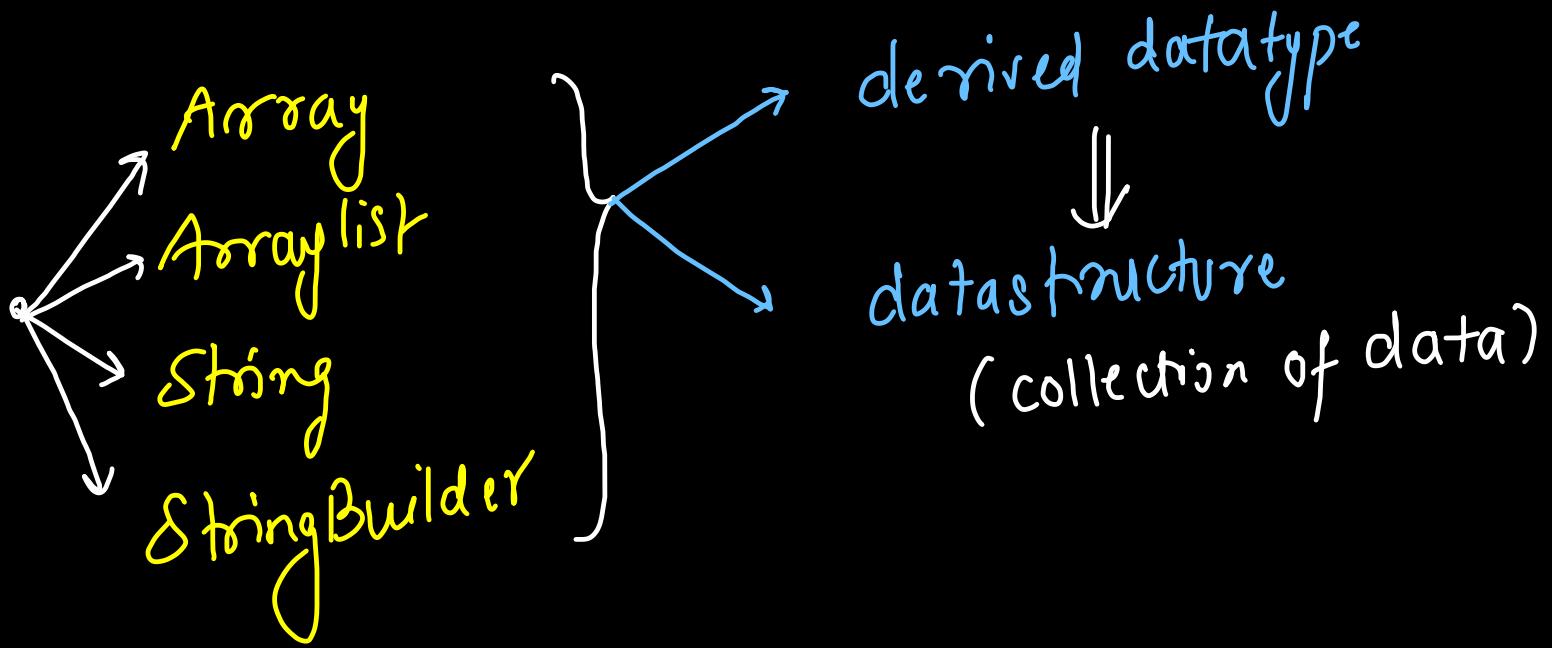


Arrays & Strings

primitive datatype ~ single unit of data



Array

- ⇒ fixed size
- ⇒ contiguous memory allocation
- ⇒ random access
- ⇒ similar datatype

Array Syntax

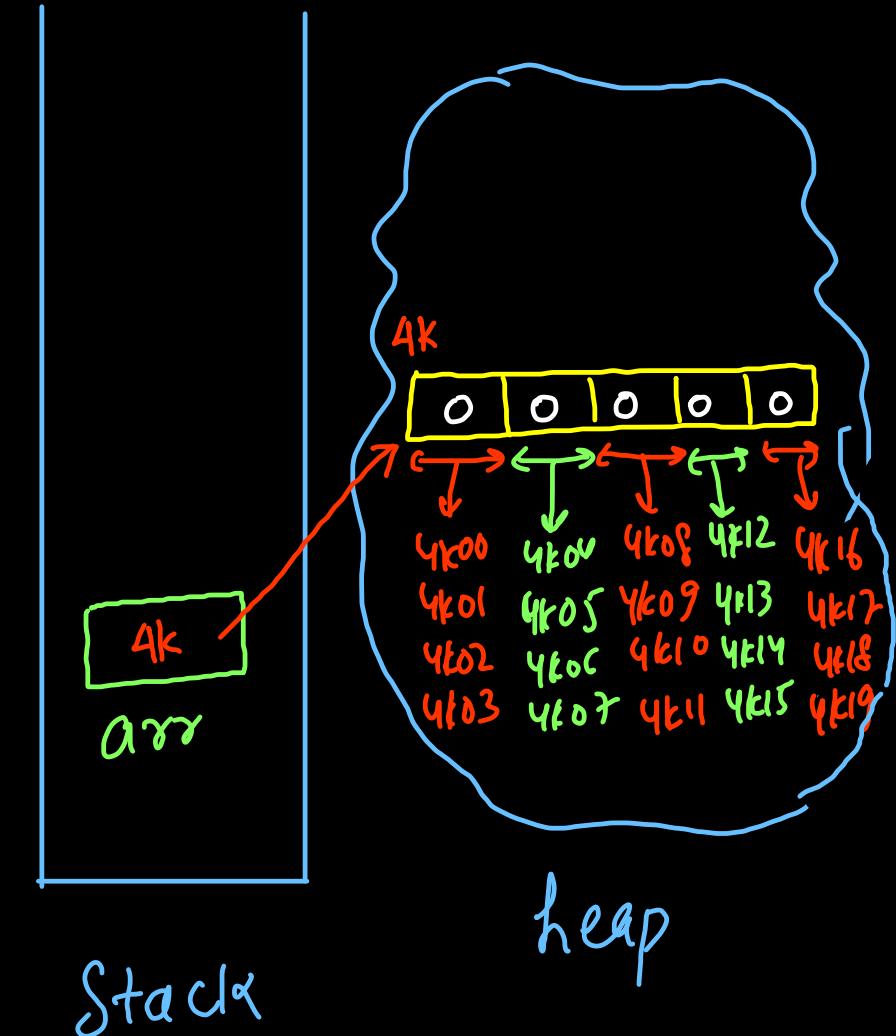
~~int~~ int [] arr = new int [5];

↑
datatype

↓
Stack ↗
reference variable
↓
declaration

↓
heap ↗
array (instance object)
↓
initialization
(array object creation)

↑
size



default value

→ int / short / byte / long ⇒ 0

→ float / double ⇒ 0.0

→ boolean ⇒ false

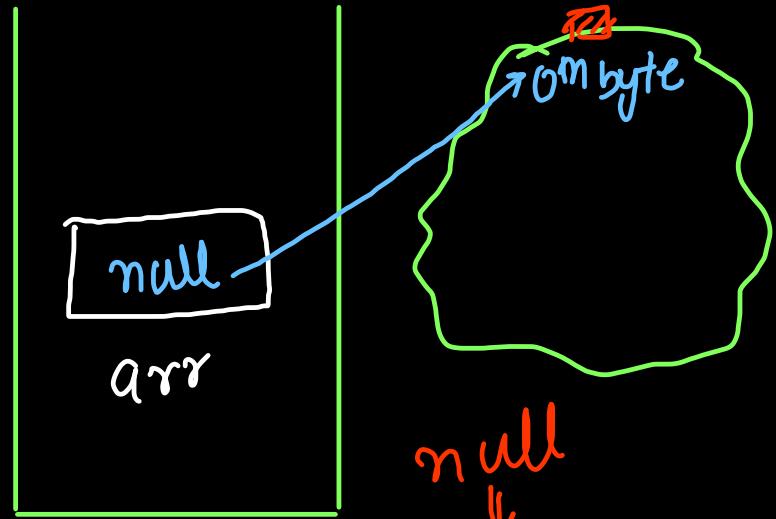
→ char ⇒ '\0' null character

Java is a fool-proof language

abstraction ↗ java ↗ address X
garbage X
reference variable X

array declaration only
and no initialization

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



arr is not
pointing to
any valid
memory
address

random access (0-based indexing)

Base address = 4k

4k	0	1	2	3	4
	0	0	0	0	0

4k 4k01 4k02 4k03 4k04 4k05 4k06 4k07 4k08, 09, 10, 4k11 4k12, 13, 14, 4k15 4k16, 17, 18, 4k19

$$\text{arr}[0] = \text{BA} + \frac{\text{size}}{\text{datatype}} * \text{index}$$

$$\text{arr}[0] = 4k + 4 * 0 = 4k$$

$$\text{arr}[1] = 4k + 4 * 1 = 4k04$$

$$\text{arr}[2] = 4k + 4 * 2 = 4k08$$

$$\text{arr}[3] = 4k + 4 * 3 = 4k12$$

$$\text{arr}[4] = 4k + 4 * 4 = 4k16$$

block address =
 base address + datatype size * index
 $\text{size} \Rightarrow 5 \Rightarrow N$
 $\text{indices} \Rightarrow [0, 4] \Rightarrow [0, N-1]$

```
// 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!

```
// 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

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

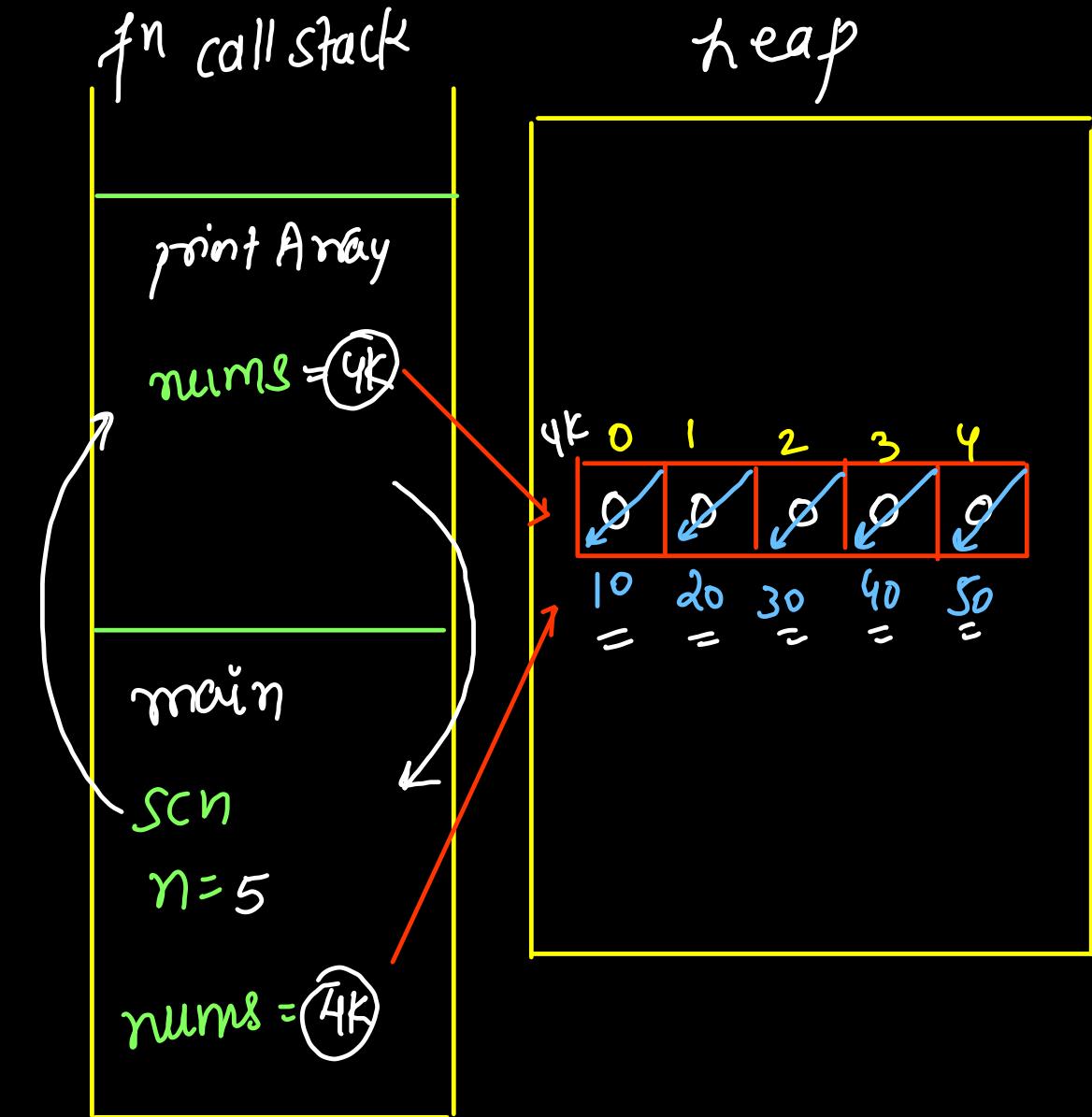
    int[] nums = new int[n];
    System.out.println(nums);
    System.out.println(Arrays.toString(nums));

    // 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] + " ");
    }
}

```



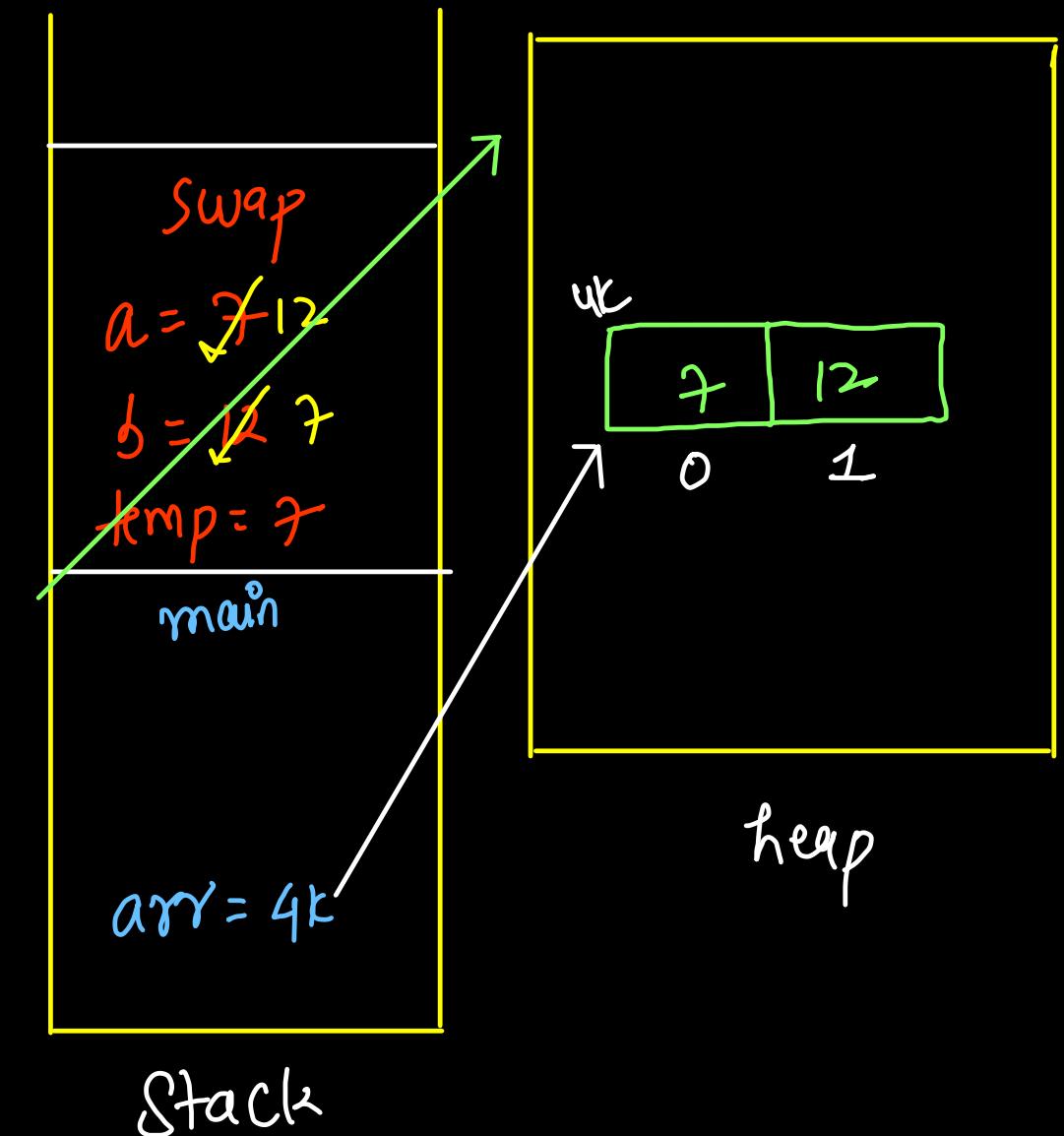
"Java is always pass by value"

```

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);
}

```



```

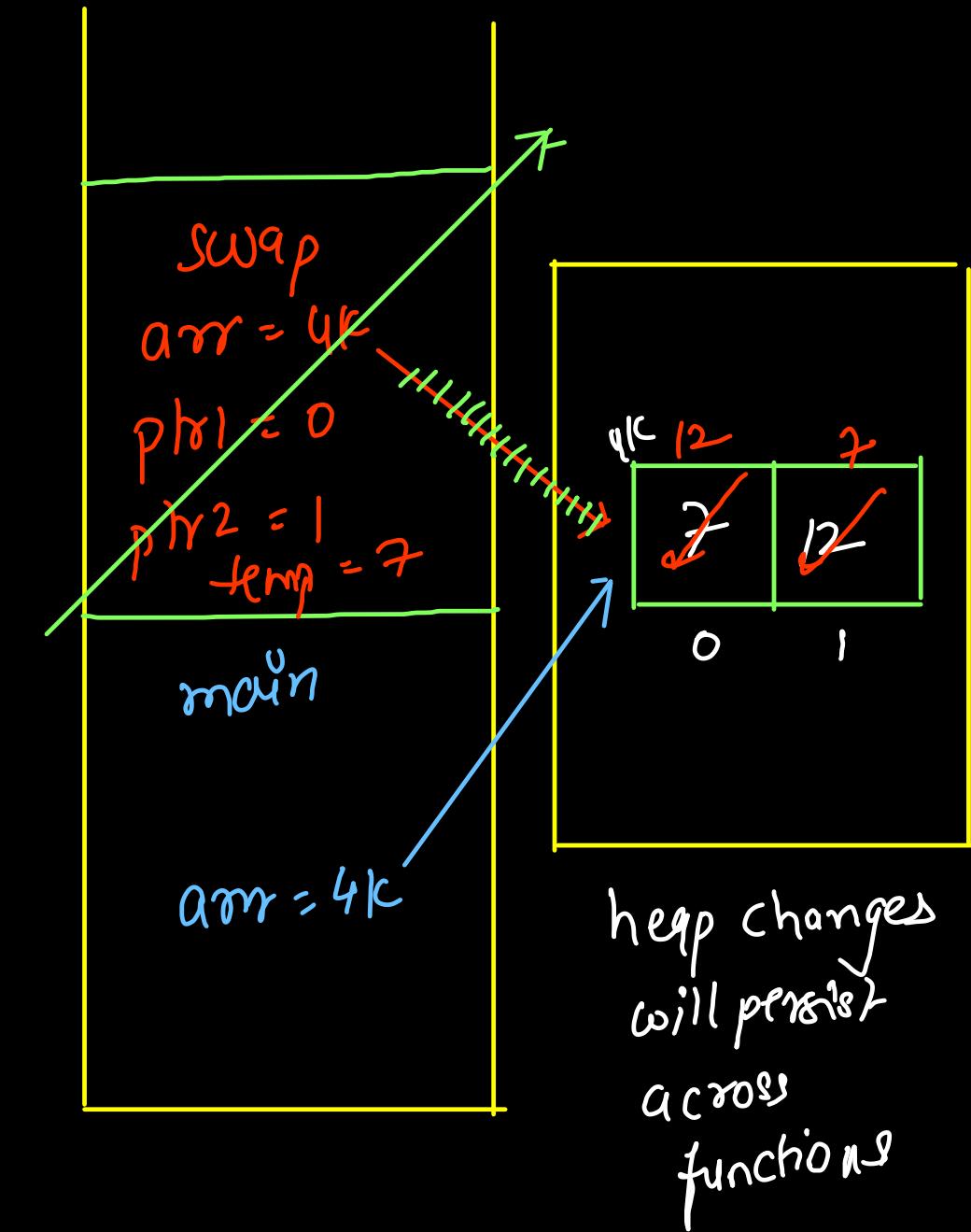
public static void main(String[] args) {
    // This Array will also be in heap
    int[] arr = {7, 12};
    // System.out.println(Arrays.toString(arr));
    // swap(arr[0], arr[1]);
    // System.out.println(Arrays.toString(arr));
    System.out.println(Arrays.toString(arr));
    swap(arr, 0, 1);
 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

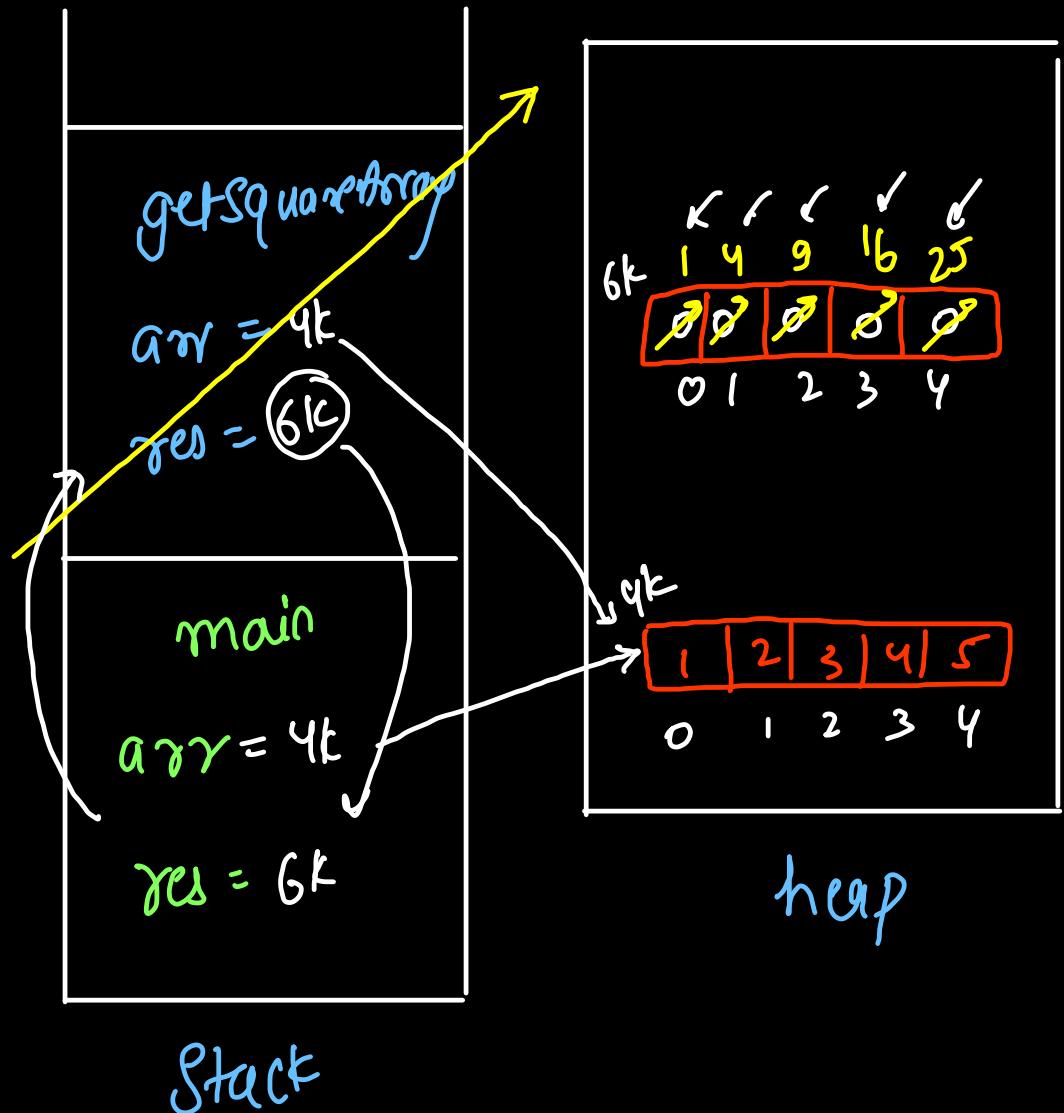


```

public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5}; 6k
    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;
} 6k

```



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;
```

```
// 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 $n = 10^6 \Rightarrow$ same time

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

~~Shallow Copy~~

```
def int[] arr = {10, 20, 30};  
// deep copy  
bt 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));
```

} different array objects

Finished in 80 ms

[100, 200, 300]

[100, 200, 300]

[10, 20, 30]

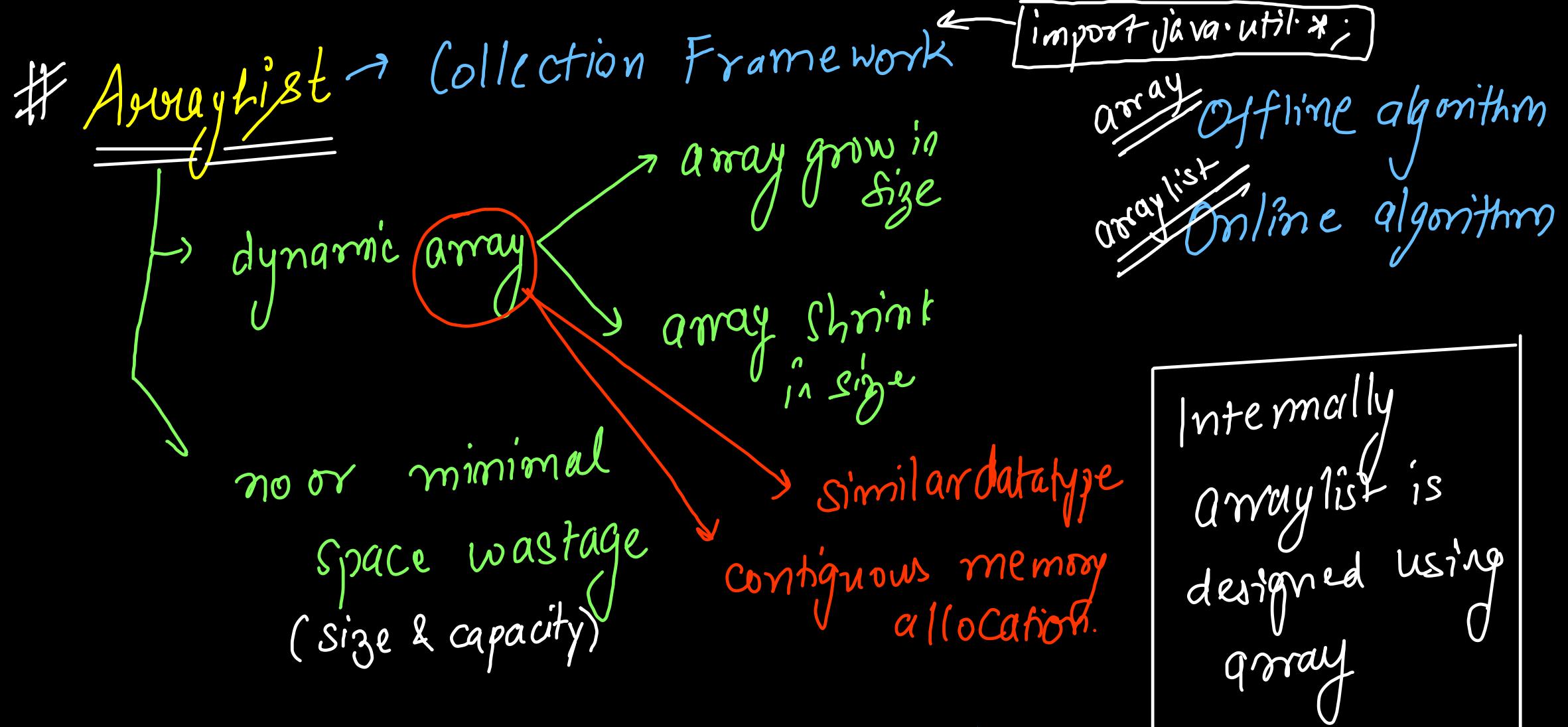
Deep copy of bigger array

will take more time

than deep copy of smaller array

Deep copy will take more time than shallow copy

updates in one array
do not effect deep copied array



ArrayList cannot be containing primitives.

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

primitive variables (stack)

Wrapper classes (heap)

① int → Integer

char → Character

long → Long

boolean → Boolean

double → Double

float → Float

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

↳ empty arr \Rightarrow size = 0 []

Creation (Inserⁿ)

add ✓

push

enqueue

put

Read (Get)

get ✓

charAt

peek

[]

Update (Set)

set ✓

setCharAt

put

Deletion / Removal

remove ✓

delete

pop

dequeue

~~add last~~

nums.add(10);

[] \rightarrow [10]
 0

nums.add(20);

[10] \rightarrow [10, 20]
 0 1

nums.add(30);

[10, 20] \rightarrow [10, 20, 30]
 0 1 2

~~get~~

nums.get(0) \Rightarrow 10

nums.get(1) \Rightarrow 20

nums.get(2) \Rightarrow 30

nums.get(-1)
nums.get(3)

Index out of
bound
exception

```

// 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);
}

```

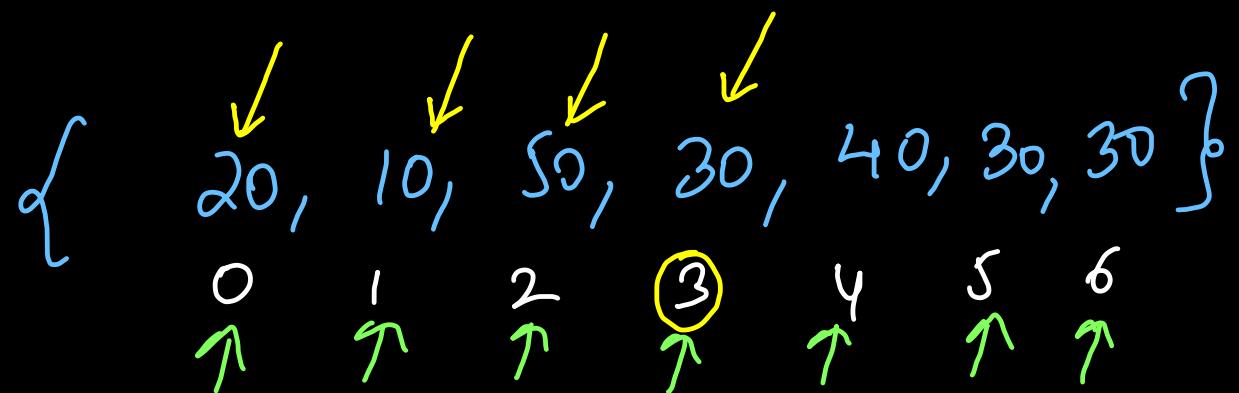
$[10, 20, 30, 40, 50] \quad n=5$

Finished in 107 ms

[]
[10]
[10, 20]
[10, 20, 30]
[10, 20, 30, 40]
[10, 20, 30, 40, 50]
0 : 10
1 : 20
2 : 30
3 : 40
4 : 50
[1, 2, 3, 4, 5]
[1, 2, 3, 4]
[1, 2, 3]
[1, 2]
[1]
[]

Linear Search Algorithm

steps / procedure \Rightarrow input \rightarrow processing \rightarrow output



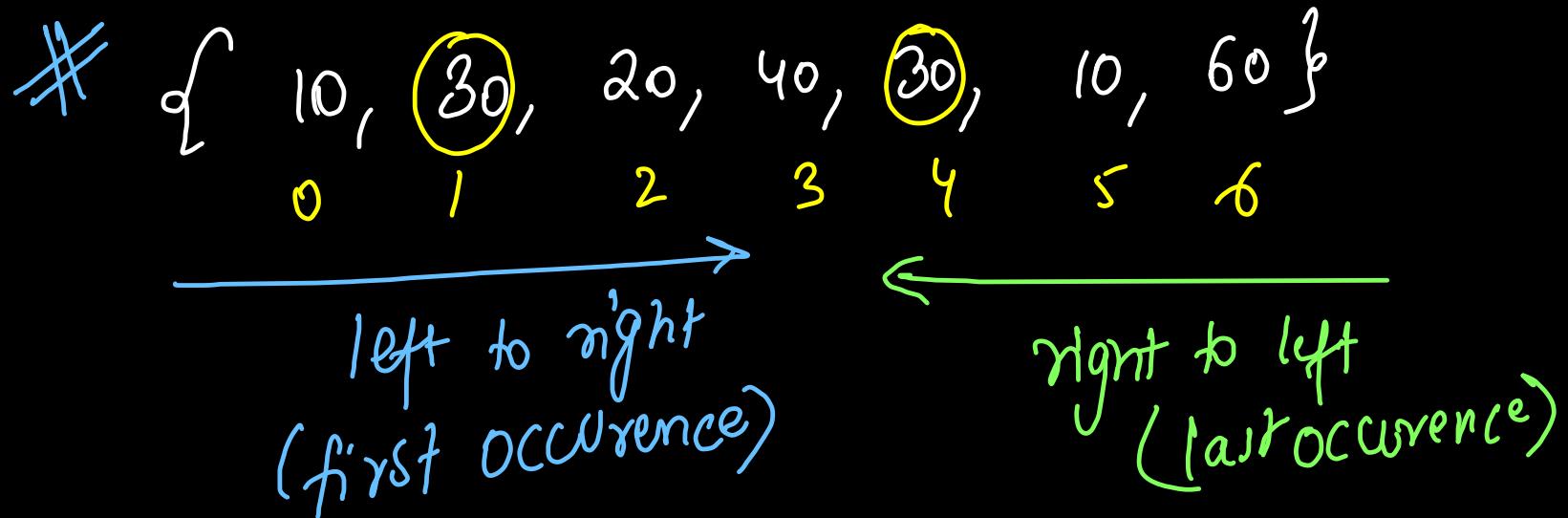
```
for(int idx=0; idx < arr.length; idx++) {  
    if(arr[idx] == target) return idx;  
} //first occurrence  
return -1; //unsuccessful search
```

target = 30
 $\Rightarrow 3$

target = 25
 $\Rightarrow -1$

Linear Search Algorithm

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



First & Last Occurrence

{ 40, 30, 20, 10, 20, 40, 20, 60, 50 }
0 1 2 3 4 5 6 7 8
LC(34)

int firstIndex = ~~2~~ 2

int lastIndex = ~~2 4 6~~ 6

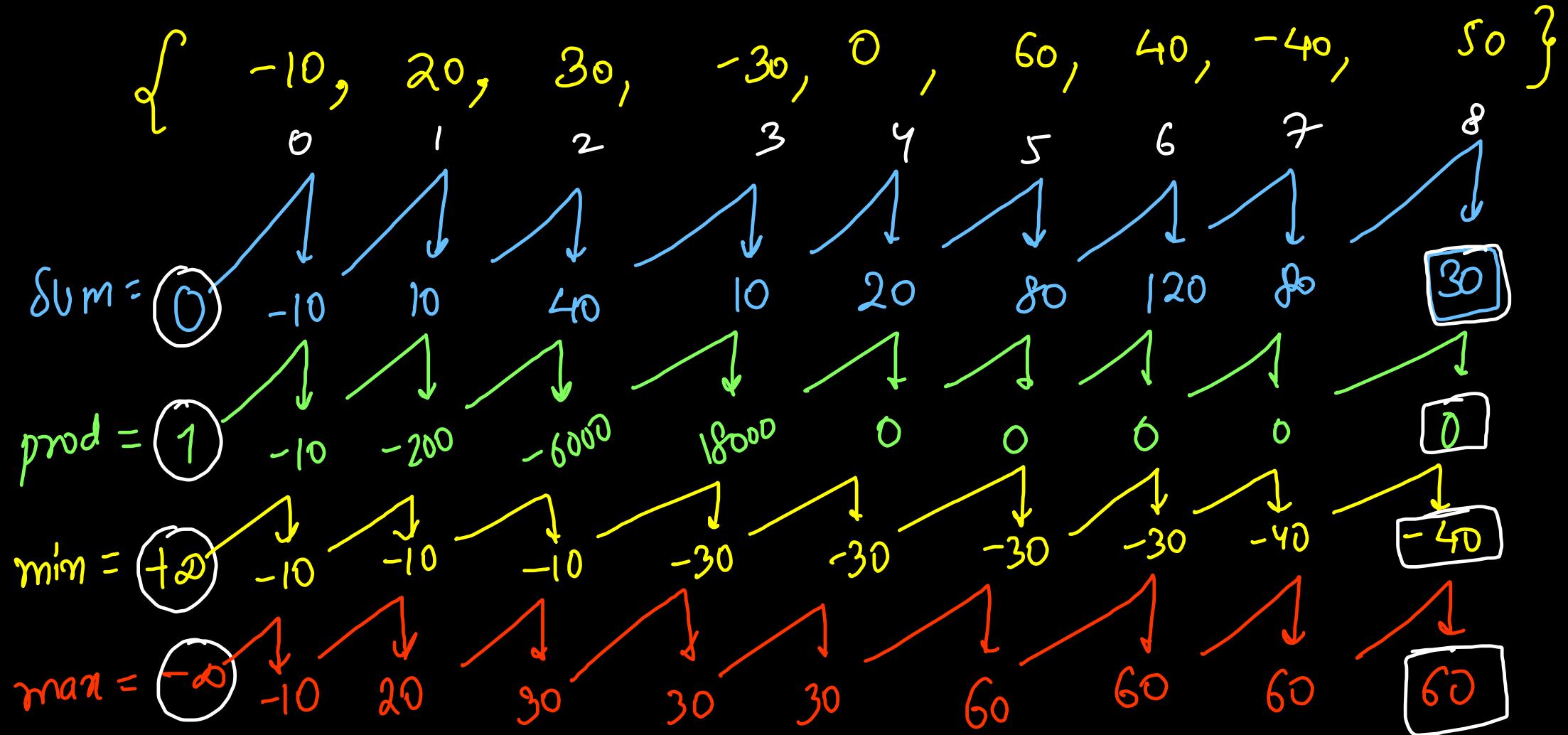
target = 20 [2, 6]

target = 60 [7, 7]

target = 35 [-1, -1]

```
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;  
}
```

Mijøm & maxm



```
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);  
}
```

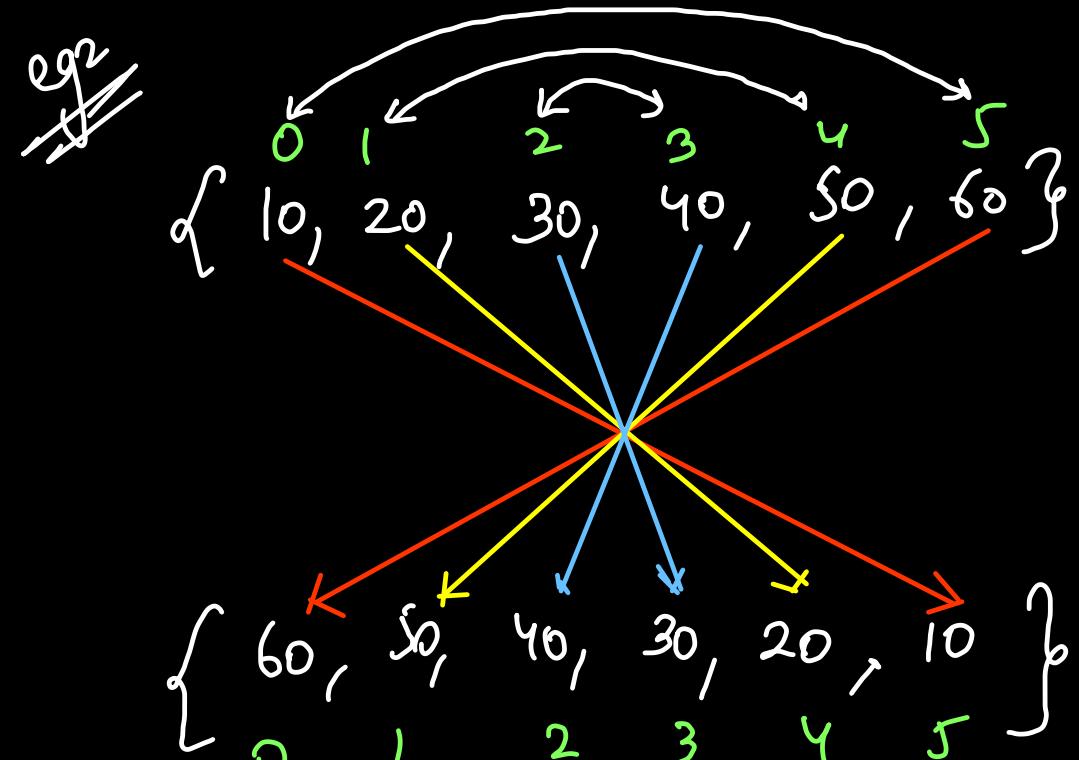
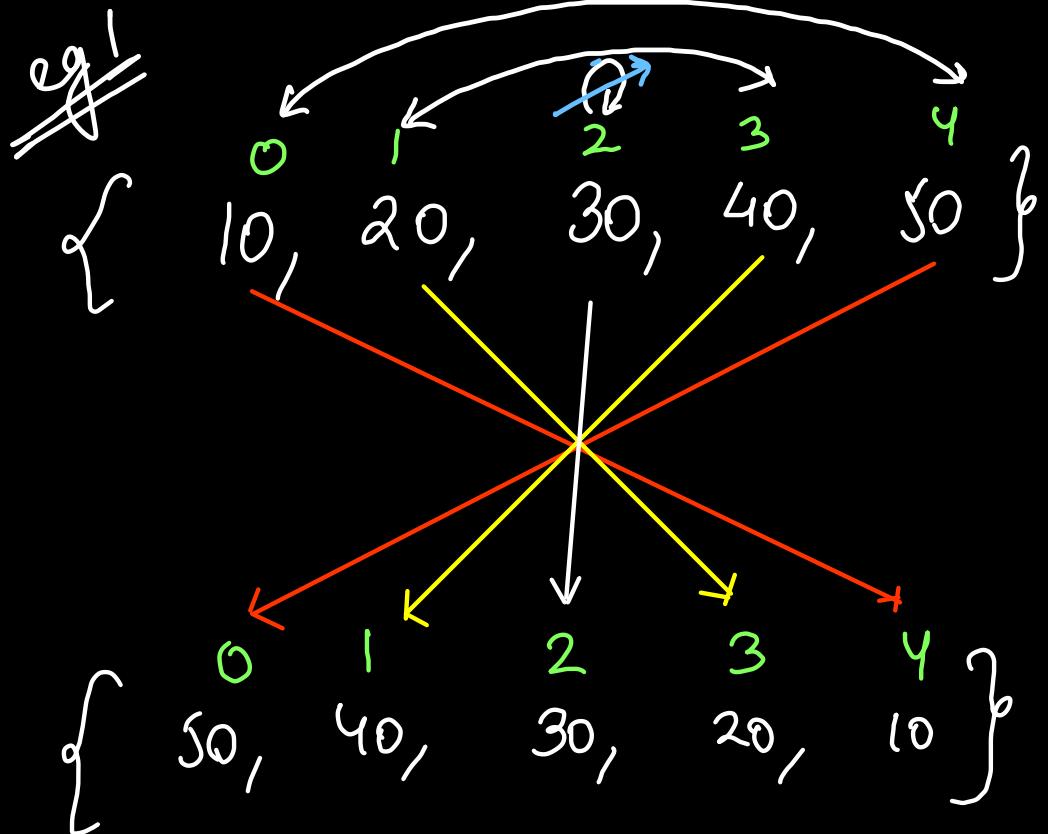
Approach ①

```
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);  
}
```

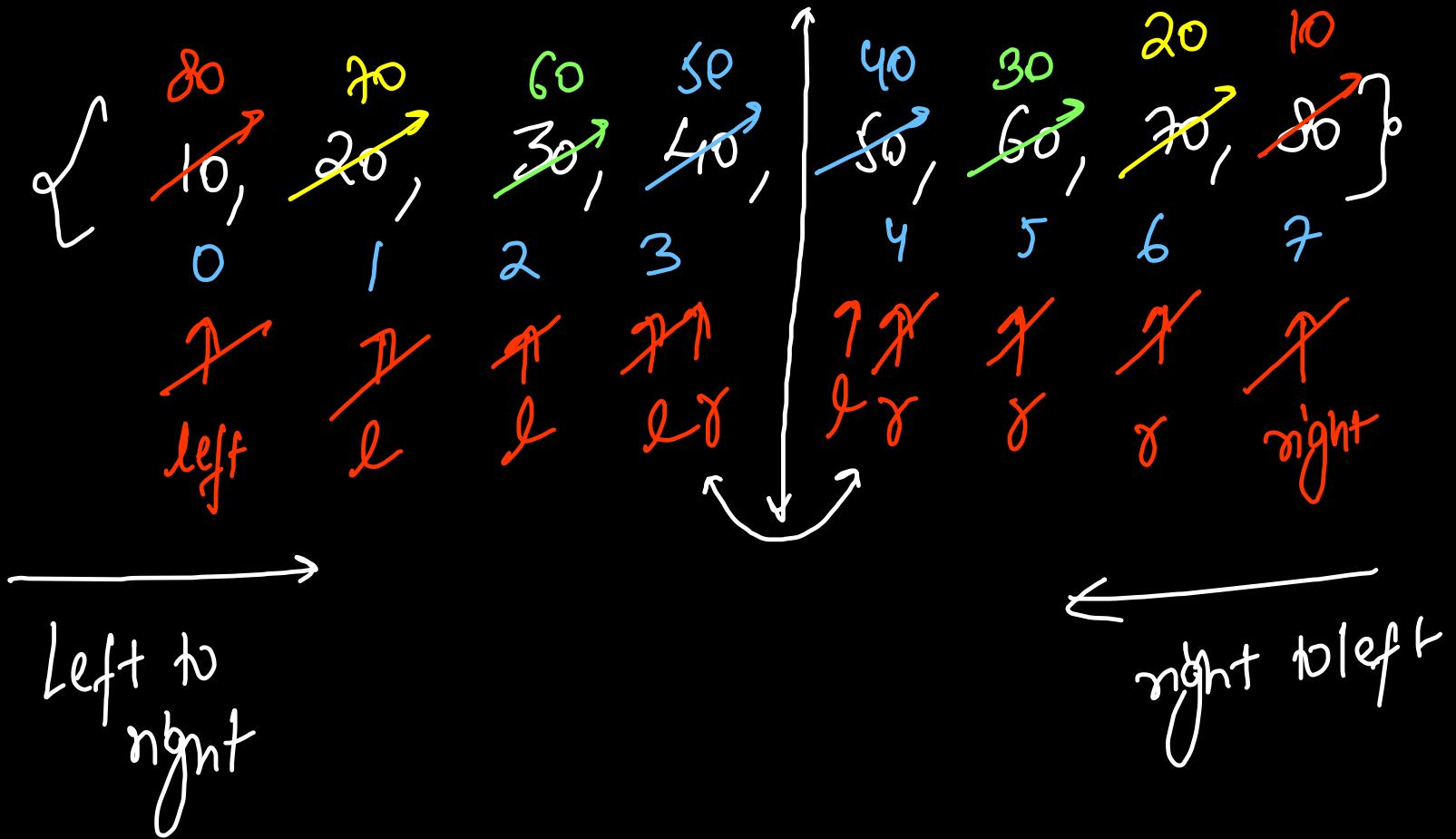
Approach ②

Reverse Array

{ inplace algorithm → change the input array }



Two pointer Technique



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

```
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--;  
    }  
}
```

Max^m prod of 2 elements

{ 10, 20, -30, 40, -20, 60, -50, -60 }

6 1 2 3 4 5 6 7

max^m product

2nd largest largest
40 * 60 = 240

-50 * -60 = 300
2nd smallest smallest

$\{$ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ $\}$
 10, 20, -30, 60, -20, 40, -50, -60
 ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

largest = ~~-∞~~ $\nearrow 10 \nearrow 20 \nearrow 60 \leftarrow$ $60 * 40 = 240$

Second largest = ~~-∞~~ $\nearrow 10 \nearrow 20 \nearrow 40 \swarrow$

smallest = ~~+∞~~ $\nearrow 10 \nearrow -30 \nearrow -50 \nearrow -60 \leftarrow$ $-60 * -50 = 300$

secondsmallest = ~~+∞~~ $\nearrow 20 \nearrow 10 \nearrow -20 \nearrow -30 \nearrow -50$

$\{ \downarrow 10, \downarrow 20, \downarrow -30, \downarrow 60, \downarrow 40 \}$

```

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);
}

```

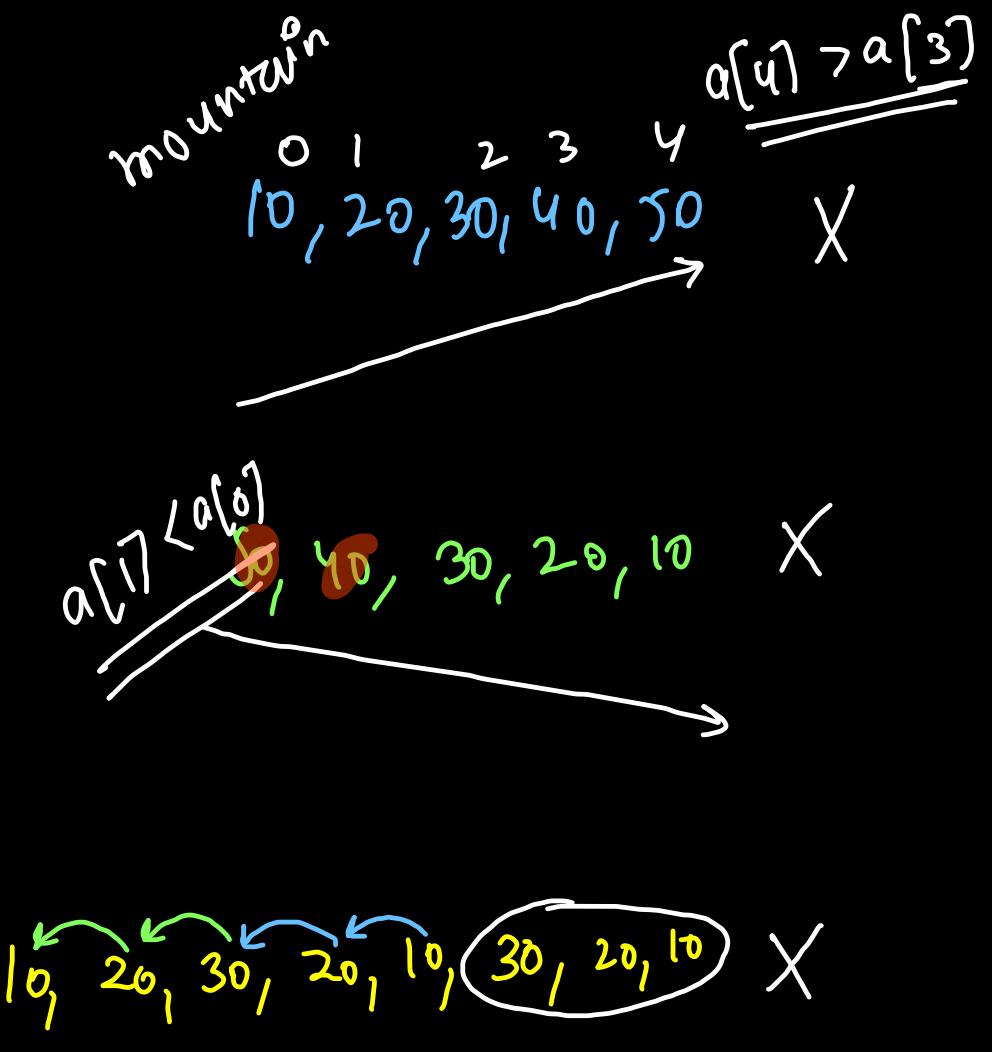
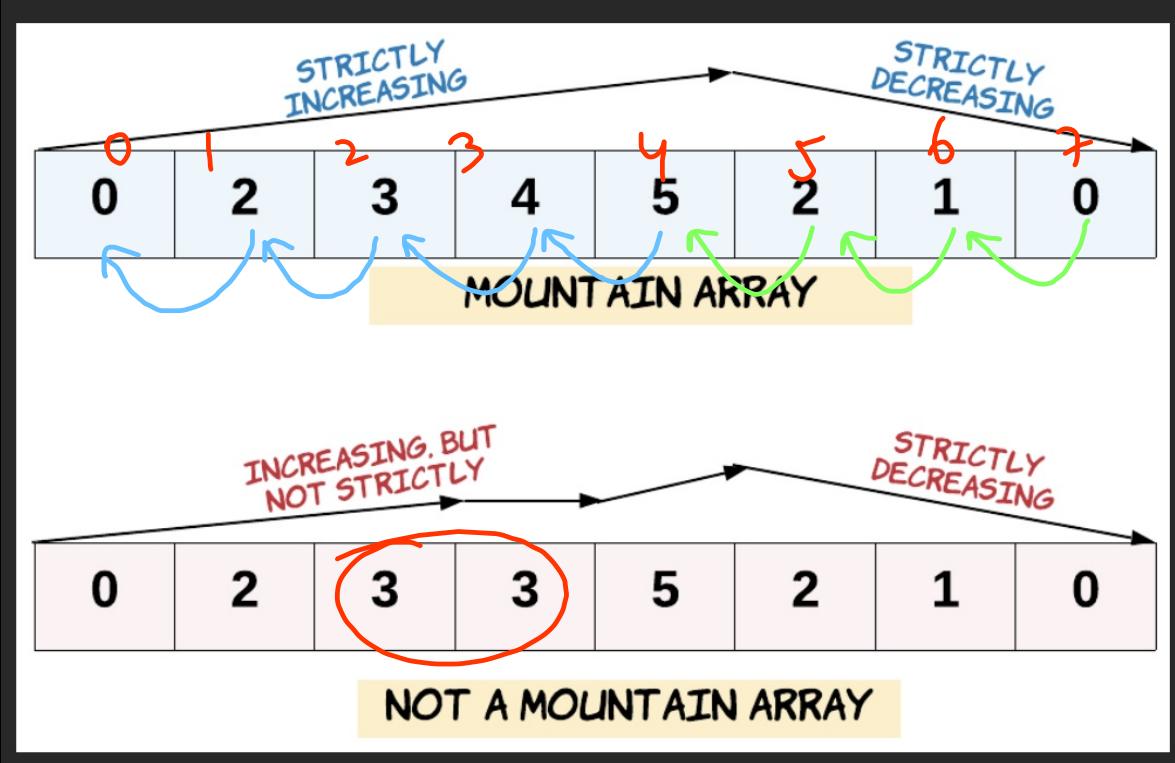
$\text{largest} = \cancel{-\infty} \nearrow 10 \nearrow 20 \nearrow 60 \Rightarrow [59 \times 39]$

$\text{Second largest} = \cancel{-\infty} \nearrow 70 \nearrow 20 \nearrow 40 \Rightarrow$

$\text{Smallest} = \cancel{+\infty} \nearrow 10 \nearrow -30$

$\text{Second smallest} = \cancel{+\infty} \nearrow 20 \nearrow 10$

-31×9



```

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

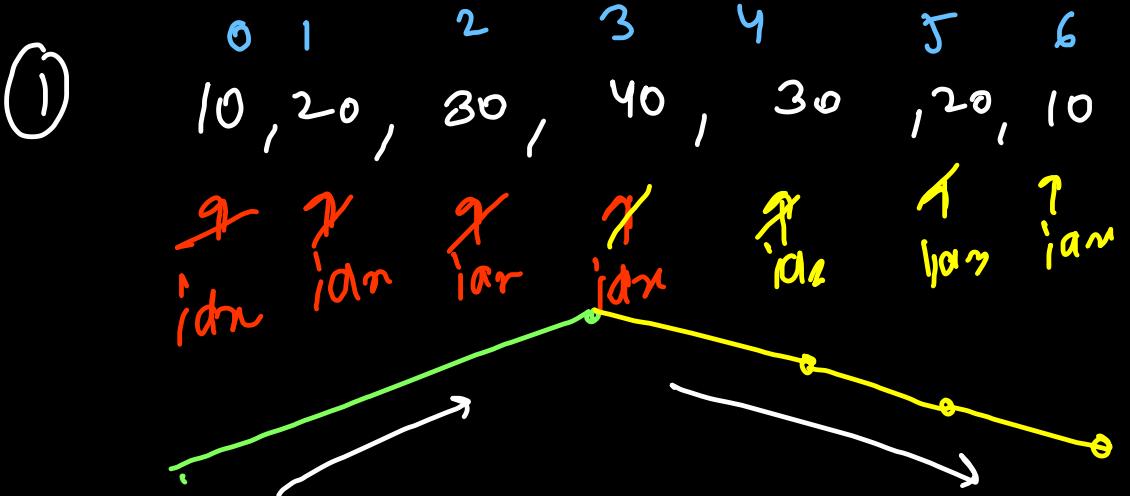
    // Increasing Part
    while(idx + 1 < arr.length){
        if(arr[idx] == arr[idx + 1]){
            return false; // not strictly increasing
        } else if(arr[idx] > arr[idx + 1]){
            break;
        }
        idx++;
    }

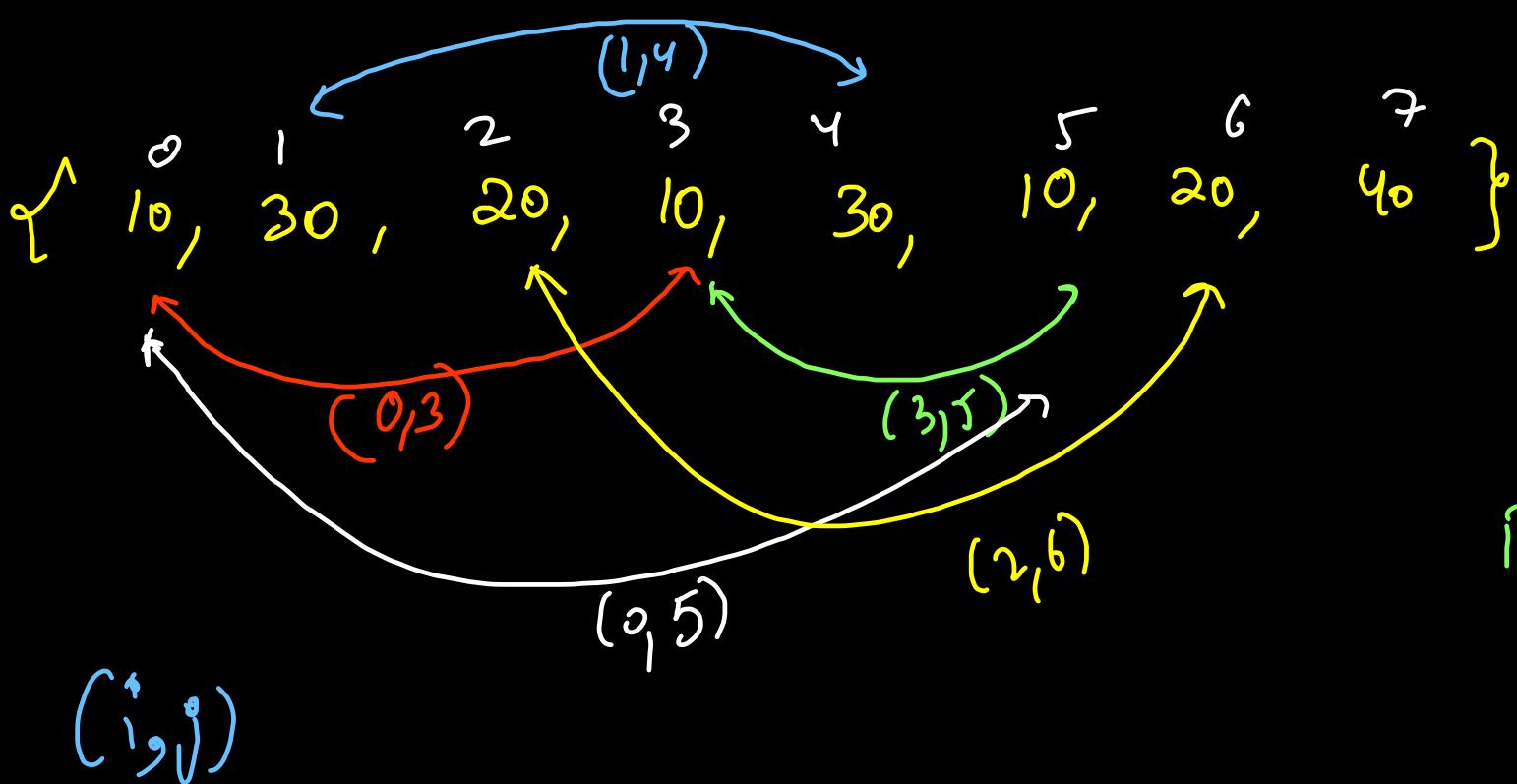
    if(idx == 0)
        return false; // no increasing part
    if(idx == arr.length - 1)
        return false; // no decreasing part

    // 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);
}

```





$\text{arr}[i] = \text{arr}[j]$

$i < j$

$\text{count} = \emptyset / \text{if } \beta \neq \emptyset$

$i=0 \quad j=1$
 2
 $3 \Leftarrow 10 == 10$
 4
 $5 \Leftarrow 10 == 10$
 6
 7

$i=1 \quad j=2$
 3
 $4 \quad 30 == 30 \Leftarrow$
 5

$i=2 \quad j=3, \dots, 7$
 $i=3 \quad j=4, \dots, 7$
 $i=4 \quad j=5, \dots, 7$
 $i=5 \quad j=6, \dots, 7$
 $i=6 \quad j=7$

```
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 = { 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]

Starting = 2

[30] [2, 2]

[30, 40] [2, 3]

Starting = 1
[20] [1, 1]
[20, 30] [1, 2]
[20, 30, 40] [1, 3]

Starting = 3
[40] [3, 3]

No of
Subarrays

$$= 1 + 2 + 3 + \dots + n$$

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

```
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

```
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

~~odd~~ { 10, 20, 30, 40, 50 }
 0 1 2 3 4

Odd = 1

① { 10 } 10

① { 20 } 20
+

① { 30 } 30
+

① { 40 } 40
+

① { 50 } 50
+

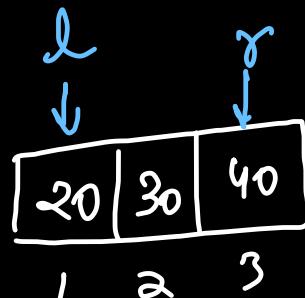
Odd = 3

{ 10, 20, 30 } + 60

{ 20, 30, 40 } + 90

{ 30, 40, 50 } + 120

Odd = 5
 { 10, 20, 30, 40, 50 } + 150



$$\Rightarrow \text{length} = r - l + 1$$

{ { 10 }, { 20 }, \dots, { 50 } },

{ } - - -

{ }

$$\text{left} = 10, \quad \text{right} = 15$$

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

$$\{10, 11, 12, 13, 14, 15\} \text{ b/e }$$

$$\text{count} \Rightarrow \text{right} - \text{left} + 1$$

$$[\text{left}, \text{right}]$$

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

$$\{11, 12, 13, 14\}$$

$$\text{count} \Rightarrow \text{right} - \text{left} - 1$$

(left, right)

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

$$(1, 2) \{10, 11, 12, 13, 14\}$$

or

$$(1, 2) \{11, 12, 13, 14, 15\}$$

$$\text{Count} = \text{right} - \text{left}$$

```

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;
}

```

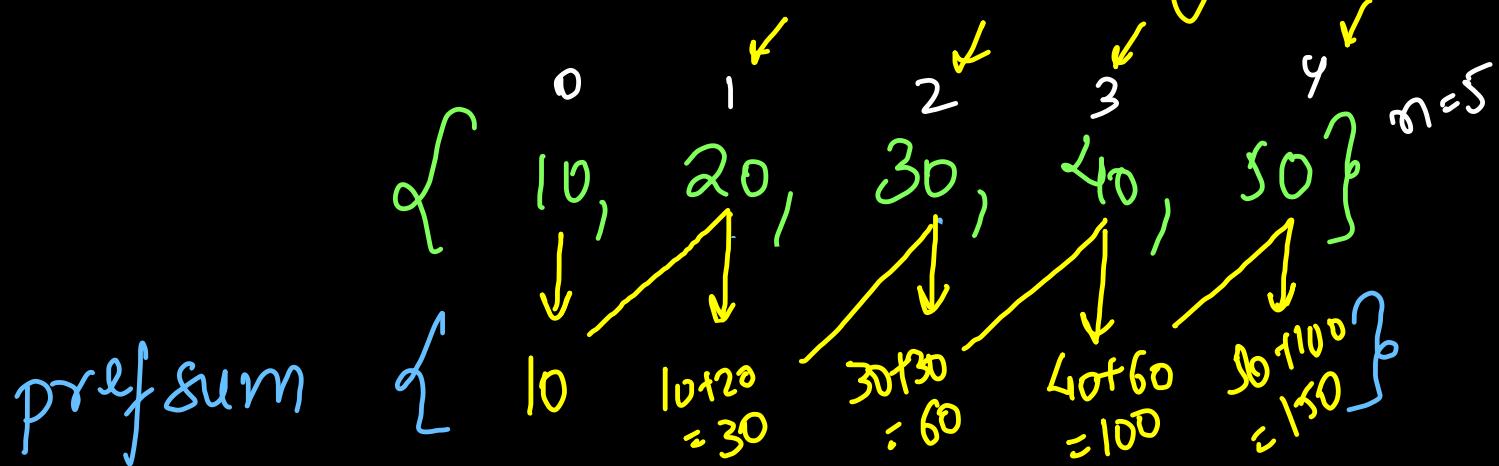
$$\text{answer} = 0 + 10 + 60 + 150 \\ + 20 + 90$$

$\{0, 1, 2, 3, 4, 5\}$

$\sum = 0$
 \downarrow
 $l=0$
 $r=0 \quad 10 \checkmark \textcircled{1}$
 \downarrow
 $r=1 \quad 30 \times \textcircled{2}$
 \downarrow
 $r=2 \quad 60 \checkmark \textcircled{3}$
 \downarrow
 $r=3 \quad 100 \times \textcircled{4}$
 \downarrow
 $r=4 \quad 150 \checkmark \textcircled{5}$

$\sum = 0$
 \downarrow
 $l=1$
 $r=1 \quad 20 \checkmark \textcircled{1}$
 \downarrow
 $r=2 \quad 50 \times \textcircled{2}$
 \downarrow
 $r=3 \quad 90 \checkmark \textcircled{3}$
 \downarrow
 $r=4 \quad 140 \times \textcircled{4}$

Prefix Sum or Running Sum



$\text{pref}[i] = \text{arr}[i] + \text{pref}[i-1]$

$i > 0$

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

1 2 3 4 5

prefix \Rightarrow subarray with starting index as 0.

~~App~~

```
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

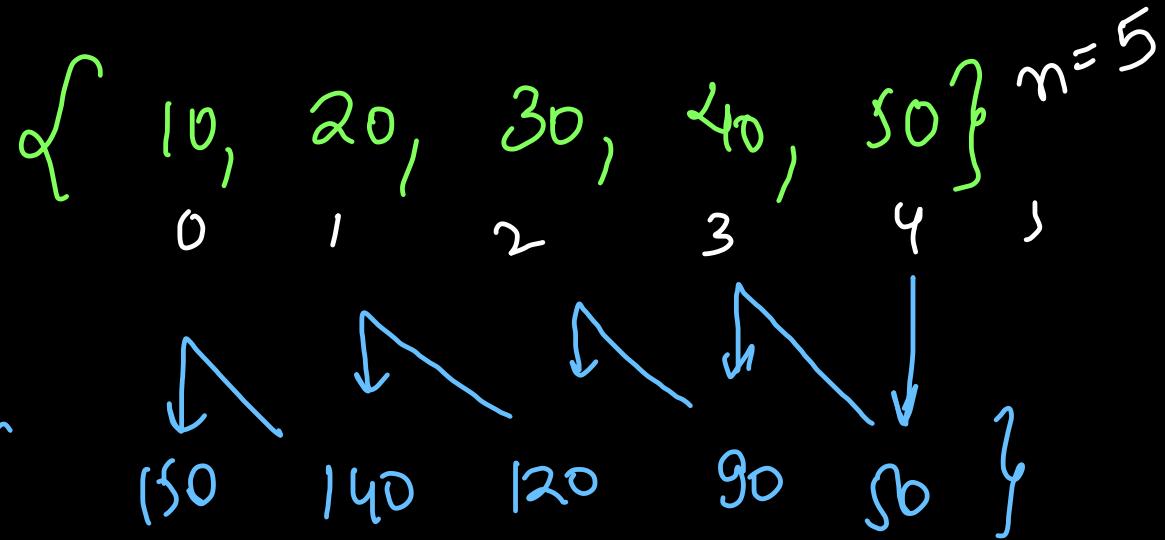
~~App~~

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

    return nums;
}
```

inplace

~~suffixes~~



$\boxed{\text{suffix}[i] = arr[i] + \text{suffix}[i+1]}$

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

{ 40, 50 } { 50 }

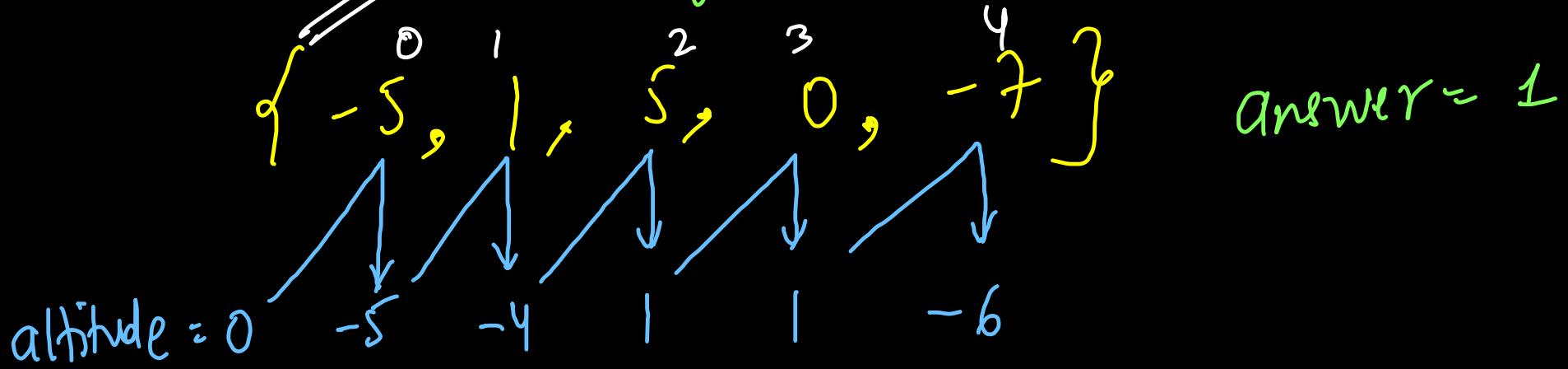
suffix \Rightarrow subarray \rightarrow with ending index as me last index

```
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;  
}
```

```
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;  
}
```

```
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;  
}
```

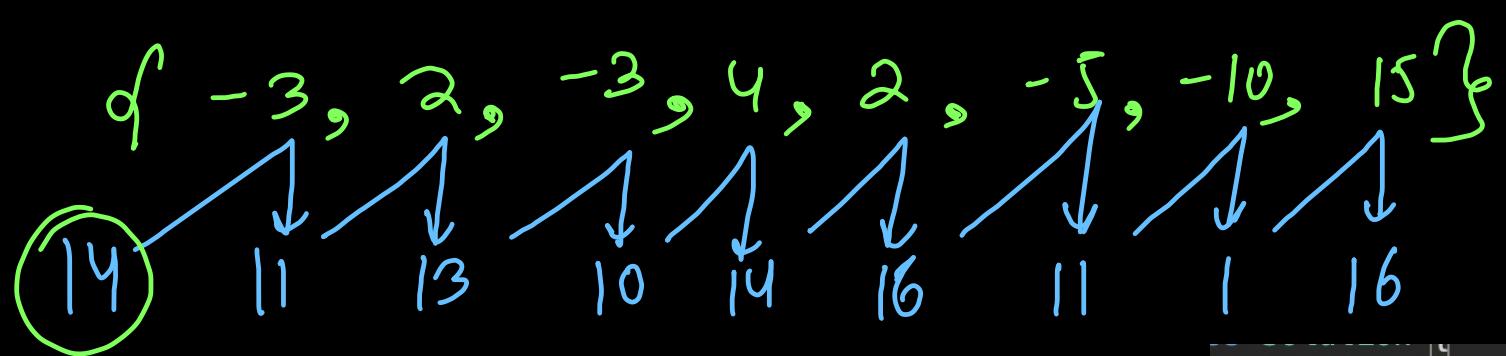
LC P22
find highest altitude



(highest)
answer = ~~0~~ $\nearrow \times \nearrow \nearrow$ 10

```
public int largestAltitude(int[] gain) {  
    int answer = 0, altitude = 0;  
  
    // For Each Loop  
    for(int val: gain){  
        altitude += val;  
        answer = Math.max(answer, altitude); ↑ same  
        // if(altitude > answer) answer = altitude;  
    }  
  
    return answer;  
}
```

~~LC 1413~~



~~Min~~
Starting value

$$= 0$$

$$\downarrow 4 = -(-3) + 1$$

$$\begin{matrix} 4 \\ 5 \end{matrix} \rightarrow 1$$

$$\cancel{2} \quad \cancel{3}$$

$$\cancel{0} \downarrow 1 = 0 \times 1$$

$$14$$

$$\cancel{-8} \downarrow 9 = -(-(-8)) + 1$$

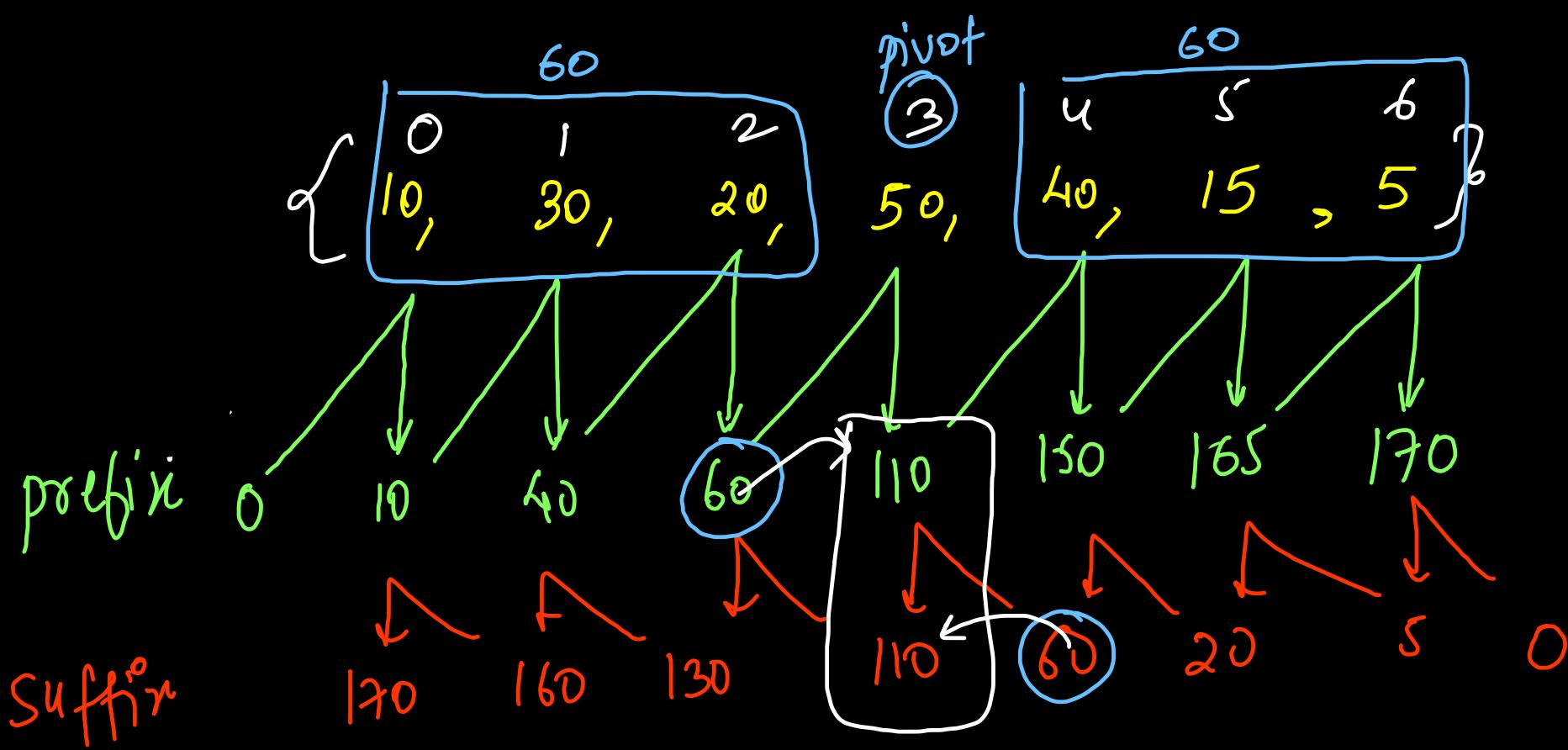
$$1$$

Note
~~0 is not considered a positive no~~

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

e.g. $[0, 2] = [4, 6] \Leftrightarrow [0, 3] = [3, 6]$

```
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;  
}
```

```
public int pivotIndex(int[] nums) {  
    int[] prefix = getPrefix(nums); Shallow copy  
    int[] suffix = getSuffix(nums); (not deep)  
  
    // 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;  
}
```

~~App using division operator~~ Product of array Except Self

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

answer: { 5x3x4x6, 2x3x4x6, 2x5x1x6, 2x5x3x6, 2x5x3x4 }

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

~~13800~~
division by 0

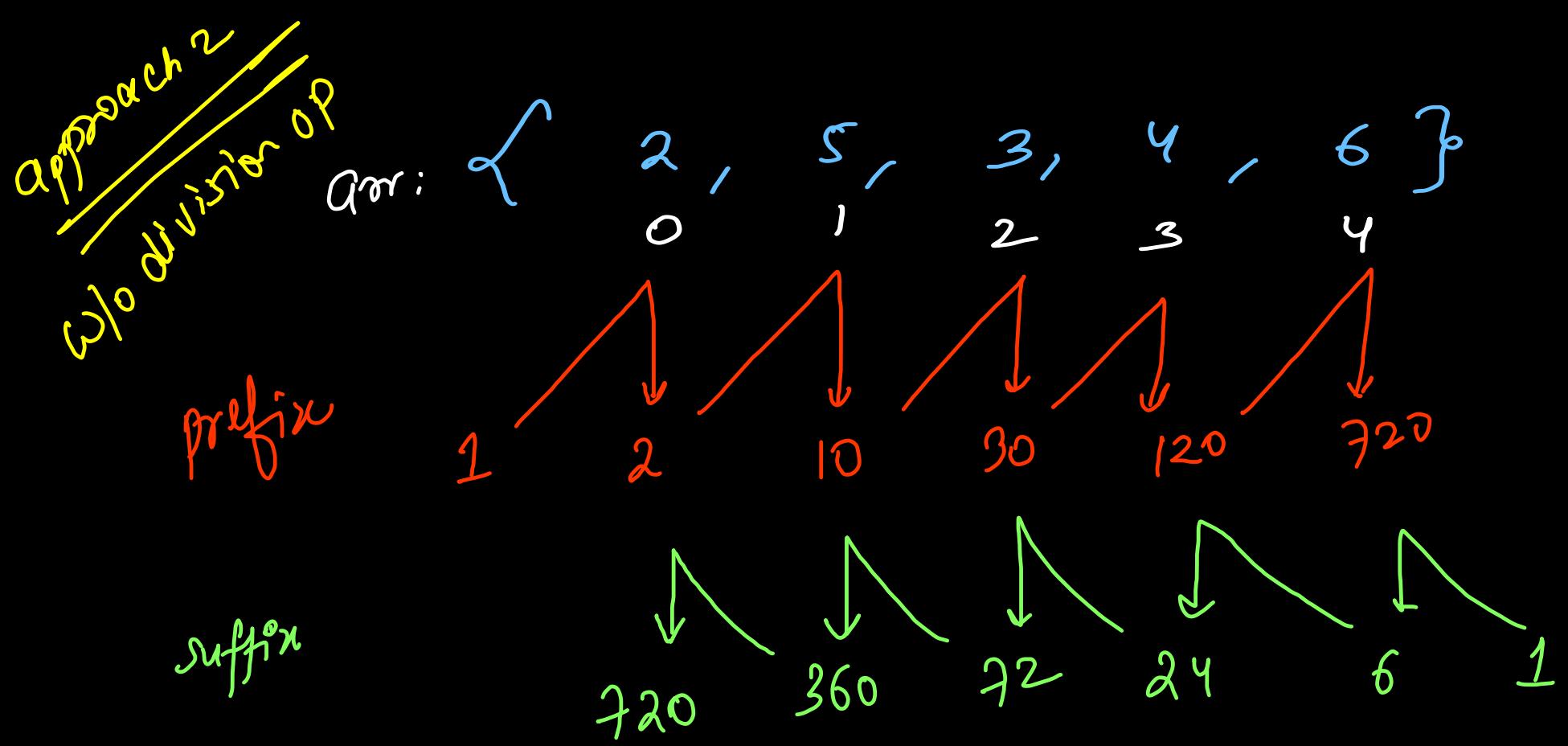
corner case
①

$$\left\{ 2, 5, 4, 0, 3 \right\}$$
$$0 \quad 0 \quad 0 \quad | \quad 120 \quad 0$$

corner cases

corner case

$$\textcircled{2} \quad \left\{ 2, 0, 4, 0, 3 \right\}$$
$$0 \quad 0 \quad 0 \quad 0 \quad 0$$



answer prod except self
 { 1x360, 2x72, 10x24, 30x6, 120x1 }

$= 360$	$= 144$	$= 240$	$= 180$	$= 120$
---------	---------	---------	---------	---------

```

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;
}

```

↓
 empty
 array
 ⇒ product
 = 1

```

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 $idx == 0 \Rightarrow pref[idx - 1]$
 will be $pref[-1]$
 which does not exist

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

~~4C169~~ Majority Element - ①

arr: $\{ \underline{2}, 3, \underline{2}, 3, \underline{2}, 4, \underline{2}, \underline{2}, 5 \}$ $n = 9$

answrr = ②

strictly more than $\lfloor n/2 \rfloor \Rightarrow \text{floor}(n/2) = \lfloor 9/2 \rfloor = 4$

Approach ①: Count frequency of each element (hashmap)

$2 \rightarrow 5$ $3 \rightarrow 2$ $5 \rightarrow 1$ $4 \rightarrow 1$

Boyer Moore Voting Algorithm

$n=14$
arr: { 10, 10, 20, 40, 40, 10, 10, 40, 40, 40, 40, 40, 20 }

majority = 

lead = 

if majority element exist
then it will pointed
by answer,
otherwise,
no majority
ele.

```

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;
}

```

↳ this code will work
if majority element
exists

$n \approx 13$

majority = 0 10 20 30 20 20 20 20

lead = 0 1 0 1 0 1 0 1 0 1

$$\begin{aligned}
 \text{majority} &= 20 \Rightarrow \text{freq} = \left\lfloor \frac{13}{2} \right\rfloor + 1 \\
 &= 6 + 1 \Rightarrow 7
 \end{aligned}$$

LC 229 Majority Element - 11

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

freq > ⌊ n/3 ⌋

answer = {}

> ⌊ 8/3 ⌋

Q { 10, 20, 30, 10, 20, 10, 30, 30 }

$\text{maj}^o A = \cancel{0} \cancel{1} \overset{\text{actual?}}{\underset{\text{freq}}{\textcircled{10}}} \cancel{2} \cancel{3}$

$\text{lead} A = \cancel{0} \cancel{1} \overset{\cancel{0}}{\cancel{2}} \cancel{3} \cancel{2} \cancel{1}$

$\text{maj} B = \cancel{0} \cancel{2} \overset{\cancel{30}}{\cancel{1}} \Rightarrow \text{freq}^2$

$\text{lead} B = \cancel{0} \cancel{1} \cancel{0} \cancel{1} \cancel{0} \cancel{1}$

eg(2) $\{ 10, \downarrow 20, \downarrow 30, \downarrow 10, \downarrow 20, \downarrow 10, \downarrow 10, \downarrow 50 \}$

$\text{maj } A = \cancel{\phi} \Rightarrow$ actual ≥ 2
yes

$\text{lead } A = \cancel{\phi} \cancel{\neq} \cancel{\neq} \cancel{\neq} 2$

$\text{maj } B = \cancel{\phi} \Rightarrow$ actual ≥ 2
no

$\text{lead } B = \cancel{\phi} \cancel{\neq} \cancel{\neq} 0$

```
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;  
}
```

LC 229

```
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;  
}
```

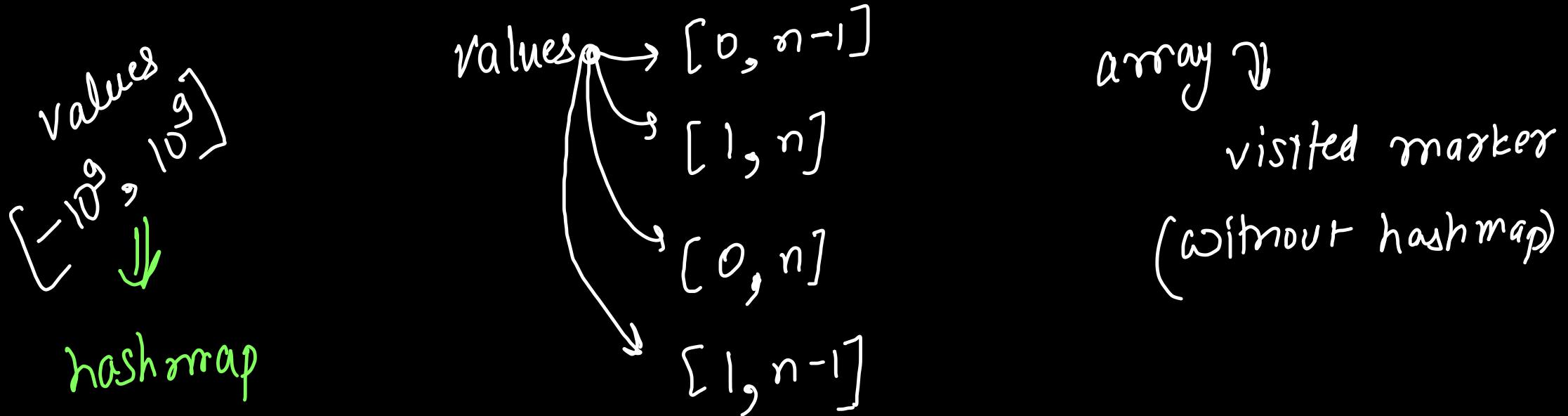
$n/2 \rightarrow 1$

$n/3 \rightarrow 2$

~~$n/k \rightarrow (k-1)$~~

majority elements

Missing & Duplicate : Indexed Based Hashmap



0 ~ 1 ~ 2 ~ 3 ~ 4 ~ 5 ~ missing ~ 6 ~ 7 ~ 8 ~
 m = 9
 [0, 9]

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

$16/10 = 1$ $14/10 = 1$ $12/10 = 1$ $19/10 = 1$ $13/10 = 1$
 $8/10 = 0$

visited \Rightarrow add $(n+1)$

original value \Rightarrow mod $(n+1)$

frequency \Rightarrow divide by $(n+1)$

\min value which
 does not exist in
 array $\Rightarrow (n+1)$

~~corner case~~

0	1	2	3	4
4	0	3	2	1
+6	+6	+6	+6	+6

$$n+1 = 6$$

[0, 5]

$$n=5$$

return 6 - 5

```
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

$$\begin{aligned} \text{Sum of } [0, n] - \text{sum of array} \\ = \frac{n \times (n+1)}{2} - \text{sum} \end{aligned}$$

```
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
DSA \Rightarrow 80% focus
Core + Project
+ LLD \Rightarrow 20%
focus

startups
DSA \Rightarrow 40%
Dev \Rightarrow 40%
Core + LLD
 \Rightarrow 20%

LC 287) Duplicate Number

Repeating or
Duplicate Number

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

[1, n-1]

n = 9

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

$$\cancel{(1+2+\dots+8)} + 5 - \cancel{(1+2+3+\dots+8)} = 5$$

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

repeating

$6/10 = 0$ $14/10 = 1$ $12/10 = 1$ $15/10 = 1$ $13/10 = 1$ $+10$ $2d/10 = 2$

(visited mark)

$$n = 9$$

$$n+1 = 10$$

2L 287

repeating / duplicate no \Rightarrow freq > 1

```

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 nos

l-based 0-based	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	1	2	3	4	5	6	7	8	
6	4	6	5	3	6	7	5	7	

$$\begin{array}{llllllllll} 6/10=0 & 4/10=0 & +10 & +10 & +10 & +10 & +10 & 5/10=0 & 2/10=0 \\ 16/10=1 & 15/10=1 & +10 & +10 & +10 & +10 & +10 & 5/10=0 & 2/10=0 \\ & & i=1 & i=2 & i=3 & i=4 & i=5 & & \\ & & 36/10=3 & & 28/10=2 & & & & \end{array}$$

missing nos = { $\checkmark 1, \checkmark 2, \checkmark 8, \checkmark 9$ }

duplicate = { 5, 6, 7 }

$$n=9$$

$$n+1=10$$

values: $[1, n]$ ↘ hash (visited)

indices: $[0, n-1]$

LC 448) Find all missing

```
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);
        }
        ↳ 1based to 0 based

        // 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);
        }
        ↳ 0based to 1 based

        return missing;
    }
}
```

LC 442) Find all duplicates

```
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

$$\begin{array}{ccccccccc} & +10 & +10 & +10 & +10 & +10 & +10 & +10 \\ 6/10 = 0 & 4/10 = 0 & 6/10 & 15/10 & 13/10 & +10 & +10 & +10 \\ & \approx 0 & \approx 1 & \approx 1 & \approx 2 & \approx 2 & \approx 2 & \approx 1 \end{array}$$

max repeating = ~~0~~ ~~3~~ ~~5~~

max freq = ~~0~~ ~~1~~ 2

most frequent element

$k \leq n$ [0, $k-1$]

\hookrightarrow [0, $n-1$]

array

range

\Rightarrow index range

tie breaker
↓

two valued with
same freq \Rightarrow pick smaller
value

```
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;  
}
```

QFG

Max 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.

$$\{ \checkmark 7, \checkmark 2, \checkmark 5, \checkmark 1, \checkmark 4, \checkmark 3, \checkmark 8, 10 \}$$

smallest missing +ve integer
⇒ ⑥

Brute force/naïve solution: → `for(int no=1; true; no++) {
 if (linear search(nums, no) == -1)
 return no;
}`

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

Corner case 1 $n=9$ [1-g] 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)

$[1]^x$ $(2)^x$ $[3]^x$ $[4]^x$ $[5]^x$ $\{6\}$ $\{7\}$ $\{8\}$ $[9]$
 0 1 2 3 4 5 6 7 8

0 10	4	0 16	5	3	-4	1	2	0
---------	---	---------	---	---	----	---	---	---

$+10$ $+10$ $+10$ $+10$ $+10$ $+10$ $+10$
 $10|10=1$ $14|10=1$ $10|10=1$ $15|10=1$ $20|10=1$ $0|10=0$

$$n=9$$

answer can be

$[1, n+1]$

① discard out of range values <0 or >n \Rightarrow make mem(0)

② visited mark

③ smallest +ve integer which is missing

```

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
}

```

[1]	[2]	[3]	[4]	[5]
0	1	2	3	4
2	4	1	5	3

+6 +6 +6 +6 +6

smallest missing +ve $\Rightarrow 6$

$$n=5$$

$$n+1=6$$