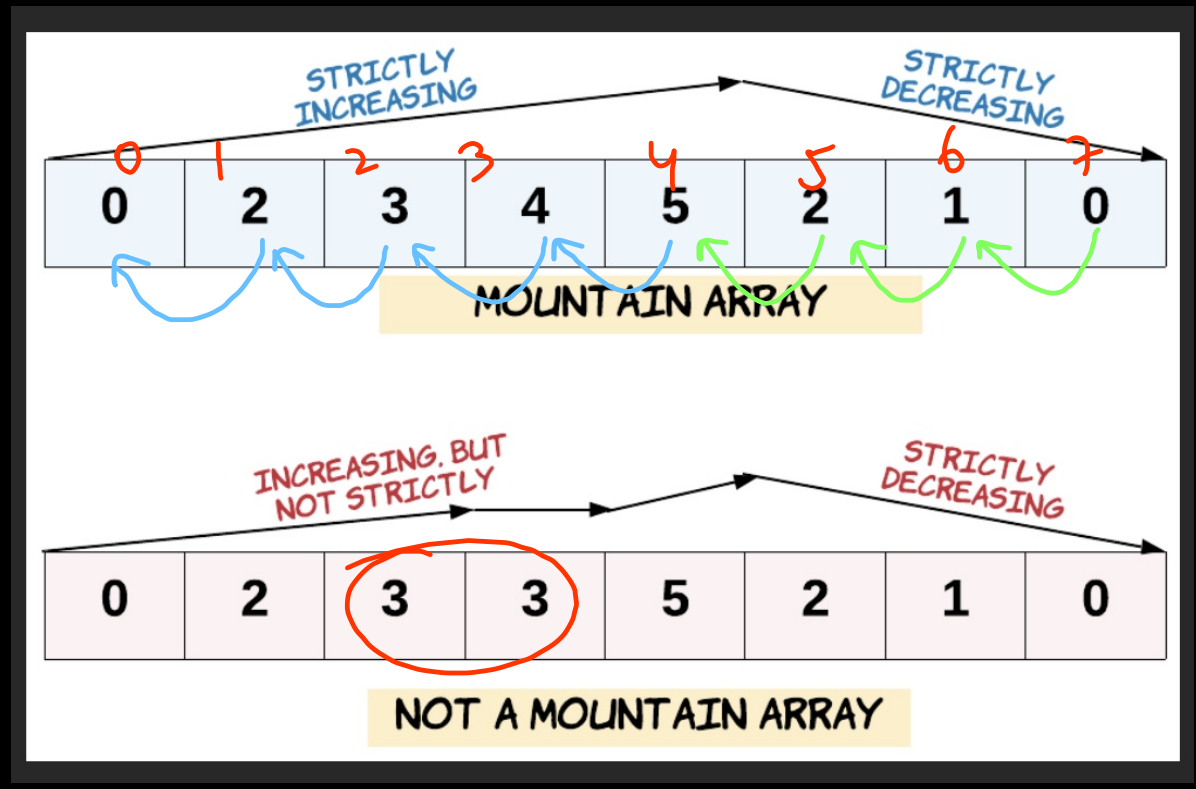
secondsmallest = ~~+∞~~ ~~20~~ ~~10~~ ~~-20~~ ~~-30~~ -50

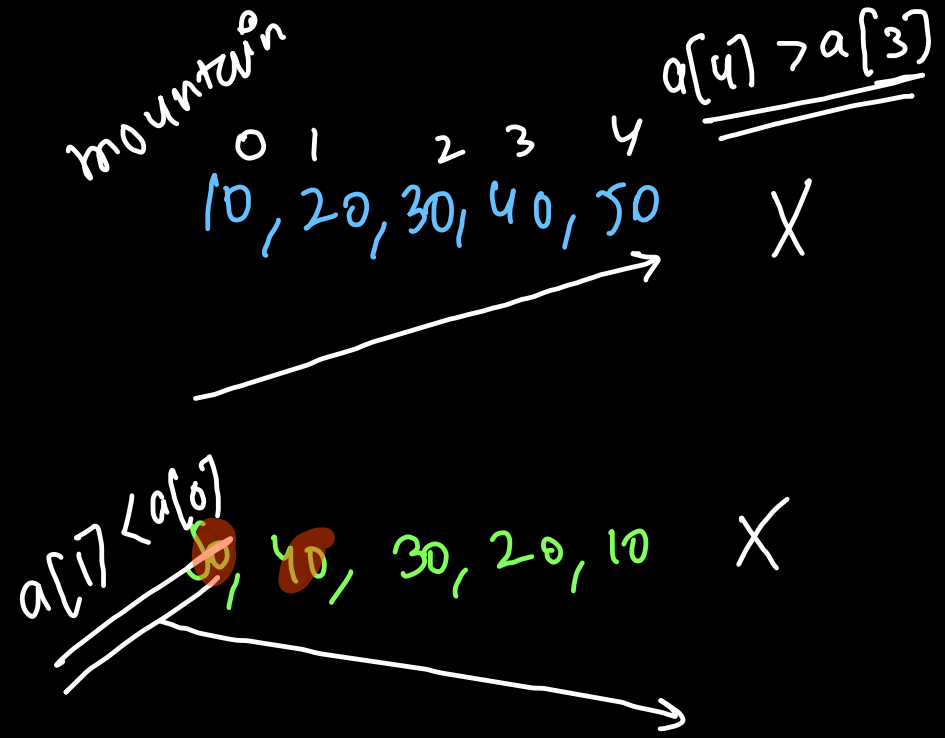
-60 * -50 = 300

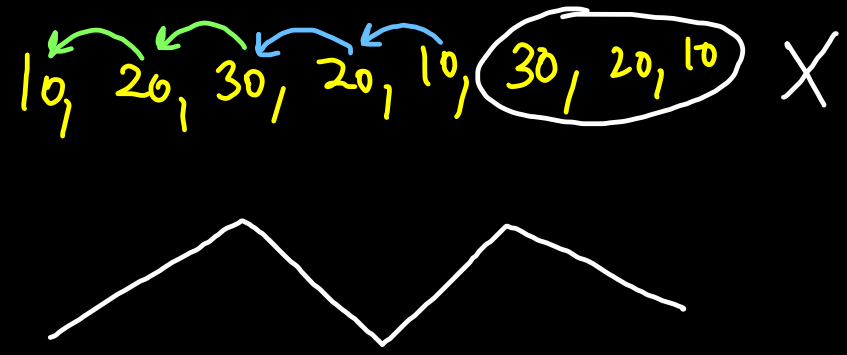



```java
public int maxProduct(int[] nums) {
    int largest = Integer.MIN_VALUE;
    int secondLargest = Integer.MIN_VALUE;

    int smallest = Integer.MAX_VALUE;
    int secondSmallest = Integer.MAX_VALUE;

    for(int idx = 0; idx < nums.length; idx++){
        if(nums[idx] >= largest){
            secondLargest = largest;
            largest = nums[idx];
        } else if(nums[idx] > secondLargest){
            secondLargest = nums[idx];
        }

        if(nums[idx] <= smallest){
            secondSmallest = smallest;
            smallest = nums[idx];
        } else if(nums[idx] < secondSmallest){
            secondSmallest = nums[idx];
        }
    }

    int choice1 = (largest - 1) * (secondLargest - 1);
    int choice2 = (smallest - 1) * (secondSmallest - 1);
    return Math.max(choice1, choice2);
}
```




STRICTLY INCREASING
STRICTLY DECREASING
0 1 2 3 4 5 6 7
0 2 3 4 5 2 1 0
MOUNTAIN ARRAY
INCREASING, BUT NOT STRICTLY
STRICTLY DECREASING
0 2 3 3 5 2 1 0
NOT A MOUNTAIN ARRAY


mountain
0 1 2 3 4
10, 20, 30, 40, 50
a[4] > a[3]
X
a[1] < a[0]
30, 40, 30, 20, 10
X


10, 20, 30, 20, 10, 30, 20, 10
X

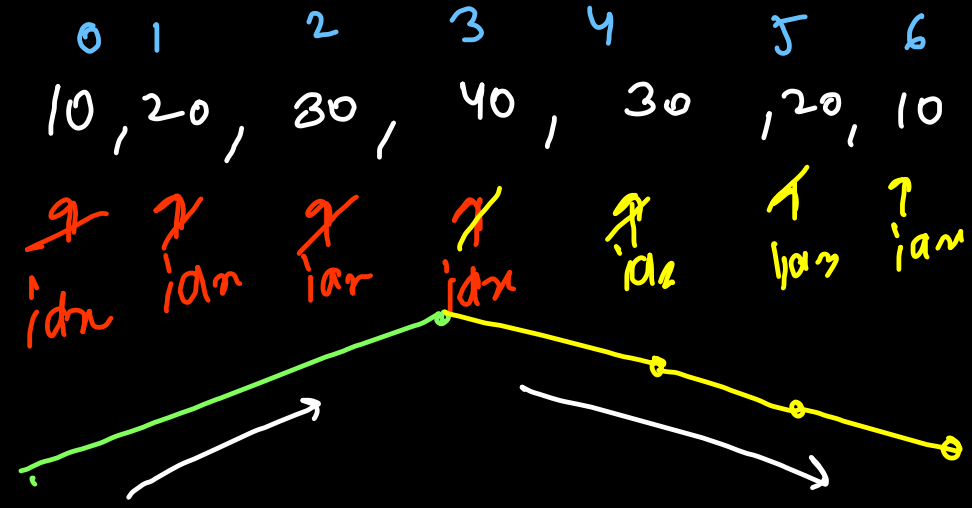
```java
public boolean validMountainArray(int[] arr) {
    if(arr.length < 3) return false;
    int idx = 0;

    // Increasing Part
    while(idx + 1 < arr.length){
        if(arr[idx] == arr[idx + 1]){
            return false; // not strictly increasing
        } else if(arr[idx] > arr[idx + 1]){
            break;
        }
        idx++;
    }

    if(idx == 0)
        return false; // no increasing part
    if(idx == arr.length - 1)
        return false; // no decreasing part

    // Decreasing Part
    while(idx + 1 < arr.length){
        if(arr[idx] == arr[idx + 1]){
            return false; // not strictly decreasing
        } else if(arr[idx] < arr[idx + 1]){
            break;
        }
        idx++;
    }

    return (idx == arr.length - 1);
}
```

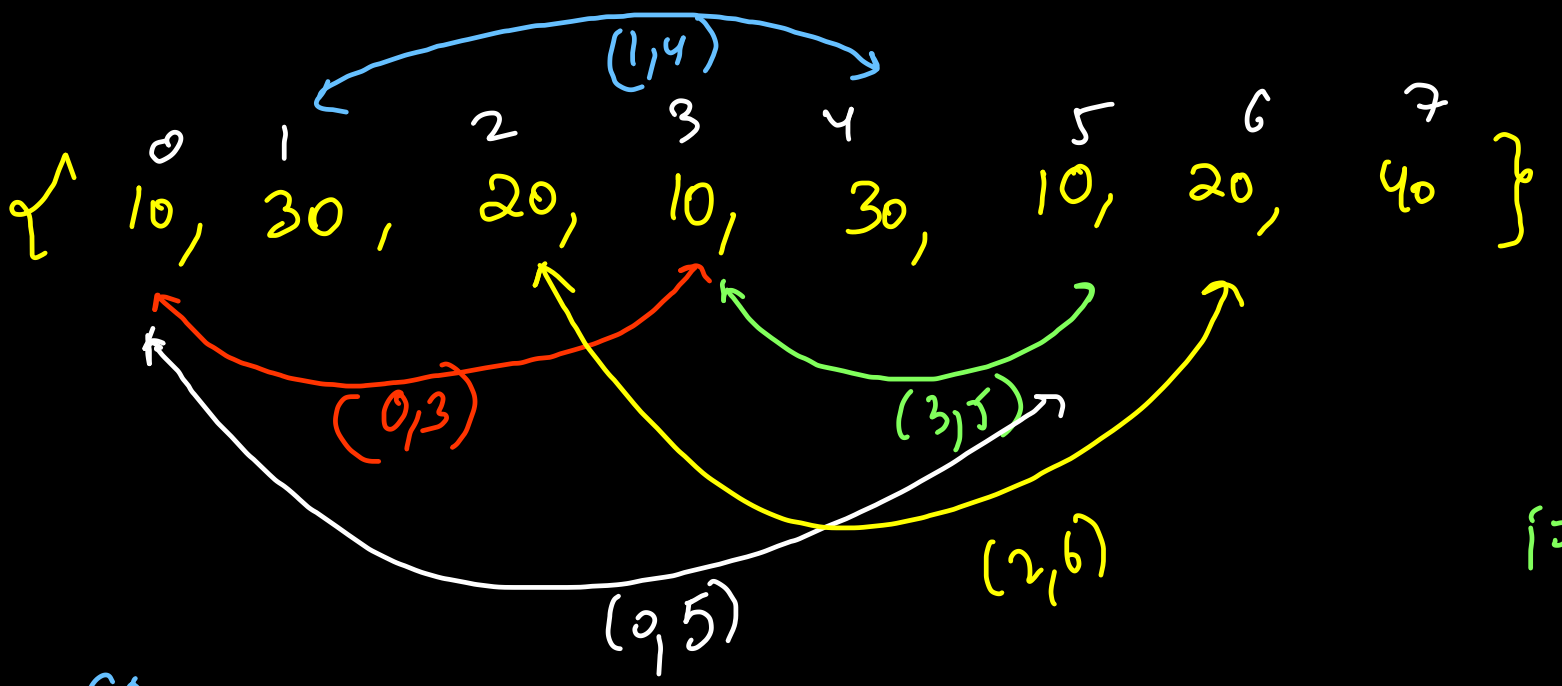
0 1 2
30, 20, 10

0 1 2
10, 20, 30

→ only 1 increasing & 1 decreasing

(1)





(i, j)

$arr[i] == arr[j]$

$i < j$

count = 0 1 2 3 4 5

$i = 0$

$j = 1$
2
3 ✓ 10 == 10
4
5 ✓ 10 == 10
6
7

$i = 1$ $j = 2$
3
4 30 == 30 ✓
5
6
7

$i = 2$ $j = 3, \ldots 7$

$i = 3$ $j = 4, \ldots 7$

$i = 4$ $j = 5, \ldots 7$

$i = 5$ $j = 6, 7$

$i = 6$

```java
class Solution {
    public int numIdenticalPairs(int[] nums) {
        int count = 0;

        for(int i = 0; i < nums.length; i++){
            for(int j = i + 1; j < nums.length; j++){
                if(nums[i] == nums[j]){
                    count++;
                }
            }
        }

        return count;
    }
}
```

Subarray

↓

part of array

→ non-empty (size ≥ 1)

→ contiguous part

→ same order as array

arr = { 10, 20, 30, 40 } (0 1 2 3) n = 4

Starting = 0

{ } → [10] [0,0]

{10} → [10,20] [0,1]

{10,20} → [10,20,30] [0,2]

{10,20,30} → [10,20,30,40] [0,3]

{10,20,30,40}

Starting = 1

{ } → [20] [1,1]

[20,30] [1,2]

[20,30,40] [1,3]

Starting = 2

[30] [2,2]

[30,40] [2,3]

Starting = 3

[40] [3,3]

no of subarrays

= 1+2+3+...n

$= \frac{n \times (n+1)}{2}$

```java
for(int left = 0; left < arr.length; left++){
    for(int right = left; right < arr.length; right++){
        for(int idx = left; idx <= right; idx++){
            System.out.print(arr[idx] + " ");
        }
        System.out.println();
    }
}
```

Approach ①

3 nested loop

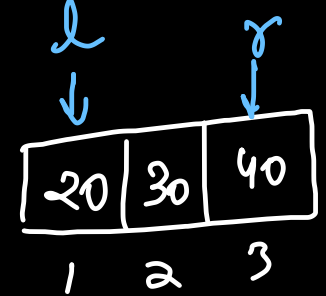
```java
for(int left = 0; left < arr.length; left++){
    ArrayList<Integer> subarray = new ArrayList<>();
    for(int right = left; right < arr.length; right++){
        subarray.add(arr[right]);
        System.out.println(subarray);
    }
}
```

Approach ②

2 nested loops

Leetcode 1588
Sum of all odd length subarrays

arr = {10, 20, 30, 40, 50}
0 1 2 3 4



⇒ length = r - l + 1

odd = 1
① {10} 10
① {20} 20
① {30} 30
① {40} 40
① {50} 50

odd = 3
{10, 20, 30} 60
{20, 30, 40} 90
{30, 40, 50} 120

odd = 5
{10, 20, 30, 40, 50} 150

+ 60 + 90 + 120 + 150

{ { {10}, {20}, ... {50},
{ } ...

{ }

left = 10, right = 15

① count elements b/w left & right including left & right

{10, 11, 12, 13, 14, 15} 6 ele

count ⇒ right - left + 1

[left, right]

② count elements b/w left & right excluding both left & right

{11, 12, 13, 14}

count ⇒ right - left - 1

(left, right)

③ count elements b/w left & right including either left or right

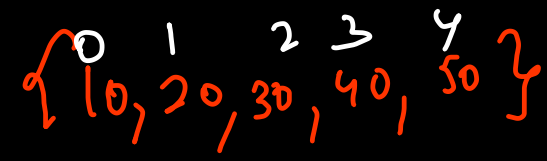
[left, right) {10, 11, 12, 13, 14}

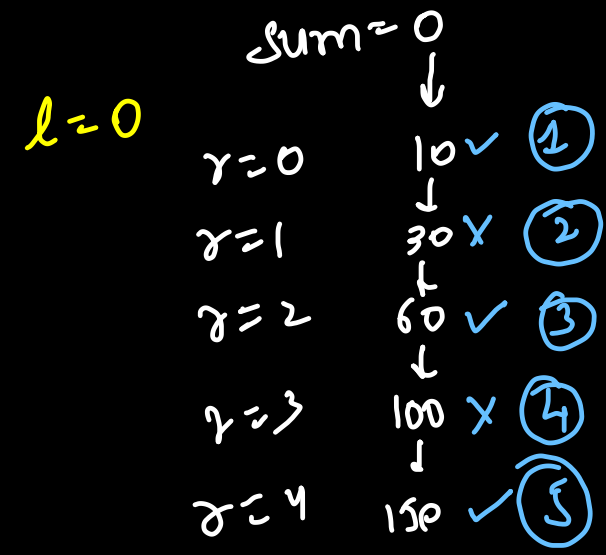
or

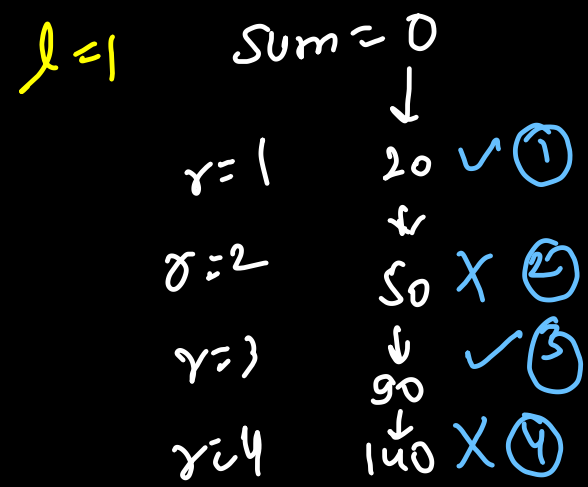
(left, right] {11, 12, 13, 14, 15}

Count = right - left

```java
public int sumOddLengthSubarrays(int[] arr) {
    int answer = 0;

    for(int left = 0; left < arr.length; left++){
        int sum = 0;
        for(int right = left; right < arr.length; right++){
            sum += arr[right];
            if((right - left + 1) % 2 == 1){
                answer += sum;
            }
        }
    }

    return answer;
}
```

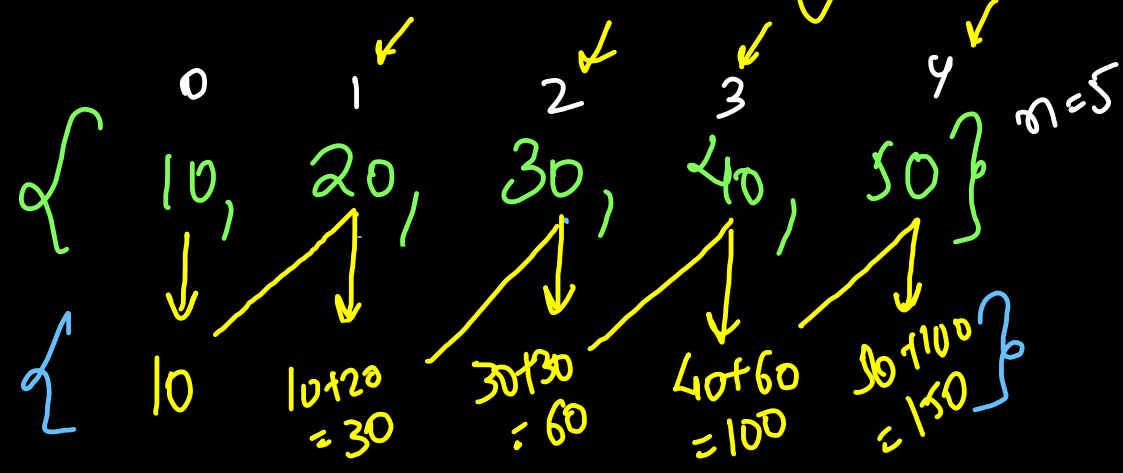






answer = 0 + 10 + 60 + 150
+ 20 + 90

Prefix Sum or Running Sum



pref[i] = arr[i] + pref[i-1]

i > 0

{10} {10,20} {10,20,30} {10,20,30,40} {10,20,30,40,50}

1 2 3 4 5

prefix ⇒ subarray with starting index as 0.

App 1

```java
public int[] runningSum(int[] nums) {
    int[] prefix = new int[nums.length];
    prefix[0] = nums[0];

    for(int idx = 1; idx < nums.length; idx++){
        prefix[idx] = nums[idx] + prefix[idx - 1];
    }

    return prefix;
}
```

LC 1480

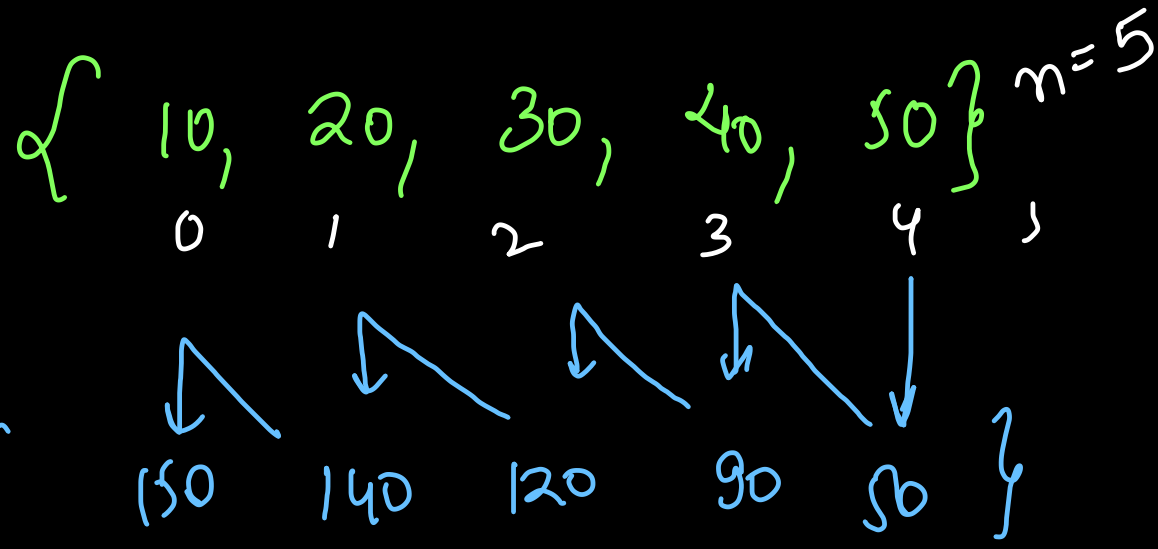
new prefix array

App 2

```java
public int[] runningSum(int[] nums) {
    for(int idx = 1; idx < nums.length; idx++){
        nums[idx] += nums[idx - 1];
    }

    return nums;
}
```

Inplace

Suffixes



suffix[i] = arr[i] + suffix[i+1]

{ 10, 20, 30, 40, 50 } { 20, 30, 40, 50 } { 30, 40, 50 }

{ 40, 50 } { 50 }

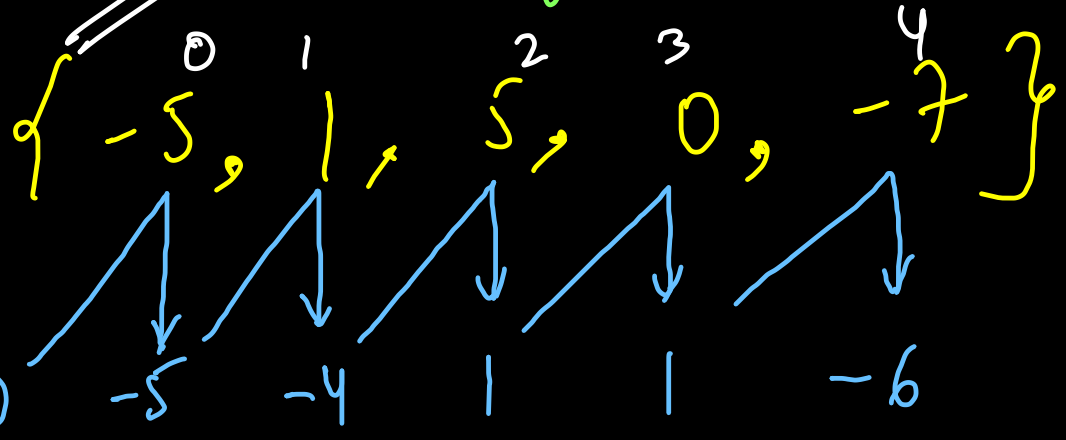
suffix ⇒ subarray → with ending index as the last index

```java
public int[] runningSum(int[] nums) {
    int[] prefix = new int[nums.length];

    for(int idx = 0; idx < nums.length; idx++){
        prefix[idx] = nums[idx] + ((idx > 0) ? prefix[idx - 1] : 0);
    }

    return prefix;
}
```

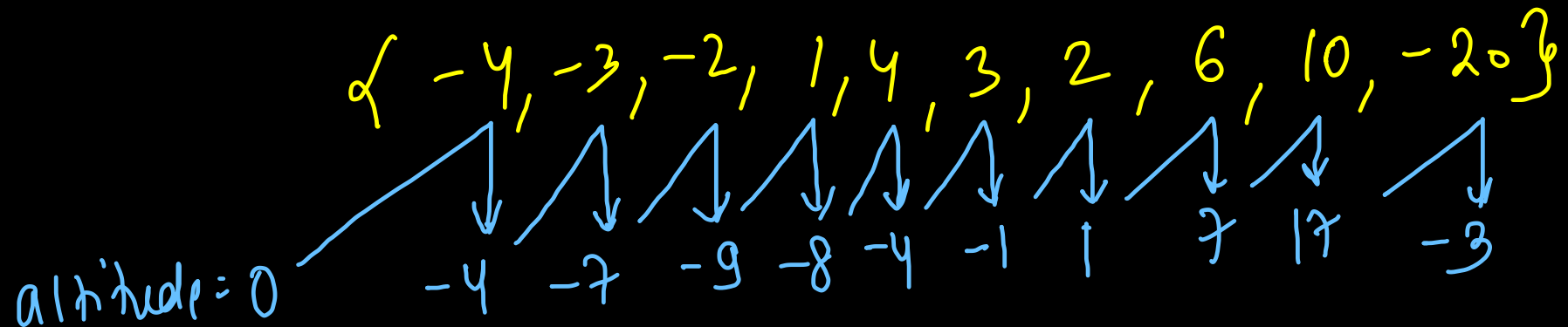
```java
public int[] runningSum(int[] nums) {
    int n = nums.length;
    int[] suffix = new int[n];
    suffix[n - 1] = nums[n - 1];

    for(int idx = n - 2; idx >= 0; idx--){
        suffix[idx] = nums[idx] + suffix[idx + 1];
    }

    return suffix;
}
```

```java
public int[] runningSum(int[] nums) {
    int n = nums.length;
    int[] suffix = new int[n];

    for(int idx = n - 1; idx >= 0; idx--){
        suffix[idx] = nums[idx] + ((idx < n - 1) ? suffix[idx + 1] : 0);
    }

    return suffix;
}
```

LC 1732 find highest altitude



answer = 1

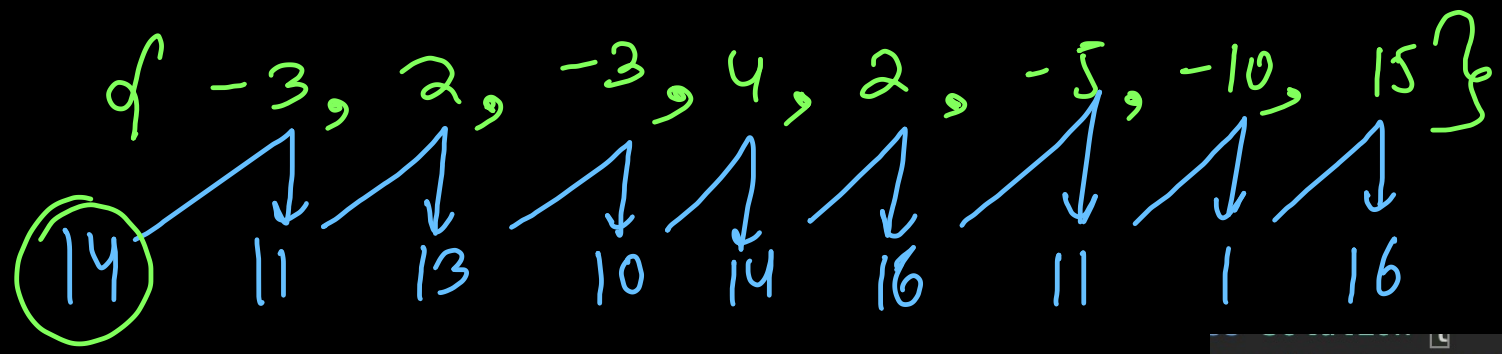


(highest) answer = ~~0~~ ~~1~~ ~~7~~ 17

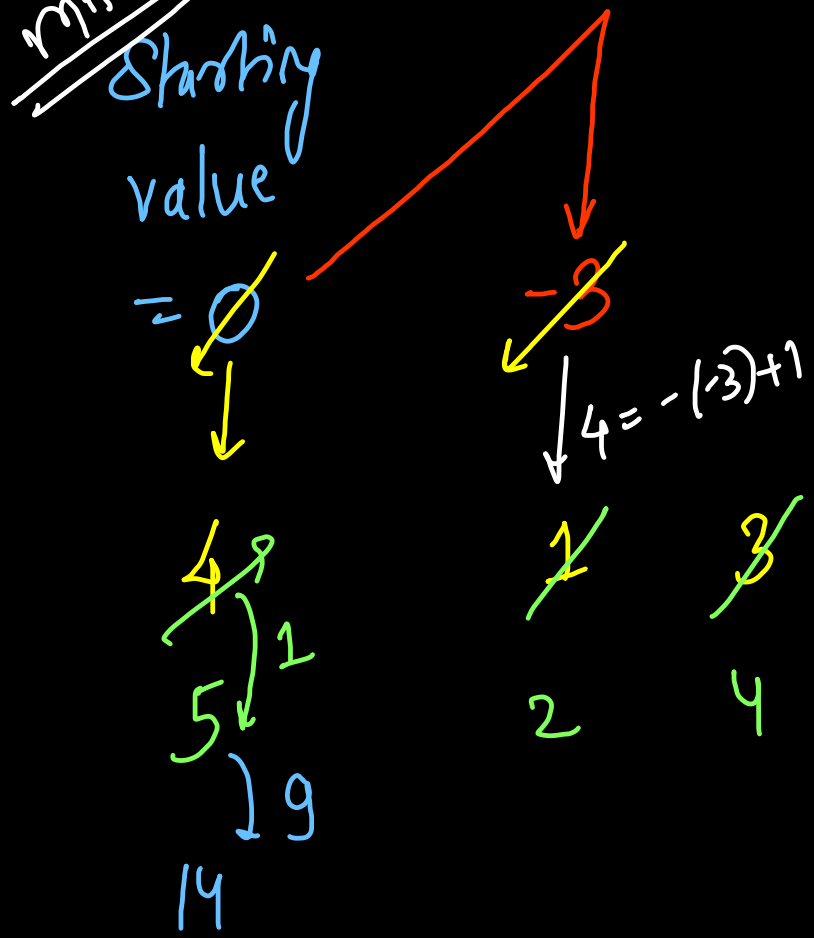
```java
public int largestAltitude(int[] gain) {
    int answer = 0, altitude = 0;

    // For Each Loop
    for(int val: gain){
        altitude += val;
        answer = Math.max(answer, altitude);
        // if(altitude > answer) answer = altitude;
    }

    return answer;
}
```

same

LC 1413

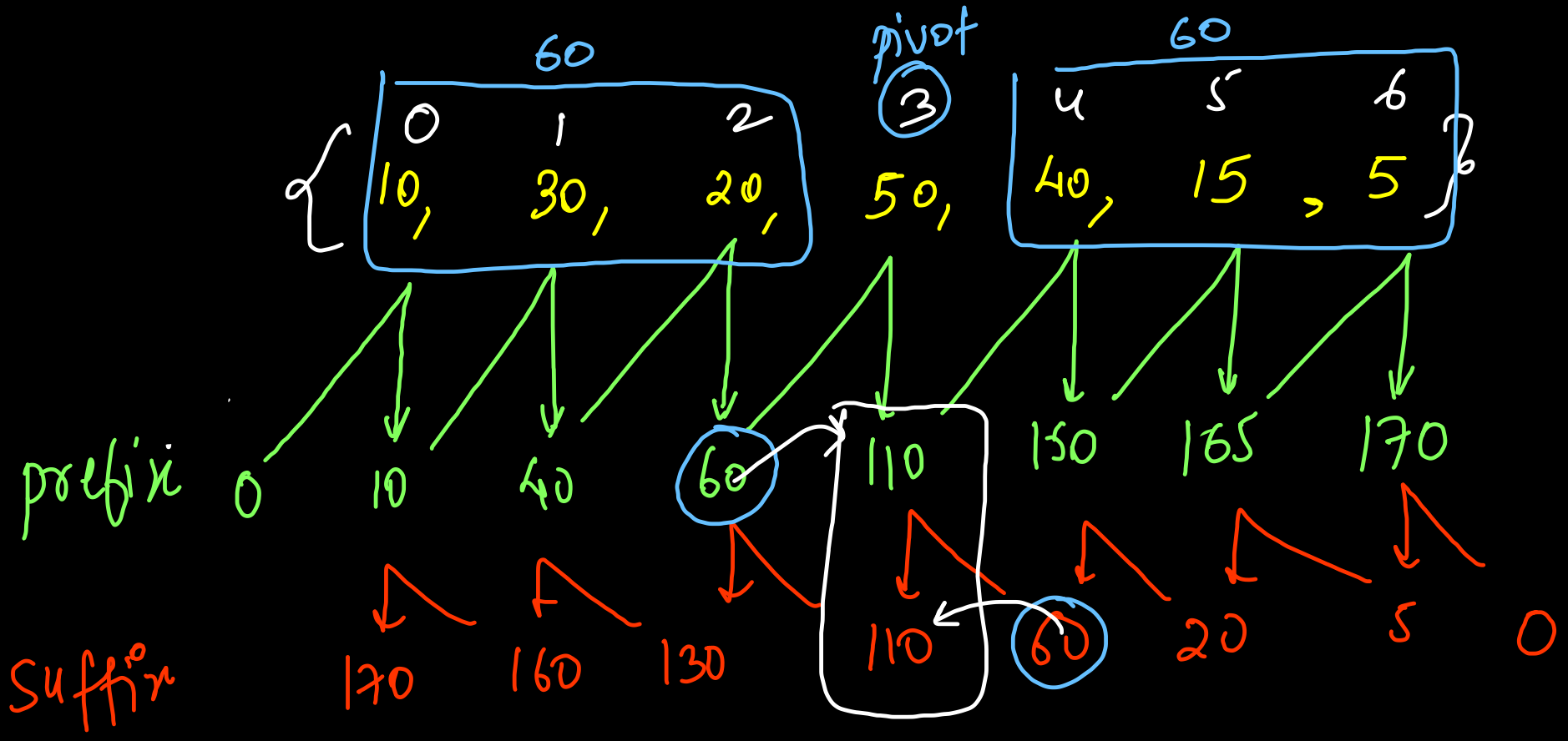


min'm Starting value



Note
↳ 0 is not considered a positive no

```java
public int minStartValue(int[] nums) {
    int sum = 1, starting = 1;

    for(int val: nums){
        sum += val;
        if(sum <= 0){
            starting += (1 - sum);
            sum = 1;
        }
    }

    return starting;
}
```



Sum of all left nos == Sum of all right nos

$[0, p-1] = [p+1, n-1] \Leftrightarrow [0,p] = [p, n-1]$

eg $[0,2) = [4,6] \Leftrightarrow [0,3] = [3,6]$

```java
public int[] getPrefix(int[] nums){
    int[] prefix = new int[nums.length];

    for(int idx = 0; idx < nums.length; idx++){
        prefix[idx] = nums[idx] + ((idx > 0) ? prefix[idx - 1] : 0);
    }

    return prefix;
}

public int[] getSuffix(int[] nums){
    int n = nums.length;
    int[] suffix = new int[n];

    for(int idx = n - 1; idx >= 0; idx--){
        suffix[idx] = nums[idx] + ((idx < n - 1) ? suffix[idx + 1] : 0);
    }

    return suffix;
}
```

```java
public int pivotIndex(int[] nums) {
    int[] prefix = getPrefix(nums);
    int[] suffix = getSuffix(nums);

    // left to right: leftmost pivot index
    for(int idx = 0; idx < nums.length; idx++){
        if(prefix[idx] == suffix[idx]){
            return idx;
        }
    }

    // no pivot index
    return -1;
}
```

Shallow copy (not deep)

Product of Array Except Self

App1 using division operator

arr: { 2, 5, 3, 4, 6 }
0 1 2 3 4

answer: { 5x3x4x6, 2x3x4x6, 2x5x4x6, 2x5x3x6, 2x5x3x4 }
product 2 | product 5 | product 3 | prod 4 | prod 6

product = 1 x 2 x 5 x 3 x 4 x 6

corner cases

division by 0

13:00

Corner case 1:

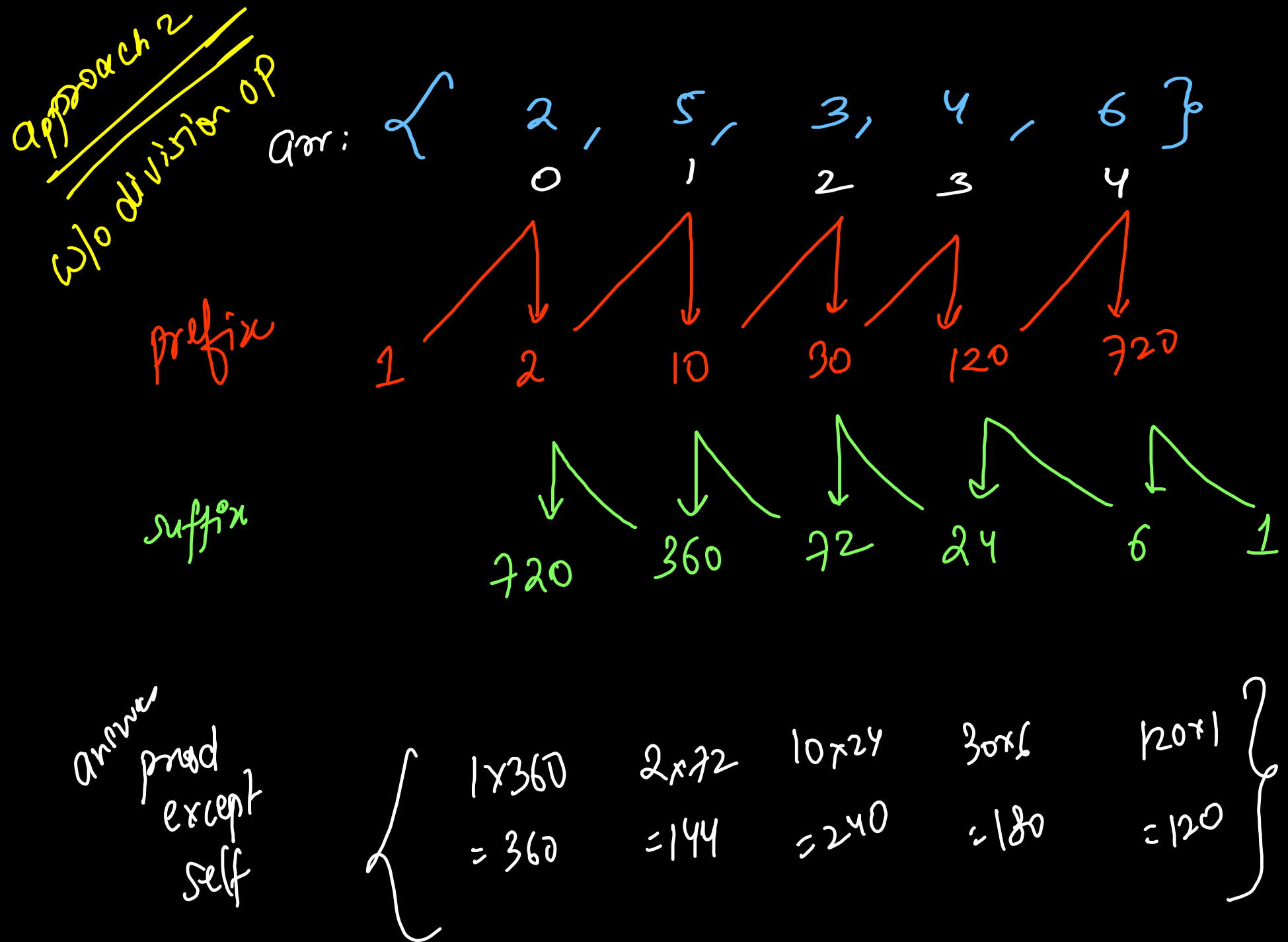
{ 2, 5, 4, 0, 8 }

0 0 0 120 0

Corner case 2

{ 2, 0, 4, 0, 3 }

0 0 0 0 0


approach 2
w/o division op
arr: { 2, 5, 3, 4, 6 }
0 1 2 3 4
prefix
1 2 10 30 120 720
suffix
720 360 72 24 6 1
answer
prod except self
{ 1x360 2x72 10x24 30x6 120x1
= 360 =144 =240 =180 =120 }

```java
public int[] getPrefix(int[] nums){
    int[] prefix = new int[nums.length];

    for(int idx = 0; idx < nums.length; idx++){
        prefix[idx] = nums[idx] * ((idx > 0) ? prefix[idx - 1] : 1);
    }

    return prefix;
}

public int[] getSuffix(int[] nums){
    int n = nums.length;
    int[] suffix = new int[n];

    for(int idx = n - 1; idx >= 0; idx--){
        suffix[idx] = nums[idx] * ((idx < n - 1) ? suffix[idx + 1] : 1);
    }

    return suffix;
}
```

If empty array ⇒ product = 1

```java
public int[] productExceptSelf(int[] nums) {
    int[] prefix = getPrefix(nums);
    int[] suffix = getSuffix(nums);

    int n = nums.length;
    int[] answer = new int[n];
    for(int idx = 0; idx < n; idx++){
        int left = (idx > 0) ? prefix[idx - 1] : 1;
        int right = (idx < n - 1) ? suffix[idx + 1] : 1;
        answer[idx] = left * right;
    }

    return answer;
}
```

if idx == 0 ⇒ pref[idx-1] will be pref[-1] which does not exist

if idx == n-1 ⇒ suff[idx+1] will be suff[n] which does not exist.

#C169 Majority Element - ①

arr :→ { 2, 3, 2, 3, 2, 4, 2, 2, 5 } n = 9

answer = ②

strictly more than $\lfloor n/2 \rfloor$ ⇒ floor $(n/2) = \lfloor 9/2 \rfloor = 4$

#Approach ① :→ Count frequency of each element (hashmap)

2 → 5 3 → 2 5 → 1 4 → 1

Boyer Moore Voting Algorithm

n = 14

arr: { 10, 10, 20, 40, 40, 10, 10, 40, 40, 40, 40, 40, 40, 20 }

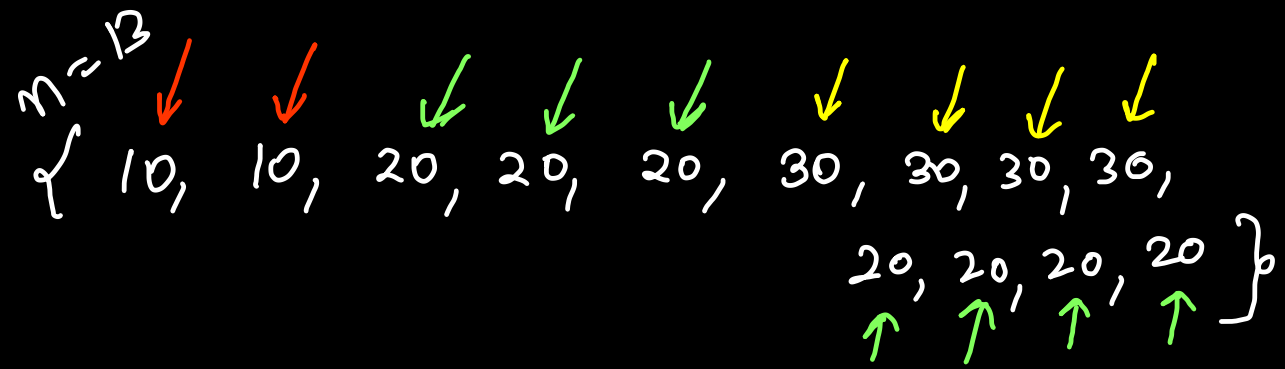
majority = 0 → 10 → 40 → 10 → (40) → answer

lead = 0 → 1 → 2 → 1 → 0 → X0 → 1 → 0 → 1 → 2 → 3 → 4 → 3 → 4

If majority element exist then it will pointed by answer,

Otherwise, no majority ele.

```java
public int majorityElement(int[] nums) {
    int majority = 0, lead = 0;

    for(int val: nums){
        if(val == majority){
            lead++;
        } else if(lead == 0){
            majority = val;
            lead = 1;
        } else {
            lead--;
        }
    }

    return majority;
}
```



majority = 20 ⇒ freq = $\lfloor 13/2 \rfloor + 1$

$= 6 + 1 \Rightarrow 7$ ✓

↳ this code will work if majority element exists

LC (229) Majority Element - (II)

eg (1) { 10, 20, 30, 10, 20, 30, 10, 30 } $n = 8$

answer = {10, 30}

eg (2) { 10, 20, 30, 10, 20, 30, 10, 50 } $n = 8$

answer = {10}

eg (3) { 10, 20, 30, 10, 20, 30, 40, 50 } $n = 8$

answer = { }

freq $> \lfloor n/3 \rfloor$

$> \lfloor 8/3 \rfloor$

eg: { 10, 20, 30, 10, 20, 10, 30, 30 }

majA = 0 → 10 → actual? i.e freq > [n/3] → > 2

leadA = 0 → 1 → 0 → 1 → 2 → 1

majB = 0 → 20 → 30 ⇒ freq > 2

leadB = 0 → 1 → 0 → 1 → 0 → 1

eg ② { 10, 20, 30, 10, 20, 10, 10, 50 }

majA = ∅ (10) ⇒ actual > 2 ? yes

leadA = ∅ ∅ ∅ 2

majB = ∅ (20) ⇒ actual > 2 ? no

leadB = ∅ ∅ 0

```java
public boolean isActualAnswer(int[] nums, int target){
    int n = nums.length;
    int freq = 0;

    for(int val: nums){
        if(val == target){
            freq++;
        }
    }

    if(freq > n / 3) return true;
    else return false;
}
```

RC 229

```java
public List<Integer> majorityElement(int[] nums) {
    int majA = Integer.MIN_VALUE, leadA = 0;
    int majB = Integer.MAX_VALUE, leadB = 0;

    for(int val: nums){
        if(val == majA){
            leadA++;
        } else if(val == majB){
            leadB++;
        } else if(leadA == 0){
            majA = val;
            leadA = 1;
        } else if(leadB == 0){
            majB = val;
            leadB = 1;
        } else {
            leadA--; leadB--;
        }
    }

    List<Integer> answers = new ArrayList<>();
    if(isActualAnswer(nums, majA) == true) answers.add(majA);
    if(isActualAnswer(nums, majB) == true) answers.add(majB);
    return answers;
}
```

$n/2 \rightarrow 1$

$n/3 \rightarrow 2$

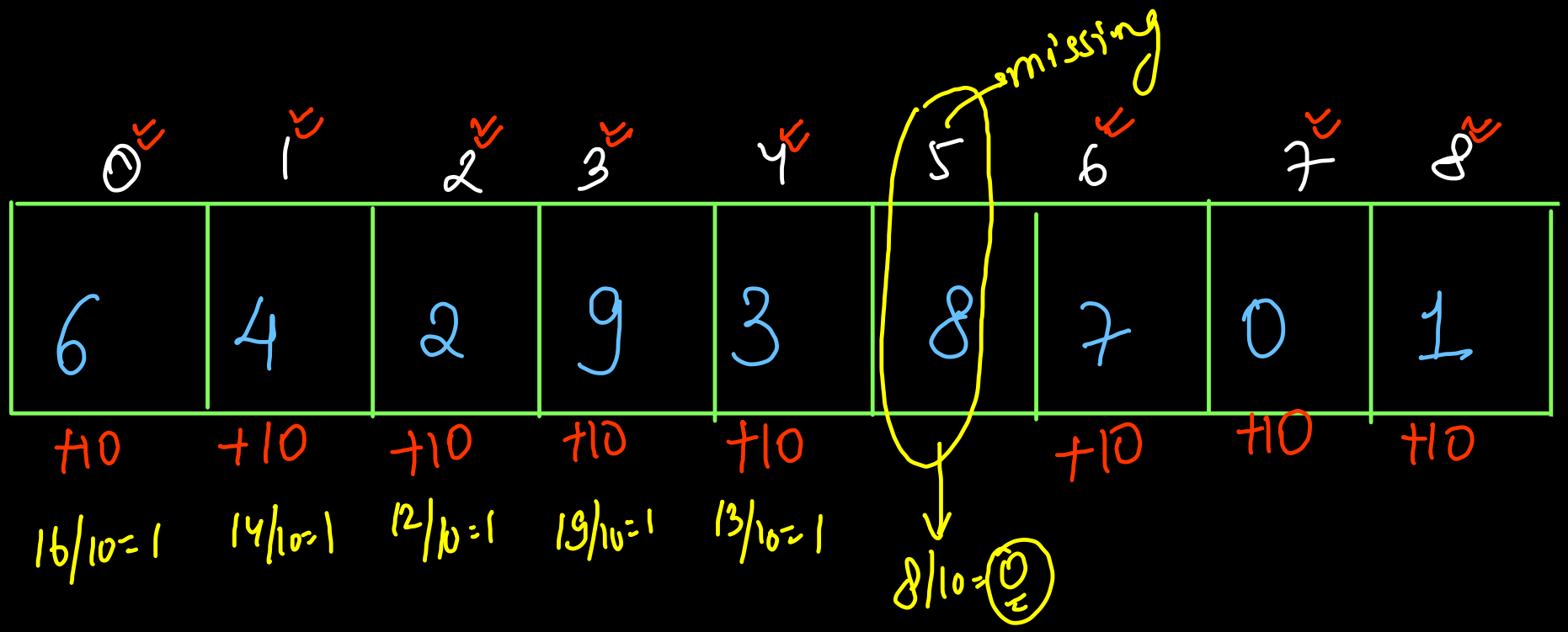
$n/k \rightarrow (k-1)$ majority elements

Missing & Duplicate : Indexed Based Hashmap

values
$[-10^9, 10^9]$
⇓
hashmap

values → $[0, n-1]$
$[1, n]$
$[0, n]$
$[1, n-1]$

array ∀
visited marker
(without hashmap)



n=9

[0,9]

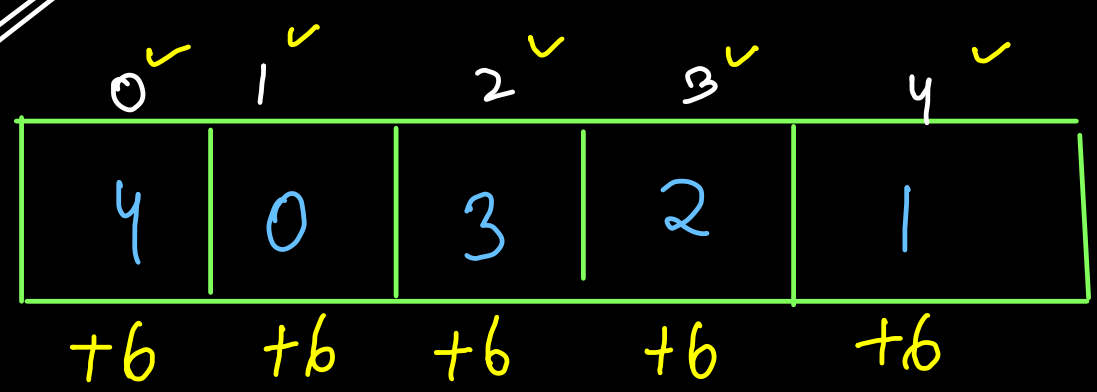
visited ⇒ add (n+1)

original value ⇒ mod (n+1)

frequency ⇒ divide by (n+1)

min value which does not exist in array ⇒ (n+1)

Corner case



return n+1 (5)

[0,5]

n+1 = 6

n = 5

```java
public int missingNumber(int[] nums) {
    int n = nums.length;
    int sum = n * (n + 1) / 2;
    for(int val: nums){
        sum -= val;
    }
    return sum;
}
```

#Other approach

Sum of [0,n] - sum of array

$= \frac{n \times (n+1)}{2} - sum$

```java
public int missingNumber(int[] nums) {
    int n = nums.length;

    // visited mark
    for(int idx = 0; idx < n; idx++){
        int original = nums[idx] % (n + 1);
        if(original < n)
            nums[original] += (n + 1);
    }

    // find missing number
    for(int idx = 0; idx < n; idx++){
        int freq = nums[idx] / (n + 1);
        if(freq == 0) return idx;
    }

    return n;
}
```

#companies

Service-based

TCS, Infosys, Wipro,
Capgemini, Cognizant,
Accenture, etc

3-7 LPA

product-based companies

"FAANG"

Microsoft, Salesforce,
Ola, Flipkart, Paytm,

tech giants

M-P

DSA ⇒ 50% focus

Core + Project + LLD ⇒ 20% focus

startups

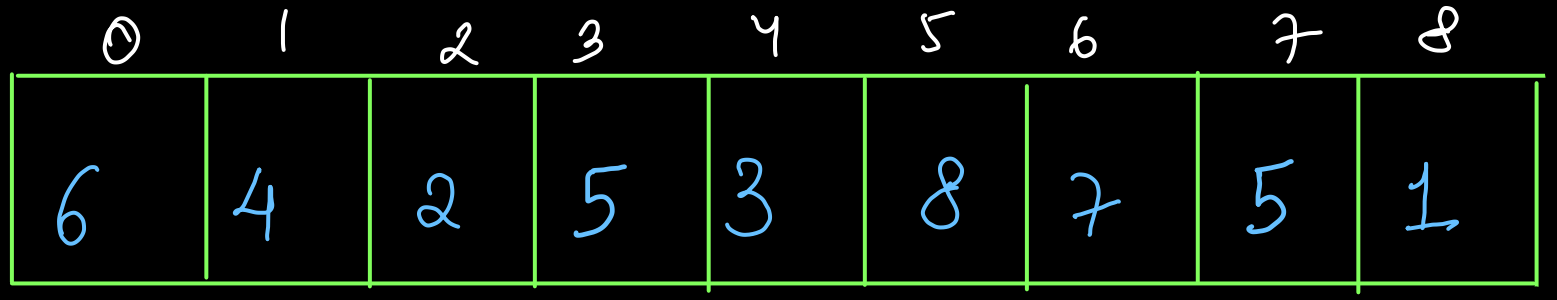
E-M

DSA ⇒ 40%

Dev ⇒ 40%

Core + LLD ⇒ 20%

LC 287) Duplicate Number (repeating or)



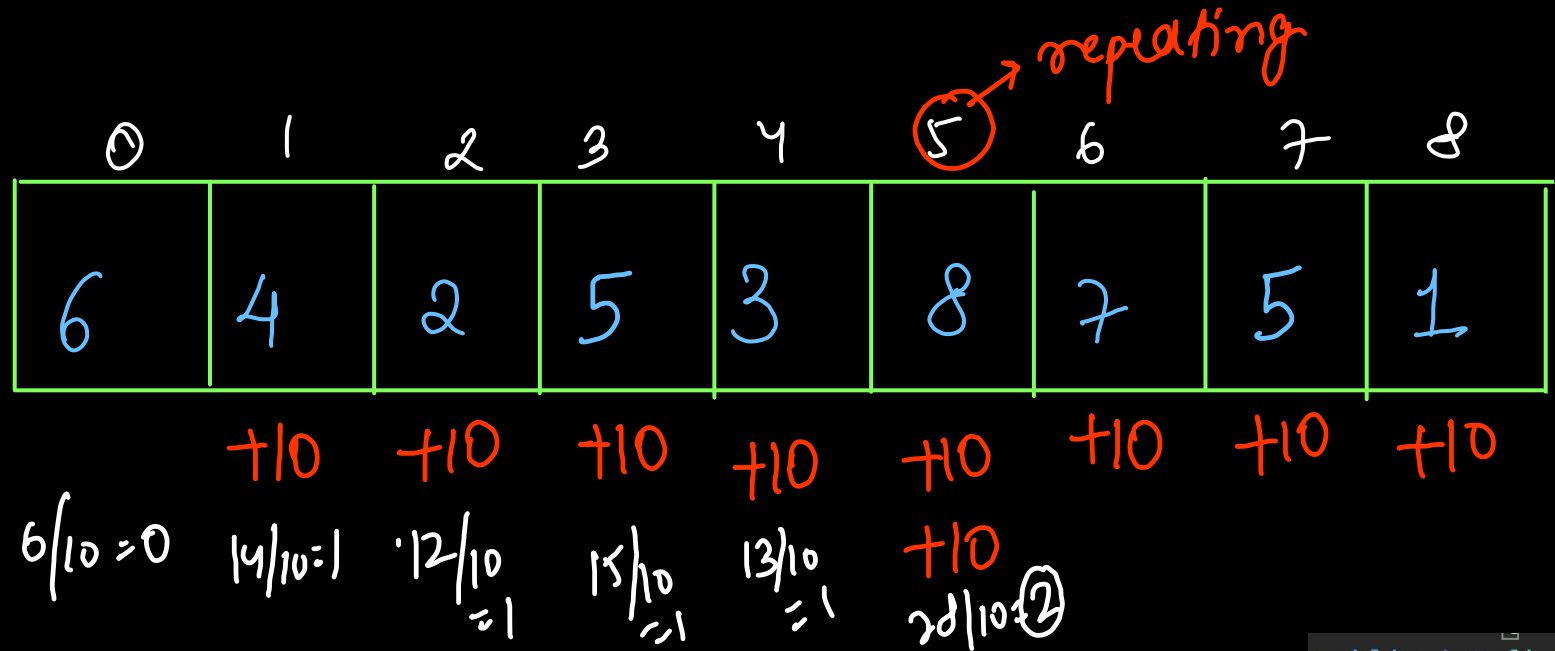
$[1, n-1]$

$n = 9$

App① Sum of array elements − sum of first N natural no's

$(1+2+\cdots 8)+5 - (1+2+3+\cdots 8)$

$= 5$



(visited mark)

$n = 9$

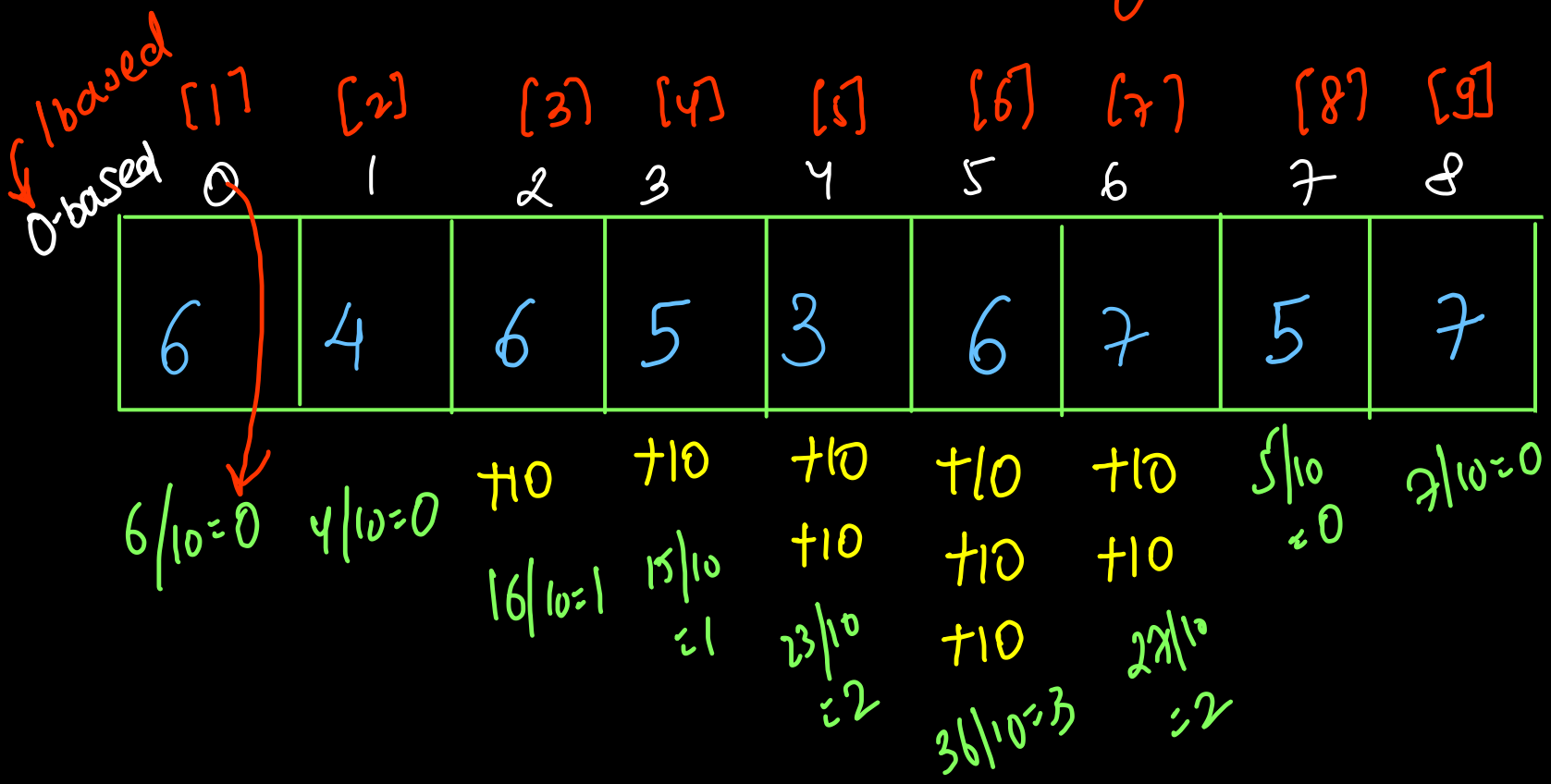
$n+1 = 10$

LC 287

repeating / duplicate no $\Rightarrow$ freq > 1

```java
public int findDuplicate(int[] nums) {
    int n = nums.length;

    // visited mark
    for(int idx = 0; idx < nums.length; idx++){
        int original = nums[idx] % (n + 1);
        nums[original] += (n + 1);
    }

    // find the repeating or duplicate
    for(int idx = 0; idx < nums.length; idx++){
        int freq = nums[idx] / (n + 1);
        if(freq > 1) return idx;
    }

    return n;
}
```

LC 448 find all missing no's



$n=9$

$n+1=10$

values: $[1, n]$ → hash (visited)

indices: $[0, n-1]$

missing no's = $\{1, 2, 8, 9\}$

duplicate = $\{5, 6, 7\}$

LC448) Find all missing

```java
class Solution {
    public List<Integer> findDisappearedNumbers(int[] nums) {
        int n = nums.length;

        // visited mark
        for(int idx = 0; idx < nums.length; idx++){
            int original = nums[idx] % (n + 1);
            nums[original - 1] += (n + 1);
        }

        // find all missing numbers
        List<Integer> missing = new ArrayList<>();

        for(int idx = 0; idx < nums.length; idx++){
            int freq = nums[idx] / (n + 1);
            if(freq == 0) missing.add(idx + 1);
        }

        return missing;
    }
}
```

↳ 1based to 0 based

↳ 0based to 1based

LC 442) Find all duplicates

```java
public List<Integer> findDuplicates(int[] nums) {
    int n = nums.length;

    // visited mark
    for(int idx = 0; idx < nums.length; idx++){
        int original = nums[idx] % (n + 1);
        nums[original - 1] += (n + 1);
    }

    // find all missing numbers
    List<Integer> duplicate = new ArrayList<>();

    for(int idx = 0; idx < nums.length; idx++){
        int freq = nums[idx] / (n + 1);
        if(freq > 1) duplicate.add(idx + 1);
    }

    return duplicate;
}
```

LC 645) Set mismatch (HW)

n=9

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

+10 +10 +10 +10 +10 +10

6/10=0 4/10=0 6/10=0 15/10=1 13/10=1 +10 +10 +10 28/10=2 27/10=2 25/10=2 17/10=1

max repeating = 0 → 3 → 5

max freq = 0 → 1 → 2

tie breaker ⇓ two values with same freq ⇒ pick smaller value

most frequent element

$k \le n$ [0, k-1] ↳ [0, n-1] array range ⇒ index range

```java
int maxRepeating(int[] nums, int n, int k) {
    // visited mark (add n + 1)
    for(int idx = 0; idx < n; idx++){
        int original = nums[idx] % (n + 1);
        nums[original] += (n + 1);
    }

    // maximum repeating number
    int maxRepeating = 0, maxFreq = 0;
    for(int idx = 0; idx < n; idx++){
        int freq = nums[idx] / (n + 1);
        if(freq > maxFreq){
            maxRepeating = idx;
            maxFreq = freq;
        }
    }

    return maxRepeating;
}
```

Max^m repeating number.

41. First Missing Positive

Hint

Hard 13.9K 1.6K

Companies

Given an unsorted integer array `nums`, return the smallest missing positive integer.

You must implement an algorithm that runs in `O(n)` time and uses constant extra space.

{ 7, 2, 5, 1, 4, 3, 8, 10 }

smallest missing +ve integer

⇒ 6

Brute force/naïve solution :→

```
for(int no=1; true; no++){
    if(linearSearch(nums, no) == -1)
        return no;
}
```

eg) $\{7, 2, 5, 1, 4, 3, 8, 10\}$ answer = ⑥

Corner case no 1 $n=9$ [1-9] values

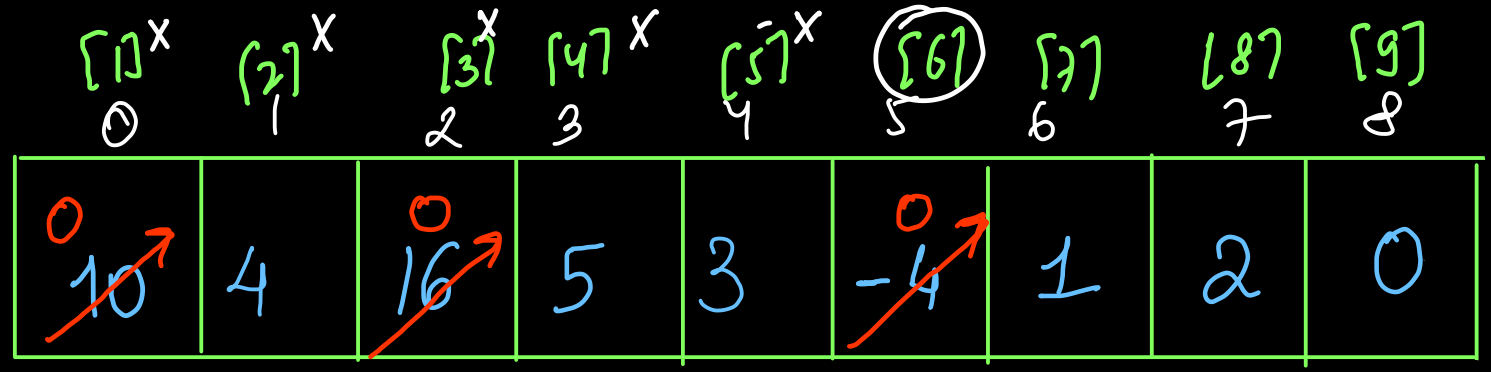
$\{7, 2, 5, 1, 4, 3, 8, 6, 9\}$ answer = ⑩

Corner case 2 $n=7$

$\{0, -100, 100, 10, 60, -2^{31}, 2^{31}-1\}$ answer = ①

Observation: Smallest missing integer will be in range $[1, n+1]$

(count sort)



$n = 9$

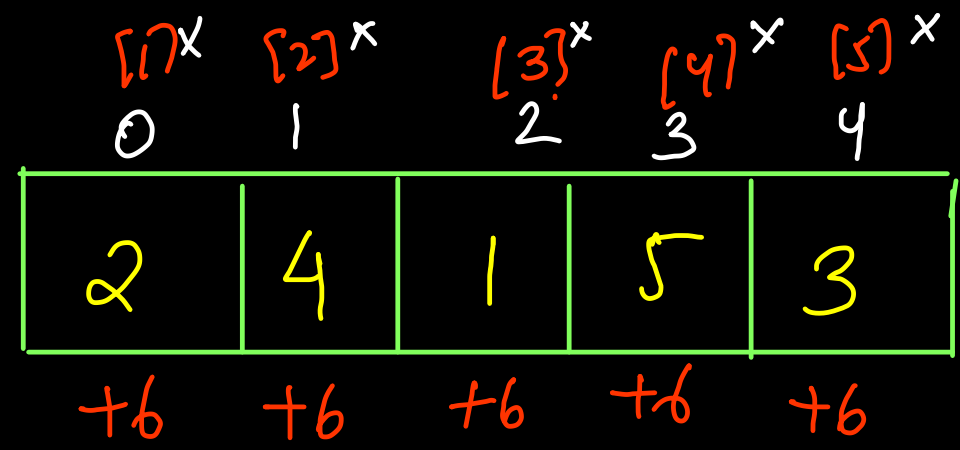
answer can be $[1, n+1]$

(1) discard out of range values $\boxed{<0 \text{ or } >n}$ ⇒ make mem(0)

(2) visited mark

(3) smallest +ve integer which is missing

```java
public int firstMissingPositive(int[] nums) {
    int n = nums.length;

    // discard the numbers out of range (< 0 or > n)
    for(int idx = 0; idx < n; idx++){
        if(nums[idx] < 0 || nums[idx] > n){
            nums[idx] = 0;
        }
    }

    // visited mark (add n + 1): 1 based indexing
    for(int idx = 0; idx < n; idx++){
        int original = nums[idx] % (n + 1);
        if(original > 0) {
            nums[original - 1] += (n + 1);
        }
    }

    // smallest missing +ve integer
    for(int idx = 0; idx < n; idx++){
        int freq = nums[idx] / (n + 1);
        if(freq == 0) return (idx + 1);
    }

    return (n + 1); // corner case
}
```



Smallest missing +ve ⇒ (6)

$n = 5$

$n + 1 = 6$