

~~Module 2~~ (40 days)

Arrays & Strings

→ 1D Array

 ↳ Linear Loop
 ↳ Nested Loop

→ Prefix / Frequency

→ Searching (Binary Search)

→ Sorting

→ Two Pointer / Sliding Window

→ 2D Arrays

→ Strings

→ Mathematics / Number Theory /
 Bit manipulation /
 Modular Arithmetic

→ Time & Space

Complexity!

Always =

marks

int rollno0 = 93;

int rollno1 = 99;

int rollno2 = 80;

int rollno3 = 30;

int rollno4 = 75;

int rollno5 = 60;

city

String city0 = "Delhi"

String city1 = "Mumbai"

String city2 = "Bangalore"

String city3 = "Kolkata"

String city4 = "Ahmedabad"

String city5 = "Hyderabad"

Array → Datastructure to store data of same datatype.

Datastructure

→ way to store
multiple data elements

→ apply operations on

data

- adding / deleting data
- updating data
- searching & sorting

Datatype

→ type of data which we are
storing

CFUD

algorithm

Array of Integers

marks

int rollno0 = 93;

int rollno1 = 99;

int rollno2 = 80;

int rollno3 = 30;

int rollno4 = 75;

int rollno5 = 60;

① Declare & Initialize the Array

int [] rollNos = new int [6];
↑
array of integers
size

② Print / Get the Element

Sysout(rollNos[0]) Sysout(rollNo[3]),

Sysout(rollNos[1]); Sysout(rollNo[4]),

Sysout(rollNos[2]); Sysout(rollNo[5]),

index out of bound & Sysout(rollNo[6]);

```
int [] arr = new int [6];
```

{ 93, 99, 89, 30, 75, 60 }
0 1 2 3 4 5

Traversal / Iteration over Array / Printing Array

↳ Loop (for)

```
for (int idx = 0; idx < arr.length; idx++) {
```

```
    System.out.println [idx]);
```

}

- ~~A~~ Error *Sys0(zollNos);*
- ~~B~~ $zollNos[0] = 93$
- ~~C~~ It will print the entire array
- ~~D~~ Garbage value ~~F~~HashCode
- ~~E~~ Address of Array

```

Scanner scn = new Scanner(System.in);
int size = scn.nextInt();
double[] percentages = new double[size];
for(int idx = 0; idx < percentages.length; idx++){
    percentages[idx] = scn.nextDouble();
}

```

Output

```

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

```

User Input of Array

size = 5

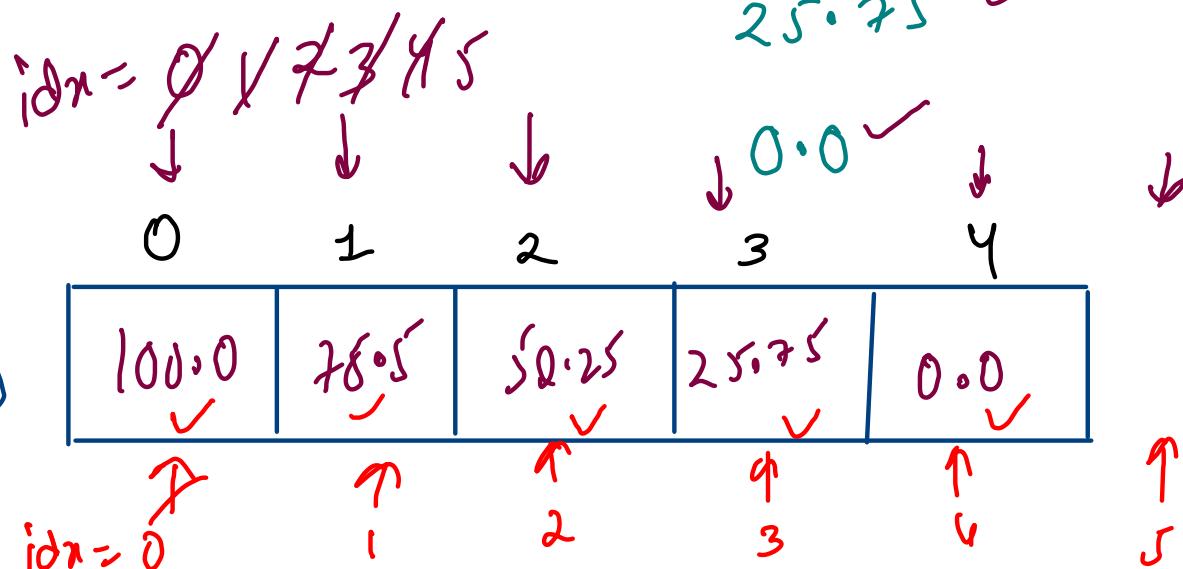
100.0 ✓

75.5 ✓

50.25 ✓

25.75 ✓

0.0 ✓



double
percentages

| | | | | |
|---------|--------|---------|---------|-------|
| 100.0 ✓ | 75.5 ✓ | 50.25 ✓ | 25.75 ✓ | 0.0 ✓ |
|---------|--------|---------|---------|-------|

idx=0 ↑ 1 ↑ 2 ↑ 3 ↑ 4 ↑ 5

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
  
    int size = scn.nextInt(); → Takes size as input  
    int[] arr = new int[size]; → Arrays are of fixed size  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    for(int idx = 0; idx < arr.length; idx++){  
        System.out.println(arr[idx]); → Traverse the array  
    }  
}
```

→ Take All Integer Elements
Inputs

Point Alternate

$$\text{size} = 9$$

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 80 | 10 | 50 | 60 | 70 | 10 | 30 | 20 | 50 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ↑ | | ↑ | | ↑ | | ↑ | | ↑ |

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    for(int idx = 0; idx < arr.length; idx++){  
        if(idx % 2 == 0)  
            System.out.println(arr[idx]);  
    }  
}
```

Approach

size iteration

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);
```

```
    int size = scn.nextInt();  
    int[] arr = new int[size];
```

```
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }
```

```
    for(int idx = 0; idx < arr.length; idx = idx + 2){  
        System.out.println(arr[idx]);  
    }
```

size/2 times

Approach2

Reverse Print Array

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 80 | 10 | 50 | 60 | 70 | 10 | 30 | 20 | 50 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];

        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        for(int idx = arr.length - 1; idx >= 0; idx--){
            System.out.print(arr[idx] + " ");
        }
    }
}
```

↑
idx = 0
ord

↑
idx = arr.length - 1
start

Odd one

Point All Elements which are odd

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| | | | | | | | | |
| 55 | 10 | 50 | 65 | 70 | 10 | 35 | 25 | 50 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

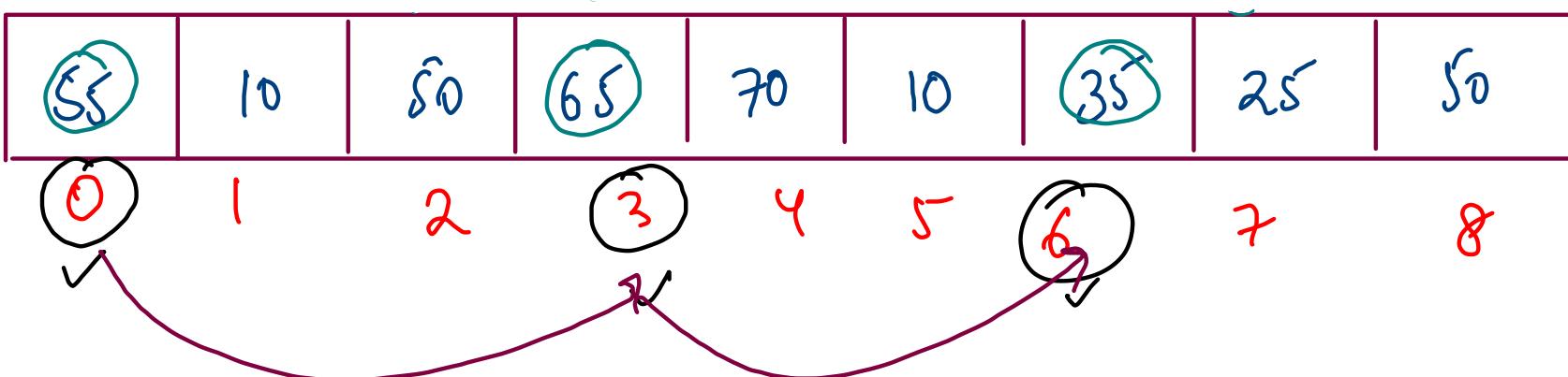
55, 65, 35, 25

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] % 2 == 1){  
            System.out.print(arr[idx] + " ");  
        }  
    }  
}
```

point values/elements /data
which are odd

$$\begin{aligned}arr[0] &= 1 \\ 55 \cdot 1 &= 1 \\ 27 \cdot 2 &= 1 \\ 35 \cdot 1 &= 1 \\ 17 \cdot 2 &= 1\end{aligned}$$

Print Elements with indexes
as multiples of 3



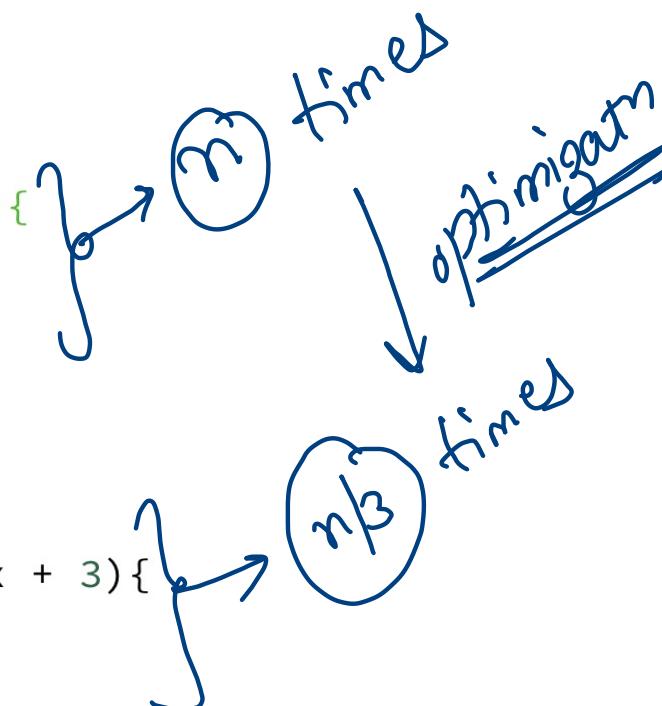
$\text{if}(idx \% 3 == 0)$

App ①

$idx = idx + 3$

App ②

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    // Approach 1  
    // for(int idx = 0; idx < arr.length; idx++){  
    //     if(idx % 3 == 0){  
    //         System.out.print(arr[idx] + " ");  
    //     }  
    // }  
  
    // Approach 2  
    for(int idx = 0; idx < arr.length; idx = idx + 3){  
        System.out.print(arr[idx] + " ");  
    }  
}
```



Print odd Element Indexes

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 55 | 10 | 50 | 65 | 70 | 10 | 35 | 25 | 50 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Print Indexes whose value is odd

0, 3, 6, 7

- ① Check Index → print index (no array)
- ② Check Index → print value (print alternate in array)
- ③ Check value → print value (print odd elements)
- ④ Check value → print index (print odd element indexes)

```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];

        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        for(int idx = 0; idx < arr.length; idx++){
            if(arr[idx] % 2 == 1){
                System.out.print(idx + " ");
            }
        }
    }
}
```

Identical Arrays

1 false
 ① $n_1 = 5 \{ 10, 30, 20, 50, 40 \}$ $n_2 = 3 \{ 10, 30, 20 \}$
 $n_1 \neq n_2 \Rightarrow$ non-identical

2 false
 ② $n_1 = 5 \{ 10, 30, 20, 50, 40 \}$ $n_2 = 5 \{ 30, 40, 10, 50, 20 \}$
 ↑ for any index $i \Rightarrow arr_1[i] = arr_2[i]$
 \Rightarrow non-identical

3 false
 ③ $n_1 = 5 \{ 10, 30, 20, 50, 40 \}$ $n_2 = 5 \{ 10, 30, 25, 50, 40 \}$
 2nd element of n_1 is circled, 3rd element of n_2 is circled

4 true
 ④ $n_1 = 5 \{ 10, 30, 20, 50, 40 \}$ $n_2 = 5 \{ 10, 30, 20, 50, 40 \}$
 Both arrays have same elements at same indices

```

public static boolean isIdentical(int[] arr1, int[] arr2){
    if(arr1.length != arr2.length) return false; // arrays of different size, arrays are not identical

    for(int idx = 0; idx < arr1.length; idx++){
        if(arr1[idx] != arr2[idx]) return false; // At any index, if value is not same, arrays are not identical
    }

    return true; // Same Size and All Indexes have same value, arrays are identical
}

```

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    // Input of first array
    int size1 = scn.nextInt();
    int[] arr1 = new int[size1];

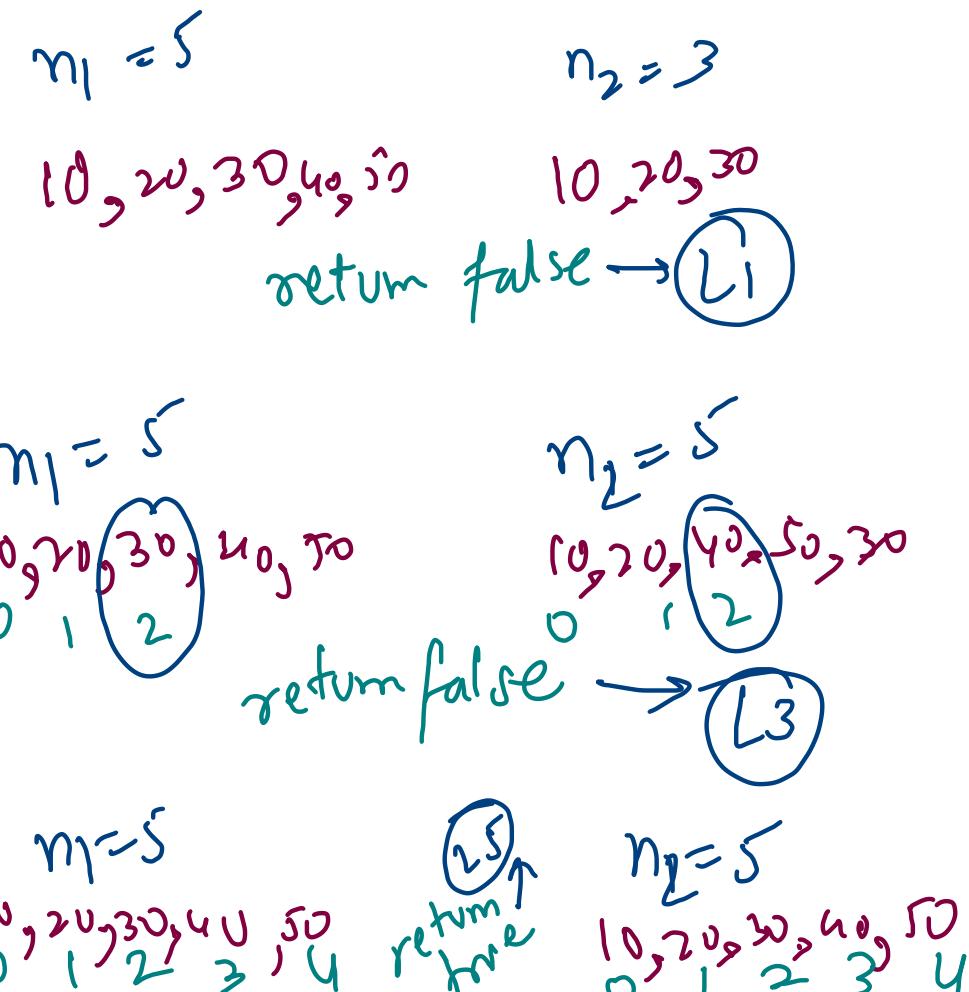
    for(int idx = 0; idx < size1; idx++){
        arr1[idx] = scn.nextInt();
    }

    // input of second array
    int size2 = scn.nextInt();
    int[] arr2 = new int[size2];

    for(int idx = 0; idx < size2; idx++){
        arr2[idx] = scn.nextInt();
    }

    // get the answer of whether arrays are identical or not
    boolean res = isIdentical(arr1, arr2);
    System.out.println(res);
}

```



Linear Search \leadsto unsorted array

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 55 | 10 | 50 | 65 | 70 | 10 | 35 | 25 | 10 |
|----|----|----|----|----|----|----|----|----|

0 1 2 3 4 5 6 7 8
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

target = 70

True

target = 5

False

target = 10

True

① successful search

② unsuccessful search

```

public static String linearSearch(int[] arr, int target){
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == target) {
            return "True";
        }
    }
    return "False";
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];

    for(int idx = 0; idx < arr.length; idx++){
        arr[idx] = scn.nextInt();
    }

    int target = scn.nextInt();

    System.out.println(linearSearch(arr, target));
}

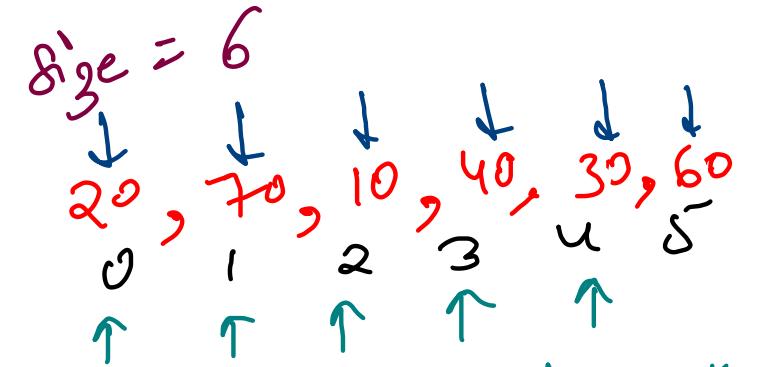
```

Best case ($\text{arr}[0] = \text{target}$)
1 iteration
(successful)

Worst case
n iterations

Average case
n/2 iterations

Linear Search
Algorithm
(*find element*)



target = 30 return "True"

target = 50 return "False"

Find the first Index
First Occurrence

Find the last Index
Last Occurrence

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| 55 | 10 | 50 | 55 | 70 | 10 | 50 | 25 | 10 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

~~leftmost~~
~~only one~~ \rightarrow $\text{first occurrence}(70) = 4$

~~rightmost~~
~~last occurrence~~ \rightarrow $\text{last occurrence}(70) = 4$

$$\text{first occurrence}(50) = 2$$

$$\text{last occurrence}(50) = 6$$

$$\text{first occurrence}(10) = 1$$

$$\text{last occurrence}(10) = 8$$

~~no occurrence~~ \rightarrow $\text{first occurrence}(35) = -1$

$$\text{last occurrence}(35) = -1$$

```
public static int firstOccurrence(int[] arr, int target){  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] == target) return idx;  
    }  
  
    return -1; // No Occurrence  
}
```

First Occurrence

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
  
    int[] arr = new int[size];  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int target = scn.nextInt();  
  
    System.out.println(firstOccurrence(arr, target));  
}
```

```
public class Solution {  
    public static int lastOccurrence(int[] arr, int target){  
        for(int idx = arr.length - 1; idx >= 0; idx--){  
            if(arr[idx] == target) return idx;  
        }  
  
        return -1; // No Occurrence  
    }  
}
```

Last Occurrence

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
  
    int[] arr = new int[size];  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int target = scn.nextInt();  
  
    System.out.println(lastOccurrence(arr, target));  
}
```

Print All Prime nos within range [2, n]

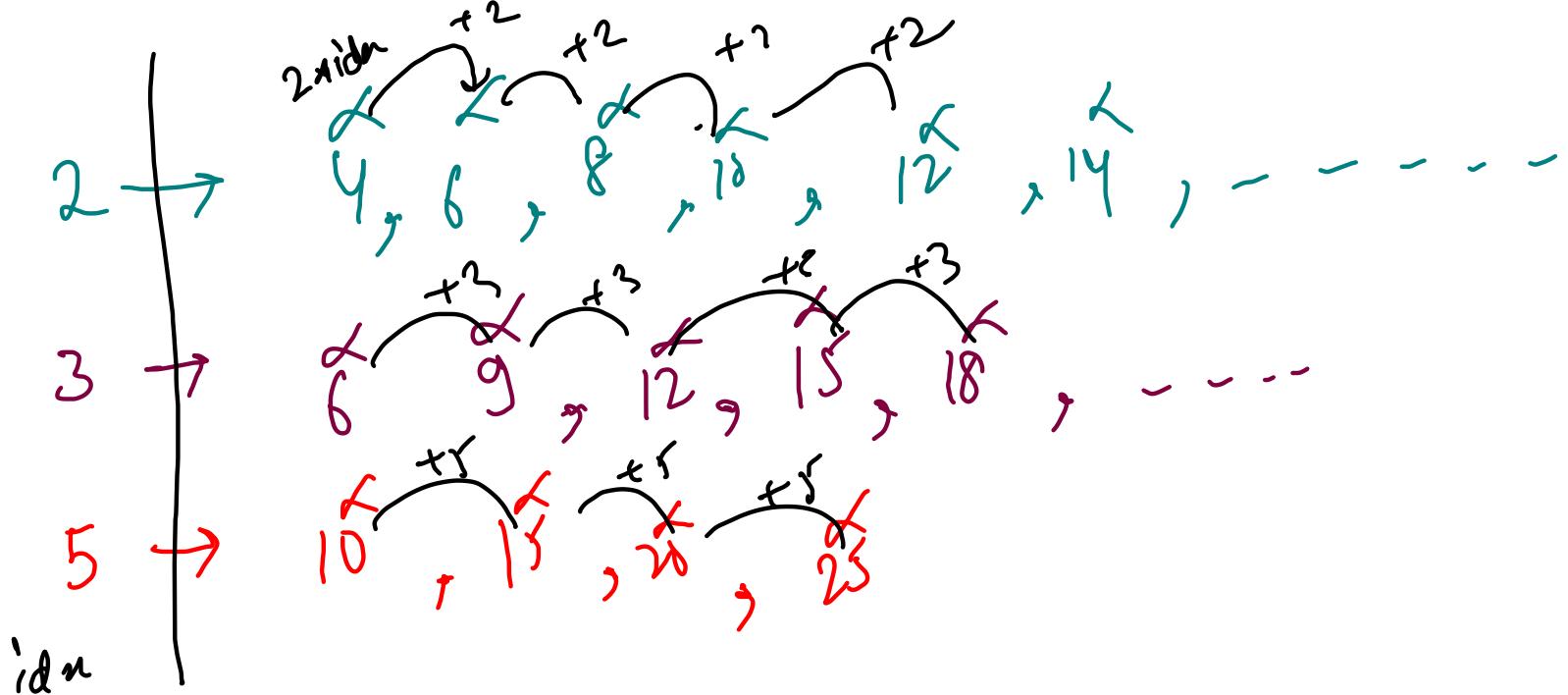
```
public class Solution {  
    public static boolean isPrime(int n){  
        int right = (int)Math.sqrt(n);  
  
        for(int idx = 2; idx <= right; idx++){  
            if(n % idx == 0) return false;  
        }  
  
        return true;  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int n = scn.nextInt();  
  
        for(int idx = 2; idx <= n; idx++){  
            if(isPrime(idx) == true) System.out.print(idx + " ");  
        }  
    }  
}
```

$n = 25$
2, 3, 5, 7, 11, 13,
17, 19, 23

Sieve of Eratosthenes

Intuition

$\rightarrow k$ is a prime number, discard all multiples of the prime m



| | | | | | | | | | |
|---|---|---|---|--------|---|--------|---|--------|--------|
| F | F | T | T | E T | T | F T | T | F T | E T |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

| | | | | | | | | | | |
|--------|----|--------|----|--------|--------|--------|--------|----|--------|---|
| E T | T | E T | T | E T | F T | E T | F T | T | E T | T |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | |

| | | | | | | | | | | |
|--------|--------|--------|----|--------|--------|--------|--------|--------|----|--------|
| E T | E T | F T | T | E T | E T | E T | E T | E T | T | E T |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |

$n=30$ $3n=23$

$$id_n = 2 \quad \checkmark$$

$$id_n = 3 \quad \checkmark$$

$$id_n = 5 \quad \checkmark$$

$$id_n = 7 \quad \checkmark$$

$$id_n = 11 \quad \checkmark$$

$$id_n = 13 \quad \checkmark$$

$$id_n = 17$$

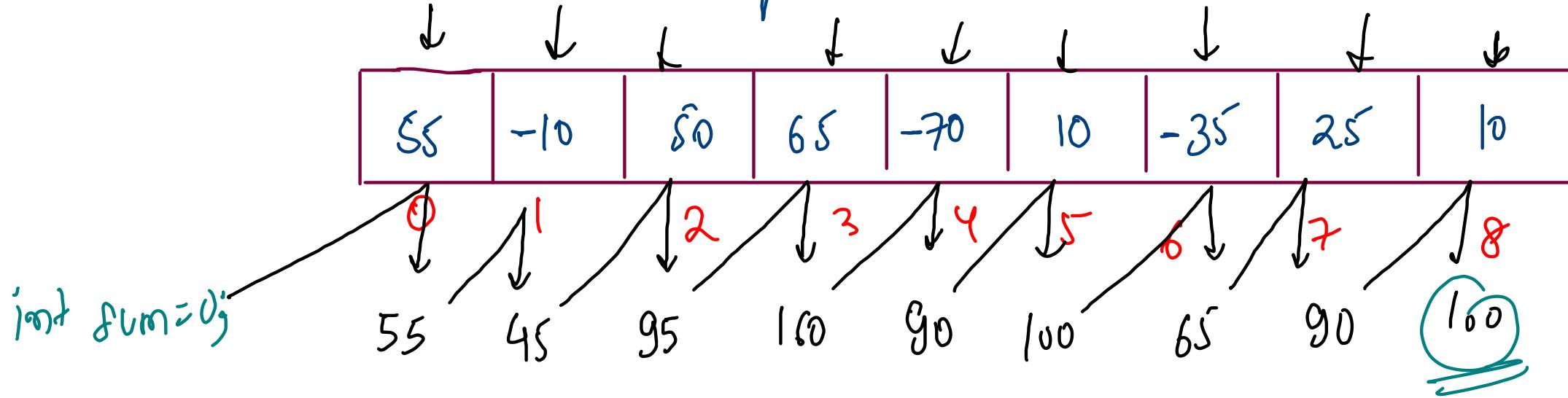
$$id_n = 19$$

$$id_n = 23$$

$$id_n = 29$$

```
public class Solution {  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int n = scn.nextInt();  
  
        boolean[] primes = new boolean[n + 1];  
        for(int idx = 2; idx <=n; idx++){  
            primes[idx] = true; // Assuming all numbers are prime initially  
        }  
  
        for(int idx = 2; idx <= n; idx++){  
            if(primes[idx] == true){  
                System.out.print(idx + " ");  
  
                for(int multiples = 2 * idx; multiples <= n; multiples = multiples + idx){  
                    primes[multiples] = false;  
                }  
            }  
        }  
    }  
}
```

Sum of All Elements of Array



$$55 + (-10) + 50 + 65 + (-70) + 10 + (-35) + 25 + 10 \\ = \underline{\underline{100}}$$

```
import java.io.*;
import java.util.*;

public class Solution {
    public static int getSum(int[] arr){
        int sum = 0;
        for(int idx = 0; idx < arr.length; idx++){
            sum = sum + arr[idx];
        }
        return sum;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];

        for(int idx=0; idx<arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        System.out.println(getSum(arr));
    }
}
```

Sum of all
Elements of Array

Ranges of Integer

| | |
|-------|--|
| 2^1 | $2 \rightarrow 0, 1$ {1 bit nos} |
| 2^2 | $4 \rightarrow -2, -1, 0, 1$ $[-2^1, 2^1 - 1]$ {2 bit nos} |
| 2^3 | $8 \rightarrow -4, -3, -2, -1, 0, 1, 2, 3$ $[-2^2, 2^2 - 1]$ {3 bit nos} |
| 2^4 | $16 \rightarrow -8, -7, -6, -5, \dots -1, 0, 1, 2, \dots 7$ $[-2^3, 2^3 - 1]$ {4 bit nos} |
| 2^5 | $32 \rightarrow -16, \dots -1, 0, +1, \dots +15$ $[-2^4, 2^4 - 1]$ {5 bit nos} |

integer
4 bytes
 $= 32$ bits

min^m value
 $-2^{31} = \underline{-2147483648}$

Integer.MAX-VALUE
 $= -\infty$

max^m value
 $+2^{31} - 1$
 $= \underline{2147483647}$

Integer.MIN-VALUE
 $= +\infty$

long → bytes → 64 bits 2^{64} types of nos

minm long value → $-2^{63} = -922\ldots807$
= long. MIN-VALUE = $-\infty$

max long value → $+2^{63} - 1 = +922\ldots807$
= long. MAX-VALUE
= $+\infty$

```
public static void main(String[] args) {  
    System.out.println("Range of Integer Datatype");  
    System.out.println(Integer.MIN_VALUE + " to " + Integer.MAX_VALUE);  
  
    System.out.println("Range of Long Datatype");  
    System.out.println(Long.MIN_VALUE + " to " + Long.MAX_VALUE);  
  
    System.out.println("Range of Char Datatype");  
    System.out.println((int)Character.MIN_VALUE + " to " + (int)Character.MAX_VALUE);  
  
    System.out.println("Range of Float Datatype");  
    System.out.println(Float.MIN_VALUE + " to " + Float.MAX_VALUE);  
}
```

$-2^{31} \text{ to } 2^{31}-1$

$-2^{63} \text{ to } 2^{63}-1$

$0 \text{ to } 65535$

Identitive property

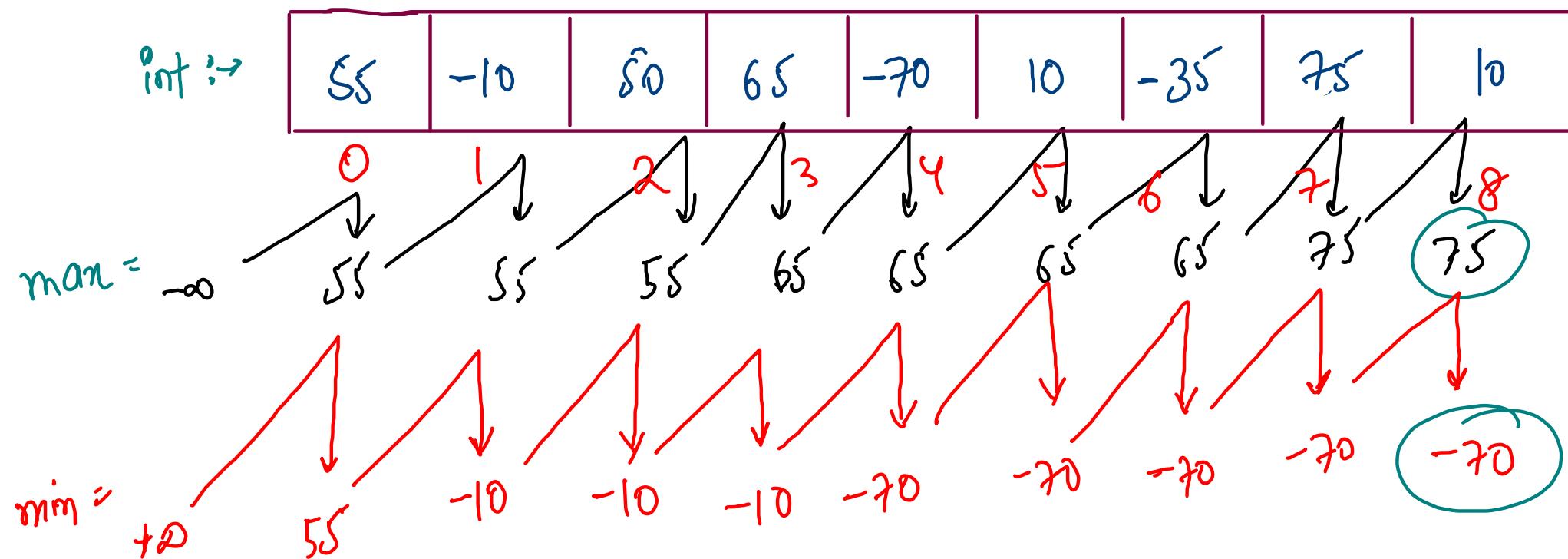
Additiv: $a + ?_e = ?_e + a = a$ $?_e = 0$

multiplicat: $a * ?_o = ?_o * a = a$ $?_o = 1$

maxim^{on}: $\max(a, ?_c) = \max(?_c, a) = a$ $?_c = -2^{31} = -2$
 (Integers) $= \text{Integer.MIN-VALUE}$

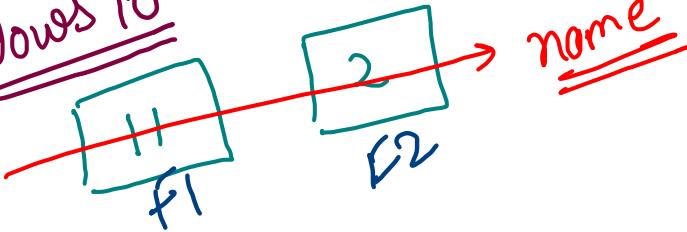
minim^{on}: $\min(a, ?_o) = \min(?_o, a) = a$ $?_o = +2^{31} - 1$
 (Integers) $= \text{Integer.MAX-VALUE}$
 $= +2$

Maximum & Minimum of Array



Homework

Windows 10



Which kind of sorting
is this?

numerical sorting?
or any other?

$$11 < 2$$

$$\text{int : } 11 \neq 2$$

?? · (1) < (2)
datatype?

```
public static int getMax(int[] arr){  
    int maximum = Integer.MIN_VALUE;  
  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] > maximum){  
            maximum = arr[idx];  
        }  
    }  
  
    return maximum;  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < size; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    System.out.println(getMax(arr));  
}
```

Max of Array

```
public static int getMin(int[] arr){  
    int minimum = Integer.MAX_VALUE;  
  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] < minimum){  
            minimum = arr[idx];  
        }  
    }  
  
    return minimum;  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < size; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    System.out.println(getMin(arr));  
}
```

Min of Array

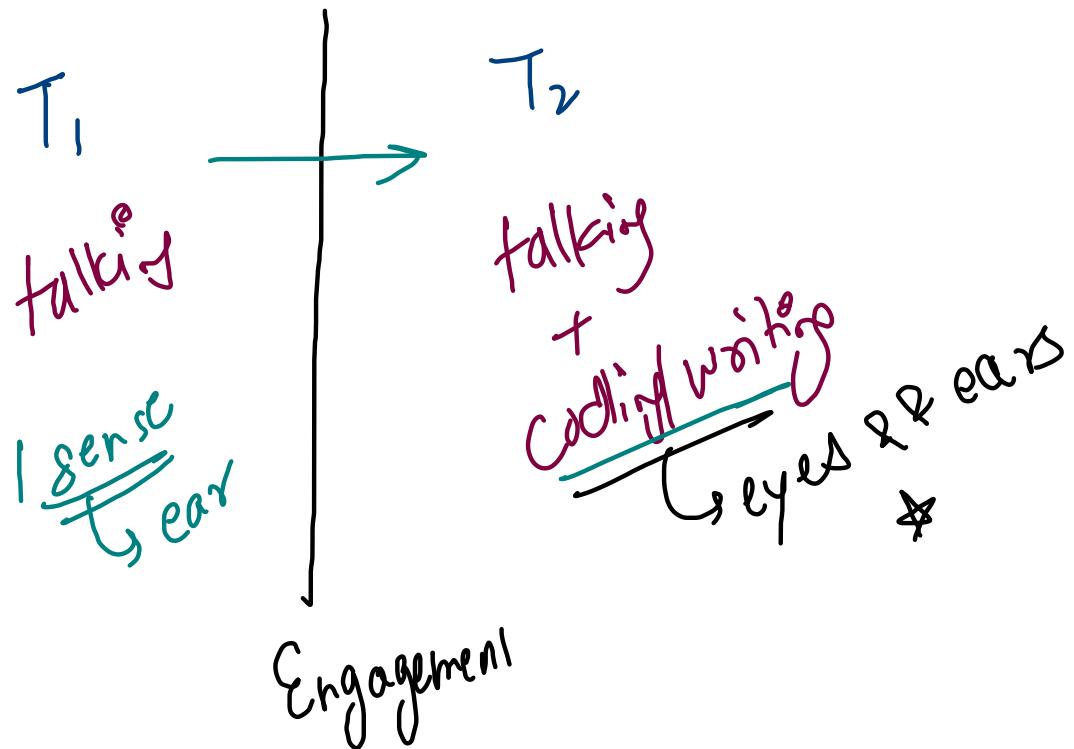
~~Cyan~~ Advice

① Be illustrative

② Be a good listener

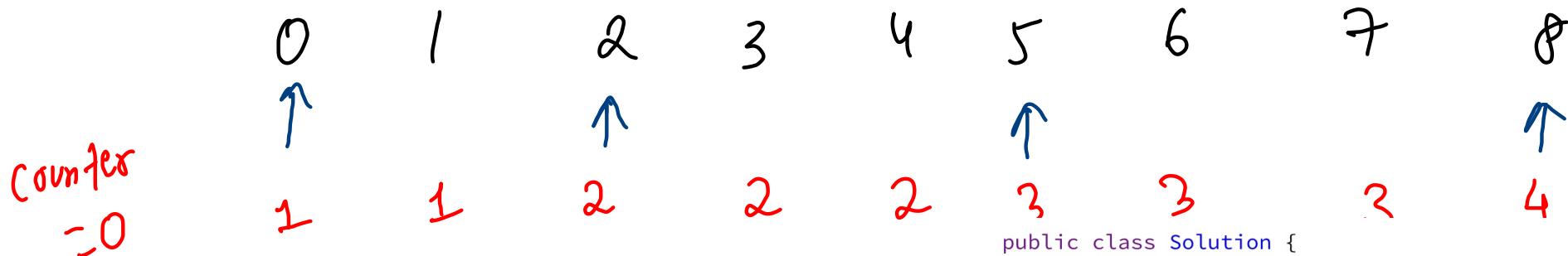
↳ Doubts?

③ Be patient / don't be rude



Count Index - value

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



$$arr[idx] == idx$$

```
public class Solution {  
    public static int getCount(int[] arr){  
        int counter = 0;  
        for(int idx = 0; idx < arr.length; idx++){  
            if(arr[idx] == idx) {  
                counter++;  
            }  
        }  
        return counter;  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int size = scn.nextInt();  
  
        int[] arr = new int[size];  
        for(int idx=0; idx<arr.length; idx++){  
            arr[idx] = scn.nextInt();  
        }  
  
        System.out.println(getCount(arr));  
    }  
}
```

First non-repeating element

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| | | | | | | | | |
| 50 | 20 | 70 | 40 | 20 | 10 | 60 | 50 | 30 |

0 1 2 3 4 5 6 7 8

← same → different

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| | | | | | | | | |
| 50 | 20 | 70 | 40 | 20 | 80 | 60 | 70 | 20 |

0 1 2 3 4 5 6 7 8

↑ ↑ ↑ ↑ ↑ ↑

Answer \Rightarrow 5th index

```
public static int getNonRepeating(int[] arr1, int[] arr2){  
    for(int idx=0; idx<arr1.length; idx++){  
        if(arr1[idx] != arr2[idx]) return idx;  
    }  
  
    return -1;  
}
```

} check identical variation

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();
```

```
int[] arr1 = new int[size];  
for(int idx=0; idx<size; idx++){  
    arr1[idx] = scn.nextInt();  
}
```

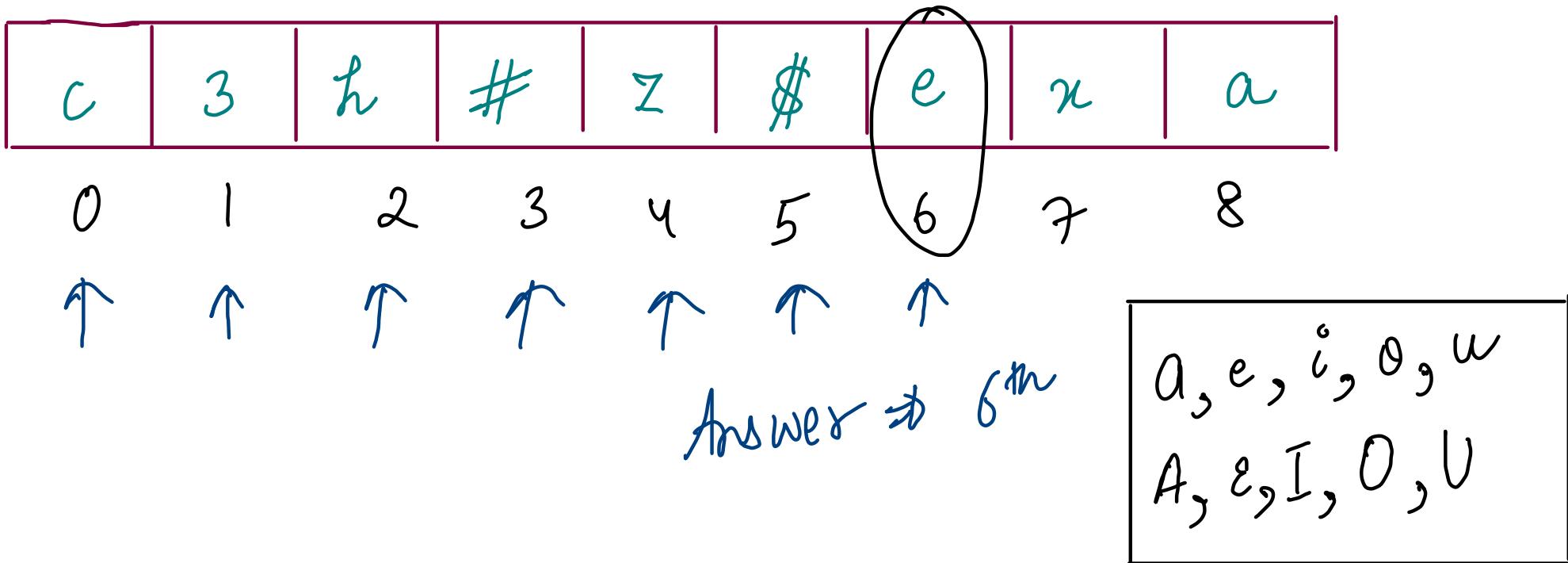
} Input of first array

```
int[] arr2 = new int[size];  
for(int idx=0; idx<size; idx++){  
    arr2[idx] = scn.nextInt();  
}
```

} Input of second array

```
System.out.println(getNonRepeating(arr1, arr2));  
}
```

First Occurrence of Vowel



if (arr[idn] == 'a' || arr[idn] == 'e' ||)

```
import java.io.*;
import java.util.*;

public class Solution {
    public static int getVowel(char[] arr){
        for(int idx = 0; idx < arr.length; idx++){
            if(arr[idx] == 'a' || arr[idx] == 'e' || arr[idx] == 'i' || arr[idx] == 'o' || arr[idx] == 'u' ||
            arr[idx] == 'A' || arr[idx] == 'E' || arr[idx] == 'I' || arr[idx] == 'O' || arr[idx] == 'U'){
                return idx;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();

        char[] arr = new char[size];

        for(int idx=0; idx<arr.length; idx++){
            arr[idx] = scn.next().charAt(0);
        }

        System.out.println(getVowel(arr));
    }
}
```

Repeating & Missing Element

| | | | | | | | | | |
|---|----|---|---|---|---|---|---|---|---|
| 6 | 10 | 8 | 7 | 4 | 8 | 1 | 9 | 5 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Repeating \Rightarrow ⑧

Missing \Rightarrow ③

$\overbrace{\text{isRepeating()}}$ of } \Rightarrow ⑧

$\overbrace{\text{isMissing()}}$ { } \Rightarrow ③

| | | | | | | | | | |
|---|----|---|---|---|---|---|---|---|---|
| 6 | 10 | 8 | 7 | 4 | 8 | 1 | 9 | 5 | 2 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

```

public static boolean isMissing(int[] arr, int target){
    int countOfOccurrences = 0;
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == target) countOfOccurrences++;
    }

    if(countOfOccurrences == 0) return true; //missing
    return false; //not missing
}

public static boolean isRepeating(int[] arr, int target){
    int countOfOccurrences = 0;
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == target) countOfOccurrences++;
    }

    if(countOfOccurrences == 2) return true;
    return false;
}

```

$\text{isMissing}(1)$ $\text{isRep}(1)$
 $\text{Count} = \emptyset / 1$ $(\text{Count} = 1)$
 $\text{isMissing}(2)$ $\text{isRep}(2)$
 $\text{Count} = \emptyset / 1$ $(\text{Count} = 1)$
 $\text{isMissing}(3)$ $\text{isRep}(3)$
 $\text{Count} = \emptyset / 0$ $(\text{Count} = 0)$
 \vdots
 $\text{isRep}(8)$
 $\text{Count} = \emptyset / 2$
 $\text{C}_{\text{repeating}}$

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();

    int[] arr = new int[size];
    for(int idx = 0; idx < arr.length; idx++){
        arr[idx] = scn.nextInt();
    }

    for(int idx = 1; idx <= size; idx++){
        if(isRepeating(arr, idx) == true) System.out.println(idx);
    }

    for(int idx = 1; idx <= size; idx++){
        if(isMissing(arr, idx) == true) System.out.println(idx);
    }
}
```

Second Largest Element

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 40 | 30 | 20 | 35 | 45 | 30 | 40 | 45 | 50 | 50 |
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |

largest = ~~-10~~ 40 45 50

secondlargest = ~~-∞~~ 30 35 40 45 ~~45~~

```
if (arr[idn] > largest){  
    secondlargest = largest;  
    largest = arr[idn];
```

```
} else if (arr[idn] > secondlargest) {  
    secondlargest = arr[idn];  
}
```

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| 40 | 30 | 20 | 35 | 45 | 30 | 40 | 45 | 50 | 50 |
|----|----|----|----|----|----|----|----|----|----|

largest = ~~-20~~ ~~30~~ ~~40~~ ~~45~~ 50

Second largest = ~~-20~~ ~~30~~ ~~35~~ ~~45~~ 45

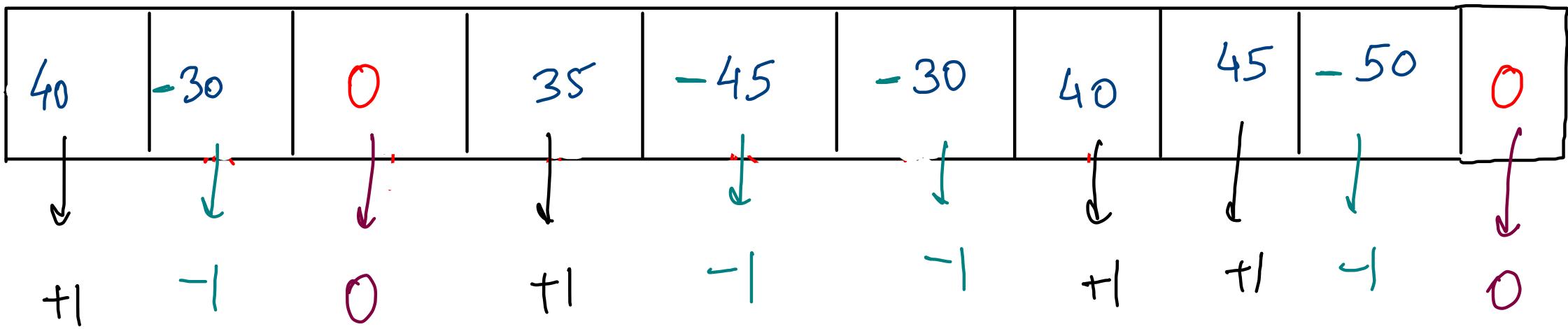
```

public static int secondMax(int[] arr){
    int largest = Integer.MIN_VALUE, secondlargest = Integer.MIN_VALUE;
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] > largest){
            secondlargest = largest;
            largest = arr[idx];
        }
        else if(arr[idx] > secondlargest && arr[idx] < largest){
            secondlargest = arr[idx];
        }
    }
    return secondlargest;
}

```

→ maintain
duplicacy as single
occurrence

Check characteristic



homework

Solve Array

Input
numbers →
indexes →

| idx: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|-----------|----|----|----|----|----|----|----|
| numbers → | 10 | 20 | 30 | 40 | 50 | 60 | 70 |
| indexes → | 3 | 6 | 0 | 1 | 4 | 2 | 5 |

Output
target:

| | | | | | | |
|----|----|----|----|----|----|----|
| 30 | 40 | 60 | 10 | 50 | 70 | 20 |
| ∅ | ∅ | ∅ | ∅ | ∅ | 6 | ∅ |

0 1 2 3 4 5 6

```

public static int[] solve(int[] numbers, int[] indexes){
    int[] res = new int[numbers.length];

    for(int idx = 0; idx < numbers.length; idx++){
        res[indexes[idx]] = numbers[idx];
    }

    return res;
}

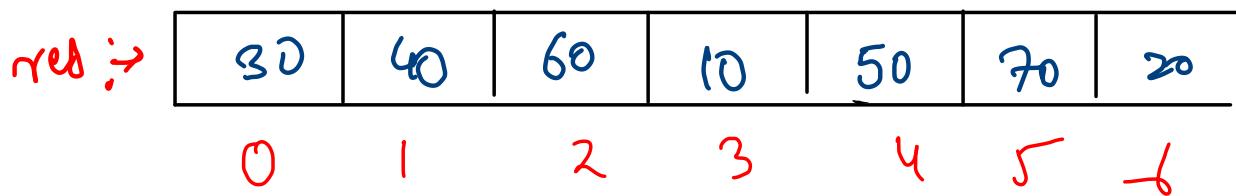
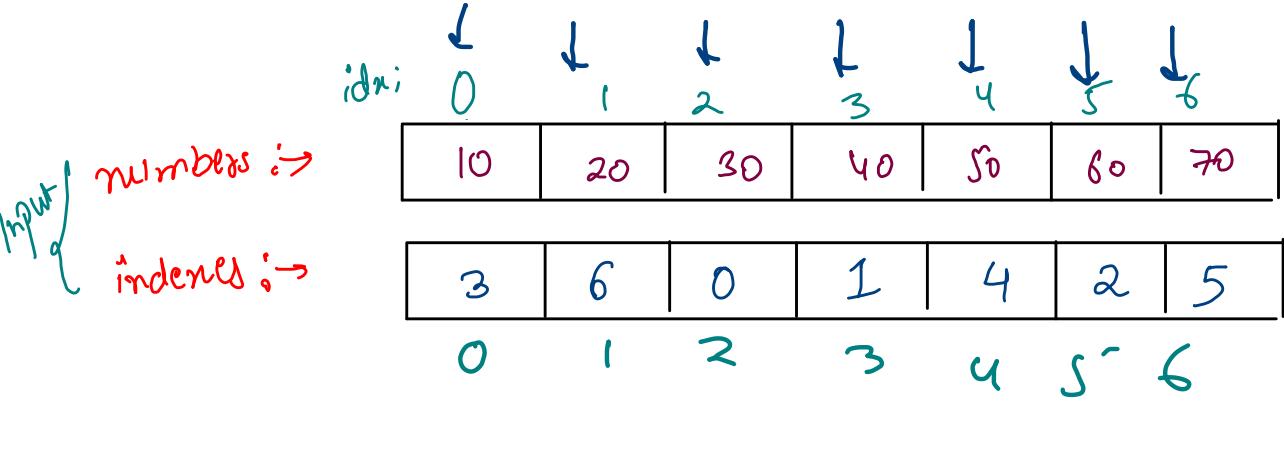
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();

    int[] numbers = new int[size];
    for(int idx = 0; idx < size; idx++){
        numbers[idx] = scn.nextInt();
    }

    int[] indexes = new int[size];
    for(int idx = 0; idx < size; idx++){
        indexes[idx] = scn.nextInt();
    }

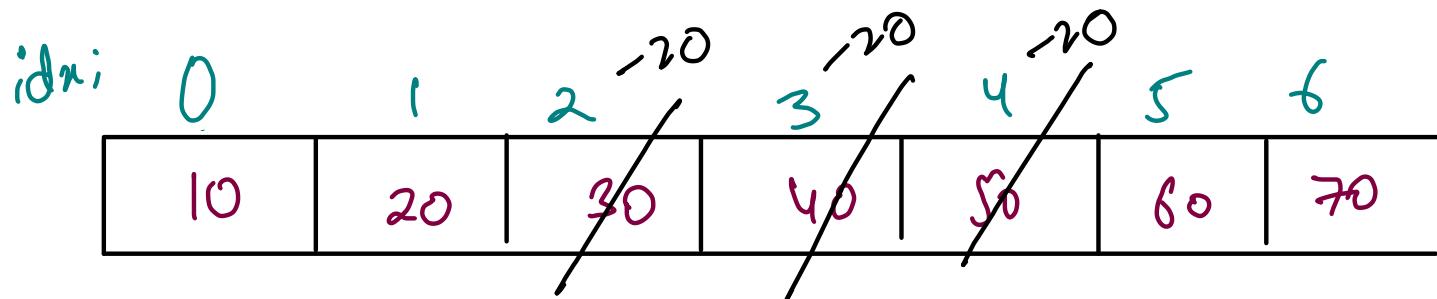
    int[] res = solve(numbers, indexes);
    for(int idx = 0; idx < size; idx++){
        System.out.print(res[idx] + " ");
    }
}

```



9:05 PM

Update Query - 1



left = 2, right = 4, $x = -20$

```
public static void updateArray(int[] arr, int left, int right, int target){  
    for(int idx = left; idx <= right; idx++){  
        arr[idx] = target;  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int size = scn.nextInt();  
        int[] arr = new int[size];  
  
        for(int idx = 0; idx < size; idx++){  
            arr[idx] = scn.nextInt();  
        }  
  
        int left = scn.nextInt();  
        int right = scn.nextInt();  
        int target = scn.nextInt();  
  
        updateArray(arr, left, right, target);  
        for(int idx = 0; idx < arr.length; idx++){  
            System.out.print(arr[idx] + " ");  
        }  
    }  
}
```

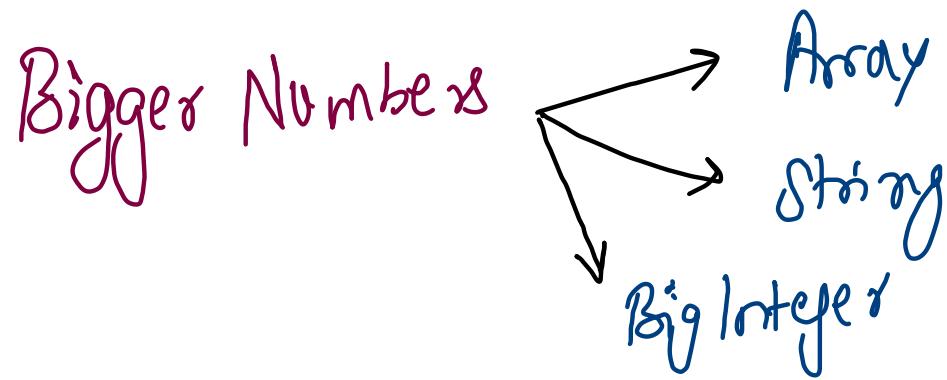
Changes
is persisting
across
functions

↓
Heap Area

Big Integers

Integer → atmost 10 digits ($2^{31}-1$)

Long → atmost 18 digits ($2^{63}-1$)



Add 1 to Array

~~Egg~~

$$\begin{array}{r}
 9, 2, 9, 9, 8, 8, 2 \\
 + 1 \\
 \hline
 9, 2, 9, 9, 8, 8, 3
 \end{array}$$

~~Egg~~

$$\begin{array}{r}
 1 0 0 0 \\
 + 1 \\
 \hline
 1 0 0 1
 \end{array}$$

~~Egg~~

$$\begin{array}{r}
 7, 8, 2, 6, 7 \\
 + 1 \\
 \hline
 7, 8, 2, 6, 8 \quad 0 \quad 0 \quad 0 \quad 0
 \end{array}$$

~~Egg~~

~~Corner Case Egg~~

~~new Array~~

$$\begin{array}{r}
 1 \quad 1 \quad 1 \quad 1 \\
 \bullet \quad 9 \quad 9 \quad 9 \quad 9 \\
 + 1 \\
 \hline
 1 \quad 0 \quad 0 \quad 0 \quad 0
 \end{array}$$

Starting 1 All 0s

| | | | | | | | |
|---|---|---|---|---------|---------|---------|---------|
| 0 | 1 | 2 | 3 | +1 4 | +1 5 | +1 6 | +1 7 |
| g | 2 | 5 | 6 | g ← | g ← | g ← | g ← |
| | | | | 7 | 0 | 0 | 0 |

| | | | | | | |
|----|---|---|---|---------|---------|---------|
| -1 | 1 | 1 | 1 | +1 3 | +1 4 | +1 5 |
| 1 | 0 | 1 | 2 | g ← | g ← | g ← |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

```
public static int[] addOne(int[] arr){  
    int idx;  
    for(idx = arr.length - 1; idx >= 0 && arr[idx] == 9; idx--){  
        arr[idx] = 0;  
    }  
  
    if(idx == -1){  
        int[] res = new int[arr.length + 1];  
        res[0] = 1;  
        return res;  
    }  
  
    arr[idx]++;  
    return arr;  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
    for(int idx = 0; idx < size; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int[] res = addOne(arr);  
    for(int idx = 0; idx < res.length; idx++){  
        System.out.print(res[idx] + " ");  
    }  
}
```

Commented

Carry

concatenate

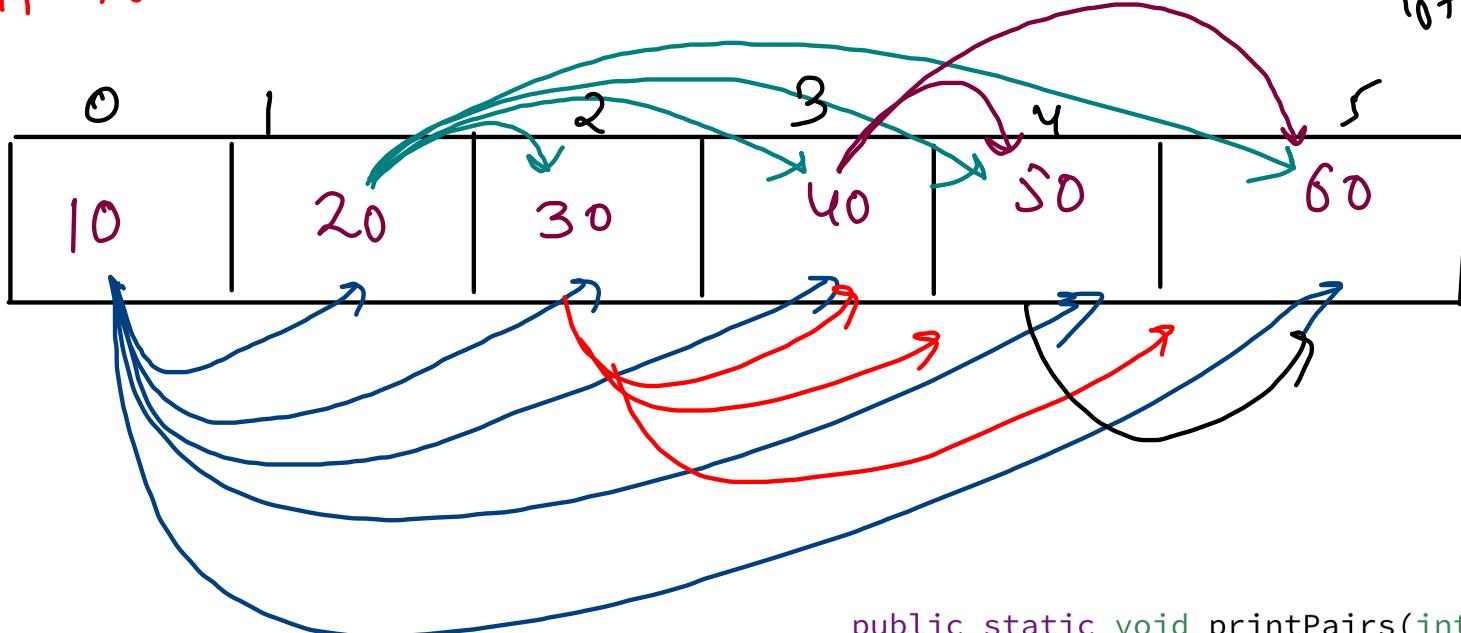
All digits are 9

arr[idx] == 9; idx--) {
 concrete case
}; } concrete case
All digits are 9

| | | | | | | | | |
|------------|---|---|---|---|---|---|---|---|
| <u>egy</u> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | 8 | 2 | 7 | 6 | 5 | 9 | 9 | 9 |
| | | | | | ↑ | ↑ | ↑ | ↑ |
| | 8 | 2 | 7 | 6 | 6 | 0 | 0 | 0 |

| leg² | 0 | 1 | 2 | 3 | 4 | 5 |
|----------------------------|---|---|---|---|---|---|
| | g | g | g | g | g | g |
| | ↑ | q | ↑ | ↑ | ↑ | T |
| | 0 | 0 | 0 | 0 | 0 | 0 |
| red array | 0 | 0 | 0 | 0 | 0 | 0 |

Print Pairs



$$\text{Total pairs} = 5 + 4 + 3 + 2 + 1 \approx 15$$

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

10 20
10 30
10 40
10 50
10 60

20 30
20 40
20 50
20 60

30 40
30 50
30 60

40 50
40 60

50 60

```
public static void printPairs(int[] arr){  
    for(int left = 0; left < arr.length; left++){  
        for(int right = left + 1; right < arr.length; right++){  
            System.out.println(arr[left] + " " + arr[right]);  
        }  
    }  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
    printPairs(arr);  
}
```

Print Pairs with Given Sum (Permutations)

20 60 10 50 30

l
↓
40
↑
r

✓ 20 60

✓ 60 20

✓ 50 30

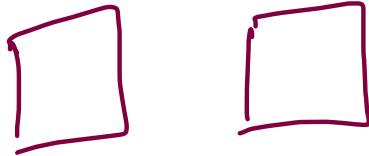
✓ 30 50

✓ 40 40

```
public static void printPairs(int[] arr, int target){  
    for(int left = 0; left < arr.length; left++){  
        for(int right = 0; right < arr.length; right++){  
            if(arr[left] + arr[right] == target){  
                System.out.println(arr[left] + " " + arr[right]);  
            }  
        }  
    }  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int target = scn.nextInt();  
    printPairs(arr, target);  
}
```

target = 80

Combinations



Select 2 seats out of 4 seats

$$= 4C_2 = \frac{4!}{2!}$$

s_1 s_2

(1) s_1 s_2 • •

(2) s_1 • s_2 •

(3) s_1 • • s_2

(4) • s_1 s_2 •

(5) • s_1 • s_2

(6) • • s_1 s_2

$$= \frac{4 \times 3}{2} = 6$$

~~permutation~~



Select 2 seats out of 4 seats

$$= {}^4P_2$$

① $s_1 s_2 \dots$, ⑦ $s_2 s_1 \dots$

$s_1 s_2$

② $s_1 \dots s_2 \dots$, ⑧ $s_2 \dots s_1 \dots$

$$= \frac{4!}{2!}$$

③ $s_1 \dots \dots s_2$, ⑨ $s_2 \dots s_1$

④ $\dots s_1 s_2 \dots$, ⑩ $\dots s_2 s_1 \dots$

$$= \frac{4!}{2!}$$

⑤ $\dots s_1 \dots s_2$, ⑪ $\dots s_2 \dots s_1$

⑫

⑥ $\dots \dots s_1 s_2$, ⑫ $\dots \dots s_2 s_1$

Print Pairs with Given Sum (Combinatorial)

20 60 10 50 30 40

✓ 20 60 ✗ 60 20

✓ 50 30 ✗ 30 50

✓ 40 40

```
public static void printPairs(int[] arr, int target){
    for(int left = 0; left < arr.length; left++){
        for(int right = left; right < arr.length; right++){
            if(arr[left] + arr[right] == target){
                System.out.println(arr[left] + " " + arr[right]);
            }
        }
    }
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];

    for(int idx = 0; idx < arr.length; idx++){
        arr[idx] = scn.nextInt();
    }

    int target = scn.nextInt();
    printPairs(arr, target);
}
```

Print Count of Elements Greater than me

Input

20 60 10 50 40 40

0 1 2 3 4 5

resultant

4 0 5 1 2 2

① Count of Elements Greater
than Target

```

public class Solution {
    public static int countGreater(int[] arr, int target){
        int count = 0;
        for(int idx = 0; idx < arr.length; idx++){
            if(arr[idx] > target) count++;
        }
        return count;
    }

    public static void solve(int[] arr){
        for(int idx = 0; idx < arr.length; idx++){
            int count = countGreater(arr, arr[idx]);
            System.out.print(count + " ");
        }
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];

        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        solve(arr);
    }
}

```

| 0 | 1 | 2 | 3 | 4 | 5 |
|-------|----|----|-------------|----|----|
| 20 | 60 | 40 | 10 | 50 | 40 |
| solve | X | X | X | X | X |
| | | | | | ↑ |
| | | | cantGreater | | |
| | | | | 4 | 0 |
| | | | | 2 | 5 |
| | | | | 1 | 2 |

$$\begin{aligned}
 & f + f + f + \dots + f \text{ times} \\
 & = f * f \\
 & = n * n \\
 & \text{where } n \text{ is} \\
 & \text{size of array}
 \end{aligned}$$

Print Elements Greater in Right

| | | | | | |
|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 20 | 60 | 40 | 10 | 50 | 40 |
| 4 | 0 | 1 | 2 | 0 | 0 |

```
public class Solution {  
    public static int countGreater(int[] arr, int target, int left){  
        int count = 0;  
        for(int right = left + 1; right < arr.length; right++){  
            if(arr[right] > target) count++;  
        }  
        return count;  
    }  
  
    public static void solve(int[] arr){  
        for(int left = 0; left < arr.length; left++){  
            int count = countGreater(arr, arr[left], left);  
            System.out.print(count + " ");  
        }  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int size = scn.nextInt();  
        int[] arr = new int[size];  
  
        for(int idx = 0; idx < arr.length; idx++){  
            arr[idx] = scn.nextInt();  
        }  
  
        solve(arr);  
    }  
}
```

Highest Frequency Element

| | | | | | | | | | |
|----|----|----|----|----|-----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 20 | 40 | 10 | 30 | 40 | 100 | 20 | 40 | 10 | 40 |

highest frequency = ~~0 20 40~~ 40

Count of occurrences = ~~0 2 4~~ 4

20 10 30 10 10 30 20 40 20 10
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

```
public static int getFrequency(int[] arr, int target){  
    int count = 0;  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] == target) count++;  
    }  
    return count;  
}
```

value = 0 10
maxFreq = 0 3/4

```
public static int maxFrequency(int[] arr){  
    int value = 0, maxFrequency = 0;  
    for(int idx = 0; idx < arr.length; idx++){  
        int currFrequency = getFrequency(arr, arr[idx]);  
  
        if(currFrequency > maxFrequency){  
            value = arr[idx];  
            maxFrequency = currFrequency;  
        }  
    }  
  
    return value;  
}
```

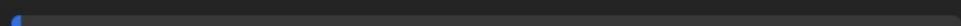
How is the pace currently

00:01:26 | 1 question | 91 of 128 (71%) participat...

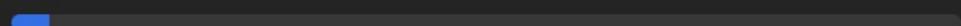
1. Slow or Fast (Single Choice) *

91/91 (100%) answered

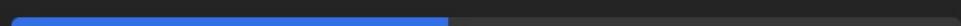
Very Slow (1/91) 1%



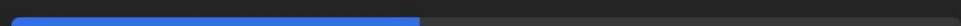
Slow (4/91) 4%



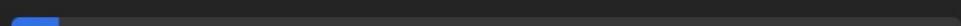
Perfect (42/91) 46%



Fast (39/91) 43%



Very Fast (5/91) 5%



Find Duplicate 3

Eg¹

20 10 60 50 40 30

"false"

Eg²

20 10 60 80 60 40

"true"

Common cases

Eg³

20 20 20 20

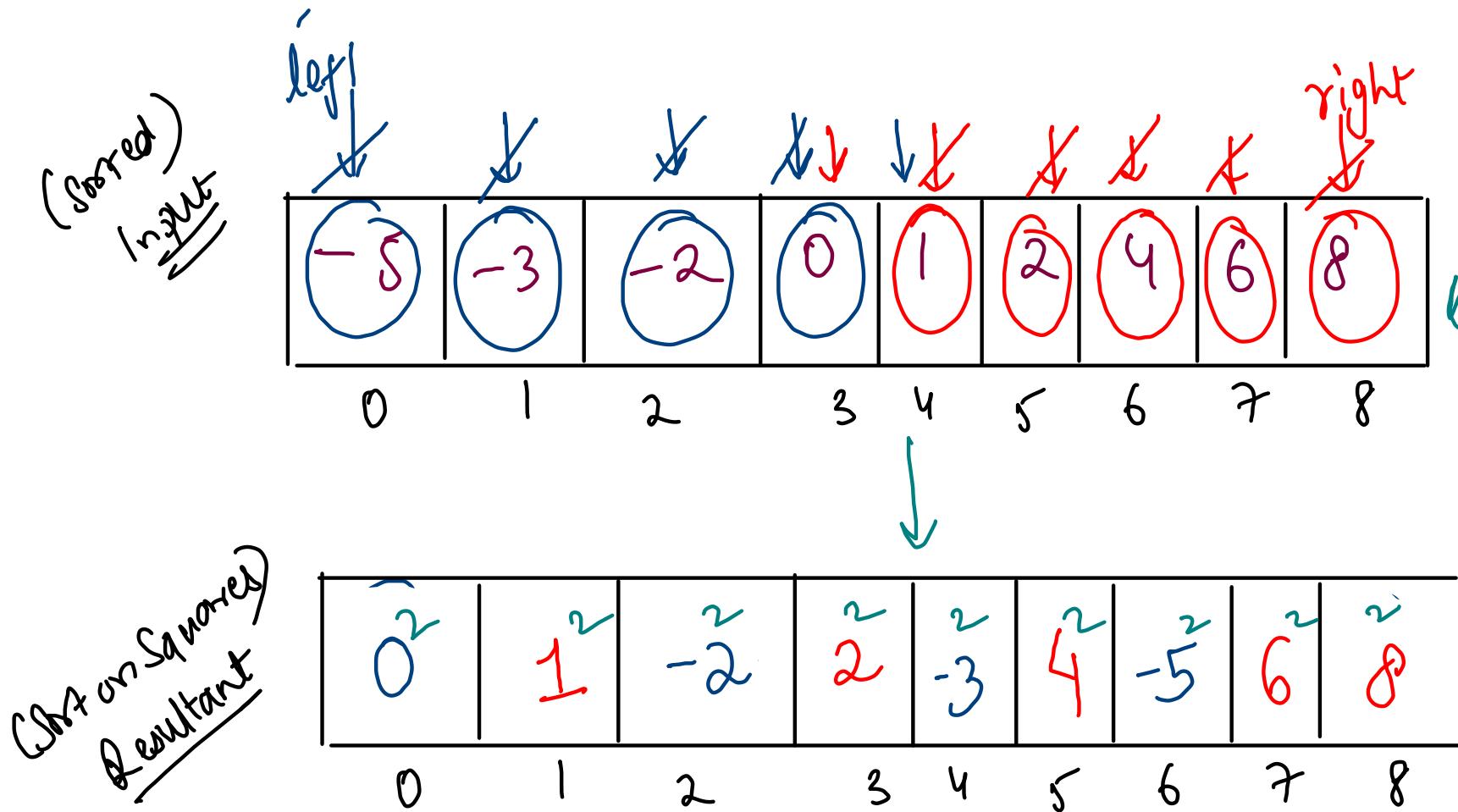
"true"

Eg⁴

10 "false"

```
public static int getFrequency(int[] arr, int target){  
    int count = 0;  
    for(int right = 0; right < arr.length; right++){  
        if(arr[right] == target) count++;  
    }  
    return count;  
}  
  
public static boolean checkDuplicate(int[] arr){  
    for(int left = 0; left < arr.length; left++){  
        int count = getFrequency(arr, arr[left]);  
        if(count > 1) return true;  
    }  
  
    return false;  
}
```

Sort Array Based on Squares

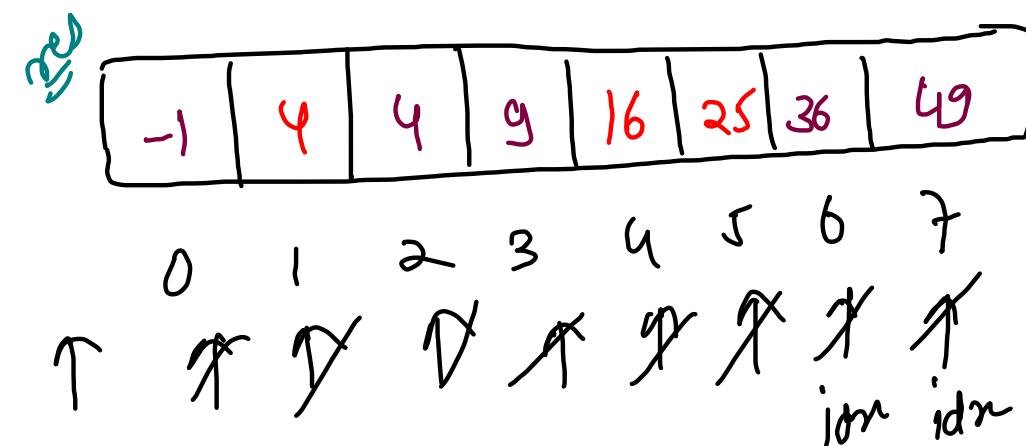
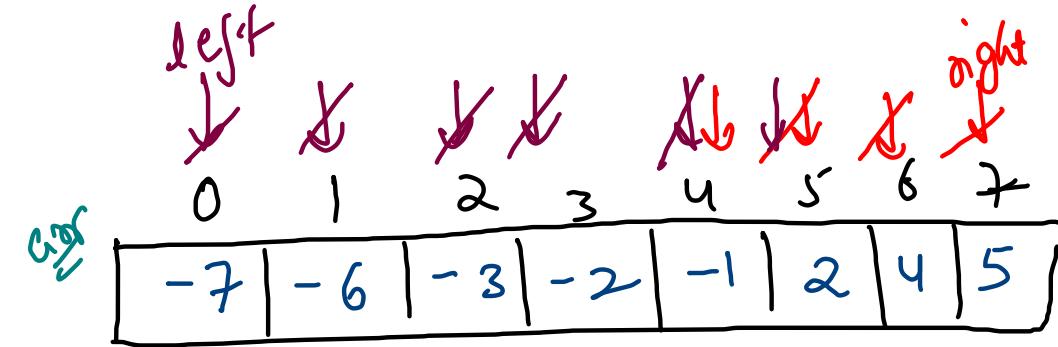


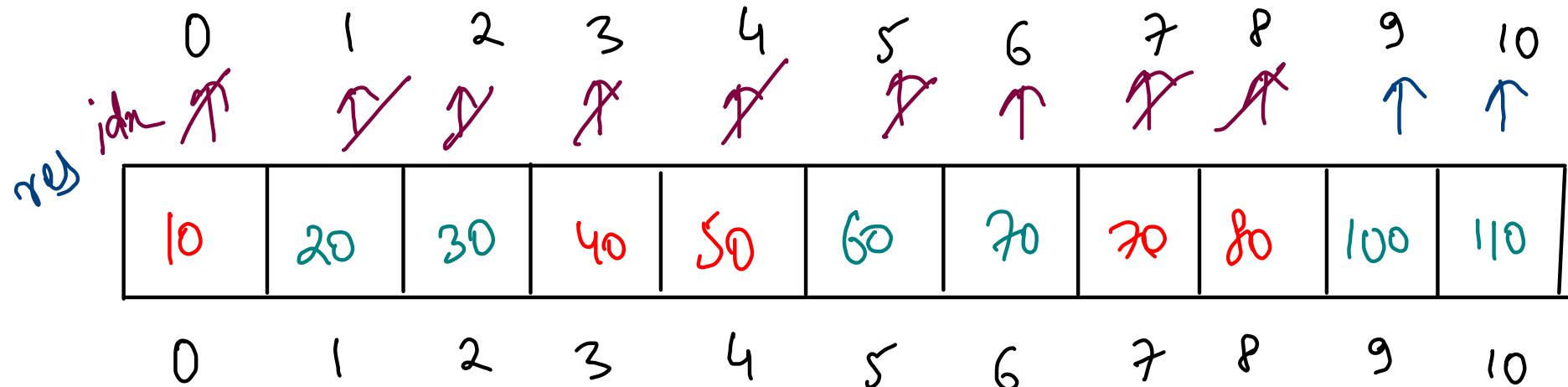
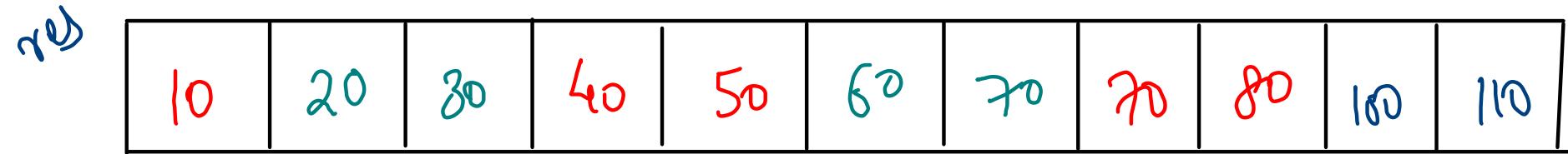
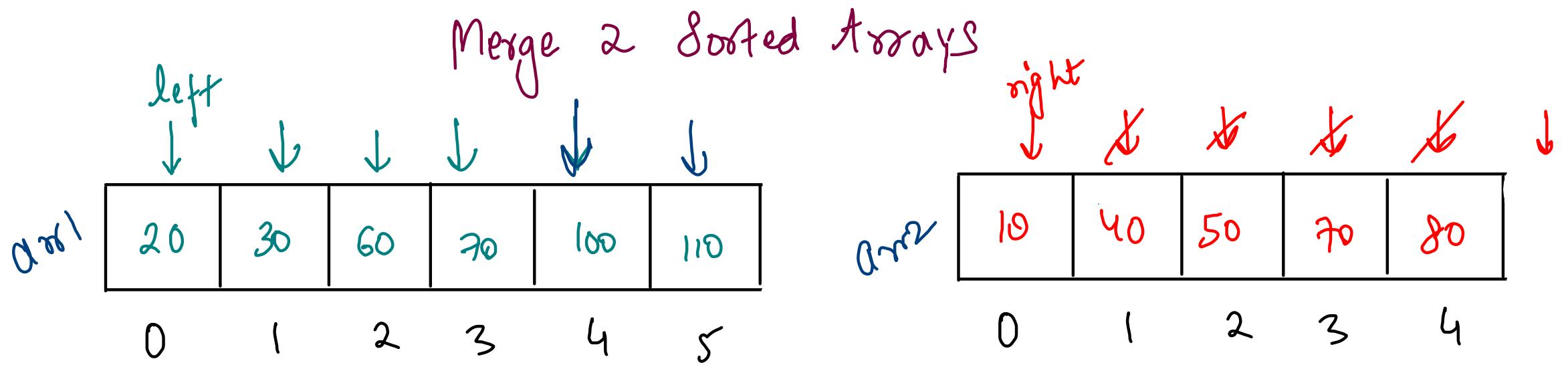
Leetcode 977 (Sort Array Squares)

```
class Solution {
    public int[] sortedSquares(int[] arr) {
        int[] res = new int[arr.length];
        int left = 0, right = arr.length - 1, idx = arr.length - 1;

        while(left <= right){
            if(arr[left] * arr[left] >= arr[right] * arr[right]){
                res[idx] = arr[left] * arr[left];
                left++;
                idx--;
            } else {
                res[idx] = arr[right] * arr[right];
                right--;
                idx--;
            }
        }

        return res;
    }
}
```





```

public static int[] merge(int[] arr1, int[] arr2){
    int first = 0, second = 0, third = 0;
    int[] res = new int[arr1.length + arr2.length];

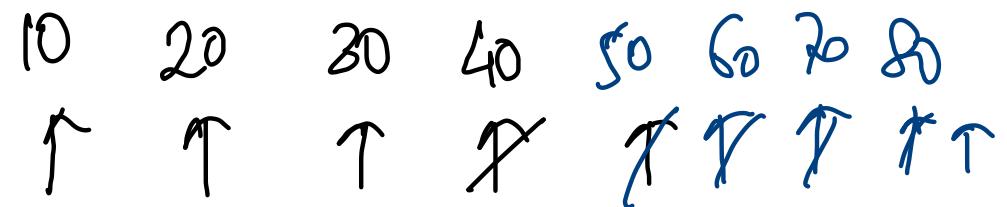
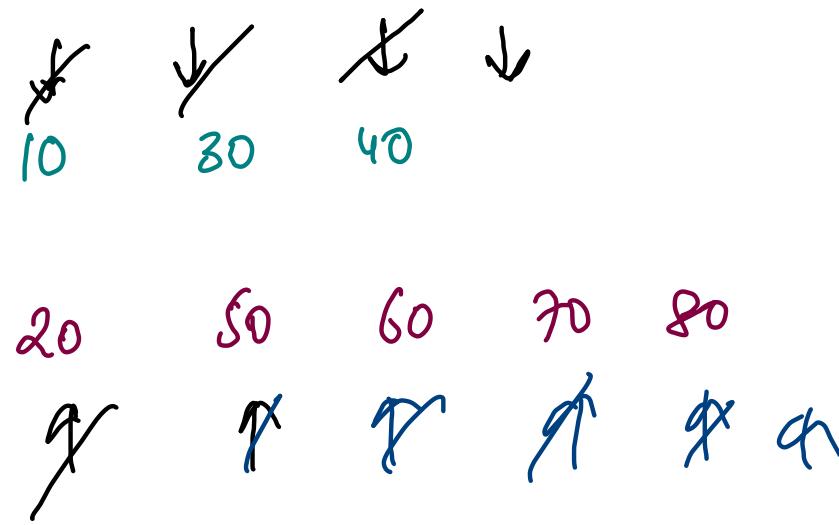
    while(first < arr1.length && second < arr2.length){
        if(arr1[first] <= arr2[second]){
            res[third] = arr1[first];
            first++;
            third++;
        } else {
            res[third] = arr2[second];
            second++;
            third++;
        }
    }

    while(first < arr1.length){
        res[third] = arr1[first];
        first++;
        third++;
    }

    while(second < arr2.length){
        res[third] = arr2[second];
        second++;
        third++;
    }

    return res;
}

```



Sorting Algorithms

Bubble Sort

70 50 20 60 10 20 30

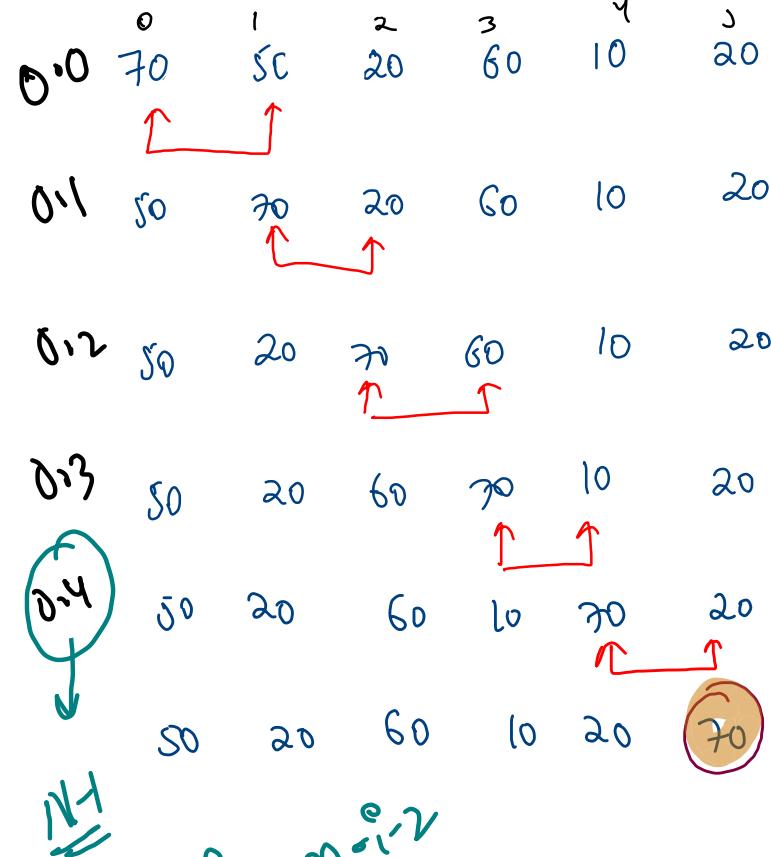
↓ sorting

10 20 20 30 50 60 70

Select Sort

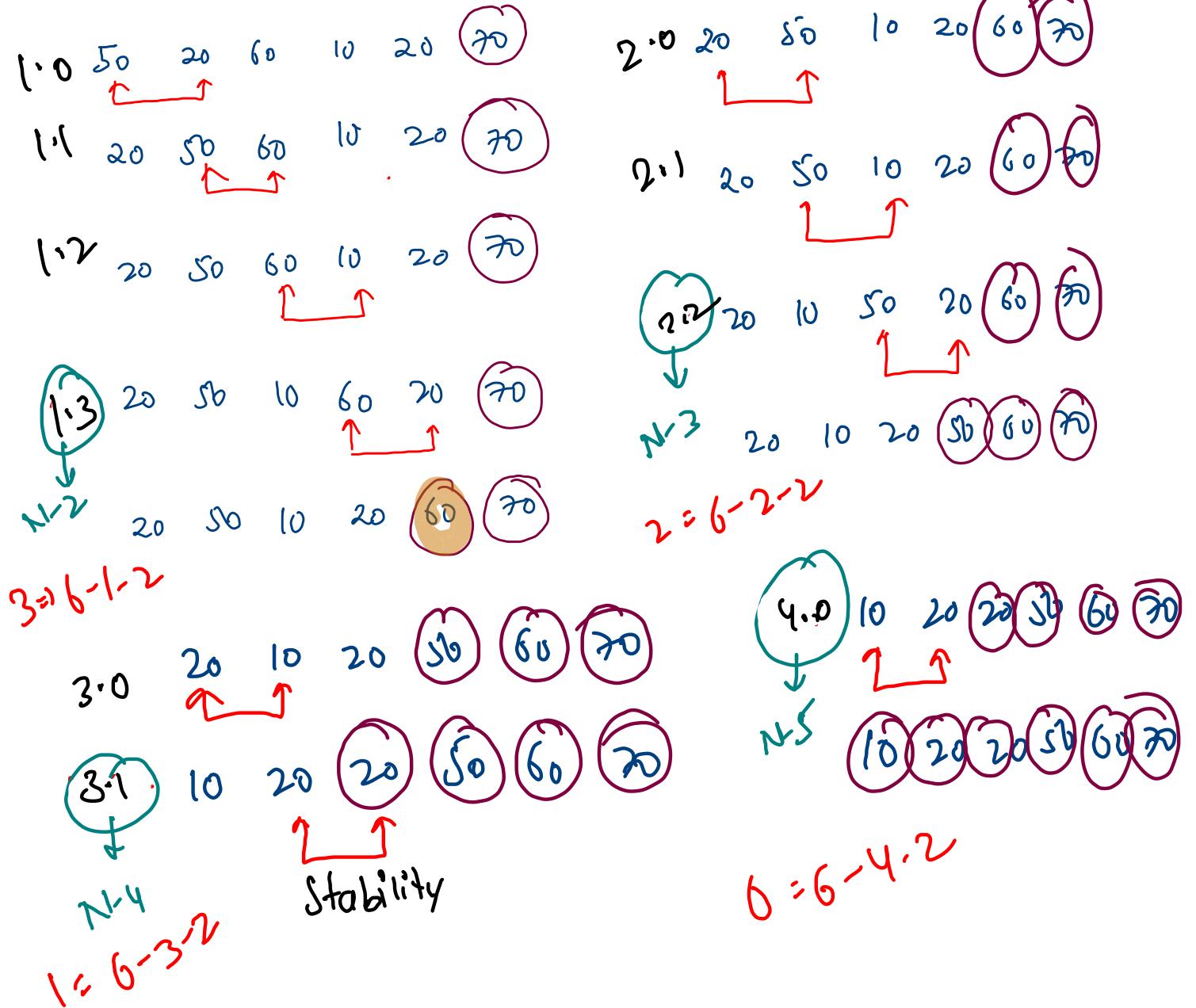
Insertion Sort

Bubble Sort



$4 \Rightarrow 7-6-2$

$4 \Rightarrow 6-0-2$



```

public static void bubbleSort(int[] arr){
    for(int i = 0; i < arr.length - 1; i++){
        for(int j = 0; j < arr.length - i - 1; j++){
            if(arr[j] > arr[j + 1]){
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

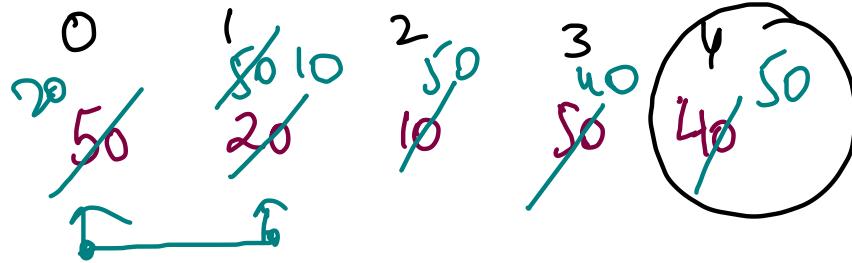
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];
    for(int idx = 0; idx < size; idx++){
        arr[idx] = scn.nextInt();
    }

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

```

→ Inplace
 → no new array
 → Stable

0: 0
 0: 1
 0: 2
 0: 3
 1: 0
 1: 1
 1: 2
 2: 0
 2: 1
 3: 0
 3



Selection Sort

0 1 2 3 4 5
 70 50 20 60 10 20

0.1 ↑
 0.2 ↑
 0.3 ↑
 0.4 ↑
 0.5 ↑
 0 1 2 3 4 5
 10 50 20 60 30 20

$$\text{minIdx} = \cancel{\phi} / \cancel{24}$$

minIdx = ~~1~~ 2
 0.2 ↑
 0.3 ↑
 0.4 ↑
 0.5 ↑
 0 1 2 3 4 5
 10 50 20 60 30 20

0 1 2 3 4 5
 10 20 50 60 70 20

2.3 ↑
 2.4 ↑
 2.5 ↑
 0 1 2 3 4 5
 10 20 20 60 70 50

3.4 ↑
 3.5 ↑
 0 1 2 3 4 5
 10 20 20 50 70 60

4.5 ↑
 0 1 2 3 4 5
 10 20 20 50 60 70

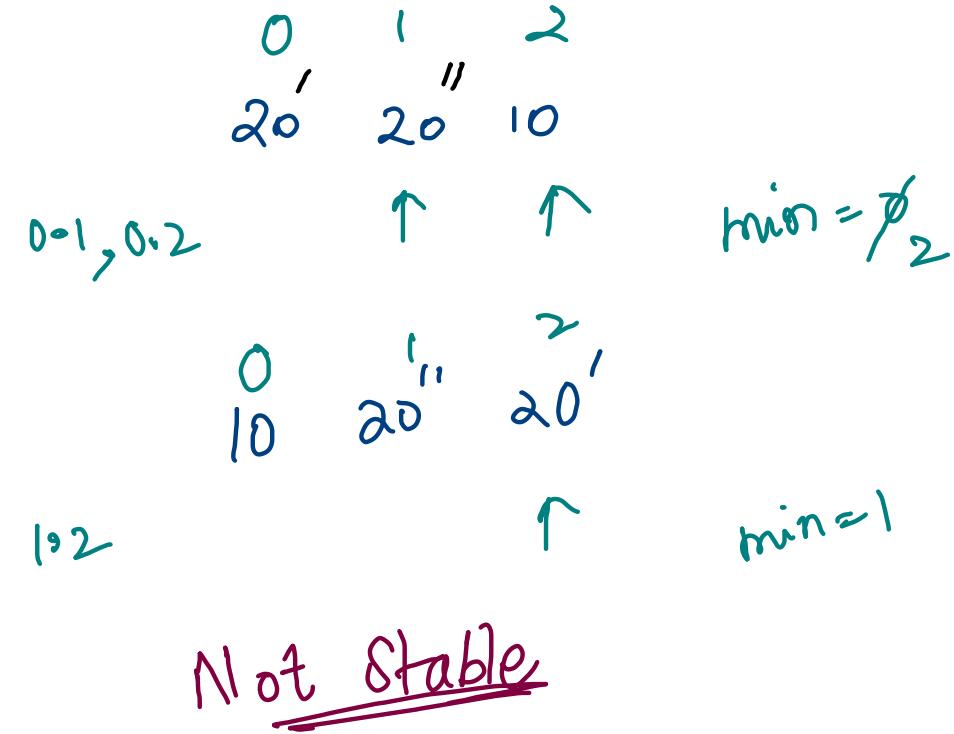
```
public static void selectionSort(int[] arr){  
    for(int i = 0; i < arr.length - 1; i++){  
        int minIdx = i;  
        for(int j = i + 1; j < arr.length; j++){  
            if(arr[j] < arr[minIdx]){  
                minIdx = j;  
            }  
        }  
        int temp = arr[i];  
        arr[i] = arr[minIdx];  
        arr[minIdx] = temp;  
    }  
}
```

Inplace ✓
Stable ✓

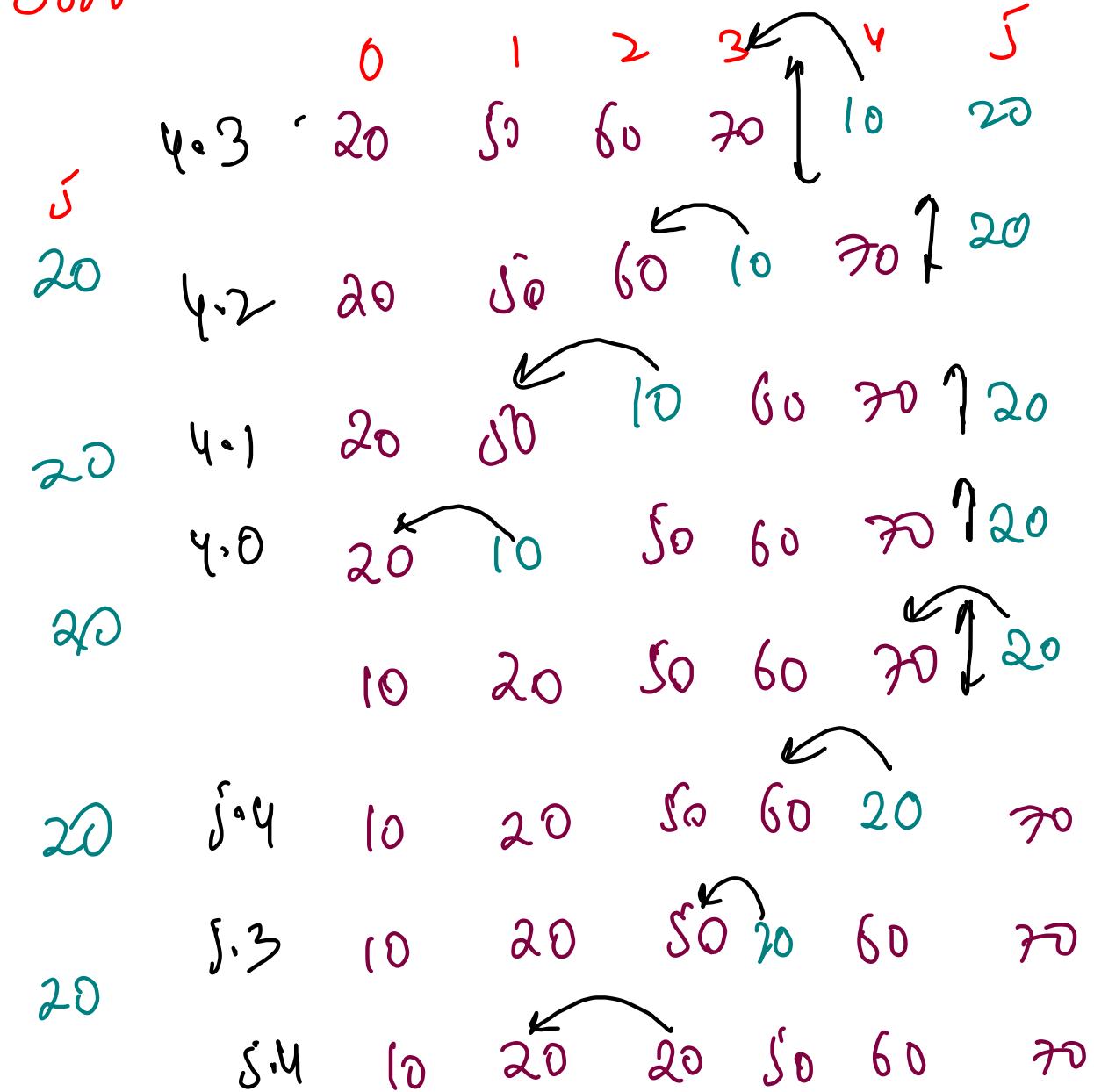
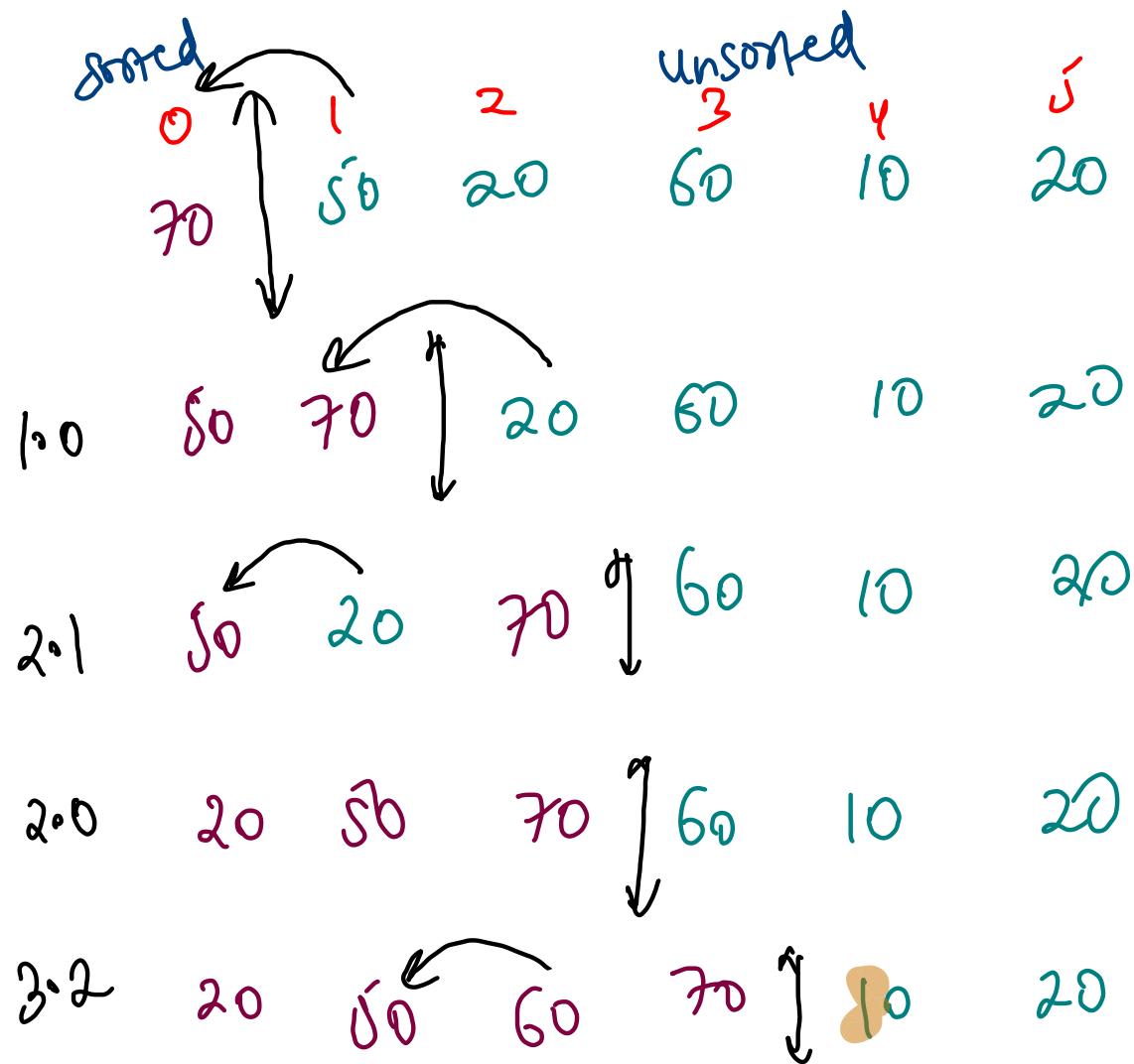
```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];
    for(int idx=0; idx<size; idx++){
        arr[idx] = scn.nextInt();
    }

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

Inplace ✓
Stable ↴



Insertion Sort



```

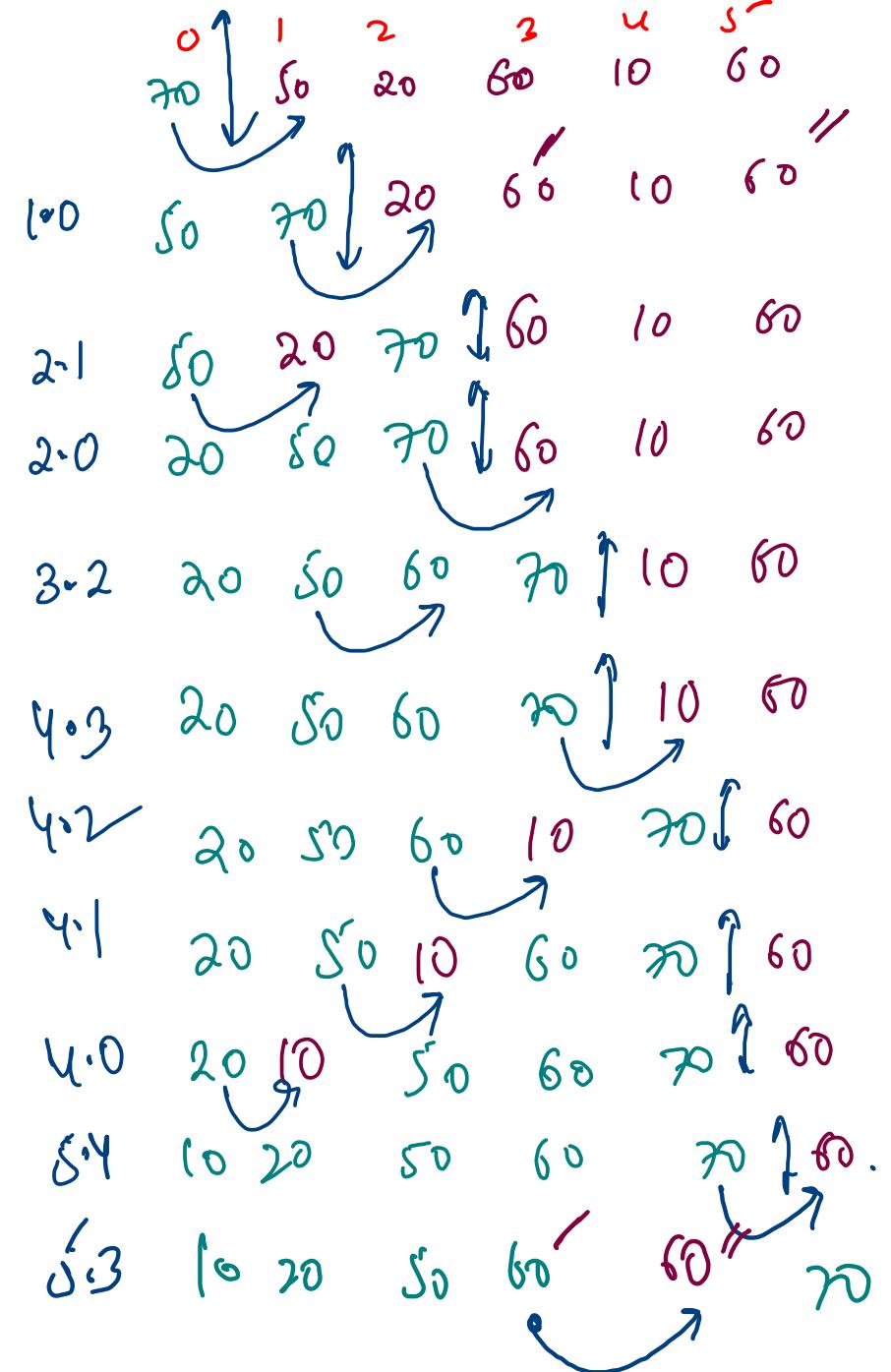
public static void insertionSort(int[] arr){
    for(int i=1; i<arr.length; i++){
        for(int j=i-1; j>=0; j--){
            if(arr[j] > arr[j + 1]){
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            } else break;
        }
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];
        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

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

```

*stable ✓
Inplace ✓*



Kth largest Element in unsorted array

0 1 2 3 4 5
 70 50 20 60 10 30

$k=1$ 70

$k=2$ 60

$k=3$ 50

$k=4$ 30

$k=5$ 20

$k=6$ 10

$k=1$ 50 30 60 10 20 70

$k=2$ 30 10 50 20 60 70

$k=3$ 30 10 20 50 60 70

$k=4$ 10 20 30 50 60 70

$k=5$ 10 20 30 50 60 70

$k=6$ 0 1 2 3 4 5

k 6 5 4 3 2 1

6^{-6}

6^{-5}

6^{-4}

6^{-3}

6^{-2}

6^{-1}

```
import java.io.*;
import java.util.*;

public class Solution {
    public static int kthlargest(int[] arr, int k){
        for(int i = 0; i < k; i++){
            for(int j = 0; j < arr.length - i - 1; j++){
                if(arr[j] > arr[j + 1]){
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        return arr[arr.length - k];
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int k = scn.nextInt();

        int[] arr = new int[size];
        for(int idx = 0; idx < size; idx++){
            arr[idx] = scn.nextInt();
        }

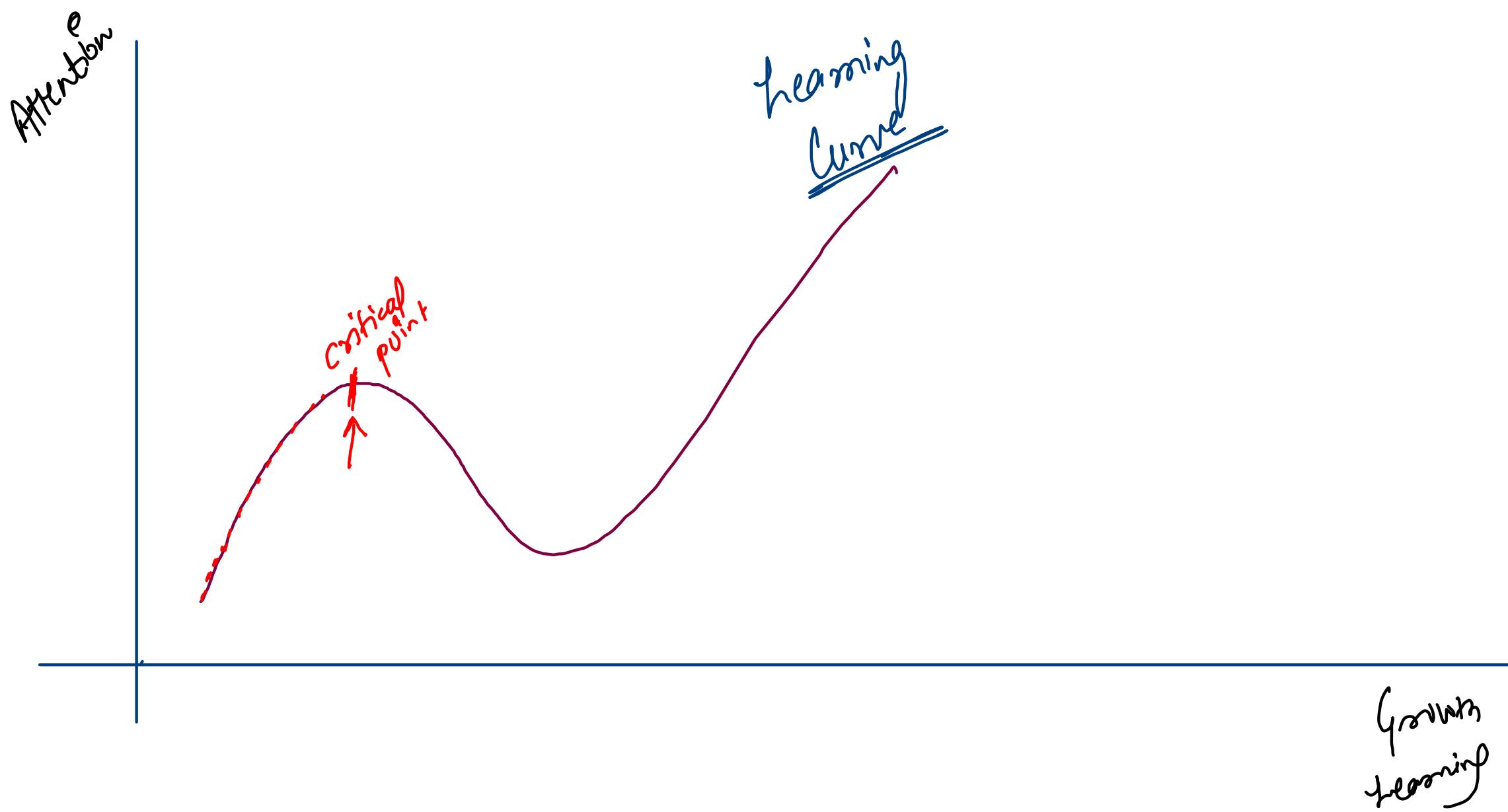
        System.out.println(kthlargest(arr, k));
    }
}
```

bubble sort junction

DSA (Fresher → 80+)

- Getting Started Module 1
- Arrays & Strings Module 2
- Object Oriented programming
in Java Module 3
 - ↳ DSA based project
 - ↳ how-level Design
- Collection frameworks Module 4
 - ↳ ArrayList
 - ↳ LinkedList
 - ↳ Stack & Queue
 - ↳ Hashmap & Heaps

- Advanced DSA ~~elite batch weekend~~
- ↳ Recursion & Backtracking
 - ↳ Binary Tree & BST
 - ↳ Graphs
 - ↳ Dynamic Programming
 - ↳ Tries
 - ↳ Bit Manipulation & Number Theory



Form a Largest No

"914"

"23"

"9"

"243"

"10"

"1"

"92"

9, 92, 914, 243, 23, 1, 10

String comparison / lexicographical comparison

97 <
↓
"archit" <
"z"
↓
larger
smaller

49 <
↓
"12345"
"gg"
↓
57

" $\downarrow \downarrow$ "
"g14"
smallest

" \downarrow "
"g"
largest

" $\downarrow \downarrow$ "
"g2"
second smallest

g, g2, g14

$$\begin{aligned} "g14" + g &< "g + g14" \\ "a + b" &\angle "b + a" \\ "g14g" &\angle "gg14" \\ \Rightarrow a &\angle b \\ "g14" &\angle \textcircled{g} \end{aligned}$$

g14

$$g14 + g2 \quad \angle \quad g2 + g14$$

$$g14g2 \quad \angle \quad g2g14$$

$$\begin{aligned} "g14" &\angle "g2" \\ a &\angle b \end{aligned}$$

Array \rightarrow Max^m Product of 3 No's

~~Max~~

-7 3 -5 2 4

$$2 \times 3 \times 4 = 24$$

$$-7 \times -5 \times 4 = 140$$

~~Min~~

3, -7, 6, 8, -5, -10, 2, 10, -8

$$\text{max} \times 2^{\text{nd}} \text{max} \times 3^{\text{rd}} \text{max}$$
$$10 \times 8 \times 6 = 480$$

Or

$$\text{min} \times 2^{\text{nd}} \text{min} \times \text{max}$$
$$-10 \times -8 \times 10 = 800$$

~~egs~~

-1, -3, -10, 4, -2, 12, 15, 10

One possibility
max * 2nd max * 3rd max
15 * 12 * 10

$$= \cancel{1800}$$

$\alpha \omega(n^{-1}) * \alpha \omega(n^{-2})$
 $* \alpha \omega(n^{-3})$

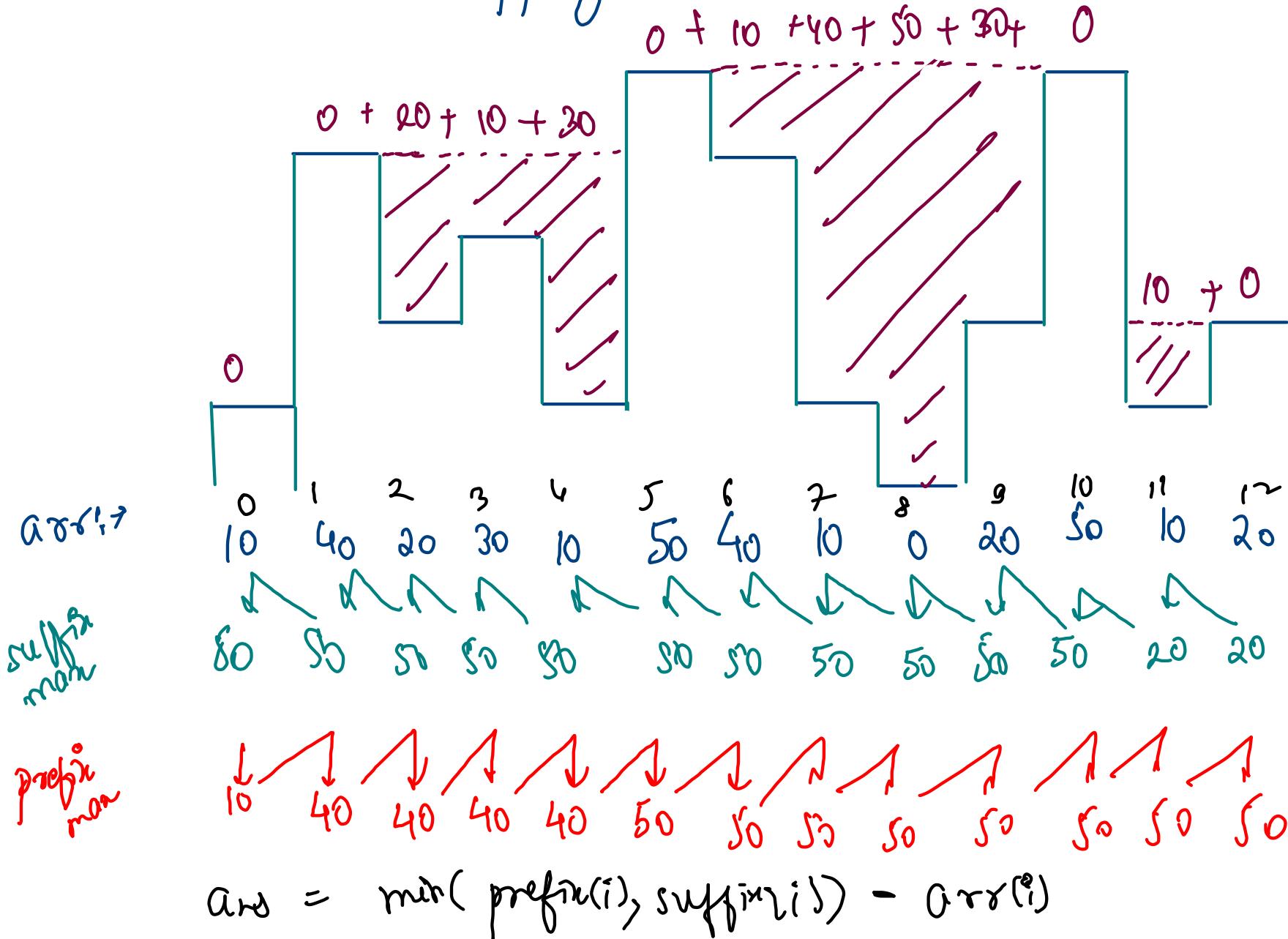
\leq

Other possibility
min * 2nd min * max
-10 * -3 * 15
= 450

$\alpha \omega(0) * \alpha \omega(1)$
 $* \alpha \omega(n^{-1})$

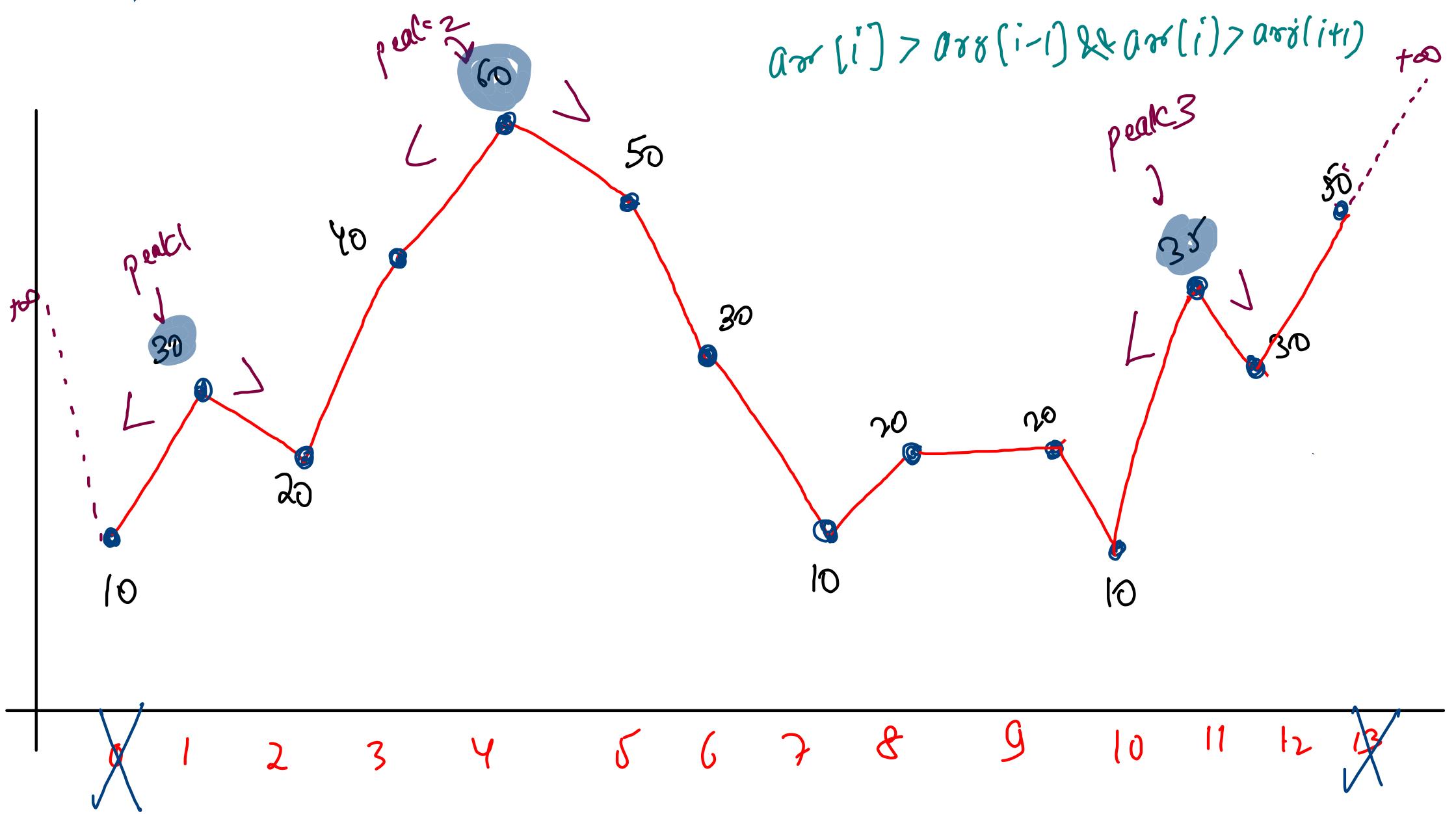
```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
    for(int idx = 0; idx < size; idx++){  
        arr[idx] = scn.nextInt();  
    }  
    Arrays.sort(arr); → sorting (Increasing)  
    int choice1 = arr[0] * arr[1] * arr[size - 1]; // min * 2nd min * max  
    int choice2 = arr[size - 1] * arr[size - 2] * arr[size - 3]; // max * 2nd max * 3rd max  
  
    if(choice1 > choice2) System.out.println(choice1);  
    else System.out.println(choice2);  
}
```

Trapping Rain Water



```
public int trap(int[] arr) {  
    int n = arr.length;  
  
    int[] prefix = new int[n];   
    prefix[0] = arr[0];  
  
    for(int i=1; i<n; i++){  
        prefix[i] = Math.max(arr[i], prefix[i - 1]);  
    }  
  
    int[] suffix = new int[n];   
    suffix[n - 1] = arr[n - 1];  
  
    for(int i=n-2; i>=0; i--){  
        suffix[i] = Math.max(arr[i], suffix[i + 1]);  
    }  
  
    int water = 0;  
    for(int i=0; i<n; i++){  
        water = water + (Math.min(prefix[i], suffix[i]) - arr[i]);  
    }  
  
    return water;  
}
```

 $\min(\text{leftwall}, \text{rightwall})$



```
public static int peakIndex(int[] arr){  
    for(int idx = 1; idx < arr.length - 1; idx++){  
        if(arr[idx] > arr[idx - 1] && arr[idx] > arr[idx + 1]){  
            return idx;  
        }  
    }  
    return -1;  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
  
    for(int idx=0; idx<arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    System.out.println(peakIndex(arr));  
}
```

Peak Index

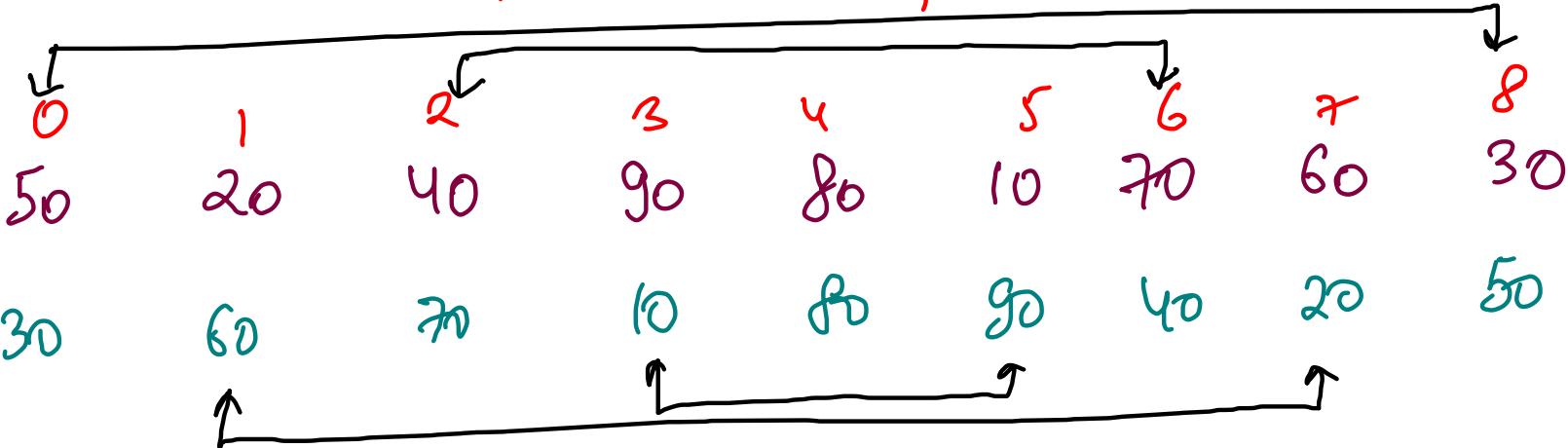
peak Elements

```
public class Solution {  
    public static void peakElements(int[] arr){  
        for(int idx = 1; idx < arr.length - 1; idx++){  
            if(arr[idx] > arr[idx - 1] && arr[idx] > arr[idx + 1]){  
                System.out.print(arr[idx] + " ");  
            }  
        }  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int size = scn.nextInt();  
        int[] arr = new int[size];  
  
        for(int idx=0; idx<arr.length; idx++){  
            arr[idx] = scn.nextInt();  
        }  
  
        peakElements(arr);  
    }  
}
```

Reverse Array

odd

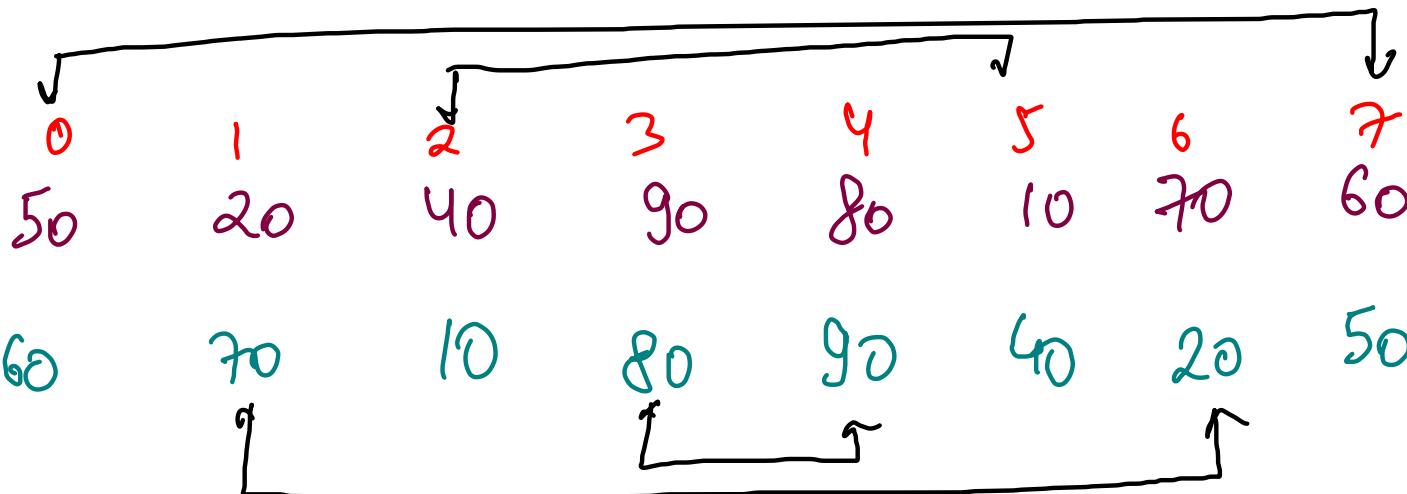
reversed:



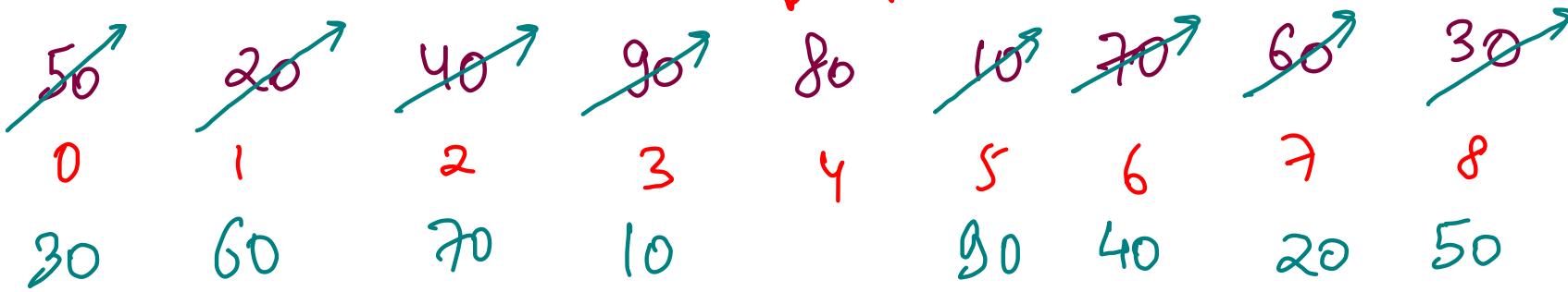
WHY
Intuition

even

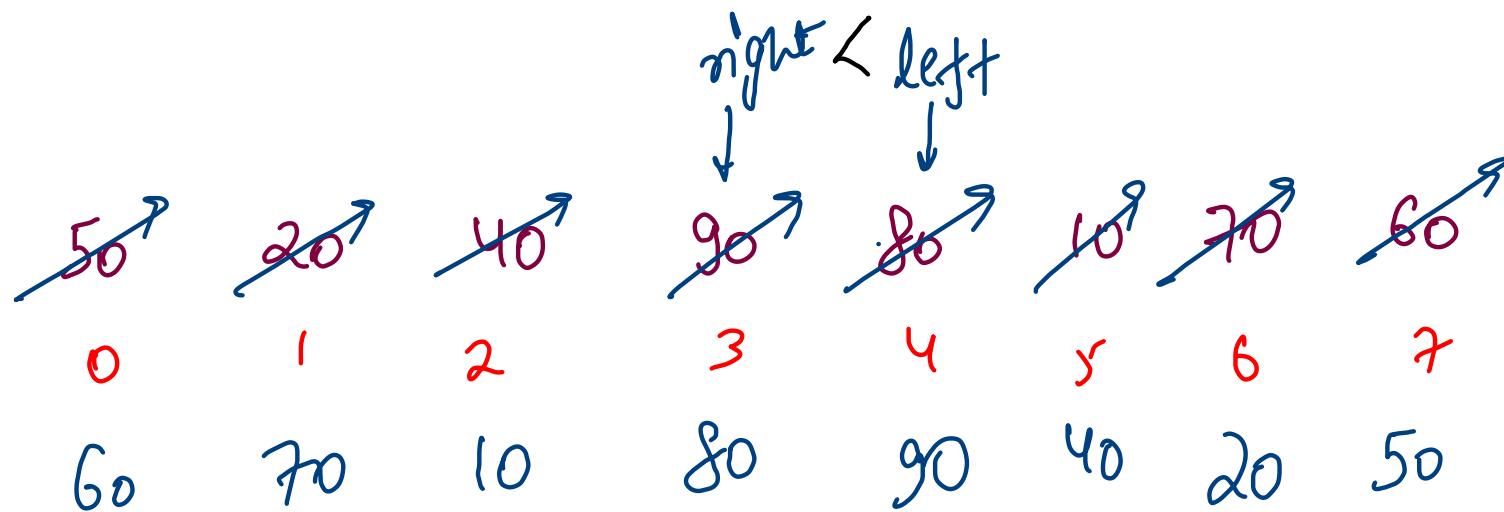
reversed:



How



left = right



right < left

```
import java.io.*;
import java.util.*;

public class Solution {
    public static void reverseArray(int[] arr){
        int left = 0, right = arr.length - 1;

        while(left < right){
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            left++; right--;
        }
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];
        for(int idx=0; idx<arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        reverseArray(arr);
        for(int idx = 0; idx < arr.length; idx++){
            System.out.println(arr[idx]);
        }
    }
}
```

$N/2$ iterations

Theory Related

Memory Mapping of Arrays

- Contiguous Memory Allocatⁿ
- Why java actually datatype
(strictly typed)
- Reference Variable vs
actual object
- Heap (Random Access Memory)
- Changes in stack vs heap variable
(lifetime & scope of variables)

Time & Space Complexity

- BigO, Omega, Theta
Notatⁿ
- Bestcase, Worstcase,
Average case
- Runtime \leq Time Comp
- Types of Space Comp.

Half Ascending Half Descending

Bitonic Array (mountain array)

increasing decreasing

1 2 3 4

1 2 4 3

1 3 4 2

2 3 4 1



1 2 3 4 9 8 7 6 5

eg 2 5 6 3 4 1 8 9 7
1 2 3 4 5 6 7 8 9
↓
↓

3 4 2 1



eg

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

\downarrow sort the array

\downarrow reverse second half of sorted arr

arraylength = 9

left = $n/2$

right = $n-1$

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

eg

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

\downarrow

arraylength = 8

left = $n/2$

right = $n-1$

```
public class Solution {
    public static void bitonicArray(int[] arr){
        Arrays.sort(arr); → Sort the array

        int left = arr.length / 2, right = arr.length - 1;
        while(left < right){
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;

            left++; right--;
        }
    }

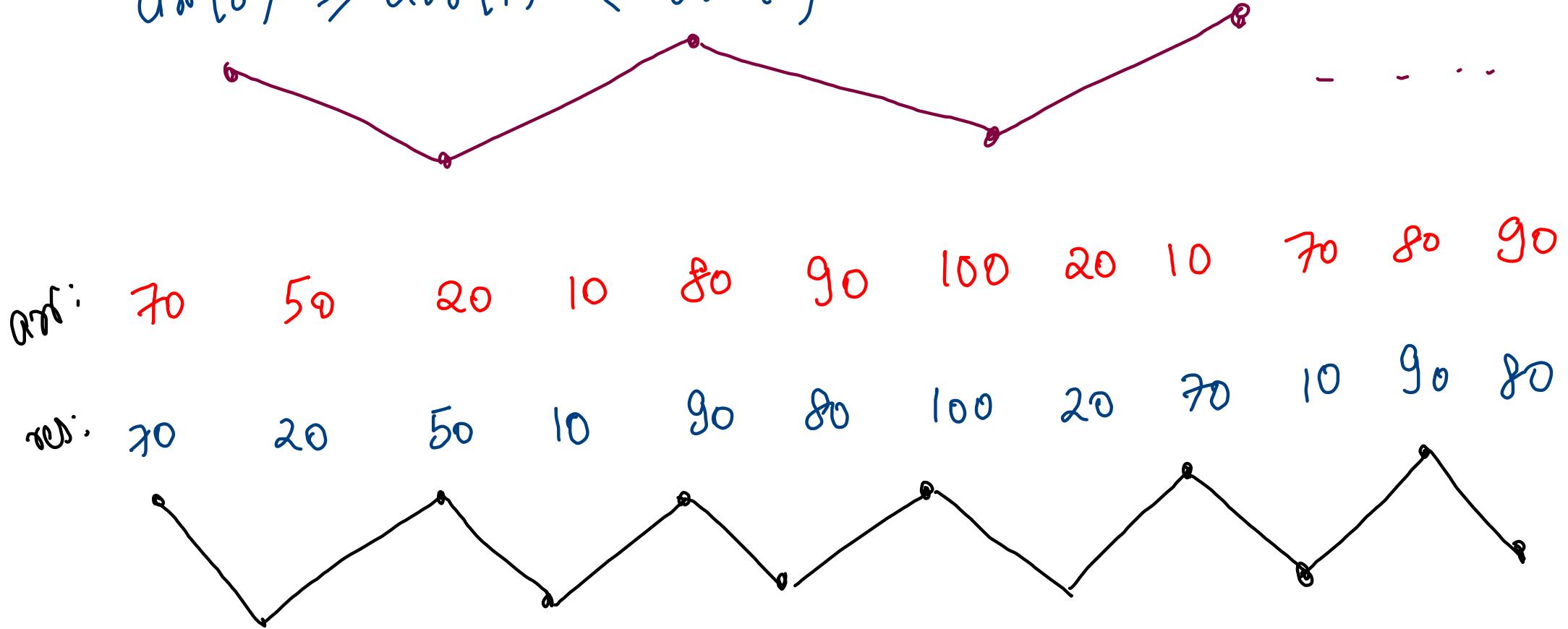
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];
        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

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

} reverse the
second
half.

Wave Sort an Array

$\text{arr}[0] \geq \text{arr}[1] \leq \text{arr}[2] \geq \text{arr}[3] \leq \text{arr}[4] \dots$

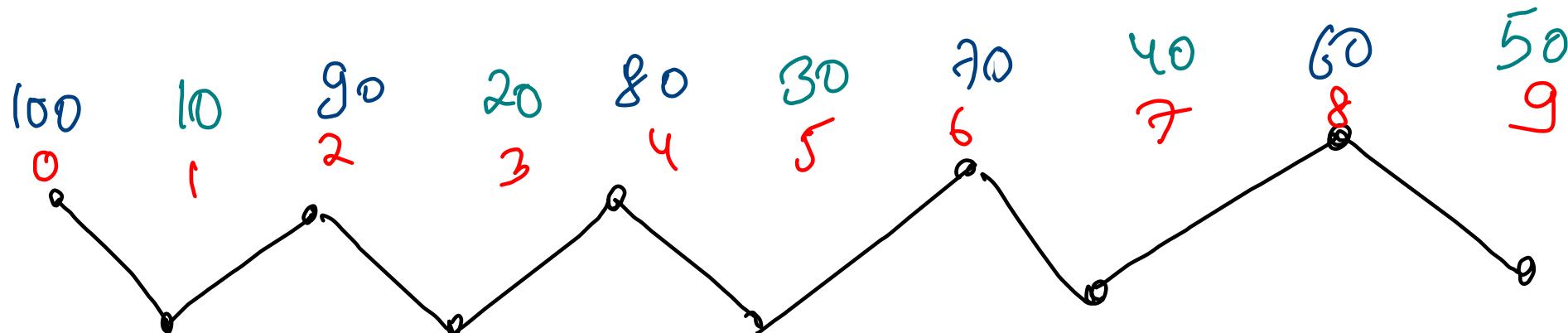


① first Array

② Take alternate highest, lowest!

right left

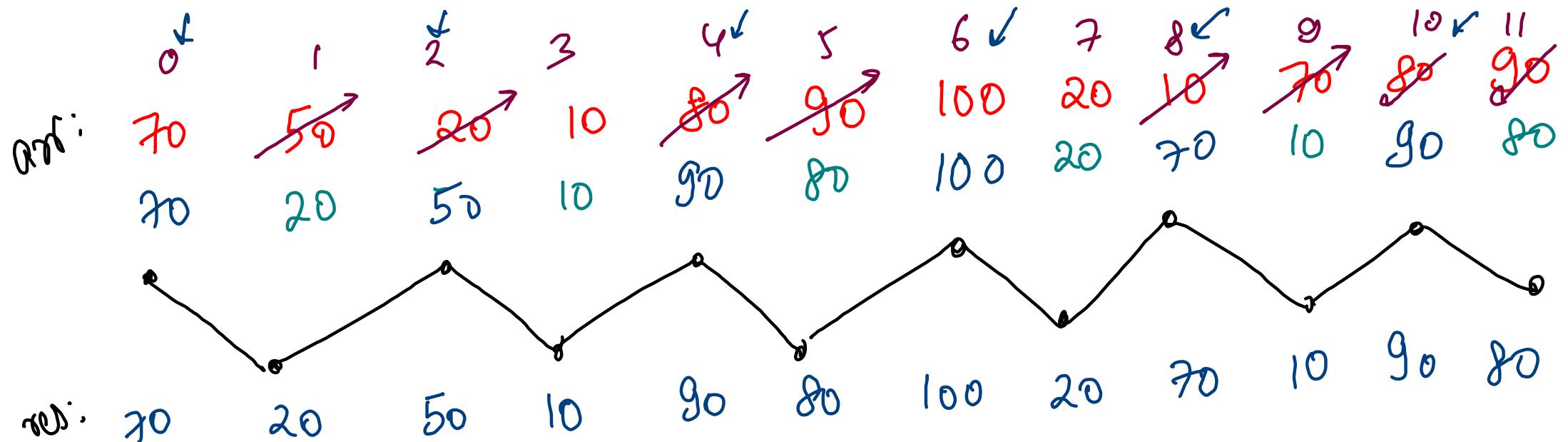
| | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|-----|
| eg | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| idx | ↓ | ↓ | ↓ | ↓ | ↓ | ↑ | ↑ | ↓ | ↑ | ↓ |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 100 | 10 | 90 | 20 | 80 | 30 | 70 | 40 | 60 | 50 | 0 |



```
public static int[] waveSort(int[] arr){  
    Arrays.sort(arr);  
  
    int left = 0, right = arr.length - 1;  
  
    int[] res = new int[arr.length];  
    for(int idx = 0; idx < arr.length; idx++){  
        if(idx % 2 == 0){  
            res[idx] = arr[right];  
            right--;  
        } else {  
            res[idx] = arr[left];  
            left++;  
        }  
    }  
  
    return res;  
}
```

Approach

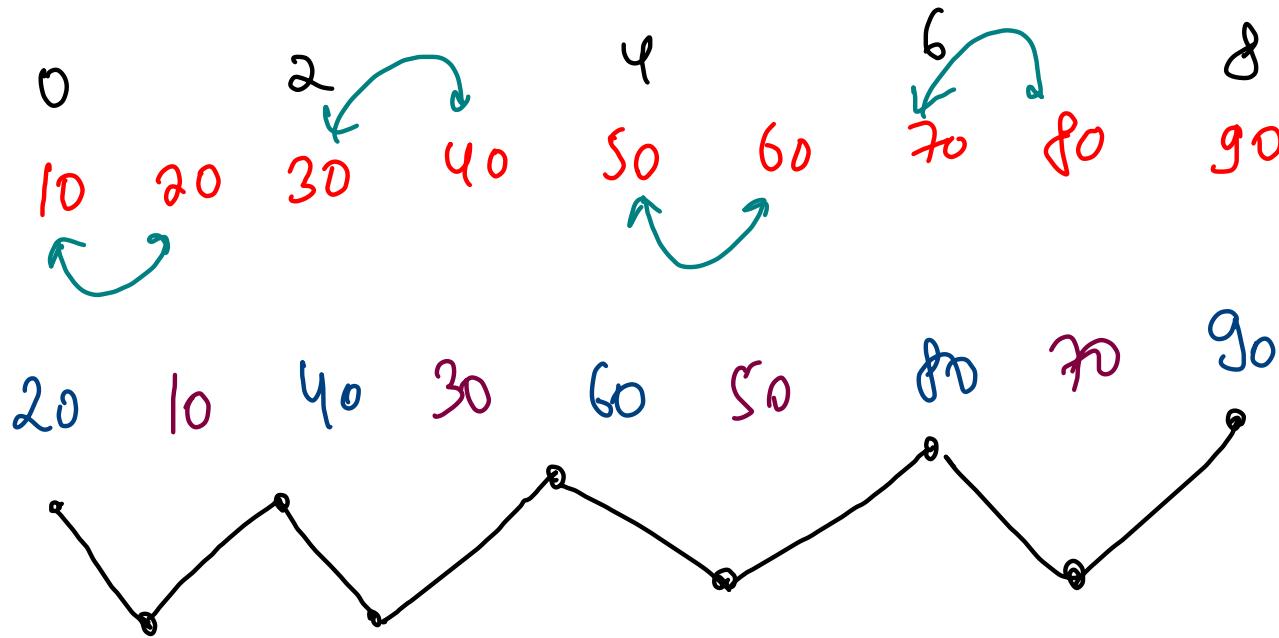
- ① Sort Array
- ② Take highest-lowest alternatively.



even indices \rightarrow peak

odd indices \rightarrow valley

- ① Traverse on even indices (peak indices)
- ② Swap with $\max(\text{arr}[i-1], \text{arr}[i])$



Approach 3

heicographically
smallest-wave sorted
array.

① Sort the array

② Swap every i with $i+1$ for even indices

Approach 3 Code

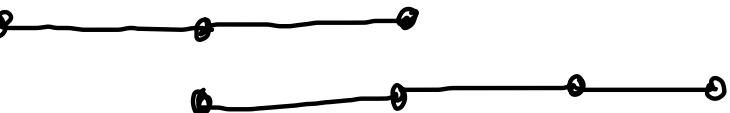
```
public static void waveSort(int[] arr){  
    Arrays.sort(arr);  
    for(int i = 0; i + 1 < arr.length; i = i + 2){  
        int temp = arr[i];  
        arr[i] = arr[i + 1];  
        arr[i + 1] = temp;  
    }  
  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int size = scn.nextInt();  
        int[] arr = new int[size];  
        for(int idx = 0; idx < arr.length; idx++){  
            arr[idx] = scn.nextInt();  
        }  
  
        waveSort(arr);  
        for(int idx = 0; idx < arr.length; idx++){  
            System.out.print(arr[idx] + " ");  
        }  
    }  
}
```

Elite Batch (advanced dsa)

- (A) after 1-2 months
- (B) 2-3 batches combined
⇒ placed at S LPA
- (C) Duration (10 hours weekly)
- (d) If I am available, I can take up the batch
- (e) Leetcode / Geeks / Code studios
(300+)

Point All Subarrays

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



contiguous part of an array
in the same order as in the array

1 2 3
20, 30, 40 ✓

2 3 4 5

30, 40, 50, 60 ✓

20, 40, 60 ✗ (non-contiguous)

10 50 60 ✗ (non-contiguous)

3 4 0 ✓ (only one element)

0 1 2 3 4 5
10 20 30 40 50 60 ✓
(entire array)

3 2 1
40, 30, 20 ✗ (diff order)

4 2 3
50, 30, 40 ✗ (diff order)

$$6 + 5 + 4 + 3 + 2 + 1 = \frac{n * (n+1)}{2} = \frac{6 * 7}{2} = 21$$

3 loops

Subarray from 10

${}^0 10^0$

${}^0 10, 20$

${}^0 10, 20, 30$

${}^0 10, 20, 30, 40$

${}^0 10, 20, 30, 40, 50$

${}^0 10, 20, 30, 40, 50, 60$

⑥

Subarrays from 20

${}^1 20^1$

${}^1 20, 30^2$

${}^1 20, 30, 40^3$

${}^1 20, 30, 40, 50^4$

${}^1 20, 30, 40, 50, 60^5$

⑤

Subarray from 30

${}^2 30^2$

${}^2 30, 40^3$

${}^2 30, 40, 50^4$

${}^2 30, 40, 50, 60^5$

④

Subarray from 40

${}^3 40^3$

${}^3 40, 50^4$

②

Subarray from 50

${}^3 50^3$

${}^3 50, 60^4$

${}^3 40, 50, 60^5$

③

Subarray from 60

${}^5 60^5$

①

Print all $(arr(i), arr(j))$
 $i \leq j$

| | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|---------|
| 0 | 1 | 2 | 3 | 4 | 5 | |
| 10 | 20 | 30 | 40 | 50 | 60 | |
| 0 0 $(10, 10)$ | 1 1 $(20, 20)$ | 2 2 $(30, 30)$ | 3 3 $(40, 40)$ | 4 4 $(50, 50)$ | 5 5 $(60, 60)$ | |
| 0 1 $(10, 20)$ | 1 2 $(20, 30)$ | 2 3 $(30, 40)$ | 3 4 $(40, 50)$ | 4 5 $(50, 60)$ | 5 6 $(60, 60)$ | |
| 0 2 $(10, 30)$ | 1 3 $(20, 40)$ | 2 4 $(30, 50)$ | 3 5 $(40, 60)$ | | | 2 loops |
| 0 3 $(10, 40)$ | 1 4 $(20, 50)$ | 2 5 $(30, 60)$ | | | | |
| 0 4 $(10, 50)$ | 1 5 $(20, 60)$ | | | | | |
| 0 5 $(10, 60)$ | | | | | | |

```

public static void printSubarrays(int[] arr){
    for(int st = 0; st < arr.length; st++){
        for(int end = st; end < arr.length; end++){
            for(int idx = st ; idx <= end; idx++){
                System.out.print(arr[idx] + " ");
            }
            System.out.println();
        }
    }
}

```

0 1 2 3 4
 10 20 30 40 50

0 0 \Rightarrow 10
 0 1 \Rightarrow 10 20
 0 2 \Rightarrow 10 20 30
 0 3 \Rightarrow 10 20 30 40
 0 4 \Rightarrow 10 20 30 40 50
 st end

1 1 \Rightarrow 20
 1 2 \Rightarrow 20, 30
 1 3 \Rightarrow 20, 30, 40
 1 4 \Rightarrow 20, 30, 40, 50

st end

2 2 \Rightarrow 30
 2 3 \Rightarrow 30, 40
 2 4 \Rightarrow 30, 40, 50

3 3 \Rightarrow 40
 3 4 \Rightarrow 40, 50

4 4 \Rightarrow 50

Point all subarrays sum

0 1 2 3 4
10 20 30 40 50

0 0 $\Rightarrow 10 = 10$
0 1 $\Rightarrow 10 + 20 = 30$
0 2 $\Rightarrow 10 + 20 + 30 = 60$
0 3 $\Rightarrow 10 + 20 + 30 + 40 = 100$
0 4 $\Rightarrow 10 + 20 + 30 + 40 + 50 = 150$
at end

1 1 $\Rightarrow 20 = 20$
1 2 $\Rightarrow 20 + 30 = 50$
1 3 $\Rightarrow 20 + 30 + 40 = 90$
1 4 $\Rightarrow 20 + 30 + 40 + 50 = 140$

at end

2 2 $\Rightarrow 30 = 30$
2 3 $\Rightarrow 30 + 40 = 70$
2 4 $\Rightarrow 30 + 40 + 50 = 120$
3 3 $\Rightarrow 40 = 40$
3 4 $\Rightarrow 40 + 50 = 90$

4 4 $\Rightarrow 50 = 50$

```
public static void printSubarrays(int[] arr){  
    for(int st = 0; st < arr.length; st++){  
        for(int end = st; end < arr.length; end++){  
            int sum = 0;  
            for(int idx = st ; idx <= end; idx++){  
                sum = sum + arr[idx];  
            }  
            System.out.println(sum);  
        }  
    }  
}
```

Approach ①

Naive approach

or

Brute force approach

```
public static void printSubarrays(int[] arr){  
    for(int st = 0; st < arr.length; st++){  
        int sum = 0;  
        for(int end = st; end < arr.length; end++){  
            sum = sum + arr[end];  
            // new subarray sum = previous subarray sum + last element  
            System.out.println(sum);  
        }  
    }  
}
```

Approach ②

Optimized approach

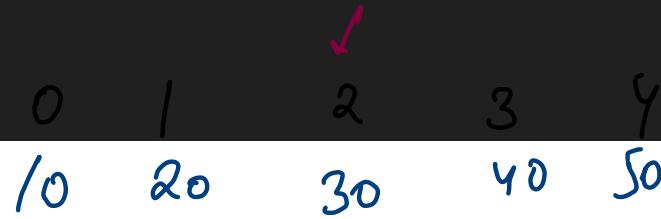
(Dynamic Programming)

prefix sum approach

```

public static void printSubarrays(int[] arr){
    for(int st = 0; st < arr.length; st++){
        int sum = 0;
        for(int end = st; end < arr.length; end++){
            sum = sum + arr[end];
            // new subarray sum = previous subarray sum + last element
            System.out.println(sum);
        }
    }
}

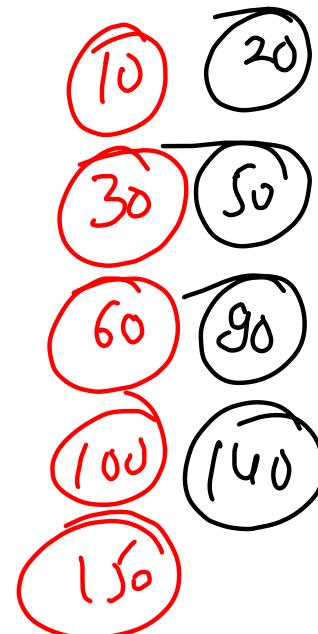
```



$st=0$

$$\text{Sum} = 0 + 10 + 20 + 30 + 40 + 50$$

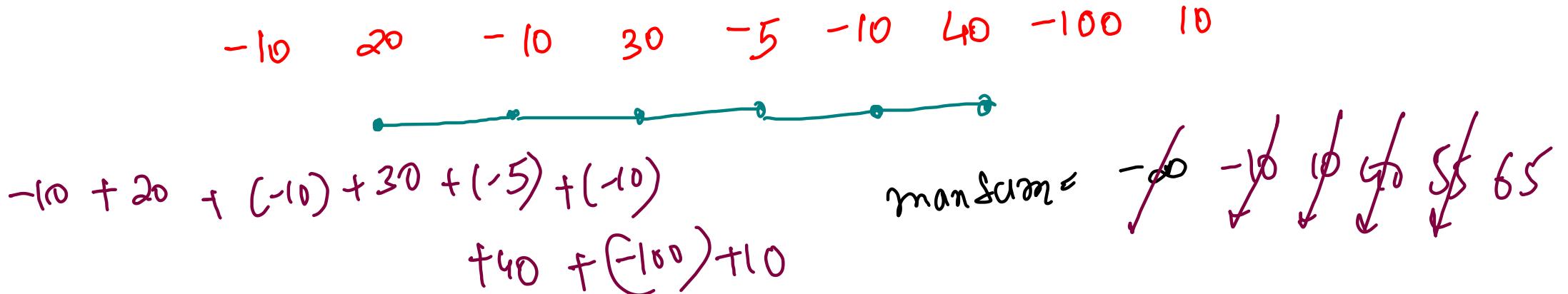
$$0 + 20 + 30 + 40 + 50 = 140$$



```
public static boolean targetSumSubarray(int[] arr, int target){  
    for(int st = 0; st < arr.length; st++){  
        int sum = 0;  
        for(int end = st; end < arr.length; end++){  
            sum = sum + arr[end];  
            if(sum == target) return true;  
        }  
    }  
    return false;  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    System.out.println(targetSumSubarray(arr, 0));  
}
```

Check subarray
sum equal
to target = 0

Maximum sum subarray

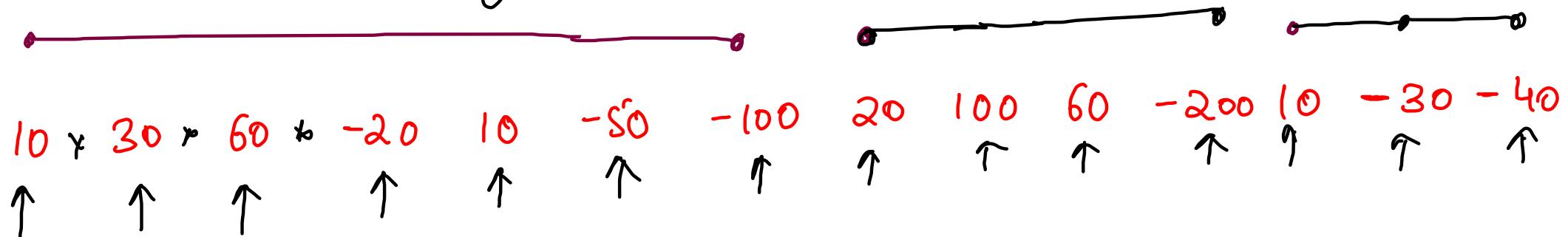


$$20 + (-10) + 30 + (-5) + (-10) + 40$$

```
public static int maxSumSubarray(int[] arr){  
    int maxSum = Integer.MIN_VALUE;  
    for(int st = 0; st < arr.length; st++){  
        int sum = 0;  
        for(int end = st; end < arr.length; end++){  
            sum = sum + arr[end];  
            maxSum = Math.max(maxSum, sum);  
        }  
    }  
    return maxSum;  
}
```

Iterations $\cancel{n \times n}$

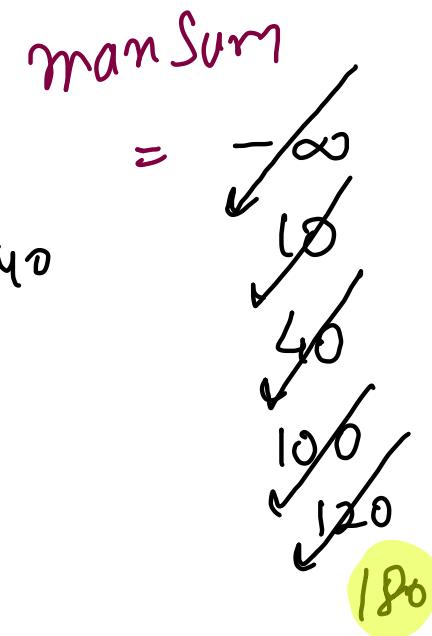
Kadane's Algorithm (Dynamic Programming)



$$\begin{aligned}
 \text{sum} &= 0 + (10 + 30 + 60 + (-20) + 10 + (-50)) \\
 &\quad + (-100) = -60 \\
 &= 20 + 100 + 60 + (-200) \\
 &= 10 + (-30) = -40
 \end{aligned}$$

option 1: \rightarrow join the previous subarray

option 2: \rightarrow start a new subarray



```
public static int maxSumSubarray(int[] arr){  
    int maxSum = Integer.MIN_VALUE, sum = 0;
```

```
    for(int idx = 0; idx < arr.length; idx++) {  
        sum = Math.max(arr[idx], sum + arr[idx]);  
        maxSum = Math.max(maxSum, sum);  
    }
```

```
    return maxSum;
```

```
}
```

start a new subarray
join the previous subarray
 $\rightarrow n$ iterations

Max^m Product Subarray

```
public static int maxProdSubarray(int[] arr){  
    int maxProduct = Integer.MIN_VALUE;  
    for(int st = 0; st < arr.length; st++){  
        int prod = 1;  
        for(int end = st; end < arr.length; end++){  
            prod = prod * arr[end];  
            maxProduct = Math.max(maxProduct, prod);  
        }  
    }  
    return maxProduct;  
}
```

Rotate Array by K

Reduce k value

$$k \% n = 0 \quad \lceil n + 0 \rceil \quad k = 0, 7, 14, 21$$

$$k \% n = 1 \quad \lceil n + 1 \rceil \quad k = 1, 8, 15, 22$$

$$k \% n = 2 \quad \lceil n + 2 \rceil \quad k = 2, 9, 16, 23$$

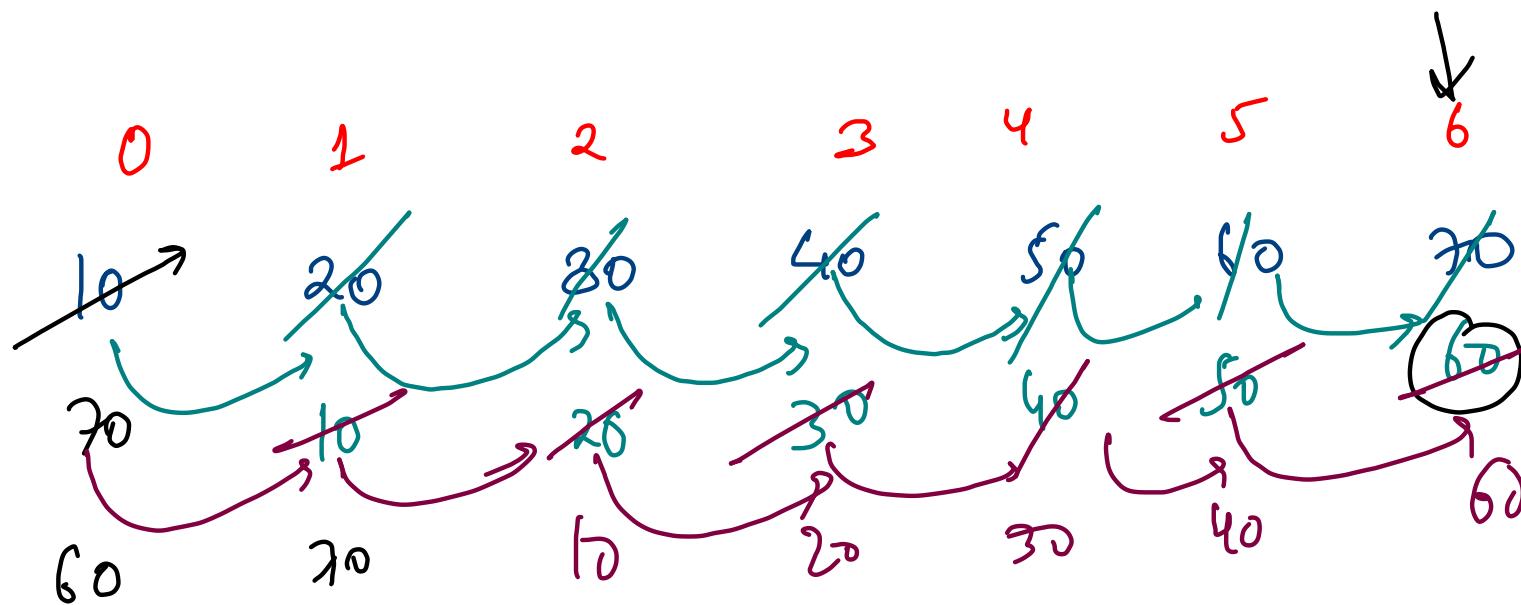
$$k \% n = 3 \quad \lceil n + 3 \rceil \quad k = 3, 10, 17, 24$$

$$k \% n = 4 \quad \lceil n + 4 \rceil \quad k = 4, 11, 18, 25$$

$$k \% n = 5 \quad \lceil n + 5 \rceil \quad k = 5, 12, 19, 26$$

$$k \% n = 6 \quad \lceil n + 6 \rceil \quad k = 6, 13, 20, 27$$

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----|----|----|----|----|----|----|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
| 70 | 10 | 20 | 30 | 40 | 50 | 60 |
| 60 | 20 | 10 | 20 | 30 | 40 | 50 |
| 50 | 60 | 20 | 10 | 20 | 30 | 40 |
| 40 | 50 | 60 | 70 | 10 | 20 | 30 |
| 30 | 40 | 50 | 60 | 70 | 10 | 20 |
| 20 | 30 | 40 | 50 | 60 | 70 | 10 |



$$\text{temp} = a \times (n-1) \\ = 70$$

$$\text{temp} = a \times (n-1) \\ = 60$$

```

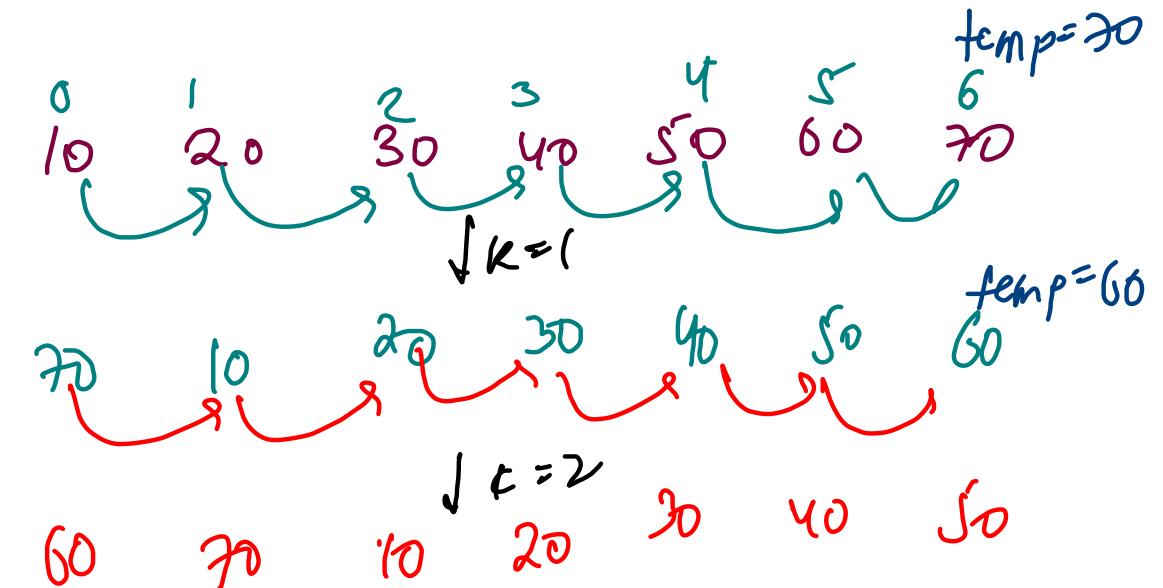
public class Solution {
    // Rotate by 1 Place
    public static void rotate(int[] arr){
        int n = arr.length;
        int temp = arr[n - 1];
        for(int i = n - 1; i > 0; i--){
            arr[i] = arr[i - 1];
        }
        arr[0] = temp;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];
        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        int k = scn.nextInt();
        k = k % arr.length; reduce no of rotations
        for(int i=0; i<k; i++){ } k=n
            rotate(arr);

        for(int i=0; i<size; i++){
            System.out.print(arr[i] + " ");
        }
    }
}

```



$$k = 100$$

$$100 \% \div 7 = 2$$

```

public class Solution {

    // Rotate by 1 Place
    public static void rotate(int[] arr){
        int n = arr.length;
        int temp = arr[n - 1];
        for(int i = n - 1; i > 0; i--){
            arr[i] = arr[i - 1];
        }

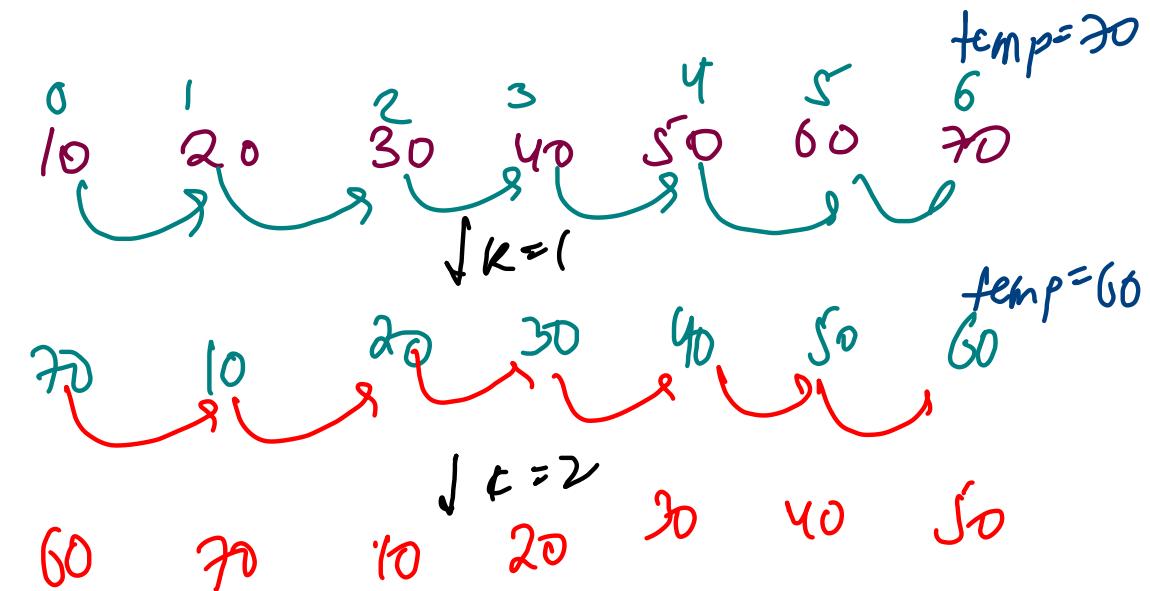
        arr[0] = temp;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];
        for(int idx = 0; idx < arr.length; idx++){
            arr[idx] = scn.nextInt();
        }

        int k = scn.nextInt();
        → reduce no of rotations
        for(int i=0; i<k; i++){
            rotate(arr);
        }

        for(int i=0; i<size; i++){
            System.out.print(arr[i] + " ");
        }
    }
}

```



~~$O(K \cdot n)$~~

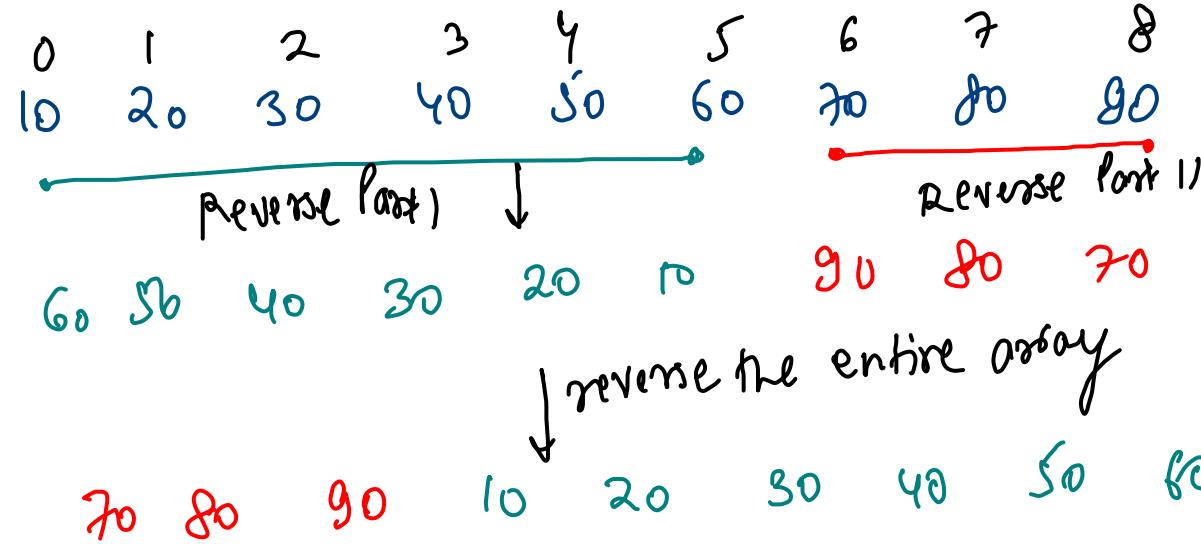
$k = 100$

$100 \% 7 = 2$

in worst case

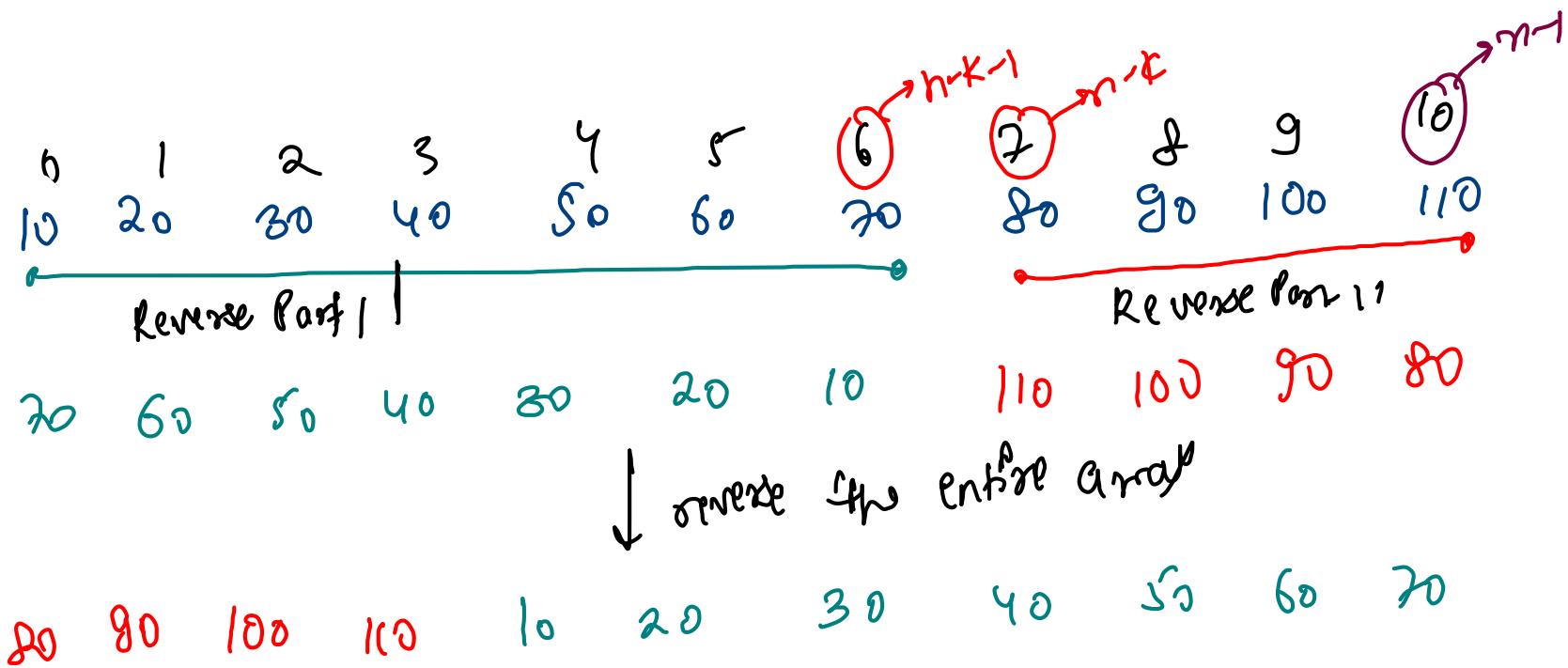
~~$O(n \cdot n)$~~

Eg1
 $k=100 \text{ kg}$
 ≈ 1



$k=3$ rotations

Eg2
 $k=100$
 $(100 \cdot 11)$
 ≈ 7



$k=4$ rotations

Mathematical
Proof

$$A = A_{\beta_1} + A_{\beta_2}$$
$$\left(\left(A_{\beta_1} \right)^T + \left(A_{\beta_2} \right)^T \right)^T$$
$$= A_{\beta_2} + A_{\beta_1}$$

$$reg = A_{\beta_2} + A_{\beta_1}$$

① reverse (arr, 0, $n-k-1$)

② reverse (arr, $n-k$, $n-1$)

③ reverse (arr, 0, $n-1$)

```

public static void reverse(int[] arr, int left, int right){
    while(left < right){
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;
        left++; right--;
    }
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];

    for(int idx = 0; idx < n; idx++){
        arr[idx] = scn.nextInt();
    }

    int k = scn.nextInt();
    k = k % n;

    reverse(arr, 0, n - k - 1); // Reverse Elements not to be
rotated
    reverse(arr, n - k, n - 1); // Reverse Elements which are
to be rotated
    reverse(arr, 0, n - 1); // Reverse the entire array

    for(int idx = 0; idx < n; idx++){
        System.out.print(arr[idx] + " ");
    }
}

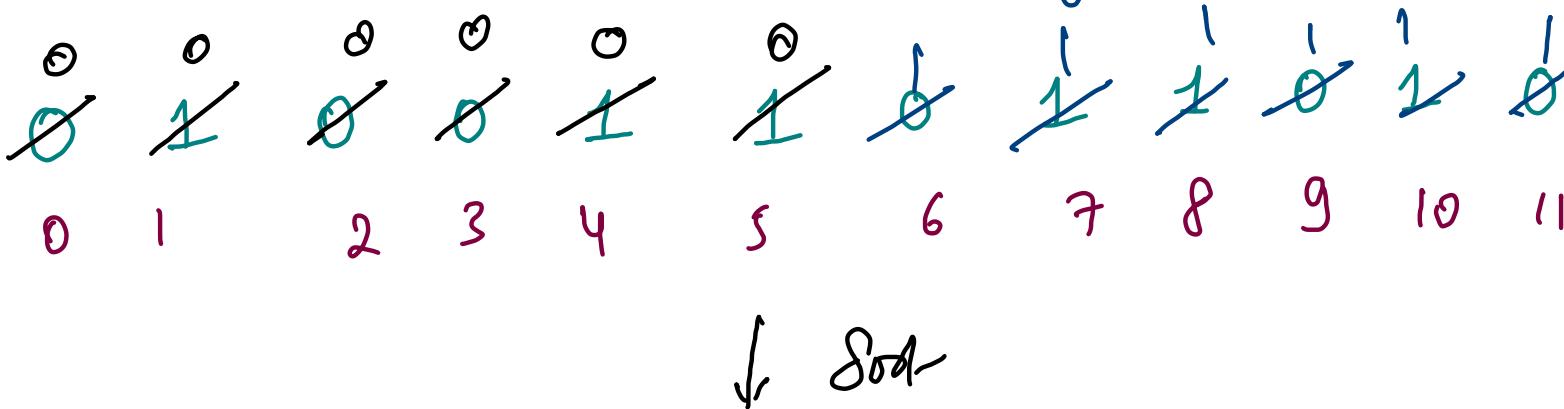
```

n iterations

$n+n+n$
 $\leq 3 \times n$ iterations

Quicksort

Sort Binary Array



0 0 0 0 0 0 1 1 1 1 1 1

- ① BubbleSort / InsertionSort / SelectionSort → $(N-1)$ pass algorithm
- ② Counting + placing → Two pass algorithm
- ③ Partition logic → One pass algorithm

```

import java.io.*;
import java.util.*;

public class Solution {
    public static void sortBinary(int[] arr){
        int countOfZeroes = 0;
        for(int idx = 0; idx < arr.length; idx++){
            if(arr[idx] == 0) countOfZeroes++;
        }

        for(int idx = 0; idx < arr.length; idx++){
            if(idx < countOfZeroes) arr[idx] = 0;
            else arr[idx] = 1;
        }
    }

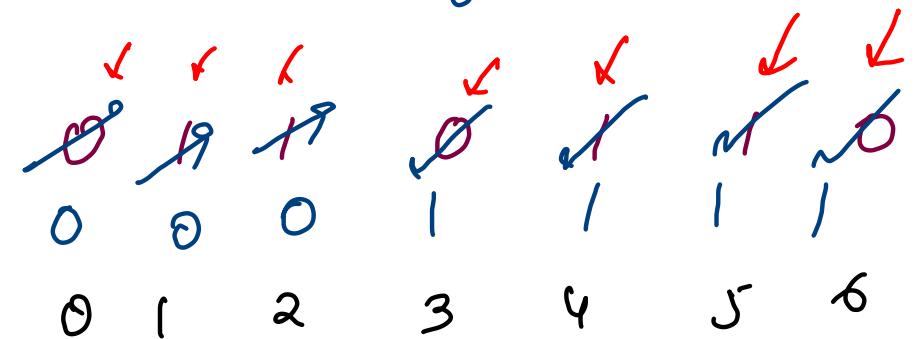
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int size = scn.nextInt();
        int[] arr = new int[size];

        for(int idx = 0; idx < size; idx++){
            arr[idx] = scn.nextInt();
        }

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

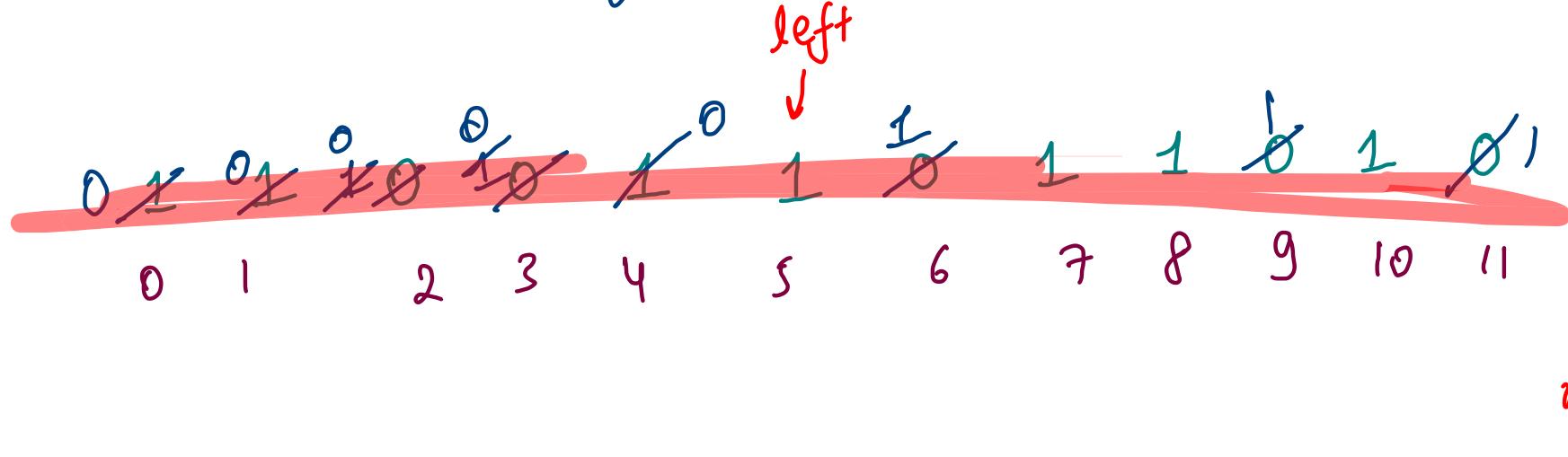
```

Two pass algorithm



$$\text{count of } 0s = \cancel{0} \cancel{1} \cancel{2} \cancel{3}$$

Partitioning Logic (Two pointers)



```
while(r < size)
    if(a[r] == 1)
        r++
    else
        swap(l, r)
        l++; r++;
```

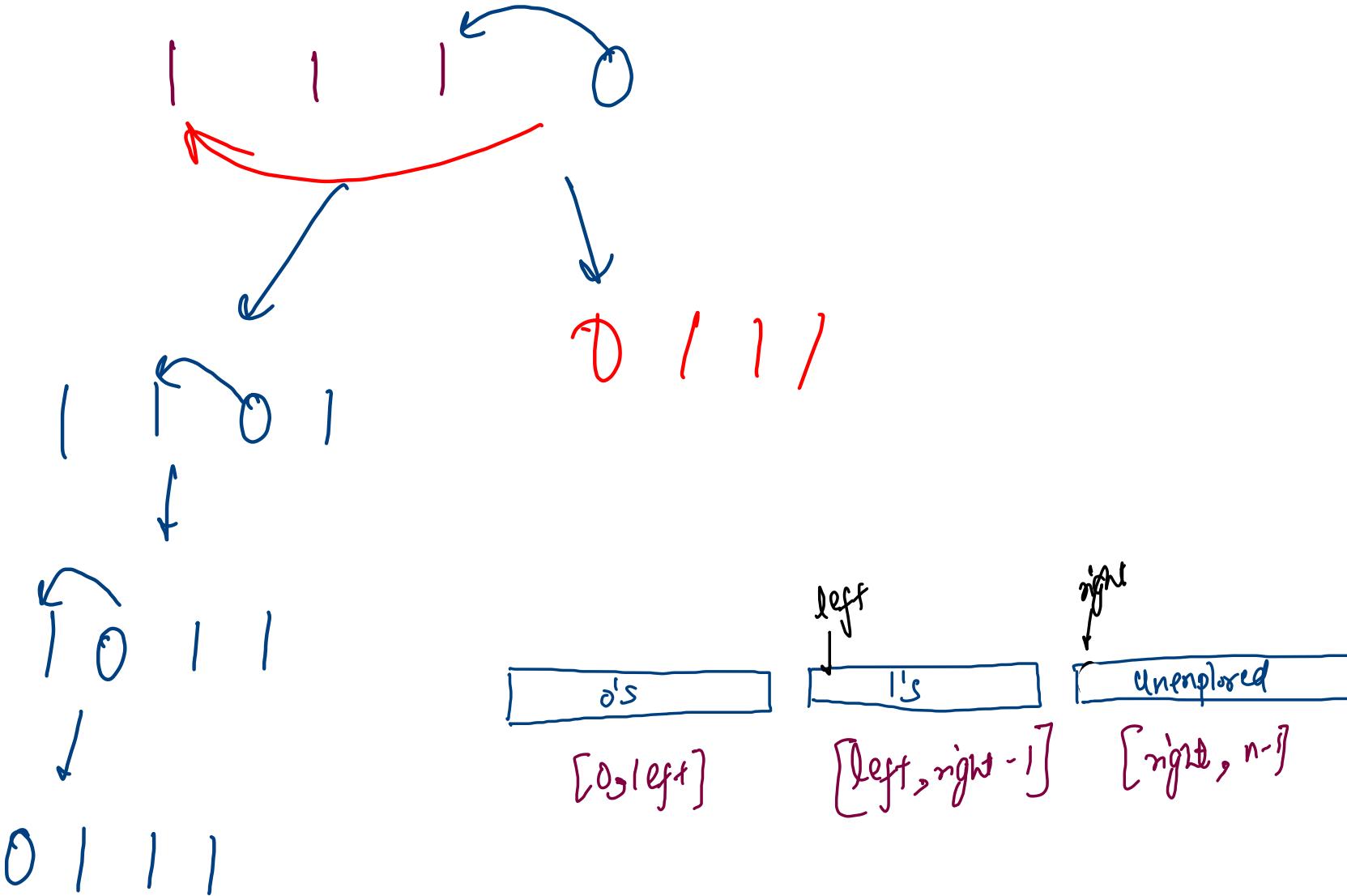
unexplored

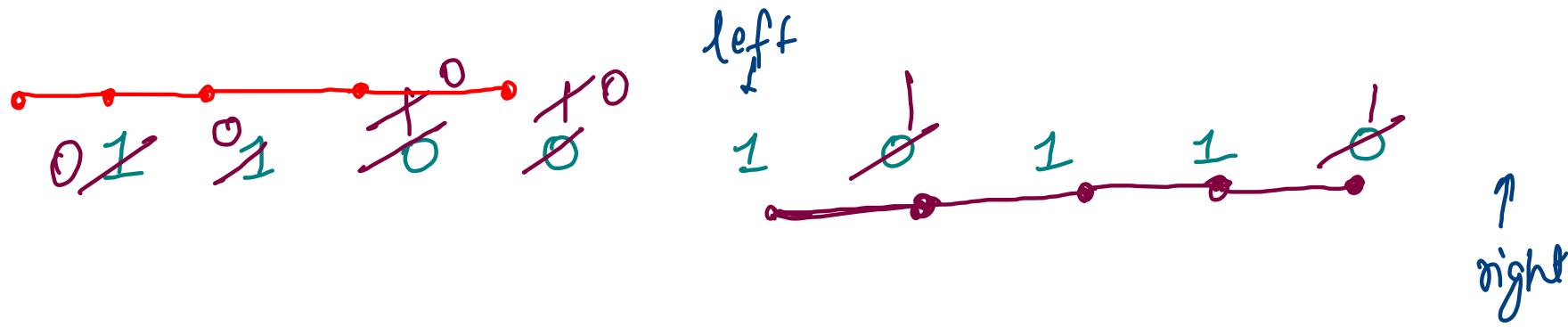
left (as)



right (ls)







```

public static void sortBinary(int[] arr){
    int left = 0, right = 0;
    while(right < arr.length){
        if(arr[right] == 1){
            right++;
        } else {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++; right++;
        }
    }
}

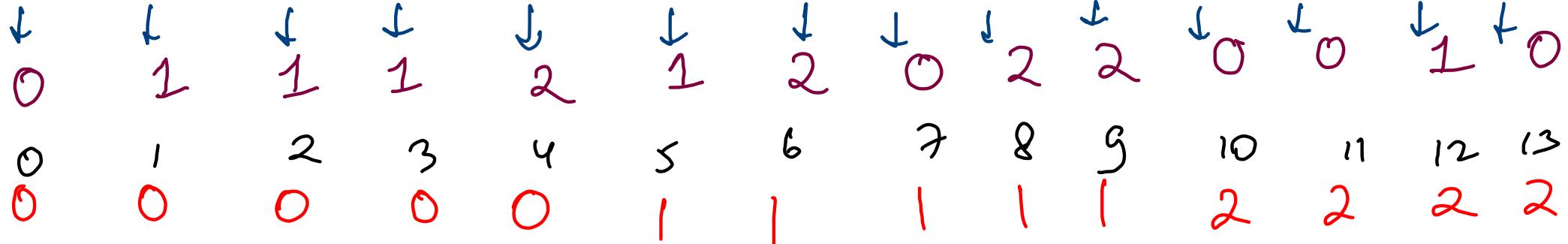
```

One pass algorithm

Sort 012

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| 0 | 1 | 1 | 1 | 2 | 1 | 2 | 0 | 2 | 2 | 0 | 0 | 1 | 0 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |

- ① Sort Array \rightarrow Insert/Select/Bubble $\Rightarrow n \times n \Rightarrow (n-i)$ pass algorithm
- ② Counted Technique $\Rightarrow 2 \times n \Rightarrow$ Two pass algorithm
- ③ Dutch National Flag Algorithm \Rightarrow One pass algorithm



```

public static void sort012(int[] arr){
    int count0 = 0, count1 = 0, count2 = 0;
    for(int idx = 0; idx < arr.length; idx++){
        if(arr[idx] == 0) count0++;
        else if(arr[idx] == 1) count1++;
        else count2++;
    }

    int idx = 0;
    for(int count = 0; count < count0; count++){
        arr[idx] = 0;
        idx++;
    }

    for(int count = 0; count < count1; count++){
        arr[idx] = 1;
        idx++;
    }

    for(int count = 0; count < count2; count++){
        arr[idx] = 2;
        idx++;
    }
}

```

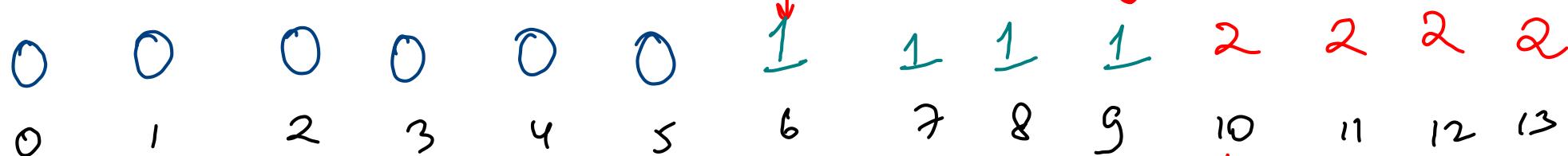
$$\text{count0} = \emptyset \times \emptyset \times \emptyset \times \emptyset \times \emptyset$$

$$\text{count1} = \emptyset \times \emptyset \times \emptyset \times \emptyset \times \emptyset$$

$$\text{count2} = \emptyset \times \emptyset \times \emptyset \times \emptyset \times \emptyset$$

idx = 0 \times 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10 \times 11 \times 12 \times 13
 13 | 4

sort 012 → One pass algorithms

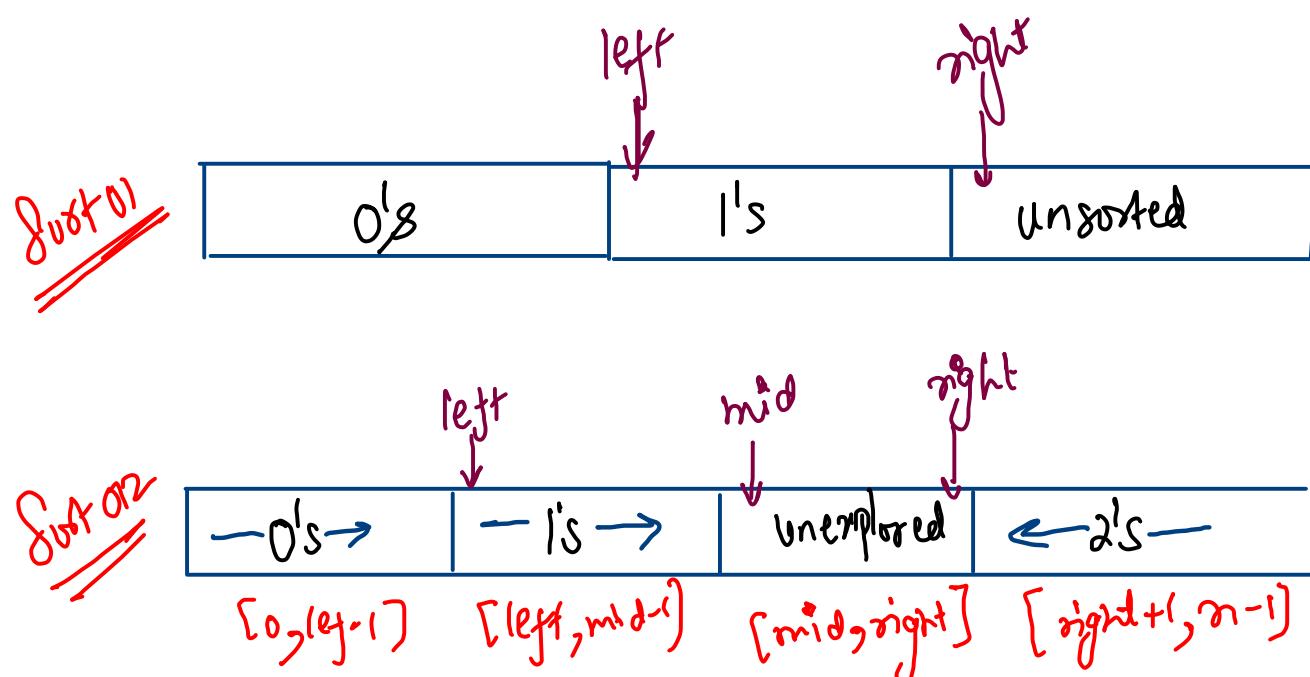


```

public static void swap(int[] arr, int p1, int p2){
    int temp = arr[p1];
    arr[p1] = arr[p2];
    arr[p2] = temp;
}

public static void sort012(int[] arr){
    int left = 0, mid = 0, right = arr.length - 1;
    while(mid <= right){
        if(arr[mid] == 0){
            swap(arr, left, mid);
            left++; mid++;
        } else if(arr[mid] == 1){
            mid++;
        } else {
            swap(arr, mid, right);
            right--;
        }
    }
}

```



```
-----  
public static void swap(int[] arr, int p1, int p2){  
    int temp = arr[p1];  
    arr[p1] = arr[p2];  
    arr[p2] = temp;  
}  
  
public static void sort012(int[] arr){  
    int left = 0, mid = 0, right = arr.length - 1;  
  
    while(mid <= right){  
        if(arr[mid] == 0){  
            swap(arr, left, mid);  
            left++; mid++;  
        } else if(arr[mid] == 1){  
            mid++;  
        } else {  
            swap(arr, mid, right);  
            right--;  
        }  
    }  
}
```

Dutch National Flag

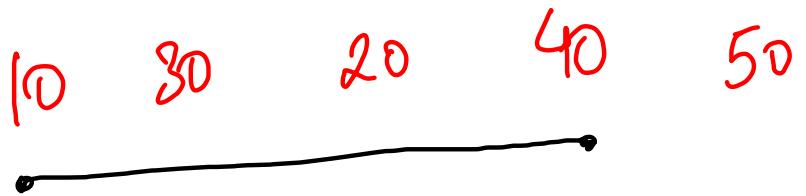
Algorithm



sort 012

Sort 01

partitioning logic in quick sort



left

60

90

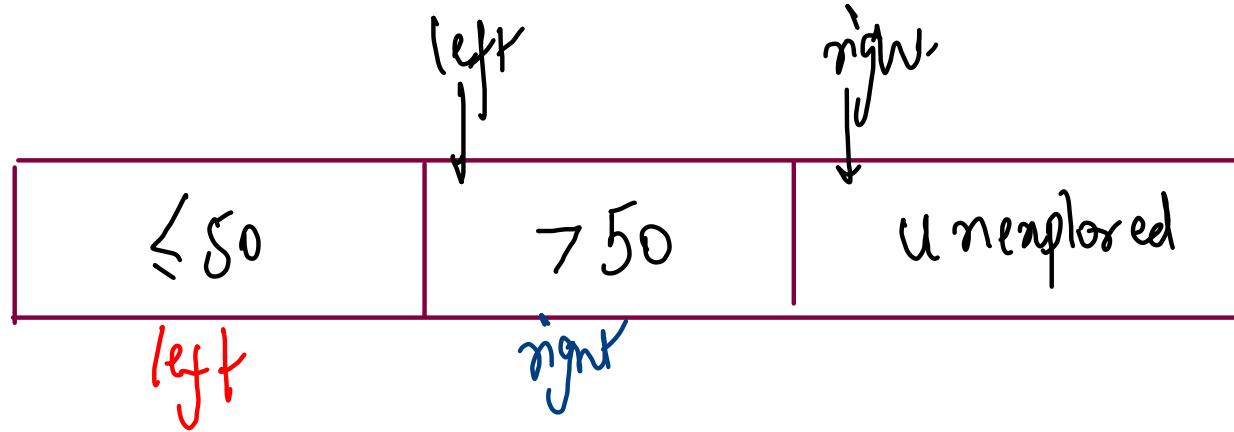
80

100

70

pivot = 50

↑
right



$\leq 50 \Rightarrow$ left
 $> 50 \Rightarrow$ right

```

public static void partition(int[] arr, int pivot){
    int left = 0, right = 0;
    while(right < arr.length){
        if(arr[right] > pivot){
            right++;
        } else {
            swap(arr, right, left);
            left++; right++;
        }
    }
}

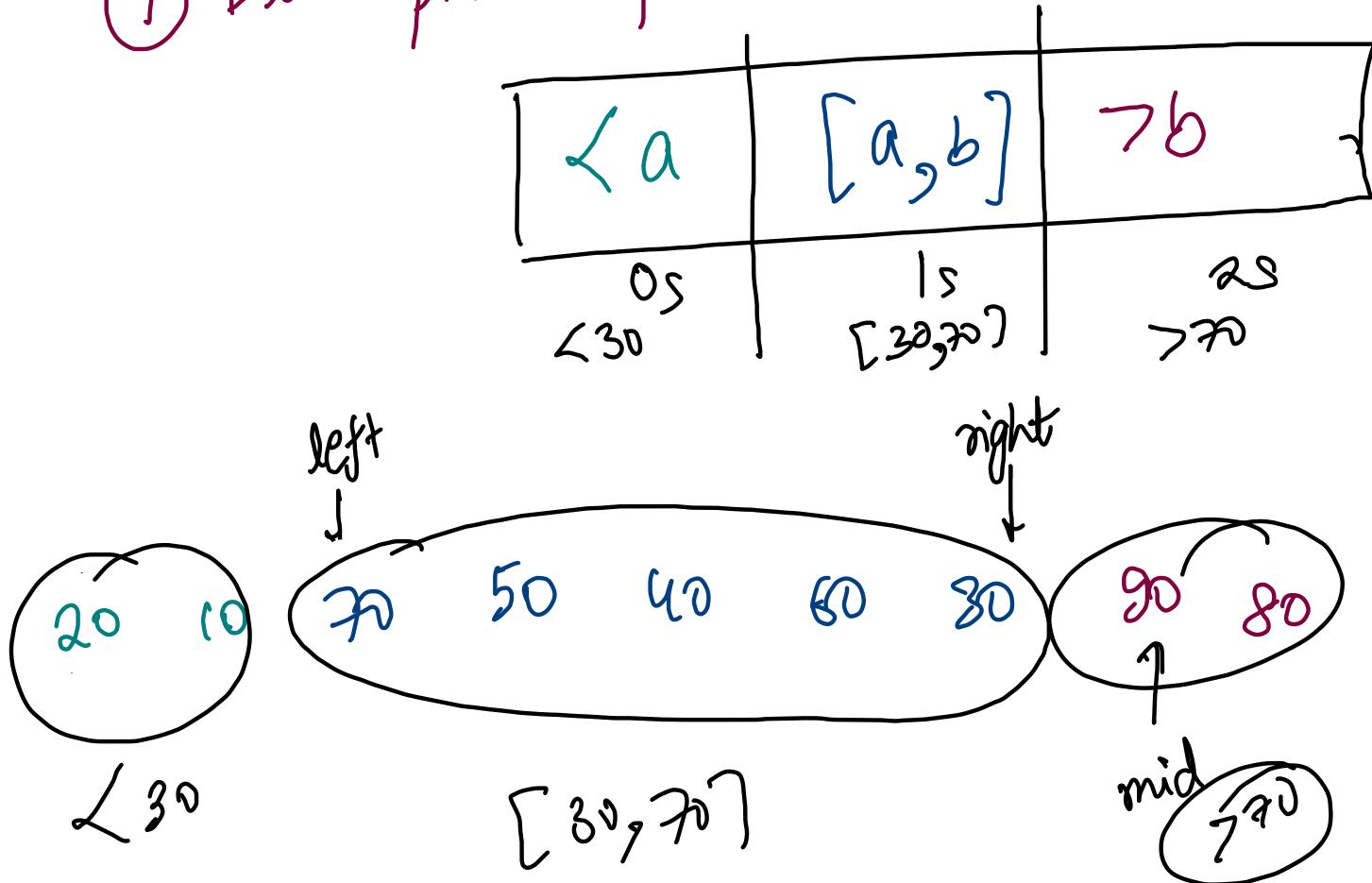
```

- ① Segregate 0's & Non'zen's
- ② Segregate +ve & -ves ($pivot = 0$)
- ③ Segregate Odd & even
- ④ Segregate 2 colors (Black & white)

- ⑤ Segregate X and Y
- ⑥ Segregate vowels & consonants
- ⑦ Segregate prime & composite

Foot 012 Variations

① Dual pivot partitioning



$a \leq b$
 Two pivots $\Rightarrow (a, b)$
 \downarrow \downarrow
 30 70

```

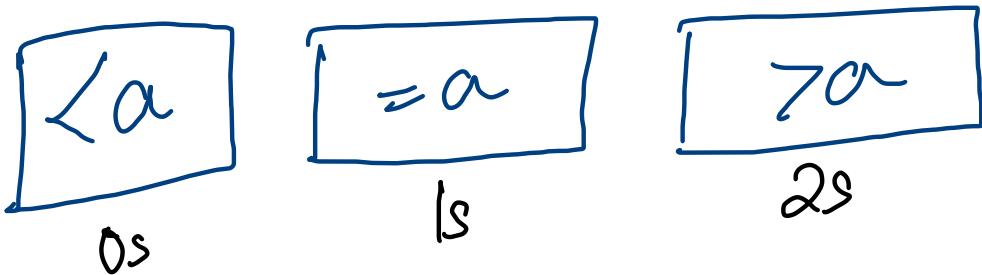
public static void swap(int[] arr, int p1, int p2){
    int temp = arr[p1];
    arr[p1] = arr[p2];
    arr[p2] = temp;
}

public void threeWayPartition(int arr[], int a, int b)
{
    int left = 0, mid = 0, right = arr.length - 1;

    while(mid <= right){
        if(arr[mid] < a){
            swap(arr, left, mid);
            left++; mid++;
        } else if(arr[mid] >= a && arr[mid] <= b){
            mid++;
        } else {
            swap(arr, mid, right);
            right--;
        }
    }
}
    
```

② Three way partitioning

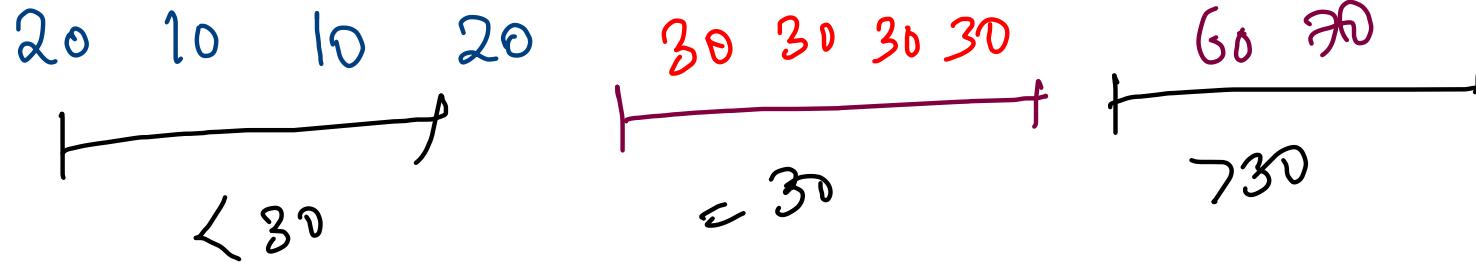
pivot = a



e.g

20 10 30 60 30 10 70 30 20 30

~~pivot = 30~~



③ Segregate - red, orange, + red

④ Segregate colors (R, G, B)

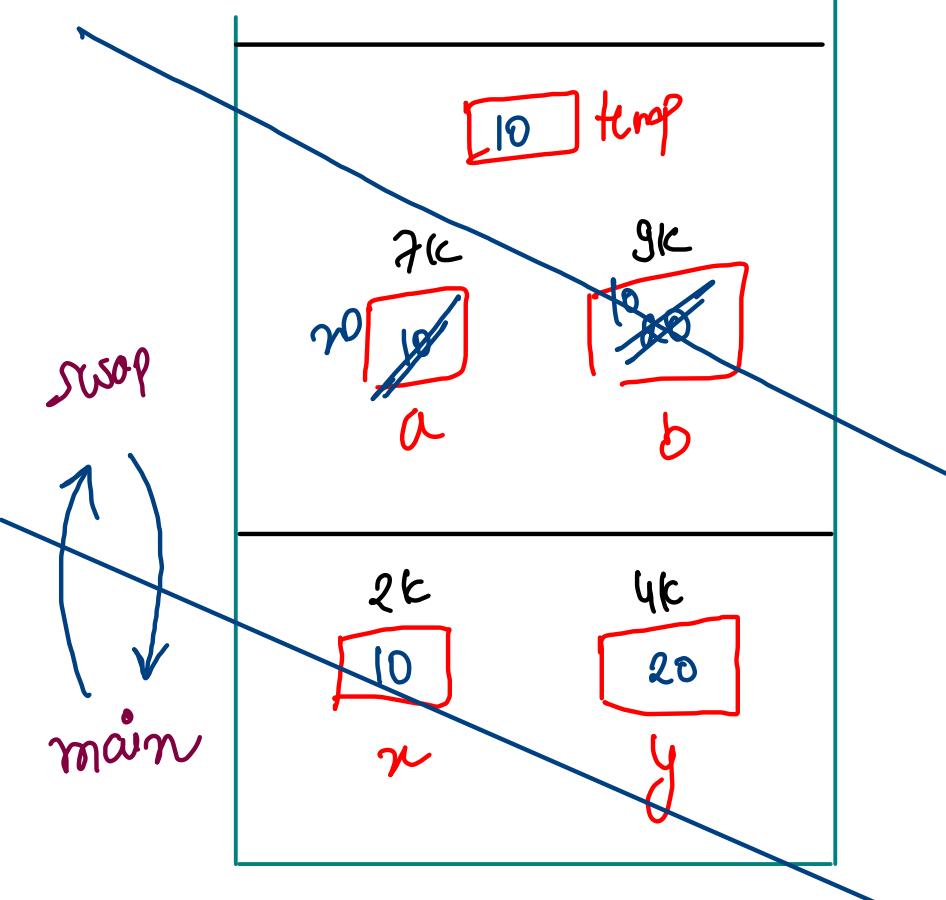
⑤ Sort Indian Flag colors (orange, white, green)

⑥ Segregate 'x', 'y' & 'z'

Arrays - Memory Management

Memory Mapping of Arrays

- ✓ → Contiguous Memory Allocation
- ✓ → Why java actually datatype
(strictly typed)
- ✓ → Reference Variable vs
actual object
- Heap (Random Access Memory)
- Changes in stack vs heap variable
(lifetime & scope of variables)



```

public static void swap(int a, int b){
    ① System.out.println(a + " " + b);      10      20
    int temp = a;
    a = b;
    b = temp;
    ③ System.out.println(a + " " + b);      20      10
}

```

```

public static void main(String[] args){
    int x = 10, y = 20;
    ① System.out.println(x + " " + y);      10      20
    swap(x, y);
    ④ System.out.println(x + " " + y);      10      20
}

```

10 20
10 20
20 10
10 20

① Local Variable

scope & lifetime
= function

② Stack Variables

changes does
not persist.

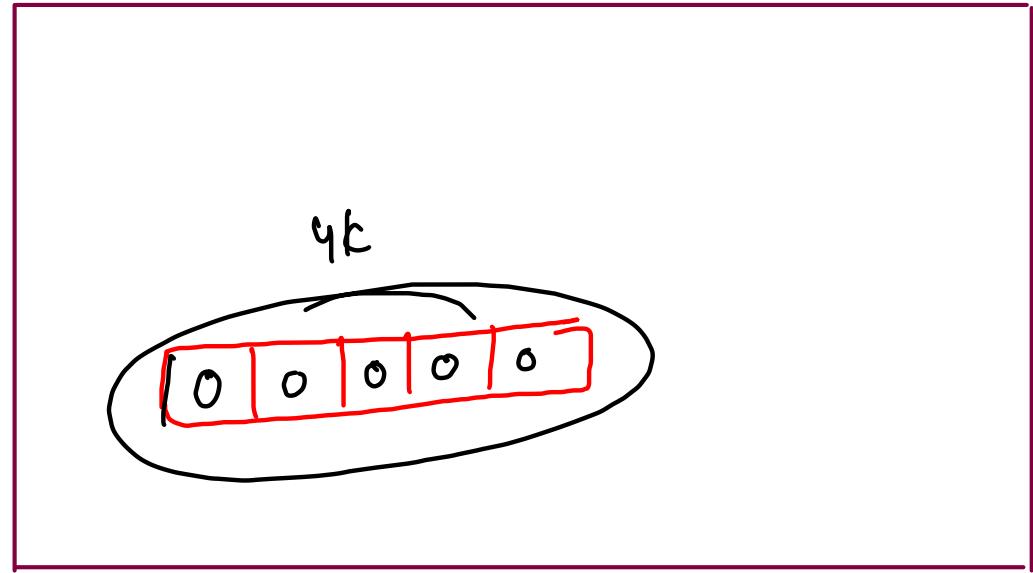
③ Java is always
pass by value!

`int [] arr;`
→ reference variables → null

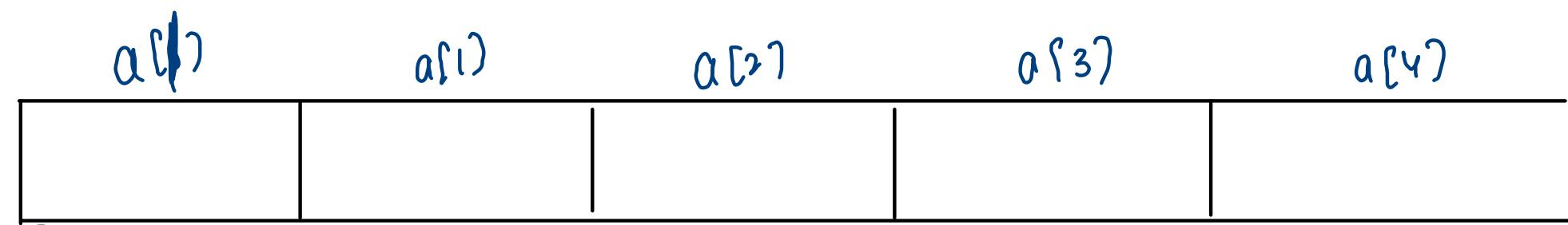
4k
`arr2 (8bytes)`
null
`arr (8bytes)`

Funcⁿ call stack

`int [] arr2`
= ~~`new int [5];`~~
↓ actual array



Heap (Random Access)
↳ arrays, strings & objects



4k 4k01 4k02 4k03 4k04 4k05 4k06 4k07 4k08 ... 4k11 4k12 ... 4k17 4k16 ... 4k19

Base address

\leftarrow 4 bytes \rightarrow \leftarrow 4 bytes \rightarrow \leftarrow 4 bytes \rightarrow \leftarrow 4 bytes \rightarrow \leftarrow 4 bytes \rightarrow

\leftarrow $4 \times 5 = 20$ bytes \rightarrow

= size of 1 block * no of blocks

= size of data type * length of array

4k \rightarrow 8k
4k \rightarrow 400k
4k \rightarrow 5k

$n=5$

Base address = 4K

$$a[0] = 4K + 0 \times 4$$

$$a[1] = 4K + 1 \times 4$$

$$a[2] = 4K + 2 \times 4$$

$$a[3] = 4K + 3 \times 4$$

$$a[4] = 4K + 4 \times 4$$

$$a[\text{idx}] = BA + \text{idx} \times \text{size of datatype}$$

Q1) Why 0-based indexing is done?

Q2) Why type of array is predefined (statically typed)?

Q3) Why indexing in array is random access?

```

public static void swap(int[] a, int[] b){
    ✓ System.out.println(a[0] + a[1] a + " " + b[0] + b[1]);
    ✓ int[] temp = a;          10   20           70   80
    ✓ a = b;
    ✓ b = temp;
    ✓ System.out.println(a[0] + a[1] a + " " + b[0] + b[1]);
}

```

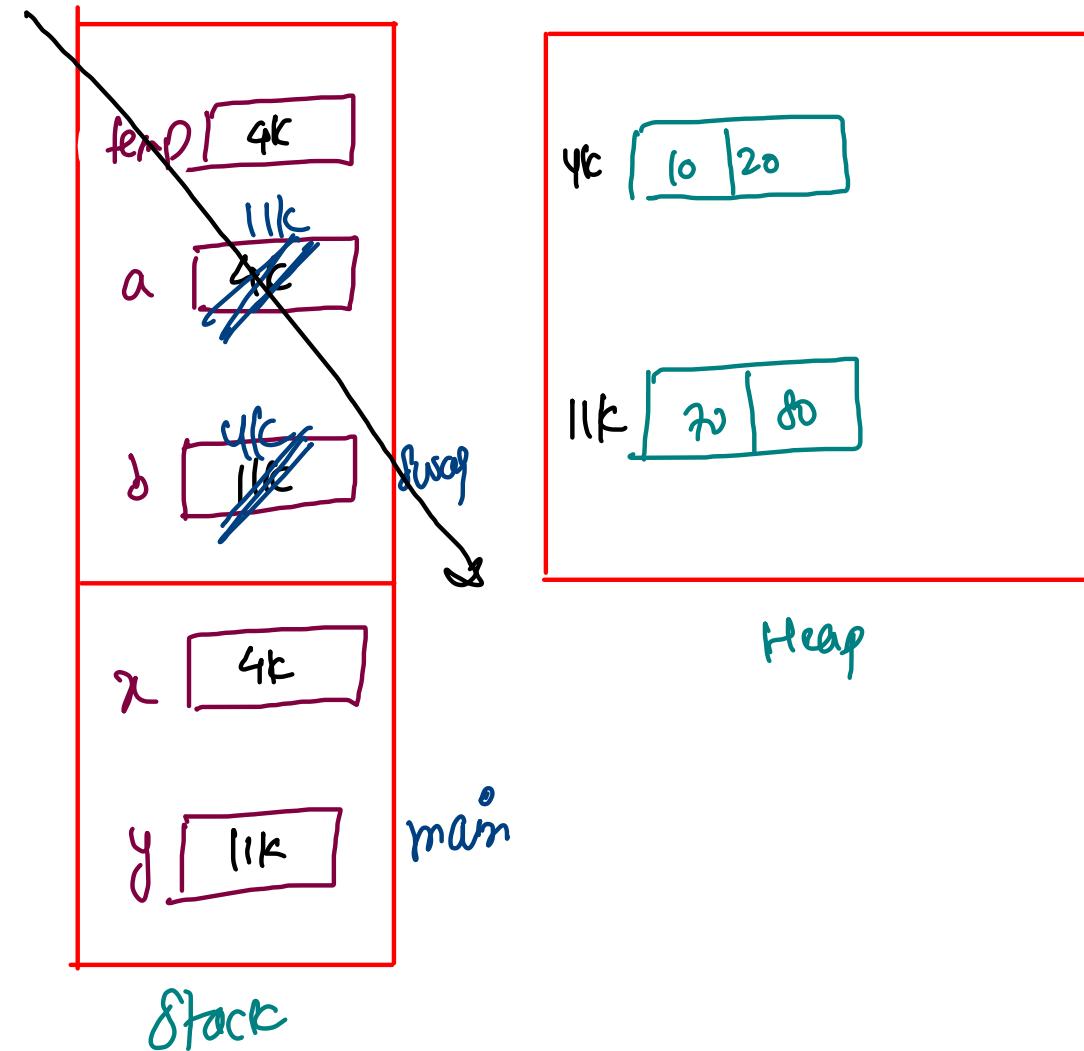
```

public static void main(String[] args){
    ✓ int[] x = {10, 20};
    ✓ int[] y = {70, 80};

    ✓ System.out.println(x[0] + x[1] y + " " + y[0] + y[1]);
    ✓ swap(x, y);
    ✓ System.out.println(x[0] + x[1] y + " " + y[0] + y[1]);
}

```

"Java is always pass by value"



```

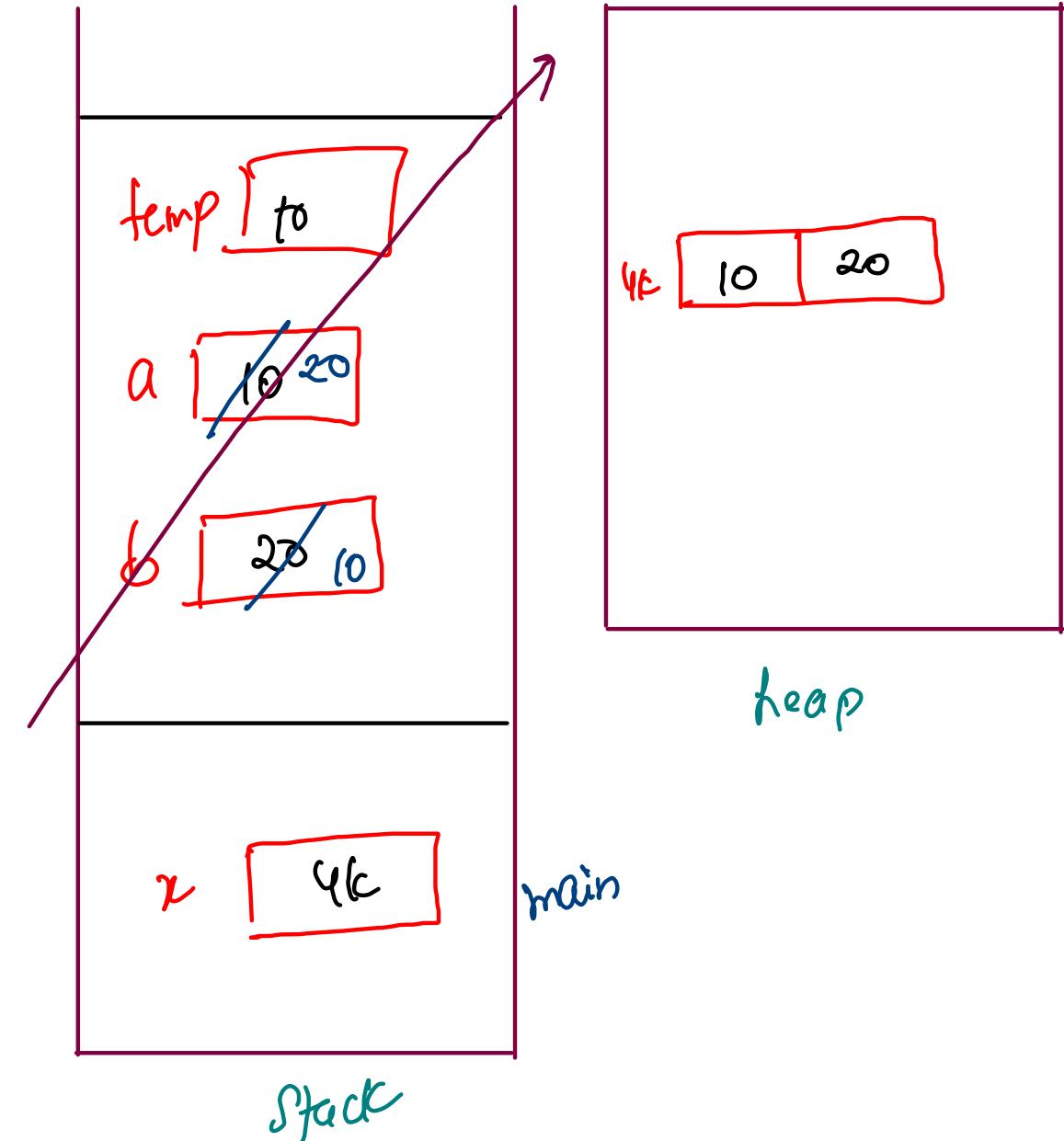
public static void swap(int a, int b){
    ✓int temp = a;
    ✓a = b;
    ✓b = temp;
}

public static void main(String[] args){
    ✓int[] x = {10, 20};

    ✓System.out.println(x[0] + " " + x[1]);
    swap(x[0], x[1]);
    ✓System.out.println(x[0] + " " + x[1]);
}

```

10 20



```

public static void swap(int[] x, int p1, int p2){
    ✓int temp = x[p1];
    ✓x[p1] = x[p2];
    ✓x[p2] = temp;
}

```

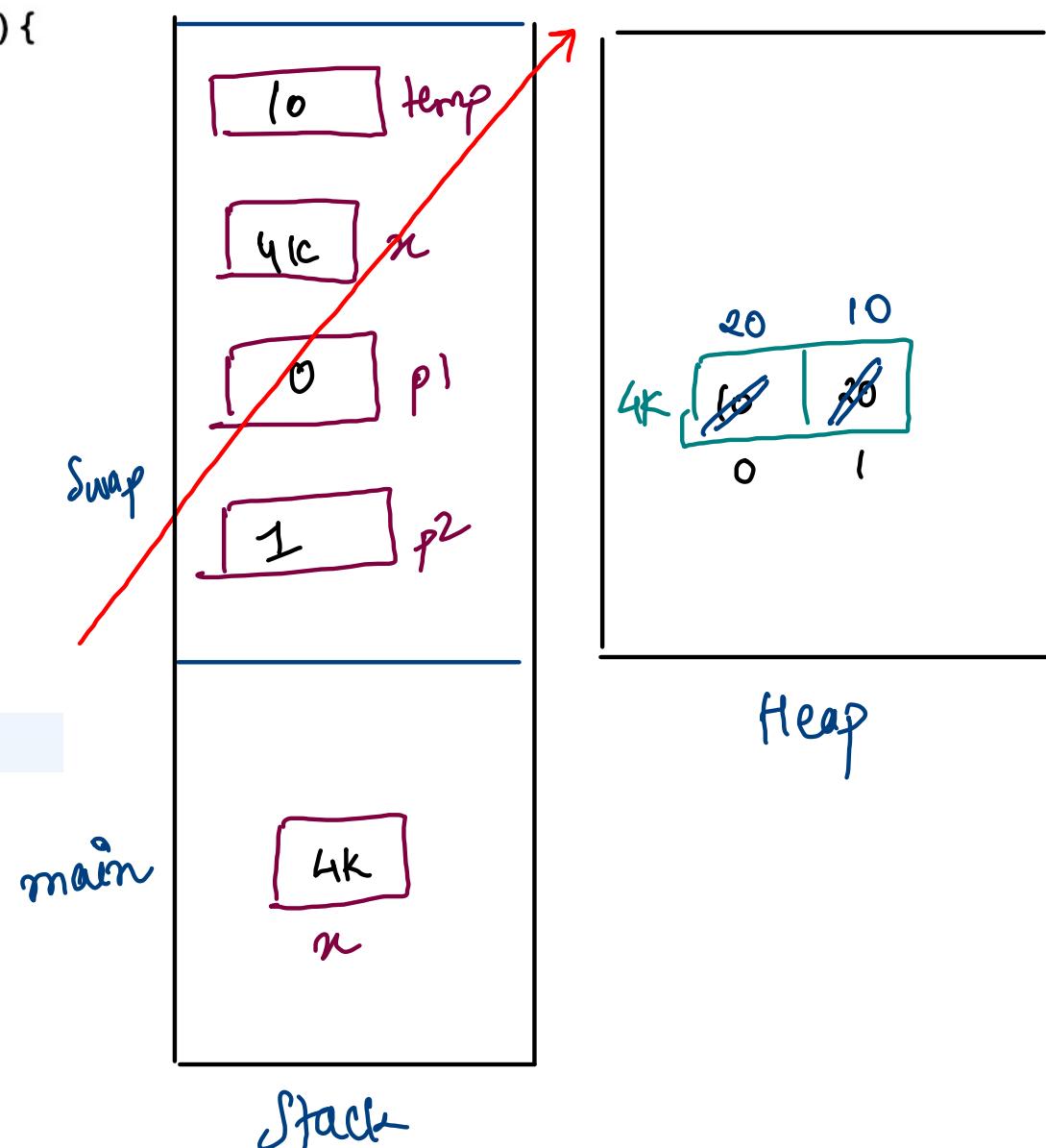
```

public static void main(String[] args){
    ✓int[] x = {10, 20};

    ✓System.out.println(x[0] + " " + x[1]);
    ✓swap(x, 0, 1);
    ✓System.out.println(x[0] + " " + x[1]);
}

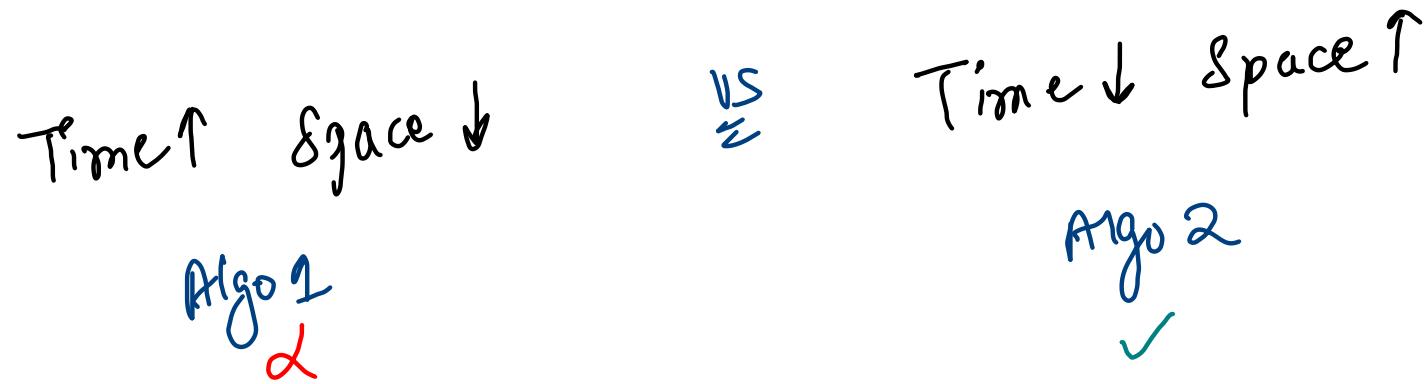
```

Heap changes will persist even if
function gets popped



Time & Space Complexity

Ideal: \rightarrow Time \downarrow and Space \downarrow (most optimized)

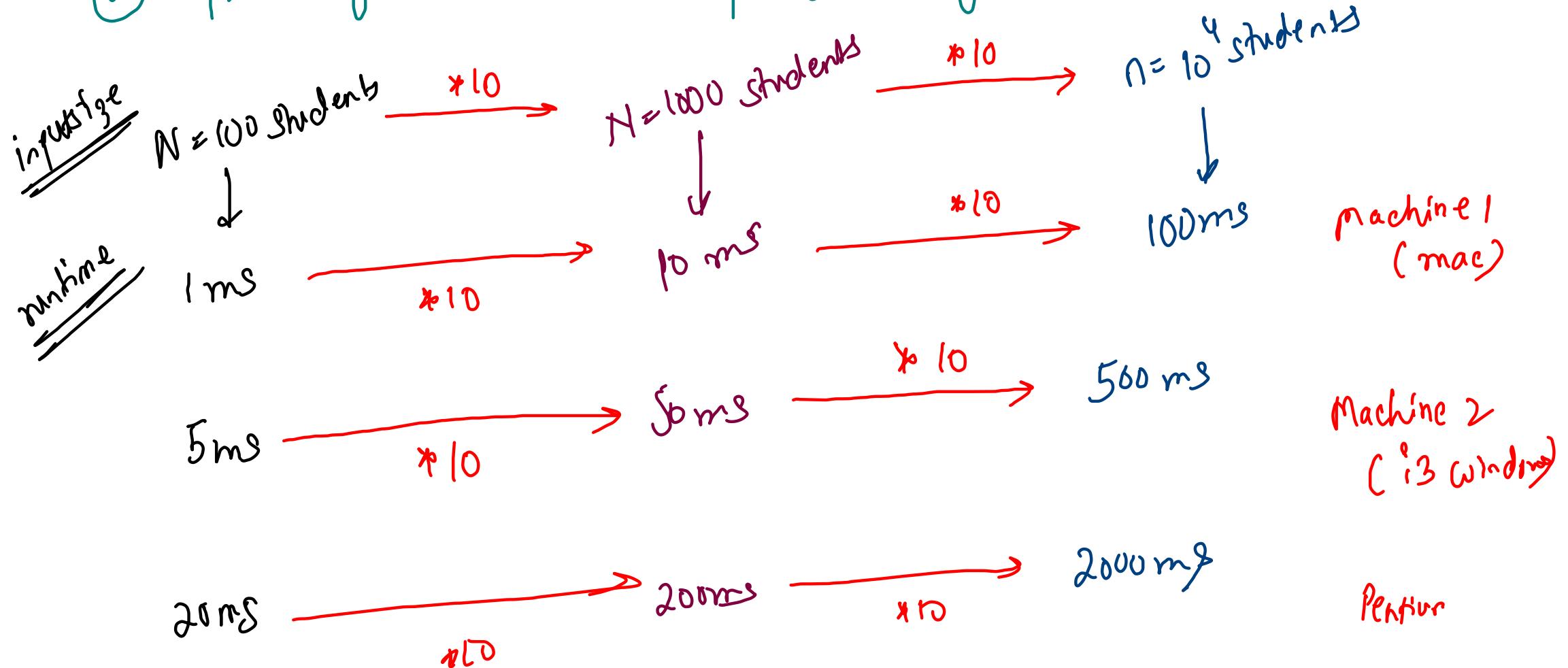


Time Complexity

① Runtime \rightarrow Machine Dependent

e.g. leetcode \rightarrow similar machines

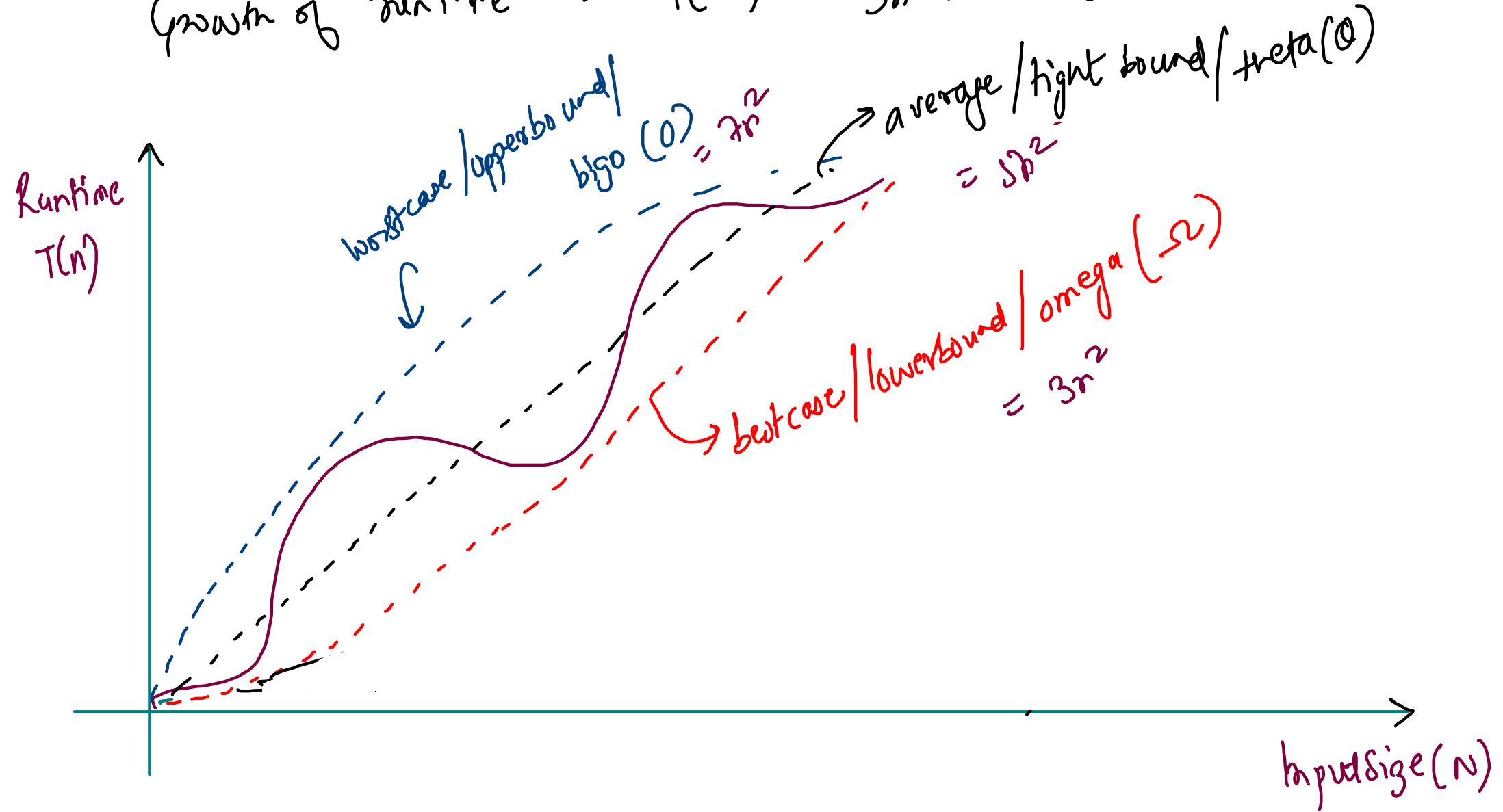
② Growth of runtime over input size (Asymptotic Analysis)



\Rightarrow Machine Independent

Algorithm Runtime \propto Input Size?

Growth of runtime $\Rightarrow T(n) = 5n^2 + 3n + 6$



$$T(n) = 5n^2 + 3n + 6$$

Ignore smaller terms

$$O(7n^2) \approx O(n^2) \text{ (Ignore constant terms)}$$

$$7n^2 > 5n^2 + 3n + 6$$

$n=1$

$7 \not> 5+3+6$ Ignore smaller terms

$n=2$

$7 \times 4 = 28 \not> 5 \times 4 + 3 \times 2 + 6$

$n=3$

$7 \times 3^2 > 5 \times 3^2 + 3 \times 3 + 6$
 $= 63 = 60$

$n \geq 3 \Rightarrow \text{Good!}$

$$O\} \quad \begin{array}{c} \text{Runtime } T(n) \\ 4n^3 + 100n + 200 \end{array}$$

Worst Case
 $\Rightarrow O(n^3)$ = Cubic

$$O\} \quad \begin{array}{c} 100n + 200n + 300n \\ = 600n \end{array}$$

$\Rightarrow O(n)$ = Linear

$$O\} \quad \begin{array}{c} 200 + 100 + 400 \\ = 700 \end{array}$$

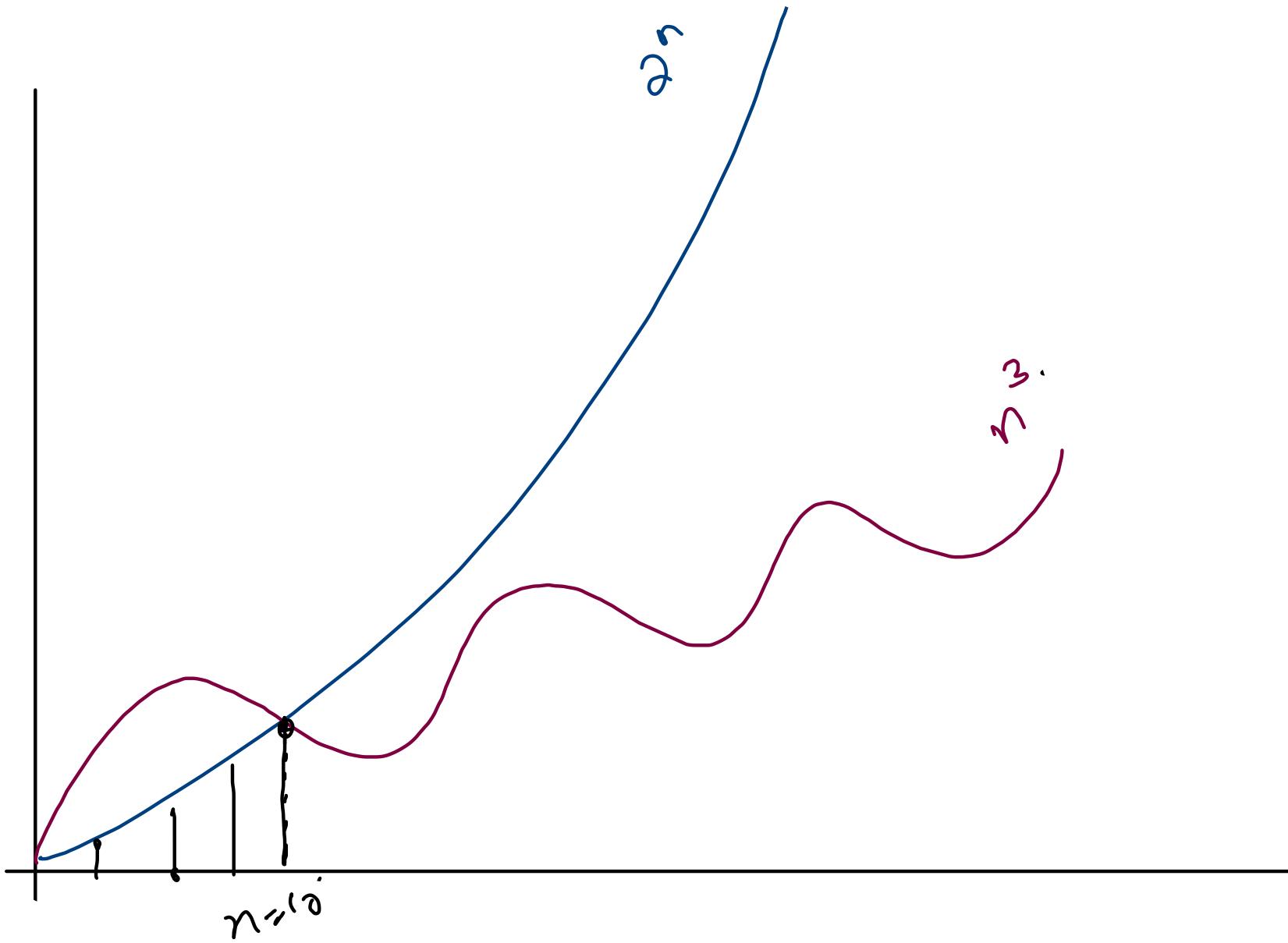
$\Rightarrow O(n^0) = O(1)$ = Constant

$$O\} \quad n^5 + n^3 + n^1 + 5$$

$\Rightarrow O(n^5)$

$$O\} \quad 2^n + n^{10} + n^9 + n^8$$

$\Rightarrow O(2^n)$ = Exponential



Time Complexity Chart

$$O(1) \prec O(\log_2 n) \ll O(\sqrt{n}) \prec O(n) \prec O(n \log n)$$

Constant logarithmic square root linear

$$\begin{array}{c}
 \text{higher} \\
 \text{order}
 \end{array}
 \quad
 \begin{array}{c}
 O(n^2) \quad \ll \quad O(n^3) \quad \ll \quad O(2^n) \quad < \quad O(3^n) \quad < \quad O(n!) \\
 \text{quadratic} \qquad \qquad \text{cubic} \qquad \qquad \qquad \qquad \qquad \text{exponential} \\
 \text{poly} \qquad \qquad \text{poly}
 \end{array}$$

O(1) Constant Time Complexity

① System.out.print() Printing

② Scanner scn = new Scanner(Systems);
int marks = scn.nextInt(); { Inputs for variable } return statement

③ logical Operators, Comparison operators → arithmetic operation
(&&, ||, !) (==, !=, >, <, >=, <=) (+, -, /, %, *)

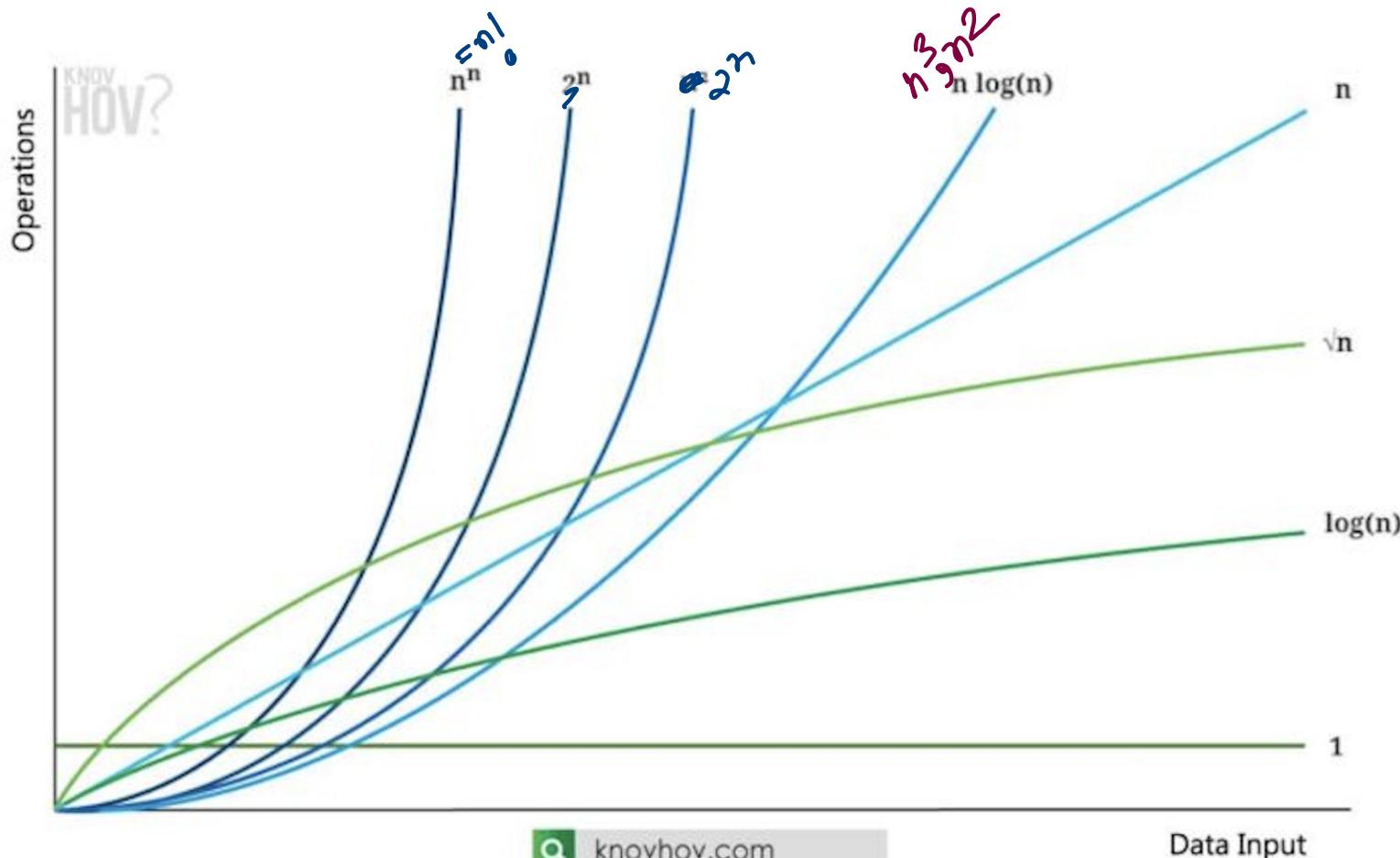
④ Assignment Operation (=)

⑤ Array Indexing (arr[ids])

⑥ Bitwise operators (&, |, ^, !, <<, >>) ⑦ Ternary operators () ?:

⑤ if-else conditions / switch

⑥ Function call
→ passing parameter



knovhov.com

BIG O' NOTATION TIME COMPLEXITY

for loops time complexity.

①

```
for(int idx = 1; idx <= n; idx++){
    System.out.print(idx + " ");
}
```

$n = 5$

| | |
|-----------|-----------------|
| $idx = 1$ | $k_{ms} = O(1)$ |
| $idx = 2$ | $k_{ms} = O(1)$ |
| $idx = 3$ | $k_{ms} = O(1)$ |
| $idx = 4$ | $k_{ms} = O(1)$ |
| $idx = 5$ | $k_{ms} = O(1)$ |

$O(n) = \text{linear}$

```

int n = scn.nextInt();

for(int idx = 1; idx <= n; idx++){
    System.out.print(idx + " ");
}

```

②

```

int m = scn.nextInt();
for(int idx = 1; idx <= m; idx++){
    System.out.print(idx + " ");
}

```

Two loops
which are
independent,
time complexity
will add up /

$n=5, m=3$

| | |
|---|--|
| $idx=1$ $idx=2$ $idx=3$ $idx=4$ $idx=5$ | $k \text{ ms}$ $k \text{ ms}$ $k \text{ ms}$ $k \text{ ms}$ $k \text{ ms}$ |
|---|--|

$\left. \begin{matrix} k \text{ ms} \\ k \text{ ms} \\ k \text{ ms} \\ k \text{ ms} \\ k \text{ ms} \end{matrix} \right\} 5 \times k \text{ ms}$
 $= 5 \times k \text{ ms}$

+

| | |
|-------------------------------|--|
| $idx=1$ $idx=2$ $idx=3$ | $k \text{ ms}$ $k \text{ ms}$ $k \text{ ms}$ |
|-------------------------------|--|

$\left. \begin{matrix} k \text{ ms} \\ k \text{ ms} \\ k \text{ ms} \end{matrix} \right\} 3 \times k \text{ ms}$
 $= 3 \times k \text{ ms}$

$(n+m) \times k \text{ ms}$
 $\Rightarrow O(n+m)$
Linear

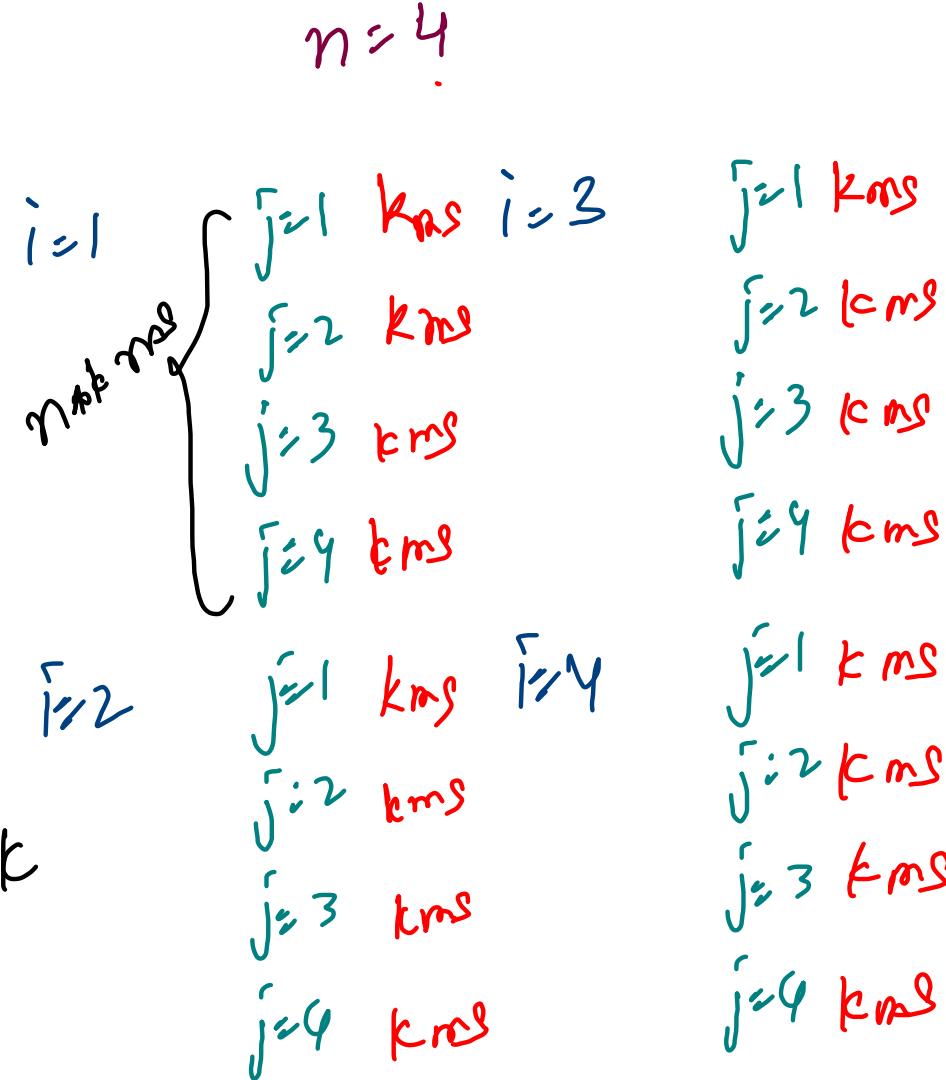
③

Nested loops

```
int n = scn.nextInt();
for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
        System.out.print(i + " " + j);
    }
}
```

Time complexity
will multiply if both loops are dependent!
 $\Rightarrow O(1)$

$$\begin{aligned} & n \times k + n \times k + n \times k \\ & = 4nk \quad \approx n \times n \times k \quad \text{MS} \\ & \Rightarrow O(n^2) \text{ quadratic} \end{aligned}$$



$n=5$, $m=3$

(4)

```
int n = scn.nextInt();
int m = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= m; j++){
        System.out.print(i + " " + j);
    }
}
```

$i=1 \quad j=1, j=2, j=3 \quad mk$
+
 $i=2 \quad j=1, j=2, j=3 \quad mk$
+
 $i=3 \quad j=1, j=2, j=3 \quad mk$
+
 $i=4 \quad j=1, j=2, j=3 \quad mk$
+
 $i=5 \quad j=1, j=2, j=3 \quad mk$

= $n \times m \times k$

$O(n \times m)$

5

```
int n = scn.nextInt();

for(int i = 1; i <= n; i++){
    for(int j = 1; j <= i; j++){
        System.out.print(i + " " + j);
    }
}

}
```

$$\gamma = 5$$

$i=1$ $j=1$ $R \text{ ms} = O(1) \Rightarrow 1 \text{ k ms}$

$$j=1 \quad k^{\text{ms}} \\ j=2 \quad l^{\text{ms}} \quad \left. \right\} \rightarrow 2 \otimes k^{\text{ms}}$$

$$\begin{matrix} i=3 \\ j=2 \\ j=3 \end{matrix} \quad \begin{matrix} j=1 \\ j=2 \\ j=3 \end{matrix} \quad \begin{matrix} k \\ + \\ n \\ + \\ c \end{matrix} \quad \left. \right\} 3 \times k \text{ ms}$$

j=4 j=1, 2, 3, 4 4 k ms

$i=5$ $j=1, 2, 3, 4, 5$ $\delta \text{ in cm}$

$$1k + 2k + 3k + 4k + 5k \\ = 1k(1+2+3+\dots+n)$$

$$= k \times \left(\frac{n(n+1)}{2} \right) = \cancel{\frac{k n^2}{2}} + \cancel{\frac{k n}{2}} \Rightarrow O(n^2)$$

$n = 5$

```

int n = scn.nextInt();

⑥ for(int i = 1; i <= n; i++){
    for(int j = i; j <= n; j++){
        System.out.print(i + " " + j);
    }
}
  
```

$i = 1$

$j = 1, 2, 3, 4, 5$

Sk

f

$i = 2$

$j = 2, 3, 4, 5$

$4K$

$i = 3$

$j = 3, 4, 5$

$3K$

$i = 4$

$j = 4, 5$

$2K$

$i = 5$

$j = 5$

$+$

$1K$

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

$$\Rightarrow \underline{\underline{O(n^2)}}$$

⑦

```
for(int i = 1; i < n; i++){
    System.out.print(i + " " unif);
}
```

$1, 2, 3, 4, \dots, n-1 \Rightarrow n-1 \text{ times}$
 $nk - k \Rightarrow O(n)$
=

⑧

```
for(int i = 1; i <= n / 2; i++){
    System.out.print(i + " " unif);
}
```

$1, 2, 3, 4, \dots, n/2 \Rightarrow n/2 \text{ times}$
 $\Rightarrow \frac{n}{2} \times k \Rightarrow O(n)$
=

n is +ve
 integer

```

int n = scn.nextInt();
  
```

(9)

```

for(int idx = n; idx <= 0; idx++){
    System.out.println(idx);
}
  
```

O iteration loops

$n = 5$
 $idx = 5$
 $idx \leq 0$ $5 \leq 0$ false

k ms
k ms

$O(1)$
Constant

(10)

```

for(int idx = n; idx >= 0; idx++){
    System.out.println(idx);
}
  
```

infinite loop

$n = 5$
 $idx = 5$ $5 \geq 0$

k ms

6 $6 \geq 0$

k ms

7 $7 \geq 0$

k ms

8 $8 \geq 0$

k ms

9

\Rightarrow infinite time
non terminating

```

int n = scn.nextInt();

for(int idx = 1; idx * idx <= n; idx++){
    System.out.println(idx);
}

```

11

06

```

int root = (int)Math.sqrt(n);

for(int idx = 1; idx <= root; idx++){
    System.out.println(idx);
}

```

$$n=4 \Rightarrow 2$$

$$\text{id}_n=1 \quad 1 \times 1 \leq 4 \quad \checkmark$$

$$\text{id}_n=2 \quad 2 \times 2 \leq 4 \quad \checkmark$$

$$n=9 \Rightarrow 3$$

$$\text{id}_n=1 \quad 1 \times 1 \leq 9 \quad \checkmark$$

$$\text{id}_n=2 \quad 2 \times 2 \leq 9 \quad \checkmark$$

$$\text{id}_n=3 \quad 3 \times 3 \leq 9 \quad \checkmark$$

$$n=25 \Rightarrow 5$$

$$\text{id}_n=1 \quad 1^2 \leq 25$$

$$\text{id}_n=2 \quad 2^2 \leq 25$$

$$\text{id}_n=3 \quad 3^2 \leq 25$$

$$\text{id}_n=4 \quad 4^2 \leq 25$$

$$\text{id}_n=5 \quad 5^2 \leq 25$$

$\Rightarrow O(\sqrt{n})$

$= O(\sqrt{N})$

$$n=16 \Rightarrow 4$$

$$\text{id}_n=1 \quad 1 \times 1 \leq 16$$

$$\text{id}_n=2 \quad 2^2 \leq 16$$

$$\text{id}_n=3 \quad 3^2 \leq 16$$

$$\text{id}_n=4 \quad 4^2 \leq 16$$

$= O(N^{1/2})$

```
int n = scn.nextInt();
```

12 **for**(int idx = 1; idx <= n; idx = idx * 2){
 System.out.println(idx);
}

$$n = 4 = 2^{\log_2 4}$$

$$\text{id} n = 1 \leq 4$$

$$\text{id} n = 2 \leq 4$$

$$\text{id} n = 4 \leq 4$$

3 times

$$n = 8 = 2^{\log_2 8}$$

$$\text{id} n = 1 \leq 8$$

$$\text{id} n = 2 \leq 8$$

$$\text{id} n = 4 \leq 8$$

$$\text{id} n = 8 \leq 8$$

4 times

$$n = 16 = 2^{\log_2 16}$$

$$\text{id} n = 1 \leq 16$$

$$\text{id} n = 2 \leq 16$$

$$\text{id} n = 4 \leq 16$$

$$\text{id} n = 8 \leq 16$$

$$\text{id} n = 16 \leq 16$$

5 times

$n = 2^n \Rightarrow \log_2 n = x$
loop is running $x+1$ times
 $= O(\underline{\underline{\log_2 n}})$

$$n = 32 = 2^5 \quad 5 = \log_2 32$$

6 times

$$n = 64 = 2^6 \quad 6 = \log_2 64$$

7 times

$$n = 128 = 2^7 \quad 7 = \log_2 128$$

8 times

13

```
for(int idx = n; idx >= 1; idx = idx / 2){    => O(log2 N)
    System.out.println(idx);
}
```

exponential \rightarrow as N increases, 2^N grows rapidly

$$N=10 \rightarrow 20 \rightarrow 30 \rightarrow 40$$

$$2^{10} \rightarrow 2^{20} \rightarrow 2^{30} \rightarrow 2^{40}$$

$$1024 \rightarrow 1048576 \rightarrow 1073741824 \rightarrow 1.09 \times 10^{12}$$

logarithmic \rightarrow as N grows, $\log_2 N$ grows very slowly

$$N \rightarrow 2^5 \rightarrow 2^{10} \rightarrow 2^{100} \rightarrow 2^{1000}$$

$$\log N \approx 5 \rightarrow 10 \rightarrow 100 \rightarrow 1000 \rightarrow \text{almost constant}$$

14)

```
for(int idx = 0; idx * idx <= n; idx++){
    System.out.println(idx);  $O(\sqrt{n})$ 
}
```

$idx = 0, 1, 2, 3, \dots, \text{infinite}$

15)

```
for(int idx = 0; idx <= n; idx = idx * 2){
    System.out.println(idx);  $\text{runtime error}$ 
}
```

16)

```
for(int idx = 1; idx <= n; idx = idx * 3){
    System.out.println(idx);  $O(\log_3 N)$ 
}
```

17)

```
for(int idx = 1; idx <= n; idx = idx * 10){
    System.out.println(idx);  $\log_{10} N$ 
}
```

18)

```
for(int idx = 1; idx <= n; idx = idx + 2){
    System.out.println(idx);
}
```

$\frac{n}{2} = O(n)$

19)

```
for(int idx = 1; idx <= n; idx = idx + 3){
    System.out.println(idx);
}
```

$\frac{n}{3} = O(n)$

20)

```
for(int idx = 1; idx <= n; idx++){
    for(int j = 0; j < 5; j++){
        System.out.println(idx);
    }
}
```

$\rightarrow n \cdot O(5) = O(n)$

21)

```
for(int idx = 1; idx <= n; idx++){
    for(int j = 1; j <= n; j = j * 2){
    }
}
```

$\log_2 N$

$= O(n \cdot \log_2 N)$

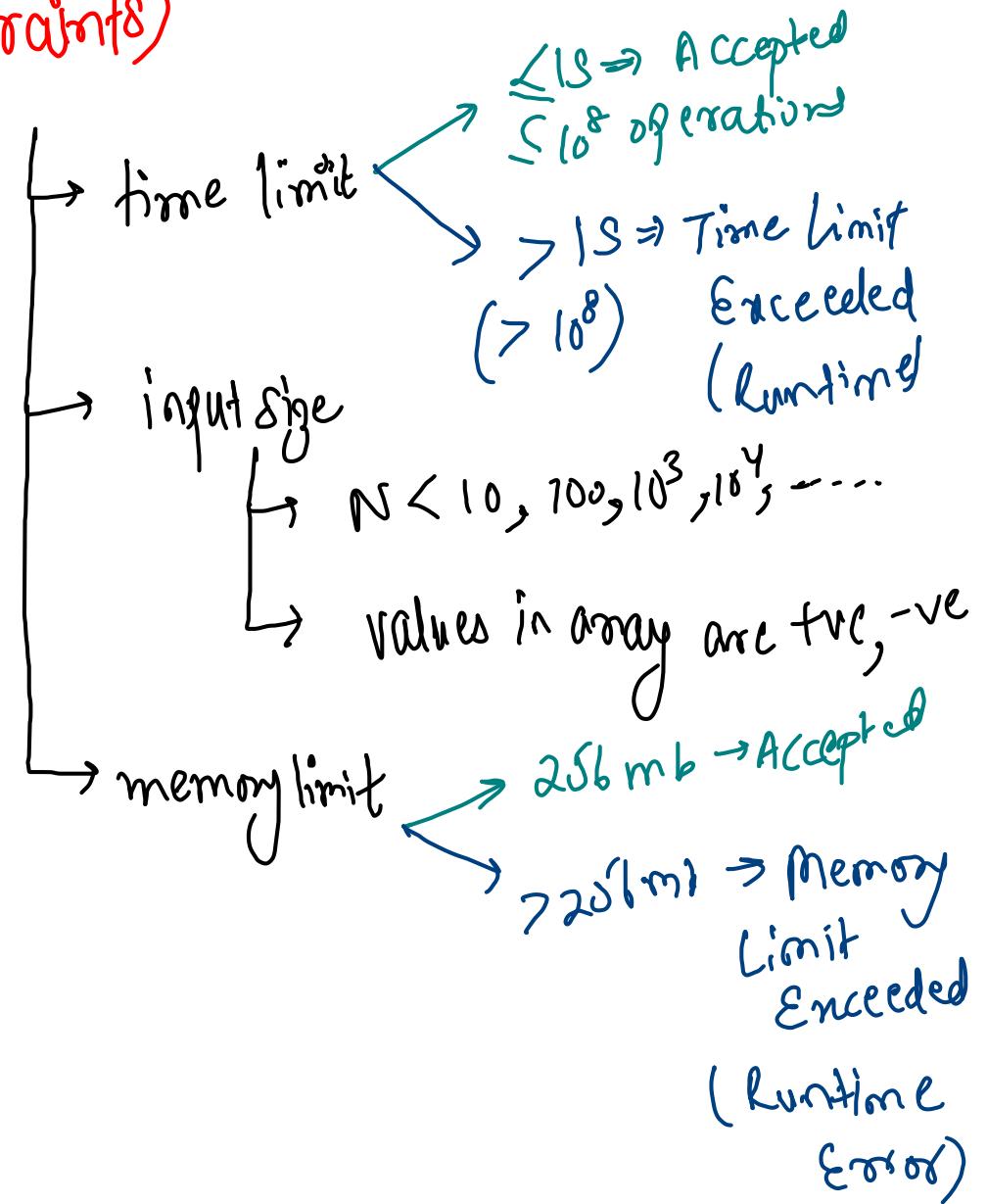
$n = 10$
 $idx = \{3, 5, 7, 9\}$

$n = 20$
 $idx = \{3, 5, 7, 9, 11, 13, 15, 17, 19\}$

$n/2$ times
 $\Rightarrow O(n)$

Competitive Programming Chart (Constraints)

$I_S = 10^8$ operations per second



Time Complexity Chart

1 second = 10^8 operations
time limit = 1s

| N_{max} | Time Complexity | Examples |
|--|---|--|
| 10^8 | $O(2^n)$ or $O(r!)$ | Recursion & Backtracking |
| 2^2 | $O(2^n \cdot n^2)$ {exponential} | Travelling Salesman (DP with Bitmasking) |
| 4×10^2 | $O(N^3)$ cubic | 3 nested loops (print all subarrays) Matrix multiplication |
| 10^4 | $O(N^2)$ quadratic | Nested loops (Bubble/Insertion/Selection), Matrix ^{2D DP} |
| 10^6 | $O(N \log N)$ | Arrays Sort (MergeSort / QuickSort / HeapSort) BS on Answer |
| 10^8 | $O(n)$ Linear | Linear search, Maxm of Array, Merge 2 sorted arrays |
| 10^{16} | $O(\sqrt{N})$ | Checking prime. |
| 10^{18} or 2^{63} long long value | $O(1)$ or $O(\log_2 N)$ constant logarithmic | Digital Translational, Binary No's, Binary Search |

① Print Hello world $\rightarrow O(1)$ Constant time complexity \Rightarrow never give TLE
Check odd or even, print a to z, print table of 4
 \Rightarrow always run

② linear loop $\rightarrow O(n)$

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
  
    for(int idx=0; idx<=n; idx++){  
        System.out.println(idx);  
    }  
}
```

Linear time complexity

$$\Rightarrow N_{max} = 10^8$$

array printing, Nth fibonacci

$$\begin{aligned} \text{Integer.MAX_VALUE} \\ = 2^{31} \times 10^9 \\ \uparrow \text{TLE} \end{aligned}$$

③ Logarithmic loop $\Rightarrow O(\log N)$ \Rightarrow It will never give TLE

```
Scanner scn = new Scanner(System.in);
int n = scn.nextInt();

for(int i = n; i > 0; i = i/3){
    System.out.println(i);
}
```

$$\left\} \log_3 N \right.$$

$$N_{\max} = \text{no. (int)}$$

$$\log N < 18 \\ N = 3^{18} = \text{very big value} = +\infty$$

$$N = \underbrace{10^{18}}_{\text{long}} \Rightarrow \text{digits} = \underline{18}$$

$$\left\} \text{Digit Digits of a No} \Rightarrow O(\log_{10} N) \right.$$

```
for(int idx=n; idx>=1; idx=idx/2){
    step=step+1;
}
```

$$\left\} \log_2 N \right.$$

④ Quadratic Time (nested loops)

```
int n = sc.nextInt();
for (int i = 0; i<n; i++){
    for (int j = 0; j<n; j++){
        System.out.print("*");
    }
    System.out.println();
}
```

```
for(int i=0 ; i<n ; i++){
    for(int j=0 ; j<=i ; j++){
        System.out.print("* ");
    }
    System.out.println();
}
```

↳ Printing Upper/lower Triangles

Grid/Bon Patterns, Bubble sort,
insert Sort, selectn sort, Bin↑Pancs

$$= O(n^2)$$

$$N_{\max} = 10^4$$

$$O(n^2) = 10^4 \times 10^4$$

$$= 10^8 \leq 10^8$$

Accepted

$$N = 10^8$$

$$O(n^2) = O(10^4 \times 10^4) \\ = O(10^{16}) \\ > 10^8$$

TLE

$$N = 10^2$$

$$O(n^2) = 10^2 \times 10^2 \\ = 10^4 \\ > 10^8$$

TLE

⑤ GCD (Euclid Algorithm) long division

```
public static int gcd(int a, int b){  
    int numerator = a, denominator = b;  
  
    while(numerator % denominator != 0){  
        int remainder = numerator % denominator;  
  
        numerator = denominator;  
        denominator = remainder;  
    }  
  
    return denominator;  
}
```



$O(\log(\min(a, b)))$
 $= \log N$

Check No is Prime

Approach①

```
public static boolean isPrime1(int n){  
    if(n == 1 || n == 2) return true; // 1 and 2 are prime  
  
    for(int idx = 2; idx <= n - 1; idx++){  
        // is idx a factor of n OR is n a multiple of idx  
        if(n % idx == 0) return false;  
    }  
  
    return true;  
}
```

$O(n)$ linear
 $N_{max} = 10^8$ (int)
brute force

$\leftarrow N = 10^9, 10^{11}, \dots, 10^{16}$
(time limit exceeded)

Approach ②

```
public static boolean isPrime2(int n){  
    if(n == 1 || n == 2) return true; // 1 and 2 are prime  
  
    int sqrt = (int)Math.sqrt(n);  
    for(int idx = 2; idx <= sqrt; idx++){  
        // is idx a factor of n OR is n a multiple of idx  
        if(n % idx == 0) return false;  
    }  
  
    return true;  
}
```

$O(\sqrt{N}) = O(N^{1/2})$

$N_{max} = (10^8)^2 = 10^{16}$ long
optimized approach

linear search Algorithm

```
public static String linearSearch(int[] arr, int target){  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] == target) {  
            return "True";  
        }  
    }  
  
    return "False";  
}
```

$$N_{\max} = 10^8$$

Worst case $\rightarrow \Theta(n)$ linear

Average case $\rightarrow \Theta(n)$ linear

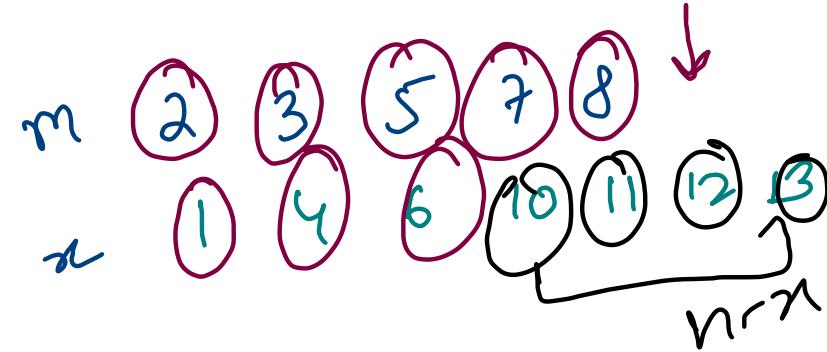
Best case $\rightarrow \Omega(1)$ constant

Merge 2 Sorted Arrays

```
public static int[] merge(int[] arr1, int[] arr2){  
    int first = 0, second = 0, third = 0;  
    int[] res = new int[arr1.length + arr2.length];  
  
    while(first < arr1.length && second < arr2.length){  
        if(arr1[first] <= arr2[second]){  
            res[third] = arr1[first];  
            first++;  
            third++;  
        } else {  
            res[third] = arr2[second];  
            second++;  
            third++;  
        }  
    }  
  
    while(first < arr1.length){  
        res[third] = arr1[first];  
        first++;  
        third++;  
    }  
  
    while(second < arr2.length){  
        res[third] = arr2[second];  
        second++;  
        third++;  
    }  
  
    return res;  
}
```

$$m = \text{arr1.length}$$

$$n = \text{arr2.length}$$



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

Print all subarrays

```
public static void printSubarrays(int[] arr){  
    for(int start = 0; start < arr.length; start++){  
        for(int end = start; end < arr.length; end++){  
            for(int idx = start; idx <= end; idx++){  
                System.out.print(arr[idx] + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

$n \times n \times n = O(n^3)$
arbit

$N_{max} = 500$
 $= 10^2 - 10^3$

Target Sum Subarray

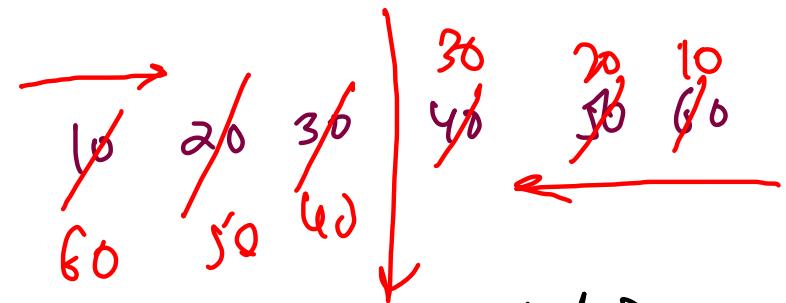
```
public static boolean targetSumSubarray(int[] arr, int target){  
    for(int st = 0; st < arr.length; st++){  
        int sum = 0;  
        for(int end = st; end < arr.length; end++){  
            sum = sum + arr[end];  
            if(sum == target) return true;  
        }  
    }  
    return false;  
}
```

$O(n^2)$ quadratic

$$N_{\max} = 10^4$$

Reverse Array

```
public static void reverseArray(int[] arr){  
    int left = 0, right = arr.length - 1;  
  
    while(left < right){  
        int temp = arr[left];  
        arr[left] = arr[right];  
        arr[right] = temp;  
  
        left++; right--;  
    }  
}
```



$$\begin{aligned} O(n/2) + O(n/2) \\ = O(n) \text{ linear} \end{aligned}$$

$$N_{\max} = 10^8$$

Sort 01

```
public static void sortBinary(int[] arr){  
    int left = 0, right = 0;  
    while(right < arr.length){  
        if(arr[right] == 1){  
            right++;  
        } else {  
            int temp = arr[left];  
            arr[left] = arr[right];  
            arr[right] = temp;  
  
            left++; right++;  
        }  
    }  
}
```

$O(n)$

Normal Solution

Two pointer
most optimized

Sort 012

```
public static void sort012(int[] arr){  
    int left = 0, mid = 0, right = arr.length-1;  
    while(mid <= right){  
        if(arr[mid] == 0){  
            swap(arr, left, mid);  
            left++;  
            mid++;  
        }  
        else if(arr[mid] == 1){  
            mid++;  
        }  
        else{  
            swap(arr, right, mid);  
            right--;  
        }  
    }  
}
```

$O(n)$

Space Complexity

- ① Input space complexity (Parameters)
- ② Output space complexity (Return type)
- ③ Recursion call stack space (Recursion)
- * ④ Extra or Auxiliary Space (Data Structures to perform Algo)
 - ↳ no extra space soln if in-place algorithm

O(1) space

↓
normal primitive
variables

O(n) space

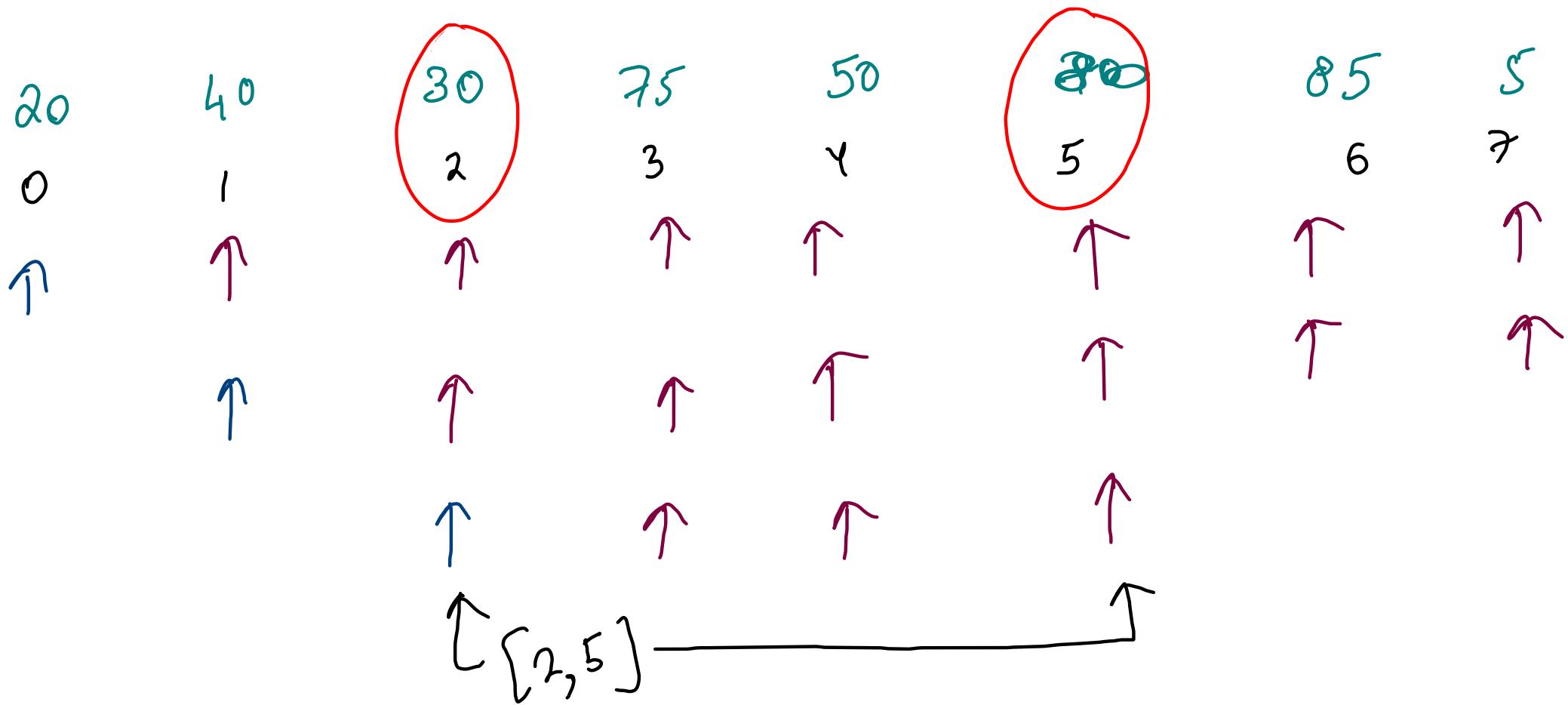
↓
1D Array / String

O(n²) space

↓
2D Array / matrix

Target sum pair (Unsorted)

target = 100



Two Sum - 1

LC ①

```
public int[] twoSum(int[] arr, int target) {  
    int[] res = new int[2];  
  
    for(int first = 0; first < arr.length; first++){  
        for(int second = first + 1; second < arr.length; second++){  
            if(arr[first] + arr[second] == target){  
                res[0] = first;  
                res[1] = second;  
            }  
        }  
    }  
    return res;  
}
```

Approach ①

Brute force



Nested loops

Time = $O(n^2)$

Space = $O(1)$
no extra space

Approach ②

HashMap

Time $\Rightarrow O(n)$

Space $\Rightarrow O(n)$
extra space

```
public static void twoSum(int[] arr, int target) {  
    for(int first = 0; first < arr.length; first++){  
        for(int second = first + 1; second < arr.length; second++){  
            if(arr[first] + arr[second] == target){  
                System.out.println(first + " " + second);  
            }  
        }  
    }  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
    int[] arr = new int[n];  
    for(int idx = 0; idx < n; idx++){  
        arr[idx] = scn.nextInt();  
    }  
    int target = scn.nextInt();  
  
    twoSum(arr, target);  
}
```

Print all
target
pairs

Reach Target - 1

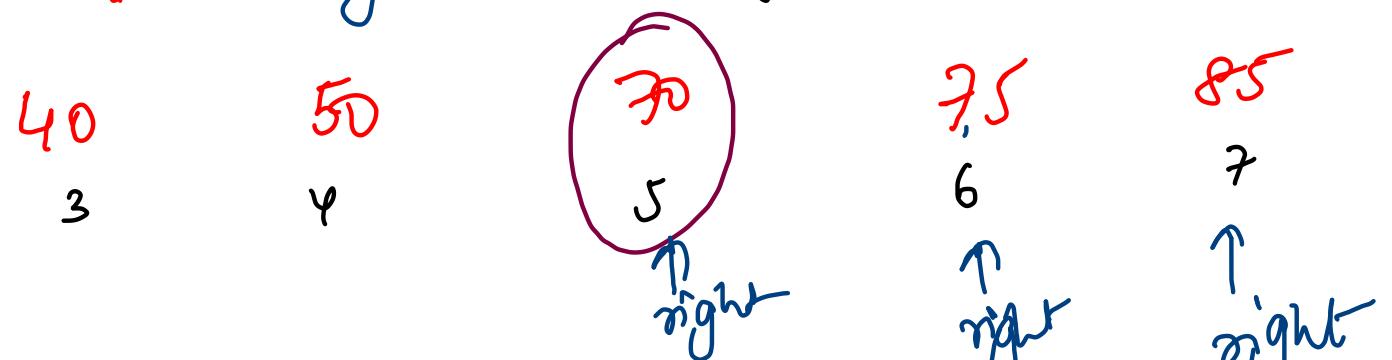
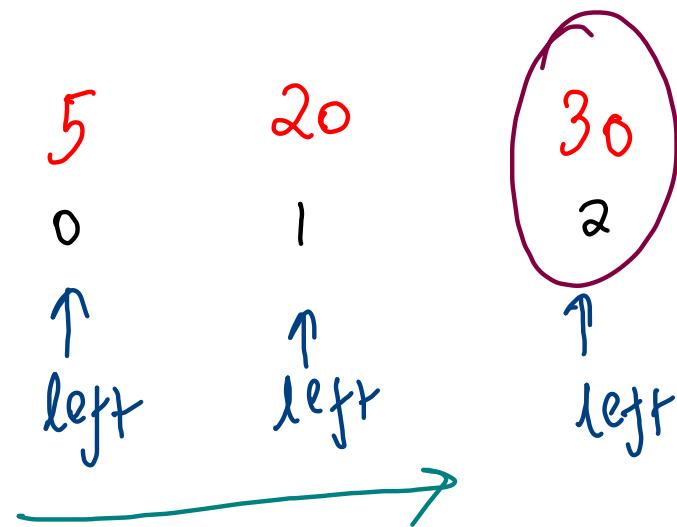
```
public static void twoSum(int[] arr, int target) {  
    for(int first = 0; first < arr.length; first++){  
        for(int second = first + 1; second < arr.length; second++){  
            if(arr[first] + arr[second] == target){  
                System.out.println(first + " " + second);  
                return;  
            }  
        }  
    }  
  
    System.out.println(-1); // no target sum pair  
    return;  
}
```

Reach Target (1)
Hackerrank

✓ ✓ ✓
 20 30 40
 0 1 2

✓ ✓ ✓
 75 50 70
 3 4 5

Arrays.sort $\Rightarrow N \log N$



$$arr(left) + arr(right)$$

$$= 100$$

< 100

return

$\{left, right\}$

$$> 100$$

$$20 + 85 = 105$$

reject $\{j\}$

$right - j$

$$30 + 75 > 100$$

$target = 100$

$$5 + 85 = 90$$

reject $\{j\}$

$left +$

$$20 + 75 = 95$$

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------------|------------|------------|------------|
| 5 | 20 | 30 | 40 | 50 | 72 | 75 | 85 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| ↑ left | ↑ left | ↑ left | ↑ left | ↑ left right | ↑ right | ↑ right | ↑ right |

$$5 + 85 = 90 < 100 \text{ left++}$$

$$20 + 85 = 105 > 100 \text{ right--}$$

$$20 + 75 = 95 < 100 \text{ left++}$$

$$30 + 75 = 105 > 100 \text{ right--}$$

$$30 + 72 = 102 > 100 \text{ right--}$$

target = 100

$$30 + 50 = 80 < 100 \text{ left++}$$

$$40 + 50 = 90 < 100 \text{ left++}$$

Two Sum - ⑪ Sorted

FC 167

Two Pointer Technique

```
public int[] twoSum(int[] arr, int target) {  
    int[] res = new int[2];  
  
    int left = 0, right = arr.length - 1;  
    while(left < right){  
        if(arr[left] + arr[right] < target){  
            left++;  
        } else if(arr[left] + arr[right] > target){  
            right--;  
        } else {  
            res[0] = left + 1; } 1-based indexing (acc. to question)  
            res[1] = right + 1;  
            break; // come out of the loop (target is found)  
    }  
  
    return res;  
}
```

Time
 $\underline{\underline{O(n)}}$

Space
 $\underline{\underline{\text{constant extra space}}}$

$\underline{\underline{O(1)}}$

Target Sum Pairs

Unsorted + duplicates + unique pairs

target = 60

| | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|---------|---------|---------|---------|---------|
| 20 | 30 | 10 | 60 | 50 | 30 | 10 | 50 | 50 | 40 | 30 |
| 10 | 10 | 20 | 30 | 30 | 30 | 40 | 50 | 50 | 50 | 60 |
| ↑ left | ↑ right |

$$10 + 60 = 70 > 60$$

$$20 + 50 = 70 > 60$$

✓ 10, 50

$$10 + 50 = 60 \checkmark$$

$$20 + 40 = 60 \checkmark$$

✓ 20, 40

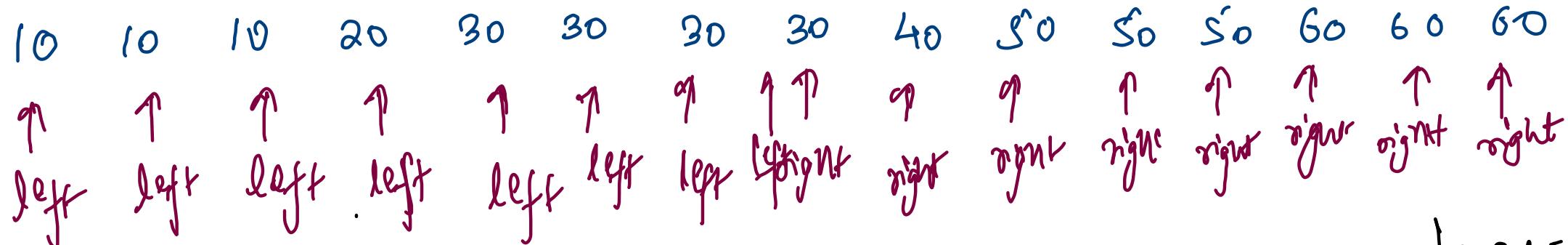
$$20 + 50 = 70 > 60$$

$$30 + 40 = 70 > 60$$

✓ 30, 40

$$20 + 50 = 70 > 60$$

$$30 + 30 = 60$$



$\text{target} = \boxed{60}$

```

public static void targetSumPairs(int[] arr, int target){
    Arrays.sort(arr); nlogn
    int left = 0, right = arr.length - 1;
    while(left < right){
        if(left > 0 && arr[left] == arr[left - 1]) left++;
        else if(arr[left] + arr[right] == target){
            System.out.println(arr[left] + " " + arr[right]);
            left++;
        } else if(arr[left] + arr[right] < target){
            left++;
        } else {
            right--;
        }
    }
}

```

$O(n^2)$

$$\begin{aligned}
 10 + 60 &= 70 > 60 \\
 30 + 40 &= 70 > 60 \\
 10 + 60 &= 70 > 60 \\
 10 + 10 &= 20 > 60 \\
 \boxed{30 + 30} &= 60 \\
 \boxed{10 + 50} &= 60 \\
 20 + 50 &= 70 > 60 \\
 20 + 50 &= 70 > 60 \\
 \boxed{20 + 40} &= 60
 \end{aligned}$$

Target Sum Triplet

target = 90

10 10 10 20 30 30 30 30 40 50 50 50 60 60 60

Brute force \Rightarrow 3 nested loops

$O(n^3)$ Cubic time

```
public static void threeSum(int[] arr, int target){  
    Arrays.sort(arr);  
    for(int first = 0; first < arr.length; first++){  
        for(int second = first + 1; second < arr.length; second++){  
            for(int third = second + 1; third < arr.length; third++){  
                if(arr[first] + arr[second] + arr[third] == target){  
                    System.out.println(arr[first] + " " + arr[second] + " " + arr[third]);  
                }  
            }  
        }  
    }  
}
```

✓ 10 + 20 + 60 = 90

✓ 10 + 30 + 50 = 90

✓ 20 + 30 + 40 = 90

✓ 30 + 30 + 30 = 90

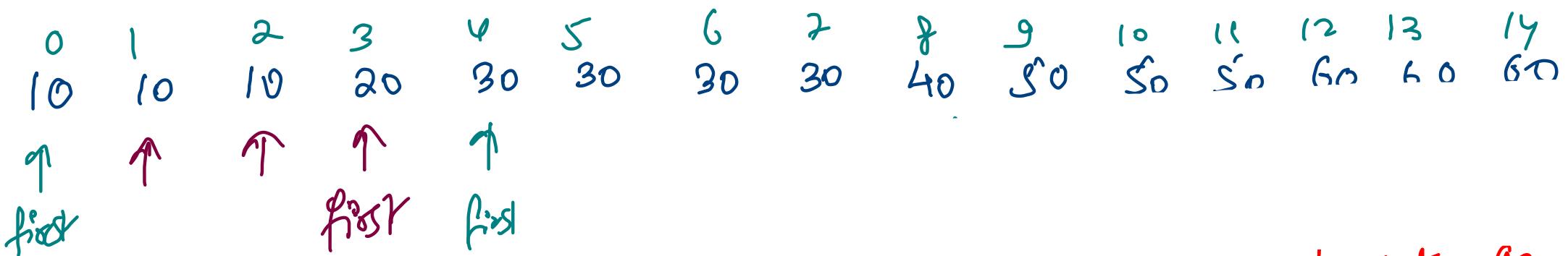
10 10 10 20 30 30 30 40 50 50 50 60 60 60

```
public static void threeSum(int[] arr, int target){  
    Arrays.sort(arr);  
    for(int first = 0; first < arr.length; first++){  
        if(first > 0 && arr[first - 1] == arr[first]) continue;  
  
        for(int second = first + 1; second < arr.length; second++){  
            for(int third = second + 1; third < arr.length; third++){  
                if(arr[first] + arr[second] + arr[third] == target){  
                    System.out.println(arr[first] + " " + arr[second] + " " + arr[third]);  
                }  
            }  
        }  
    }  
}
```

10 10 10 20 30 30 30 40 50 50 50 60 60
↑ ↑ S ↑ t

```
public static void threeSum(int[] arr, int target){  
    Arrays.sort(arr);  
    for(int first = 0; first < arr.length; first++){  
        if(first > 0 && arr[first - 1] == arr[first]) continue;  
  
        for(int second = first + 1; second < arr.length; second++){  
            if(second > first + 1 && arr[second - 1] == arr[second]) continue;  
  
            for(int third = second + 1; third < arr.length; third++){  
                if(third > second + 1 && arr[third - 1] == arr[third]) continue;  
  
                if(arr[first] + arr[second] + arr[third] == target){  
                    System.out.println(arr[first] + " " + arr[second] + " " + arr[third]);  
                }  
            }  
        }  
    }  
}
```

target=90
10+20+60
10+30+50



```

public static void twoSum(int[] arr, int target, int first, int left, int right){
    while(left < right){
        if(left > first + 1 && arr[left] == arr[left - 1]) left++;
        else if(arr[left] + arr[right] == target){
            System.out.println(arr[first] + " " + arr[left] + " " + arr[right]);
            left++;
        } else if(arr[left] + arr[right] < target){
            left++;
        } else {
            right--;
        }
    }
}

```

Time $\uparrow O(n^2)$ quadratic

```

public static void threeSum(int[] arr, int target){
    Arrays.sort(arr);
    for(int first = 0; first < arr.length; first++){
        if(first > 0 && arr[first] == arr[first - 1]) continue;
        twoSum(arr, target - arr[first], first, first + 1, arr.length - 1);
    }
}

```

$$10 + ? + ? = 90$$

$$\text{target} = 80$$

$$10 + 20 + 60$$

$$10 + 30 + 50$$

$$20 + 30 + 40$$

(9 Sept) → 7:30 to 8:30 PM (Archit)

8:30 PM to 9:30 PM (Shreya)

20 Sept, 21 Sept → Shreya

22nd Sept → Thursday (Archit)

7:30 PM
to 9:30 PM

2 sum

Brute force \rightarrow 2 loops $O(n^2)$



Two pointer \rightarrow 1 loop $O(n)$

3 sum

Brute force \rightarrow 3 loops $O(n^3)$



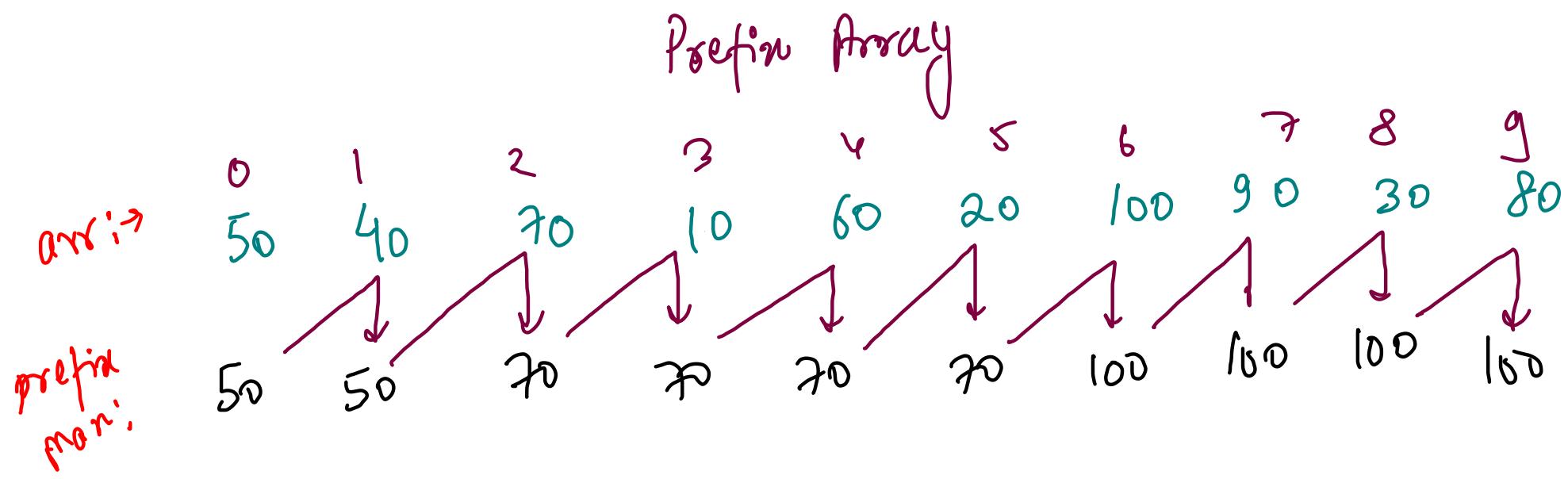
Two pointer \rightarrow 2 loops $O(n^2)$

4 sum

Brute force \rightarrow 4 loops $O(n^4)$ \rightarrow Two pointer \rightarrow 3 loops $O(n^3)$



Hash map \rightarrow 2 loops $O(n^2)$



```

public static int[] prefixMax(int[] arr){
    int[] prefix = new int[arr.length];

    for(int i = 0; i < arr.length; i++){
        int ans = Integer.MIN_VALUE;
        for(int j = 0; j <= i; j++){
            ans = Math.max(ans, arr[j]);
        }
        prefix[i] = ans;
    }

    return prefix;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for(int idx = 0; idx < arr.length; idx++){
        arr[idx] = scn.nextInt();
    }

    int[] prefix = prefixMax(arr);
    for(int idx = 0; idx < arr.length; idx++){
        System.out.println(prefix[idx]);
    }
}

```

Brute force $O(n^2)$

$N_{max} \sim 10^4$

```

public static int[] prefixMax(int[] arr){
    int[] prefix = new int[arr.length];

    for(int i = 0; i < arr.length; i++){
        int previousMax = (i == 0) ? Integer.MIN_VALUE : prefix[i - 1];
        prefix[i] = Math.max(arr[i], previousMax);
    }

    return prefix;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for(int idx = 0; idx < arr.length; idx++){
        arr[idx] = scn.nextInt();
    }

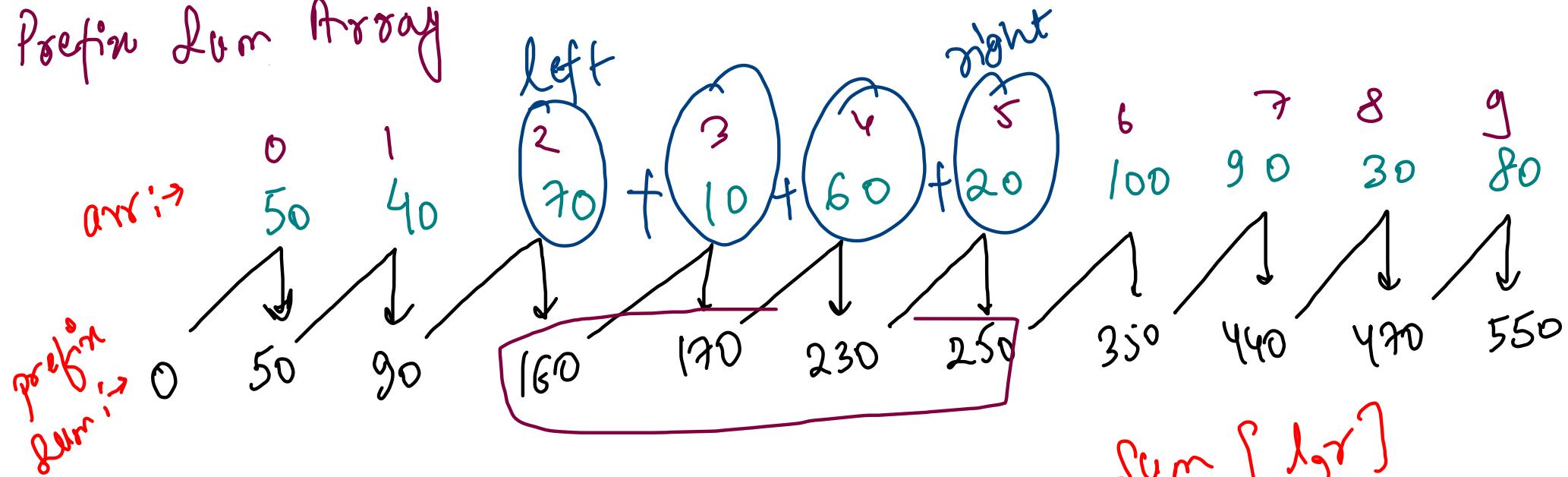
    int[] prefix = prefixMax(arr);
    for(int idx = 0; idx < arr.length; idx++){
        System.out.println(prefix[idx]);
    }
}

```

Optimized Approach $O(n)$

$N_{max} \leq 10^8$

Prefix Sum Array



```
public static int[] prefixSum(int[] arr){
    int[] prefix = new int[arr.length];

    for(int i = 0; i < arr.length; i++){
        int previousSum = (i == 0) ? 0 : prefix[i - 1];
        prefix[i] = arr[i] + previousSum;
    }
    return prefix;
}
```

6 7 8 9
 100 90 30 80
 350 440 470 550

sum [l:r]

$$\begin{aligned} &= \text{prefix}[r] \\ &\quad - \text{prefix}[l-1] \end{aligned}$$

$$\begin{aligned} &50 + 40 + 70 + 10 + 60 + 20 \\ &- (50 + 40) = \text{prefix}[5] \\ &\quad - \text{prefix}[1] \end{aligned}$$

```
public static int[] prefixSum(int[] arr){  
    int[] prefix = new int[arr.length];  
  
    for(int i = 0; i < arr.length; i++){  
        int previousSum = (i == 0) ? 0 : prefix[i - 1];  
        prefix[i] = arr[i] + previousSum;  
    }  
  
    return prefix;  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
    int[] arr = new int[n];  
    for(int idx = 0; idx < arr.length; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int[] prefix = prefixSum(arr);  
    int left = scn.nextInt();  
    int right = scn.nextInt();  
    for(int idx = left; idx <= right; idx++){  
        System.out.println(prefix[idx]);  
    }  
}
```

Frequency Related problems

Point Frequency of Alphabet

A hand-drawn signature "Archit Aggarwal" in red ink, with black arrows above it pointing downwards and green arrows below it pointing upwards.

'a', 'b', 'c' ... 'x', 'y', 'z'

Lower case English alphabets

⑦ Build the frequency array

a-4

1 - 2

C-1

b1

5
(-)

t-1

9,2

CP-1

J - I

'a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm'
0 1 2 3 4 5 6 7 8 9 10 11 12

| | | | | | | | | | | | | |
|--------------|---|--------------|---|---|---|--------------|--------------|---|---|--------------|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
|--------------|---|--------------|---|---|---|--------------|--------------|---|---|--------------|---|---|

'n' 'o' 'g' 'g' 'u' 's' 't' 'u' 'r' 'w' 'x' 'y' 'z'
13 14 15 16 17 18 19 20 21 22 23 24 25

0 0 0 0 ~~0~~₁₂ 0 ~~0~~₁ 0 0 0 ~~0~~₁ 0 0 0

$$'a' \rightarrow 0 \quad ('a' - 'a' = 97 - 97)$$

$$'b' \rightarrow 1 \quad ('b' - 'a' = 98 - 97)$$

$$'c' \rightarrow 2 \quad ('c' - 'a' = 99 - 97)$$

$$'d' \rightarrow 3 \quad ('d' - 'a' = 100 - 97)$$

$$'e' \rightarrow 4 \quad ('e' - 'a' = 101 - 97)$$

$$'f' \rightarrow 5 \quad ('f' - 'a' = 102 - 97)$$

$$w \rightarrow 22 \quad ('w' - 'a' = 119 - 97 = 22)$$

$$x \rightarrow 23 \quad ('x' - 'a' = 120 - 97 = 23)$$

$$y \rightarrow 24 \quad ('y' - 'a' = 121 - 97 = 24)$$

$$z \rightarrow 25 \quad ('z' - 'a' = 122 - 97 = 25)$$

$$\text{index of character } ch = ch - 'a' = ch - 97$$

$$\text{character at index } i = (\text{char})(\text{idx} + 'a')$$

```

public static int[] frequencyArray(String str){
    int[] freq = new int[26];

    for(int idx=0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        freq[ch - 'a']++;
    }

    return freq;
}

public static void printFrequency(String str, int[] freq){
    for(int idx=0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        if(freq[ch - 'a'] > 0){
            System.out.println(ch + "-" + freq[ch - 'a']);
            freq[ch - 'a'] = -1;
        }
    }
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();

    int[] freq = frequencyArray(str);
    printFrequency(str, freq);
}

```

$O(n)$

$\hookrightarrow str.length()$

+

$O(n)$

Total time Comp

$$= O(n) + O(n) = 2 \cdot O(n)$$

$= O(n)$ linear

Max frequency character

| | | | | | | | | | | | | |
|--------------|--------------|------------|--------------|--------------|--------------|--------------|------------|--------------|--------------|--------------|--------------|--------------|
| \downarrow | \downarrow | \uparrow | \downarrow | \downarrow | \downarrow | \downarrow | \uparrow | \downarrow | \downarrow | \downarrow | \downarrow | \downarrow |
| 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 1 | 1 | 0 | 0 | 2 | 2 | 4 | 2 | 0 | 0 | 0 | 0 | 0 |

\uparrow \uparrow

\downarrow
 " datastructures n t lgo

ch - null 'd' 'r' 'f' 'l'
 max freq - d f r l

```

public static int[] frequencyArray(String str){
    int[] freq = new int[26];

    for(int idx=0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        freq[ch - 'a']++;
    }

    return freq;
}

public static char maxFrequency(int[] freq){
    char res = 'N';
    int maxFreq = 0;

    for(int idx = 0; idx < 26; idx++){
        if(freq[idx] > maxFreq){
            maxFreq = freq[idx];
            res = (char)(idx + 'a');
        }
    }

    return res;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();

    int[] freq = frequencyArray(str);
    System.out.println(maxFrequency(freq));
}

```

$O(n)$

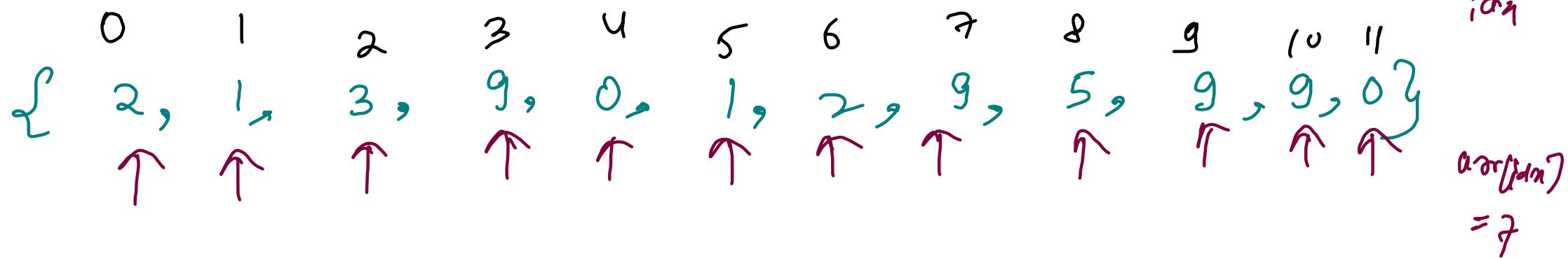
+

$O(26) = O(1)$

~~Total time comp~~

$O(n+26) = O(n)$

Non-frequency of Digs



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 4 |
|---|---|---|---|---|---|---|---|---|---|

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |

$$\text{freq} = \cancel{-1}/\cancel{0}, \quad \text{freq} = \cancel{0}/24$$

```

public static int[] frequencyArray(int[] arr){
    int[] freq = new int[10];

    for(int idx=0; idx < arr.length; idx++){
        freq[arr[idx]]++;
    }

    return freq;
}

public static int maxFrequency(int[] freq){
    int res = 0;

    for(int idx = 0; idx < 10; idx++){
        if(freq[idx] > freq[res])
            res = idx;
    }

    return res;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];
    for(int idx = 0; idx < size; idx++){
        arr[idx] = scn.nextInt();
    }

    int[] freq = frequencyArray(arr);
    System.out.println(maxFrequency(freq));
}

```

$O(n)$

+

| | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|---|
| | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| | 2 | 2 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 4 |

$O(10) \approx O(1)$

Time

$O(n+10) \approx O(n)$

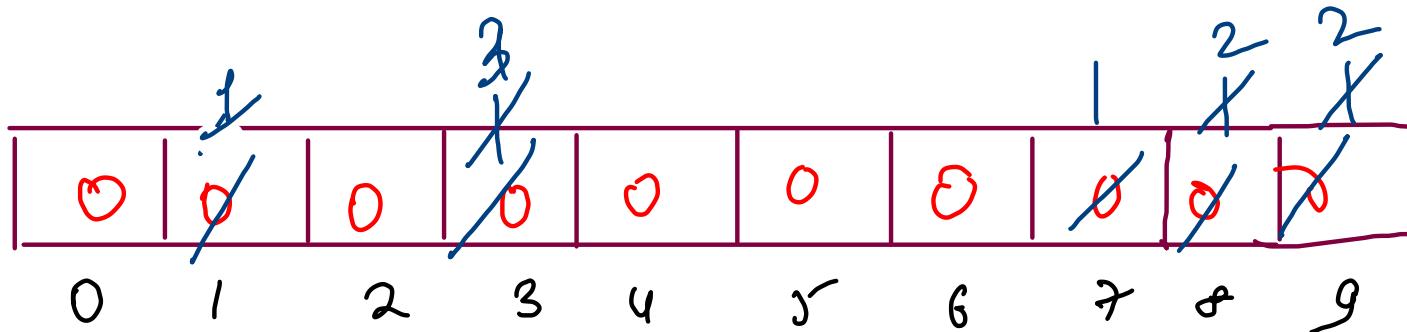
Linear

Extra Space \rightarrow Constant
 $O(1)$

Digit with max Frequency

integer \Rightarrow

931933788
• • • • • • •



frequency Array

```

public static int[] frequencyArray(int val){
    int[] freq = new int[10];

    while(val > 0){
        freq[val % 10]++;
        val = val / 10;
    }

    return freq;
}

public static int maxFrequency(int[] freq){
    int res = 0;

    for(int idx = 0; idx < 10; idx++){
        if(freq[idx] > freq[res])
            res = idx;
    }

    return res;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int val = scn.nextInt();

    int[] freq = frequencyArray(val);
    System.out.println(maxFrequency(freq));
}

```

$$O(\log_{10} n)$$

+

$$O(10) = O(1)$$

$$O(\log_{10} n) \approx O(1)$$

$$\text{int} \rightarrow 2^{31}-1 \approx 10^{10}$$

10 digits

$$\text{long} \rightarrow 2^{63}-1 \approx 10^{18}$$

18 digits

Hucky Integer

| | | | | | | | | | | |
|---|---|---|---|---|---|----|---|---|---|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 1 | 4 | 2 | 4 | 4 | 30 | 4 | 0 | 5 | 100 |

value frequency

✓ 1 = 1

✓ 2 = 2

✗ 3 ≠ 0

✓ 4 = 4 answer

✗ 5 ≠ 1

size = N

max^m frequency = n

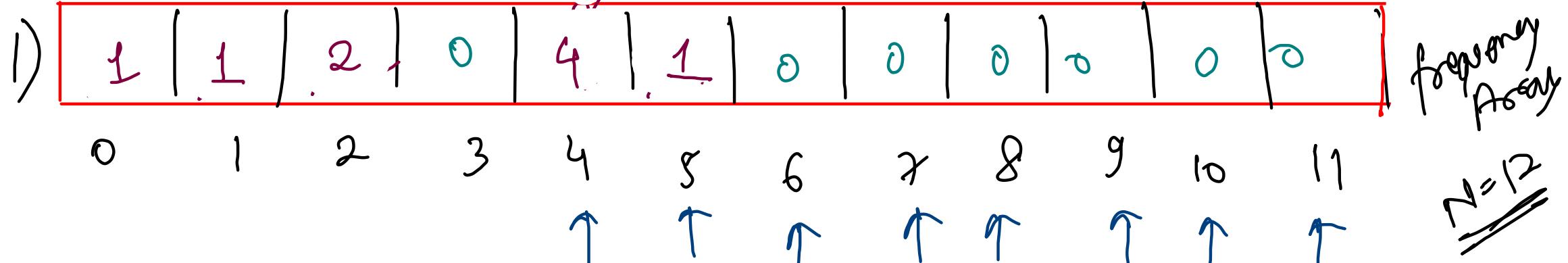
Largest hucky No = π

atmosr

size of frequency

array = $n+1$

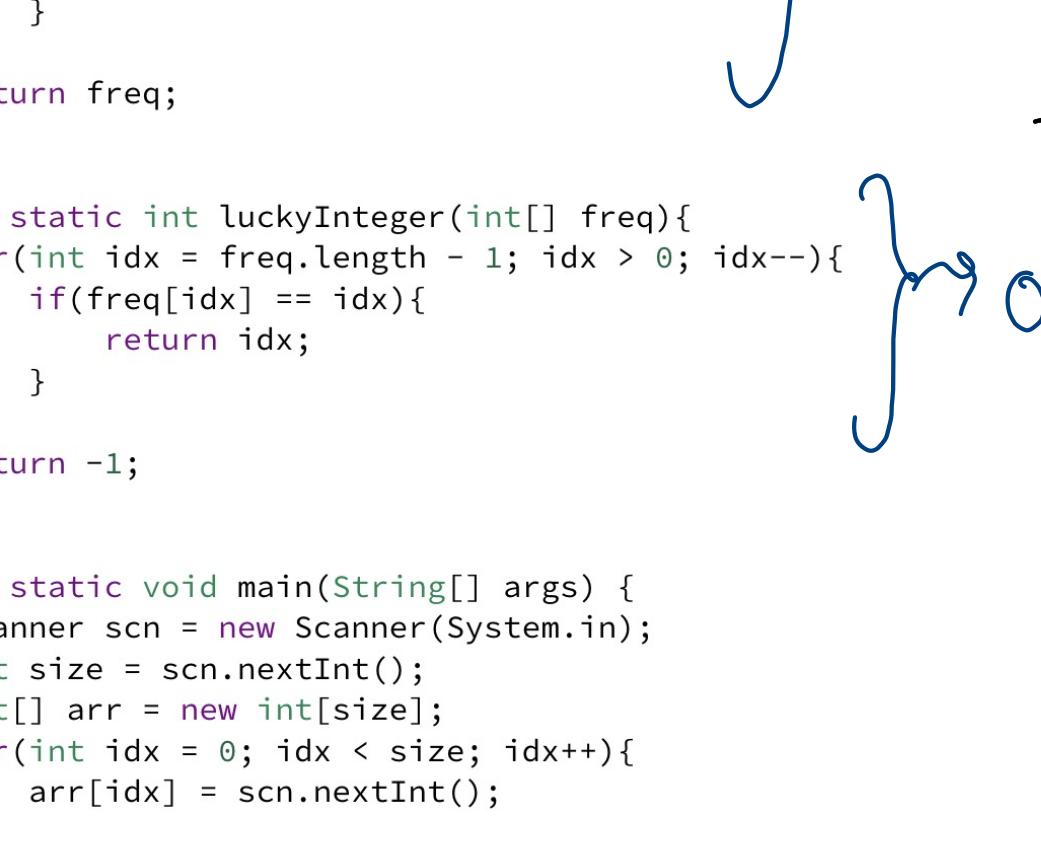
| | | | | | | | | | | | |
|---|---|---|---|---|---|----|---|---|---|-----|---------------------------------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | <u><u>$N=11$</u></u> |
| 2 | 1 | 4 | 2 | 4 | 4 | 30 | 4 | 0 | 5 | 100 | |
| ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | |



2) largest hacky integer (value = freq[value])

Traverse from right to left

```
public static int[] frequencyArray(int[] arr){  
    int[] freq = new int[arr.length + 1];  
    for(int idx = 0; idx < arr.length; idx++){  
        if(arr[idx] <= arr.length){  
            freq[arr[idx]]++;  
        }  
    }  
    return freq;  
}  
  
public static int luckyInteger(int[] freq){  
    for(int idx = freq.length - 1; idx > 0; idx--){  
        if(freq[idx] == idx){  
            return idx;  
        }  
    }  
    return -1;  
}  
  
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int size = scn.nextInt();  
    int[] arr = new int[size];  
    for(int idx = 0; idx < size; idx++){  
        arr[idx] = scn.nextInt();  
    }  
  
    int[] freq = frequencyArray(arr);  
    System.out.println(luckyInteger(freq));  
}
```



The code consists of three parts: a frequency array function, a lucky integer function, and a main function for input. The frequency array function has a time complexity of $O(n)$, indicated by a curly brace and the handwritten label $O(n)$. The lucky integer function also has a time complexity of $O(n)$, indicated by another curly brace and the handwritten label $O(n)$. The main function has a time complexity of $O(n)$ due to its loop, indicated by a curly brace and the handwritten label $O(n)$.

| | | | | | |
|---|---|---|---|---|---|
| 6 | 6 | 6 | 6 | 6 | 6 |
| 0 | 0 | 0 | 0 | 0 | 6 |
| 0 | 1 | 2 | 3 | 4 | 5 |

outer loop
f

2D Arrays or Matrix

| | c0 | c1 | c2 | c3 | c4 |
|----|------|------|------|------|------|
| r0 | r0c0 | r0c1 | r0c2 | r0c3 | r0c4 |
| r1 | r1c0 | r1c1 | r1c2 | r1c3 | r1c4 |
| r2 | r2c0 | r2c1 | r2c2 | r2c3 | r2c4 |
| r3 | r3c0 | r3c1 | r3c2 | r3c3 | r3c4 |
| r4 | r4c0 | r4c1 | r4c2 | r4c3 | r4c4 |
| r5 | r5c0 | r5c1 | r5c2 | r5c3 | r5c4 |

1D array
int []

linear traversal
row = new int[5];

int [][] mat = new int[6][5];

2d array

matrix

2 loops

forwards

no of rows no of columns

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();

    int[][] mat = new int[rows][cols];
    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            System.out.print(mat[row][col] + " ");
        }
        System.out.println();
    }
}

```

Input space
 $O(rows * cols)$
 $\leq O(n^2)$

Inputting a matrix → $O(rows * cols)$
 $= O(n^2)$

Outputting a matrix → $O(rows * cols)$
 $= O(n^2)$

+

```

import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int rows = scn.nextInt();
        int cols = scn.nextInt();

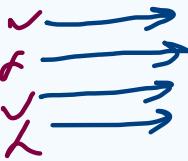
        int[][] mat = new int[rows][cols];

        for(int row = 0; row < rows; row++){
            for(int col = 0; col < cols; col++){
                mat[row][col] = scn.nextInt();
            }
        }

        for(int row = 0; row < rows; row=row+2){
            for(int col = 0; col < cols; col++){
                System.out.print(mat[row][col] + " ");
            }
            System.out.println();
        }
    }
}

```

Alternate Rows



```

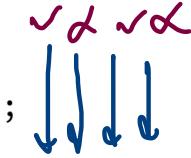
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();

    int[][] mat = new int[rows][cols];

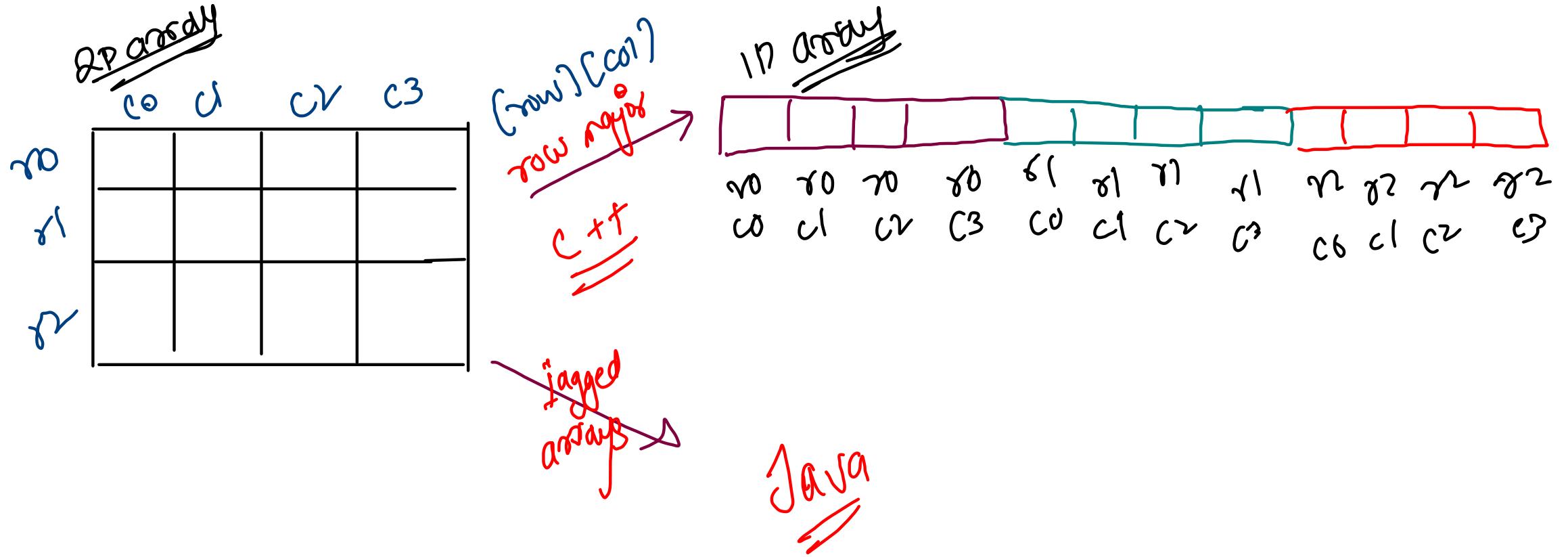
    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int col = 0; col < cols; col=col+2){
        for(int row = 0; row < rows; row++){
            System.out.print(mat[row][col] + " ");
        }
        System.out.println();
    }
}

```



Alternate Col



Upper Triangle matrix

| | c_0 | c_1 | c_2 | c_3 | c_4 | |
|-------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|--|
| r_0 | 0_0 | 0_1 | 0_2 | 0_3 | 0_4 | |
| r_1 | 1_0 | 1_1 | 1_2 | 1_3 | 1_4 | |
| r_2 | 2_0 | 2_1 | 2_2 | 2_3 | 2_4 | |
| r_3 | 3_0 | 3_1 | 3_2 | 3_3 | 3_4 | |
| r_4 | 4_0 | 4_1 | 4_2 | 4_3 | 4_4 | |
| r_5 | 5_0 | 5_1 | 5_2 | 5_3 | 5_4 | |

upper right triangle

↑

$row L = c_0$

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();

    int[][] mat = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            if(row <= col)
                System.out.print(mat[row][col] + " ");
            else System.out.print(0 + " ");
        }
        System.out.println();
    }
}
```

| | c_0 | c_1 | c_2 | c_3 | c_4 |
|-------|-------|-------|-------|-------|-------|
| r_0 | 00 | 01 | 02 | 03 | 04 |
| r_1 | 10 | 11 | 12 | 13 | 14 |
| r_2 | 20 | 21 | 22 | 23 | 24 |
| r_3 | 30 | 31 | 32 | 33 | 34 |
| r_4 | 40 | 41 | 42 | 43 | 44 |
| r_5 | 50 | 51 | 52 | 53 | 54 |

Lower Left Triangle

Upper Right
Triangle
 $\Rightarrow 0$

$m[m] = col$

Lower Triangular Matrix

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();
    int[][] mat = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            if(col <= row){
                System.out.print(mat[row][col] + " "); // lower left including diagonal
            } else {
                System.out.print(0 + " "); // upper right
            }
        }
        System.out.println();
    }
}
```

Row by Row Wave
Traversal

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ | |
|----------------|------------------|------------------|------------------|------------------|------------------|------|
| r ₀ | r _{0c0} | r _{0c1} | r _{0c2} | r _{0c3} | r _{0c4} | Even |
| r ₁ | r _{1c0} | r _{1c1} | r _{1c2} | r _{1c3} | r _{1c4} | Odd |
| r ₂ | r _{2c0} | r _{2c1} | r _{2c2} | r _{2c3} | r _{2c4} | Even |
| r ₃ | r _{3c0} | r _{3c1} | r _{3c2} | r _{3c3} | r _{3c4} | Odd |
| r ₄ | r _{4c0} | r _{4c1} | r _{4c2} | r _{4c3} | r _{4c4} | Even |
| r ₅ | r _{5c0} | r _{5c1} | r _{5c2} | r _{5c3} | r _{5c4} | Odd |

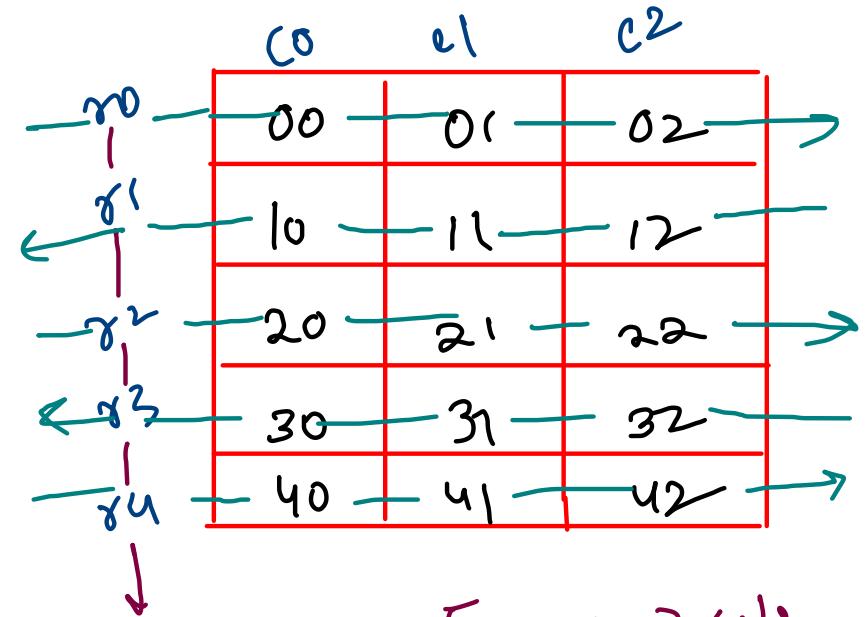
```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();
    int[][] mat = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int row = 0; row < rows; row++){
        if(row % 2 == 0){ → even row
            for(int col = 0; col < cols; col++){ → left to right
                System.out.print(mat[row][col] + " ");
            }
        } else { → odd row
            for(int col = cols - 1; col >= 0; col--){ → right to left
                System.out.print(mat[row][col] + " ");
            }
        }
        System.out.println();
    }
}

```



5 rows, 3 cols

00 01 02
12 11 10
20 21 22

Print col by col in many
order

| | c0 | c1 | c2 | c3 | c4 |
|----|------|------|------|------|------|
| r0 | r0c0 | r0c1 | r0c2 | r0c3 | r0c4 |
| r1 | r1c0 | r1c1 | r1c2 | r1c3 | r1c4 |
| r2 | r2c0 | r2c1 | r2c2 | r2c3 | r2c4 |
| r3 | r3c0 | r3c1 | r3c2 | r3c3 | r3c4 |
| r4 | r4c0 | r4c1 | r4c2 | r4c3 | r4c4 |
| r5 | r5c0 | r5c1 | r5c2 | r5c3 | r5c4 |

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();
    int[][] mat = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    for(int col = 0; col < cols; col++){
        if(col % 2 == 0){ → even col
            for(int row = 0; row < rows; row++){ → top to bottom
                System.out.print(mat[row][col] + " ");
            }
        } else { → odd col
            for(int row = rows - 1; row >= 0; row--){ → bottom to top
                System.out.print(mat[row][col] + " ");
            }
        }
        System.out.println();
    }
}

```

→ Column by column

→ top to bottom

→ bottom to top

Time $\Rightarrow \mathcal{O}(n^2)$

Space \Rightarrow

① New Space $\Rightarrow \mathcal{O}(n^2)$

~~② Extra Space $\Rightarrow \mathcal{O}(1)$~~

Identical Matrices

2×3
mat₁

| | | |
|----|----|----|
| 10 | 20 | 30 |
| 40 | 50 | 60 |

3×2
mat₂

No

\neq

| | |
|----|----|
| 10 | 40 |
| 20 | 50 |
| 30 | 60 |

- ① 2×4
 $row_1 \neq row_2$
 5^{th}
- ② $col_1 \neq col_2$
 3×4

| | c ₀ | c ₁ | c ₂ |
|----------------|----------------|----------------|----------------|
| r ₀ | 10 | 20 | 30 |
| r ₁ | 40 | 50 | 100 |
| r ₂ | 70 | 80 | 90 |
| r ₃ | 100 | 110 | 120 |

| | c ₀ | c ₁ | c ₂ |
|----------------|----------------|----------------|----------------|
| r ₀ | 10 | 20 | 30 |
| r ₁ | 40 | 50 | 100 |
| r ₂ | 70 | 80 | 90 |
| r ₃ | 100 | 110 | 120 |

\neq

```
public static String identical(int[][] mat1, int[][] mat2,
                             int r1, int c1, int r2, int c2){
    if(r1 != r2 || c1 != c2) return "Not Same";

    for(int row = 0; row < r1; row++){
        for(int col = 0; col < c1; col++){
            if(mat1[row][col] != mat2[row][col])
                return "Not Same";
        }
    }

    return "Same";
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    // Inputting First Matrix
    int r1 = scn.nextInt();
    int c1 = scn.nextInt();
    int[][] mat1 = new int[r1][c1];
    for(int row=0; row<r1; row++){
        for(int col=0; col<c1; col++){
            mat1[row][col] = scn.nextInt();
        }
    }

    // Inputting Second Matrix
    int r2 = scn.nextInt();
    int c2 = scn.nextInt();
    int[][] mat2 = new int[r2][c2];
    for(int row=0; row<r2; row++){
        for(int col=0; col<c2; col++){
            mat2[row][col] = scn.nextInt();
        }
    }
}

System.out.println(identical(mat1, mat2, r1, c1, r2, c2));
}
```

2×3
mat 1

| | | |
|----|----|----|
| 10 | 20 | 30 |
| 40 | 50 | 60 |

3×2
mat 2

+

| | |
|----|----|
| 10 | 40 |
| 20 | 50 |
| 30 | 60 |

Print -1

Addition Not possible

| | c0 | c1 | c2 |
|----|-----|-----|-----|
| r0 | 10 | 20 | 30 |
| r1 | 40 | 50 | 100 |
| r2 | 20 | 80 | 90 |
| r3 | 100 | 110 | 120 |

mat 1

+

| | c0 | c1 | c2 |
|----|-----|----|----|
| r0 | 30 | 10 | 20 |
| r1 | 40 | 50 | 10 |
| r2 | 60 | 70 | 90 |
| r3 | 100 | 10 | 20 |

mat 2

=

| | c0 | c1 | c2 |
|----|-----------------|---------------|-----------------|
| r0 | $10+30 = 40$ | $20+10 = 30$ | $30+20 = 50$ |
| r1 | $40+40 = 80$ | $50+50 = 100$ | $10+10 = 110$ |
| r2 | $60+60 = 120$ | $70+70 = 140$ | $90+90 = 180$ |
| r3 | $100+100 = 200$ | $10+10 = 20$ | $120+120 = 240$ |

yes (3rd)

```
public static void add2Matrices(int[][] mat1, int[][] mat2, int r1, int c1, int r2, int c2){  
    if(r1 != r2 || c1 != c2) {  
        System.out.println(-1);  
        return;  
    }  
  
    int[][] res = new int[r1][c1];  
    for(int row = 0; row < r1; row++){  
        for(int col = 0; col < c1; col++){  
            res[row][col] = mat1[row][col] + mat2[row][col];  
            System.out.print(res[row][col] + " ");  
        }  
        System.out.println();  
    }  
}
```

Interchange Row & Column

| | c_0 | c_1 | c_2 | c_3 | c_4 |
|-------|-----------|-----------|-----------|-----------|-----------|
| r_0 | $r_0 c_0$ | $r_0 c_1$ | $r_0 c_2$ | $r_0 c_3$ | $r_0 c_4$ |
| r_1 | $r_1 c_0$ | $r_1 c_1$ | $r_1 c_2$ | $r_1 c_3$ | $r_1 c_4$ |
| r_2 | $r_2 c_0$ | $r_2 c_1$ | $r_2 c_2$ | $r_2 c_3$ | $r_2 c_4$ |
| r_3 | $r_3 c_0$ | $r_3 c_1$ | $r_3 c_2$ | $r_3 c_3$ | $r_3 c_4$ |
| r_4 | $r_4 c_0$ | $r_4 c_1$ | $r_4 c_2$ | $r_4 c_3$ | $r_4 c_4$ |
| r_5 | $r_5 c_0$ | $r_5 c_1$ | $r_5 c_2$ | $r_5 c_3$ | $r_5 c_4$ |



| | c_0 | c_1 | c_2 | c_3 | c_4 |
|-------|-----------|-----------|-----------|-----------|-----------|
| r_0 | $r_5 c_0$ | $r_5 c_1$ | $r_5 c_2$ | $r_5 c_3$ | $r_5 c_4$ |
| r_1 | $r_1 c_0$ | $r_1 c_1$ | $r_1 c_2$ | $r_1 c_3$ | $r_1 c_4$ |
| r_2 | $r_2 c_0$ | $r_2 c_1$ | $r_2 c_2$ | $r_2 c_3$ | $r_2 c_4$ |
| r_3 | $r_3 c_0$ | $r_3 c_1$ | $r_3 c_2$ | $r_3 c_3$ | $r_3 c_4$ |
| r_4 | $r_4 c_0$ | $r_4 c_1$ | $r_4 c_2$ | $r_4 c_3$ | $r_4 c_4$ |
| r_5 | $r_0 c_0$ | $r_0 c_1$ | $r_0 c_2$ | $r_0 c_3$ | $r_0 c_4$ |

```

public static void interchangeMat(int[][] mat, int rows, int cols){
    for(int col = 0; col < cols; col++){
        // Swap Elements From First Row with Last Row for Every Column
        int temp = mat[0][col];
        mat[0][col] = mat[rows - 1][col];
        mat[rows - 1][col] = temp;
    }
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int rows = scn.nextInt();
    int cols = scn.nextInt();

    int[][] mat = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    interchangeMat(mat, rows, cols);

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            System.out.print(mat[row][col] + " ");
        }
        System.out.println();
    }
}

```

Fixing a row k
 ↓
 $[k][col]$
 Iterating over columns

⚡
 Fixing a column k
 ↓
 $[row][k]$
 Iterating over rows

Each row of
Reverse Matrix of A of Square Matrix

| | c_0 | c_1 | c_2 | c_3 | c_4 |
|-------|-------|-------|-------|-------|-------|
| r_0 | 00 | 01 | 02 | 03 | 04 |
| r_1 | 10 | 11 | 12 | 13 | 14 |
| r_2 | 20 | 21 | 22 | 23 | 24 |
| r_3 | 30 | 31 | 32 | 33 | 34 |
| r_4 | 40 | 41 | 42 | 43 | 44 |

| | c_0 | c_1 | c_2 | c_3 | c_4 |
|-------|-------|-------|-------|-------|-------|
| r_0 | 04 | 03 | 02 | 01 | 00 |
| r_1 | 14 | 13 | 12 | 11 | 10 |
| r_2 | 24 | 23 | 22 | 21 | 20 |
| r_3 | 34 | 33 | 32 | 31 | 30 |
| r_4 | 44 | 43 | 42 | 41 | 40 |

Traverse on each row & reverse it!

```

public static void reverseRows(int[][] mat, int n){

    // Iterate on all rows
    for(int row = 0; row < n; row++){
        int left = 0, right = n - 1;

        // reverse the given row
        while(left < right){
            // Swap (row, left) with (row, right)
            int temp = mat[row][left];
            mat[row][left] = mat[row][right];
            mat[row][right] = temp;

            left++; right--;
        }
    }
}

```

Reverse Each Column

((left, col) with (right, col))

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    int n = scn.nextInt();
    int[][] mat = new int[n][n];
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    reverseRows(mat, n);
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            System.out.print(mat[row][col] + "\t");
            // '\t' is tabspace character
        }
        System.out.println();
    }
}

```

Reverse Rows & columns

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ₀ | 00 | 01 | 02 | 03 | 04 |
| r ₁ | 10 | 11 | 12 | 13 | 14 |
| r ₂ | 20 | 21 | 22 | 23 | 24 |
| r ₃ | 30 | 31 | 32 | 33 | 34 |
| r ₄ | 40 | 41 | 42 | 43 | 44 |

Reverse
Rows

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ₀ | 04 | 03 | 02 | 01 | 00 |
| r ₁ | 14 | 13 | 12 | 11 | 10 |
| r ₂ | 24 | 23 | 22 | 21 | 20 |
| r ₃ | 34 | 33 | 32 | 31 | 30 |
| r ₄ | 44 | 43 | 42 | 41 | 40 |

Single step

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ₀ | 44 | 43 | 42 | 41 | 40 |
| r ₁ | 34 | 33 | 32 | 31 | 30 |
| r ₂ | 24 | 23 | 22 | 21 | 20 |
| r ₃ | 14 | 13 | 12 | 11 | 10 |
| r ₄ | 04 | 03 | 02 | 01 | 00 |

→
Reverse
columns

Transpose of Matrix (always square)

rows \rightarrow columns

columns \rightarrow rows

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ₀ | 00 | 01 | 02 | 03 | 04 |
| r ₁ | 10 | 11 | 12 | 13 | 14 |
| r ₂ | 20 | 21 | 22 | 23 | 24 |
| r ₃ | 30 | 31 | 32 | 33 | 34 |
| r ₄ | 40 | 41 | 42 | 43 | 44 |

| | c ₀ | c ₁ | c ₂ | c ₃ | c ₄ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ₀ | 00 | 10 | 20 | 30 | 40 |
| r ₁ | 01 | 11 | 21 | 31 | 41 |
| r ₂ | 02 | 12 | 22 | 32 | 42 |
| r ₃ | 03 | 13 | 23 | 33 | 43 |
| r ₄ | 04 | 14 | 24 | 34 | 44 |

| | c ⁰ | c ¹ | c ² | c ³ | c ⁴ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ⁰ | 00 ✓ | 01 ✓ | 02 ✓ | 03 ✓ | 04 ✓ |
| r ¹ | 10 ✓ | 11 ✓ | 12 ✓ | 13 ✓ | 14 ✓ |
| r ² | 20 ✓ | 21 ✓ | 22 ✓ | 23 ✓ | 24 ✓ |
| r ³ | 30 ✓ | 31 ✓ | 32 ✓ | 33 ✓ | 34 ✓ |
| r ⁴ | 40 ✓ | 41 ✓ | 42 ✓ | 43 ✓ | 44 ✓ |

| | c ⁰ | c ¹ | c ² | c ³ | c ⁴ |
|----------------|----------------|----------------|----------------|----------------|----------------|
| r ⁰ | 00 | 10 | 20 | 30 | 40 |
| r ¹ | 01 | 11 | 21 | 31 | 41 |
| r ² | 02 | 12 | 22 | 32 | 42 |
| r ³ | 03 | 13 | 23 | 33 | 43 |
| r ⁴ | 04 | 14 | 24 | 34 | 44 |

```
public static void transpose(int[][] mat, int n){
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            int temp = mat[row][col];
            mat[row][col] = mat[col][row];
            mat[col][row] = temp;
        }
    }
}
```

Upper right

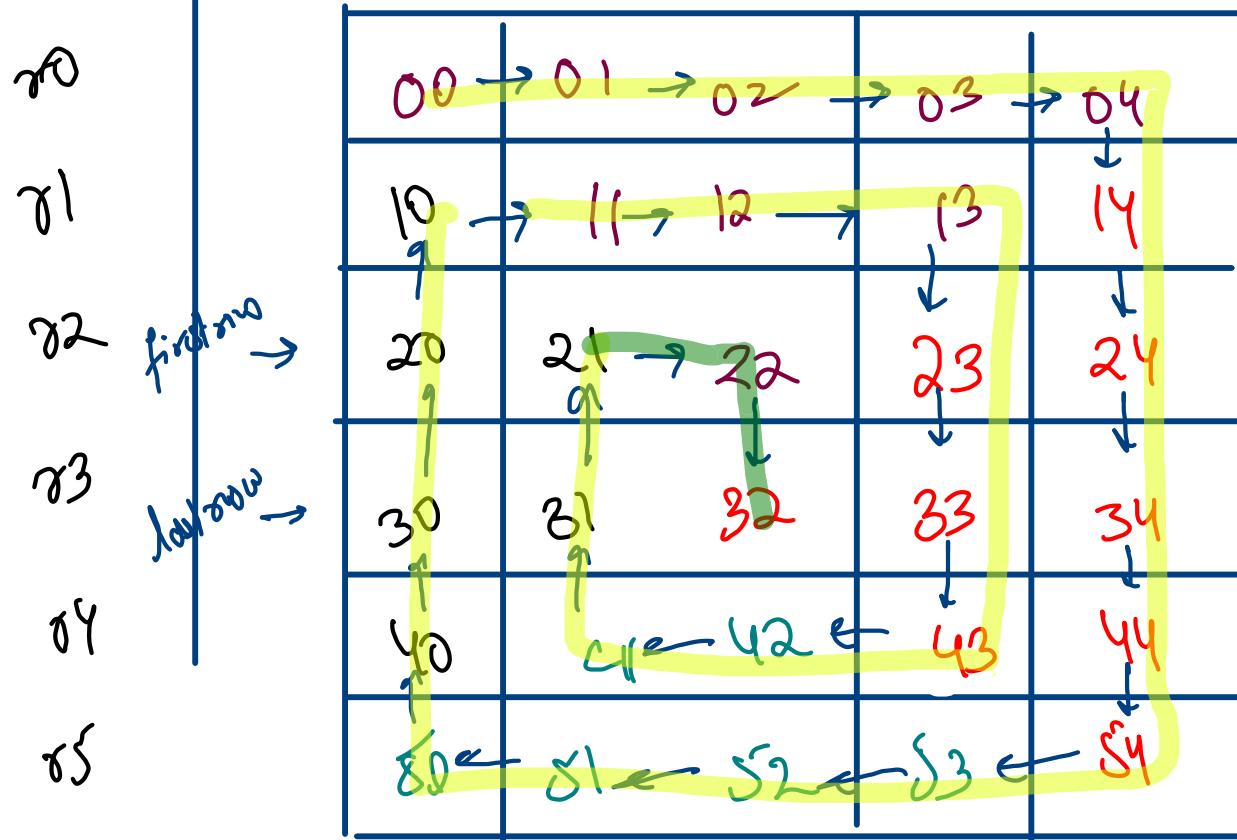
```
public static void transpose(int[][] mat, int n){
    for(int row = 0; row < n; row++){
        for(int col = 0; col < row; col++){
            int temp = mat[row][col];
            mat[row][col] = mat[col][row];
            mat[col][row] = temp;
        }
    }
}
```

Lower left

Spiral Matrix Traversal

c0 c1 c2 c3 c4

firstCol



```

public static void spiralMatrix(int[][] mat, int rows, int cols){
    int firstRow = 0, lastRow = rows - 1;
    int firstCol = 0, lastCol = cols - 1;

    // TOP WALL: Left to Right
    for(int col = firstCol; col <= lastCol; col++){
        System.out.print(mat[firstRow][col] + "\t");
    }
    firstRow++; // SKIP THE TOP RIGHT CORNER

    // RIGHT WALL: Top to Bottom
    for(int row = firstRow; row <= lastRow; row++){
        System.out.print(mat[row][lastCol] + "\t");
    }
    lastCol--; // SKIP THE BOTTOM RIGHT CORNER

    // BOTTOM WALL: Right to Left
    for(int col = lastCol; col >= firstCol; col--){
        System.out.print(mat[lastRow][col] + "\t");
    }
    lastRow--; // SKIP THE BOTTOM LEFT CORNER

    // LEFT WALL: Bottom to Top
    for(int row = lastRow; row >= firstRow; row--){
        System.out.print(mat[row][firstCol] + "\t");
    }
}

```

```

int firstRow = 0, lastRow = rows - 1;
int firstCol = 0, lastCol = cols - 1;

while(firstRow <= lastRow && firstCol <= lastCol){
    // TOP WALL: Left to Right
    for(int col = firstCol; col <= lastCol; col++){
        System.out.print(mat[firstRow][col] + "   ");
    }

    firstRow++; // SKIP THE TOP RIGHT CORNER

    if(firstRow > lastRow || firstCol > lastCol) break;

    // RIGHT WALL: Top to Bottom
    for(int row = firstRow; row <= lastRow; row++){
        System.out.print(mat[row][lastCol] + "   ");
    }

    lastCol--; // SKIP THE BOTTOM RIGHT CORNER

    if(firstRow > lastRow || firstCol > lastCol) break;

    // BOTTOM WALL: Right to Left
    for(int col = lastCol; col >= firstCol; col--){
        System.out.print(mat[lastRow][col] + "   ");
    }

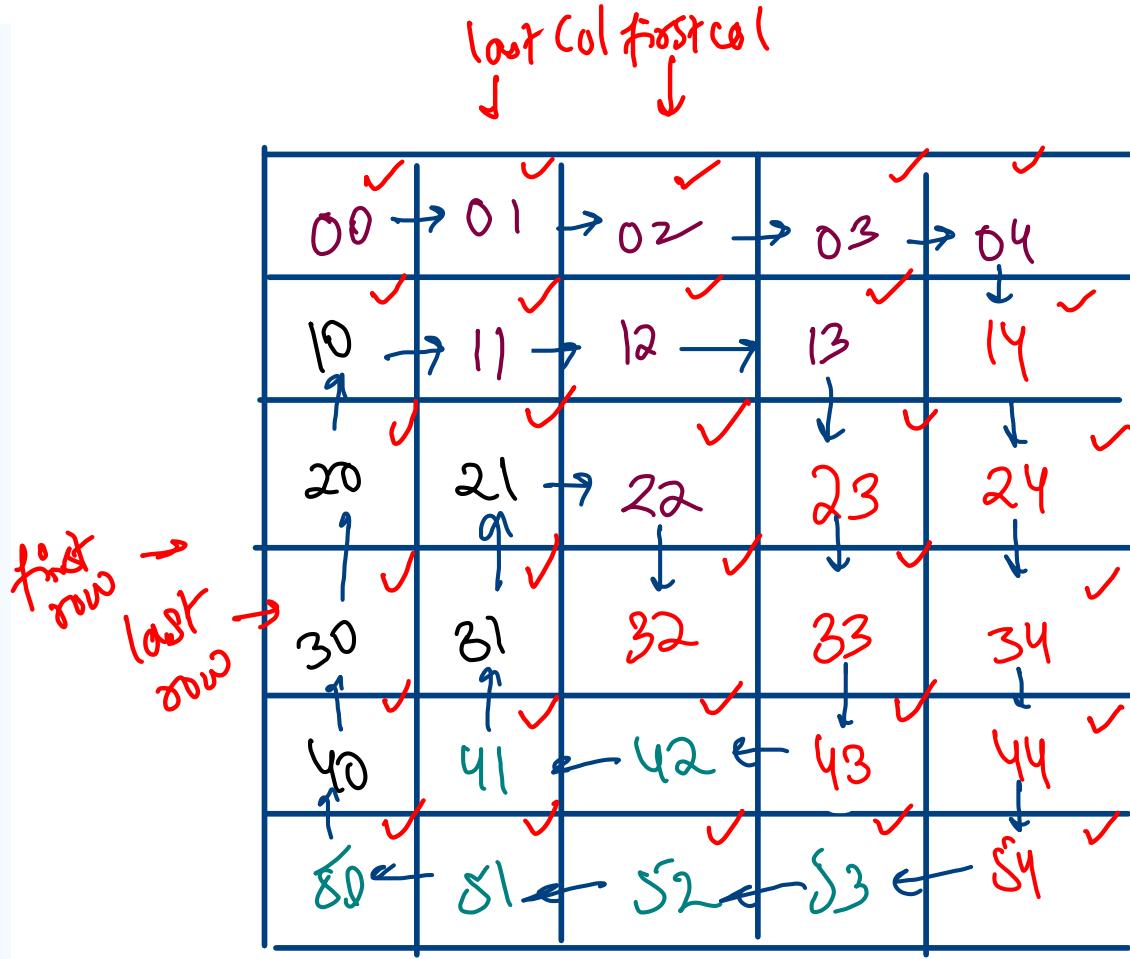
    lastRow--; // SKIP THE BOTTOM LEFT CORNER

    if(firstRow > lastRow || firstCol > lastCol) break;

    // LEFT WALL: Bottom to Top
    for(int row = lastRow; row >= firstRow; row--){
        System.out.print(mat[row][firstCol] + "   ");
    }

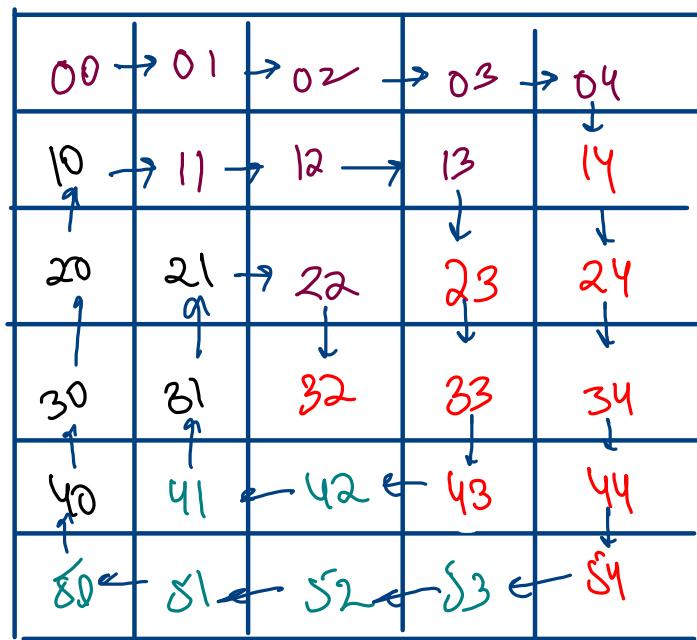
    firstCol++; // SKIP THE TOP LEFT CORNER
}

```

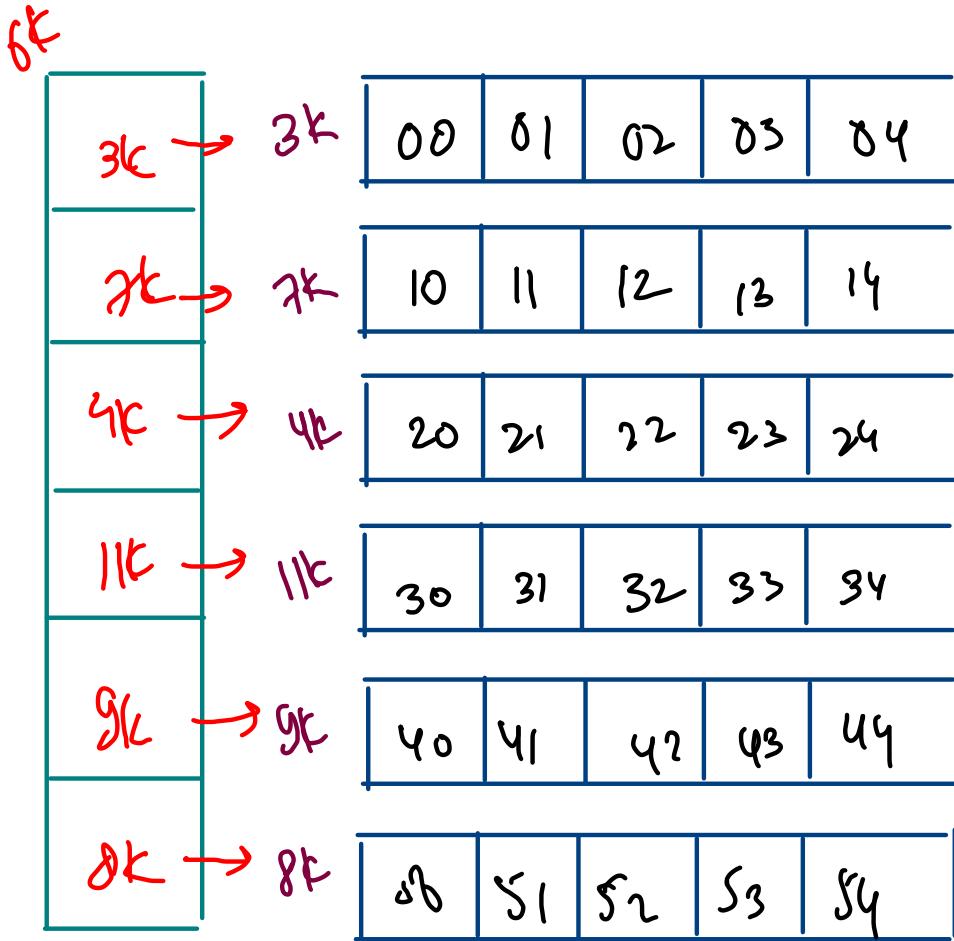


Memory Mapping of 2D Arrays

`int[][] mat = new int[6][5];`



`mat = 6K`



matrix of 6 rows * 5 columns
6 rows of 5 columns each
→ 6 arrays of 5 elements
→ no of 1D arrays = no of rows
size of 1D array = no of columns

$\text{mat} = \text{fk}$ Stack variable
(reference variable)

| | | | | | | | |
|-------|-----------------------|--|----|----|--|----|----|
| $6k$ | $3k \rightarrow 3k$ | <table border="1"> <tr><td>00</td><td>01</td><td>02</td><td>03</td><td>04</td></tr> </table> | 00 | 01 | 02 | 03 | 04 |
| 00 | 01 | 02 | 03 | 04 | | | |
| | $7k \rightarrow 7k$ | <table border="1"> <tr><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td></tr> </table> | 10 | 11 | 12 | 13 | 14 |
| 10 | 11 | 12 | 13 | 14 | | | |
| | $4k \rightarrow 4k$ | <table border="1"> <tr><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td></tr> </table> | 20 | 21 | 22 | 23 | 24 |
| 20 | 21 | 22 | 23 | 24 | | | |
| $11k$ | $11k \rightarrow 11k$ | <table border="1"> <tr><td>30</td><td>31</td><td>32</td><td>33</td><td>34</td></tr> </table> | 30 | 31 | 32 | 33 | 34 |
| 30 | 31 | 32 | 33 | 34 | | | |
| | $9k \rightarrow 9k$ | <table border="1"> <tr><td>40</td><td>41</td><td>42</td><td>43</td><td>44</td></tr> </table> | 40 | 41 | 42 | 43 | 44 |
| 40 | 41 | 42 | 43 | 44 | | | |
| | $8k \rightarrow 8k$ | <table border="1"> <tr><td>s0</td><td>s1</td><td>s2</td><td>s3</td><td>s4</td></tr> </table> | s0 | s1 | s2 | s3 | s4 |
| s0 | s1 | s2 | s3 | s4 | | | |

$\text{mat}[i][j]$ Rows = ?

eg $\text{mat}[3][4]$ Cols =

$$= 6k[3][4]$$

$$= 11k[4]$$

$$= 34$$

eg $\text{mat}[5][2]$

$$6k[5][2] = 8k[2] \\ = s2$$

```

public static void main(String[] args){
    int[] arr = new int[5];

    System.out.println(arr.length);

    int[][] mat = new int[6][4];

    System.out.println("Rows : " + mat.length);
    System.out.println("Cols : " + mat[0].length);

}

```

Jagged Arrays

C++



Java

$\text{arr}[6] = 1000;$

```

int[][] mat = new int[6][];
mat[0] = new int[5];
mat[1] = new int[10];
mat[2] = new int[3];
mat[3] = new int[4];
mat[4] = new int[1];
mat[5] = new int[2];

for(int row = 0; row < mat.length; row++){
    for(int col = 0; col < mat[row].length; col++){
        System.out.print(mat[row][col] + " ");
    }
    System.out.println();
}

```

Finished in 109 ms

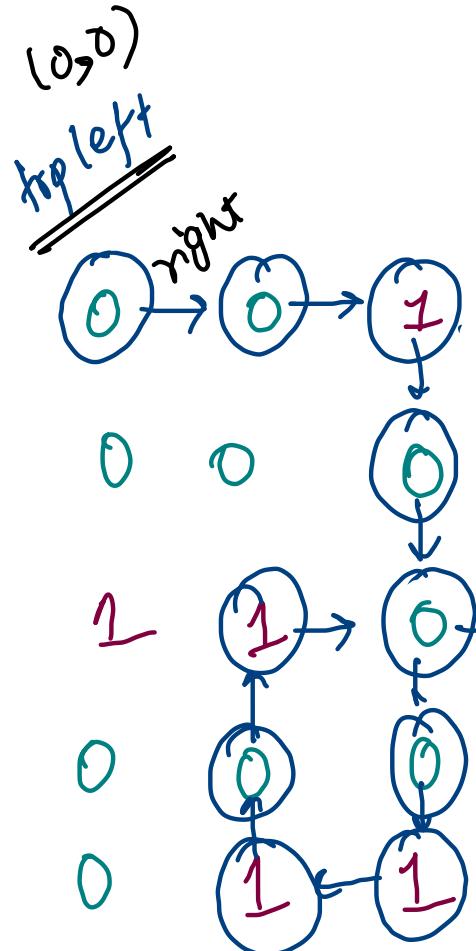
```

5
0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0
0 0 0 0
0
0 0

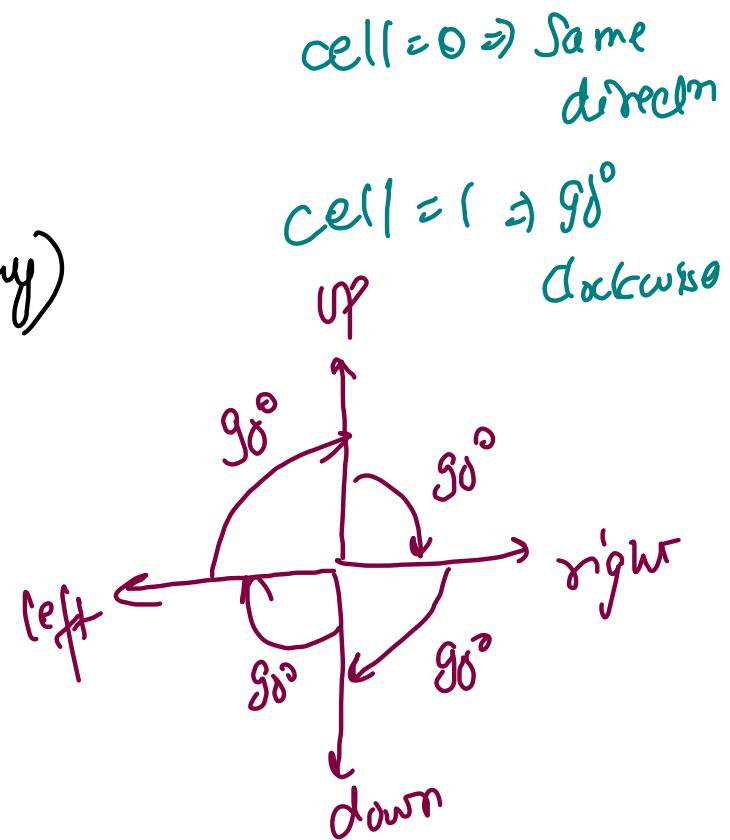
```

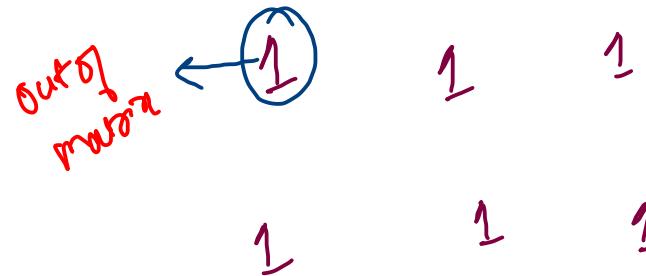
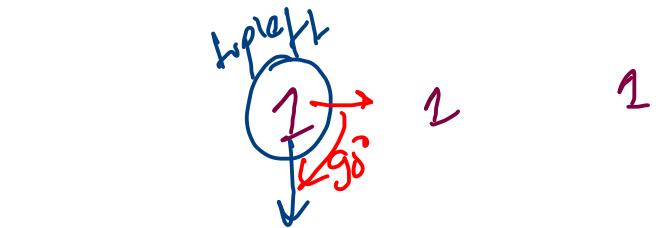
Make DS on GFG & Leetcode

&& Import DSA bookmark



Exit Point of Matrix (Binary)





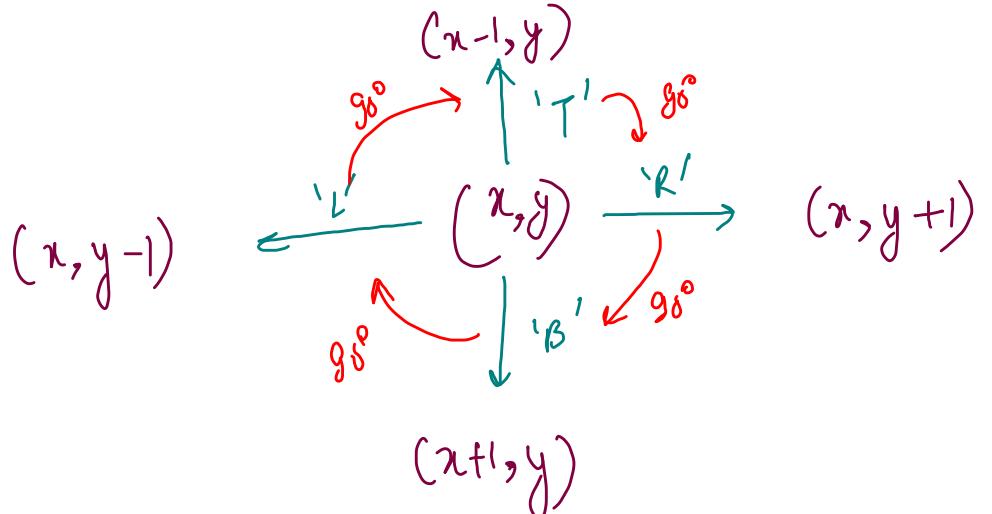
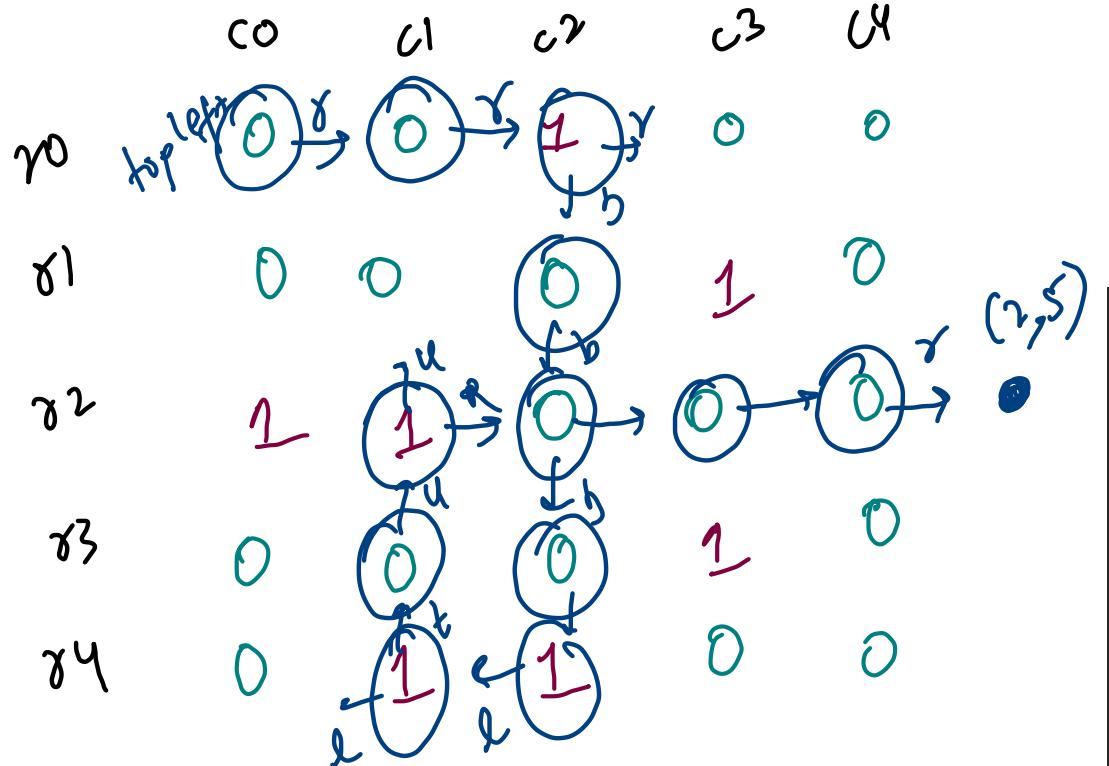
There will always be a exit point!

$$x = 0$$

$$y = 0$$

direction = 'T', 'R', 'L', 'B'

| | c_0 | c_1 | c_2 | c_3 | c_4 |
|------------|--------------|-------|-------|-------|-------|
| γ_0 | top left (0) | 0 | 1 | 0 | 0 |
| γ_1 | 0 | 0 | 0 | 1 | 0 |
| γ_2 | 1 | 1 | 0 | 0 | 0 |
| γ_3 | 0 | 0 | 0 | 1 | 0 |
| γ_4 | 0 | 1 | 1 | 0 | 0 |



```

public int[] FindExitPoint(int[][] mat)
{
    int rows = mat.length, cols = mat[0].length;
    int x = 0, y = 0;
    char direction = 'R';

    while(x >= 0 && x < rows && y >= 0 && y < cols){
        if(mat[x][y] == 1){
            if(direction == 'R') direction = 'B';
            else if(direction == 'B') direction = 'L';
            else if(direction == 'L') direction = 'T';
            else direction = 'R';
        }

        if(direction == 'R') y++;
        else if(direction == 'L') y--;
        else if(direction == 'T') x--;
        else x++;
    }
}

```

} shifting

```

public int[] FindExitPoint(int[][] mat)
{
    int rows = mat.length, cols = mat[0].length;

    int x = 0, y = 0;
    char direction = 'R';

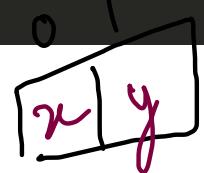
    while(x >= 0 && x < rows && y >= 0 && y < cols){
        if(mat[x][y] == 1){
            if(direction == 'R') direction = 'B';
            else if(direction == 'B') direction = 'L';
            else if(direction == 'L') direction = 'T';
            else direction = 'R';
            mat[x][y] = 0;
        }

        if(direction == 'R') y++;
        else if(direction == 'L') y--;
        else if(direction == 'T') x--;
        else x++;

    }

    int[] res = {x, y};
    if(direction == 'R') res[1]--;
    else if(direction == 'T') res[0]++;
    else if(direction == 'B') res[0]--;
    else res[1]++;
    return res;
}

```

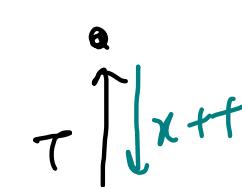
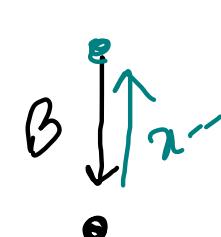
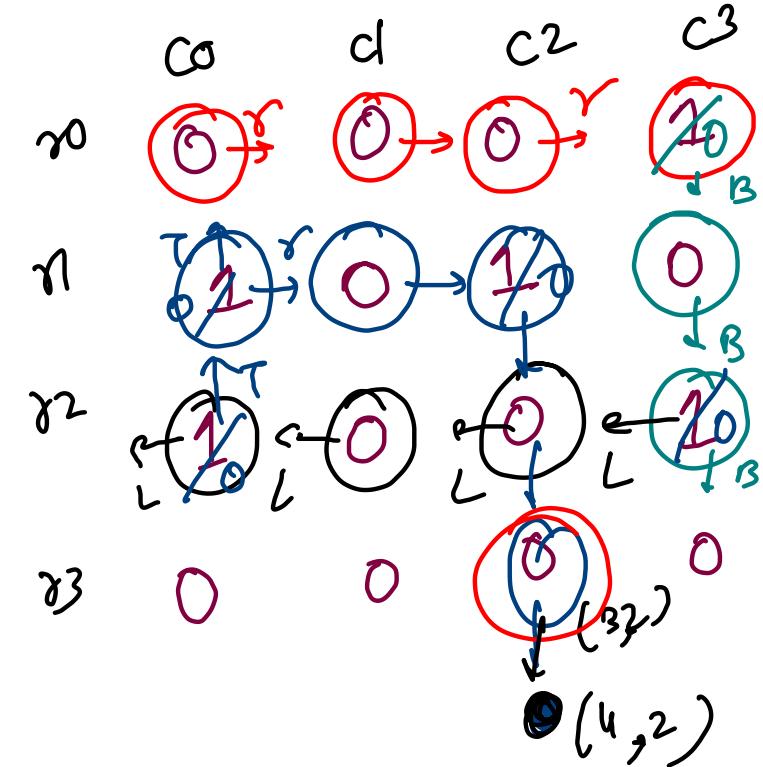


Checking out of matrix

rotate 90° clockwise

} shift in 4 direction

} exit poi



Convert 1D Array to 2D Array

1D
 $n=12$

0 1 2 3 4 5 6 7 8 9 10 11
10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120
↓ ↓ ↓ ↓ ↓ ↓ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

2D
 $rows=4$
 $cols=3$

| | | | |
|----|-----|-----|-----|
| 00 | 10 | 20 | 30 |
| 01 | 40 | 50 | 60 |
| 02 | 70 | 80 | 90 |
| 03 | 100 | 110 | 120 |

c0 c1 c2

12
↓
{ rows * 3 cols }

```

class Solution {
    public int[][] construct2DArray(int[] original, int rows, int cols) {
        int[][] mat = new int[rows][cols];
        if(rows * cols != original.length) return new int[0][0];

        int idx = 0;

        for(int row = 0; row < rows; row++){
            for(int col = 0; col < cols; col++){
                mat[row][col] = original[idx++];
            }
        }

        return mat;
    }
}

```

- ① `int[][]` mat
- ② `int[][]` mat² null;
- ③ `int[][]` mat³ = new int[5][0];
- Difference

- ① int[][] mat1;
- ② int[][] mat2 = null;
- ③ int[][] mat3 = new int[0][0];
- ④ int[][] mat4 = new int[5][];

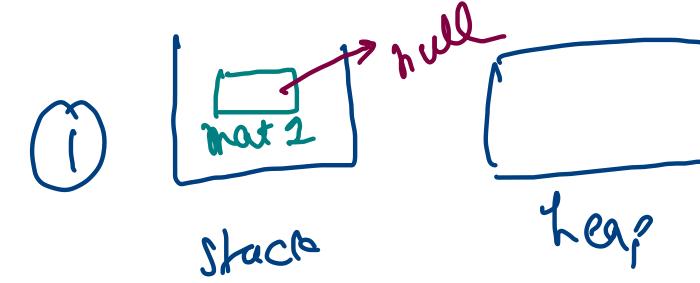
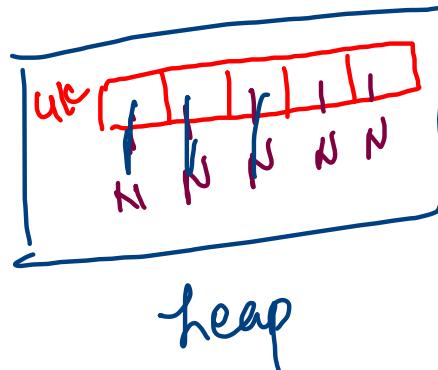
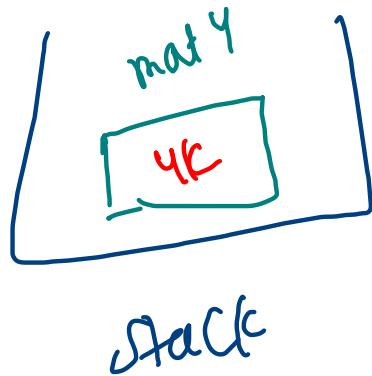
same

```

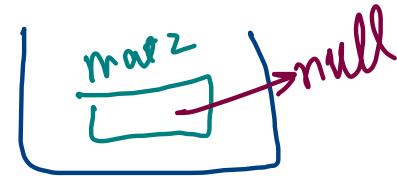
int[] mat1; → pointer
int[] mat2 = null; → pointer + initialization
int[] mat3 = new int[0][0];
int[] mat4 = new int[5][];

```

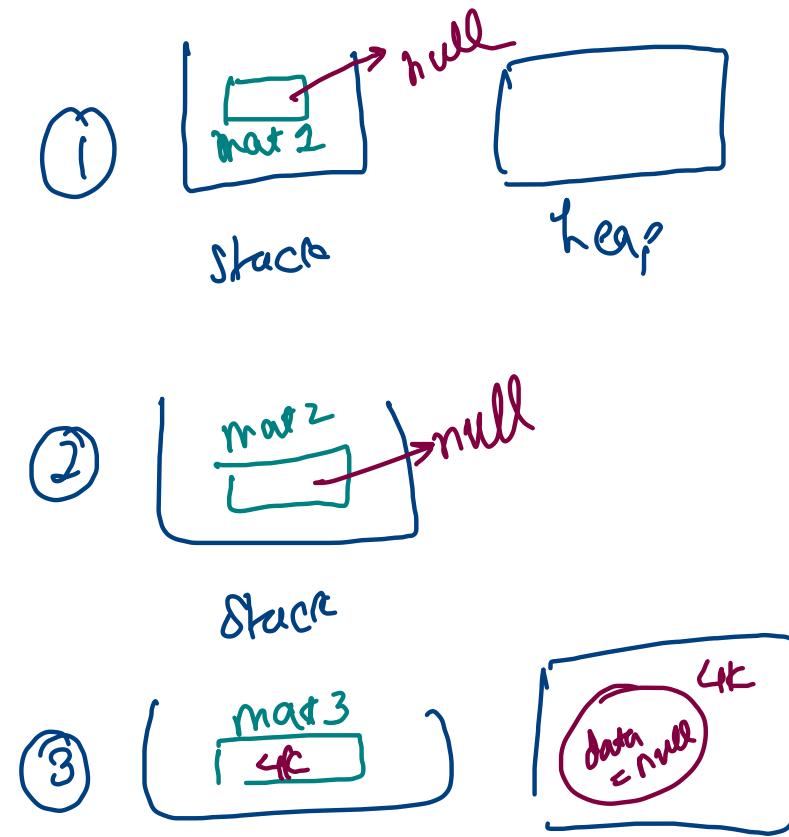
①



②



③



$1D \rightarrow 2D$ & $2D \rightarrow 1D$ index mapping

| | | | | | | | | | | | |
|-------------|-------------|-------------|---------------|---|---|---|---|---|---|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 10, 20, 30, | 40, 50, 60, | 70, 80, 90, | 100, 110, 120 | | | | | | | | |

1D
 $n=12$



idx $\rightarrow (?, ?)$

| | | | r_0 | r_1 | r_2 | r_3 |
|------------|--|--|----------------|----------------|----------------|-------|
| | | | c_0 | c_1 | c_2 | |
| $2D$ | | | $0*3+0$ 10 | $0*3+1$ 20 | $0*3+2$ 30 | |
| $rows = 4$ | | | $1*3+0$ 40 | $1*3+1$ 50 | $1*3+2$ 60 | |
| $cols = 3$ | | | $2*3+0$ 70 | $2*3+1$ 80 | $2*3+2$ 90 | |
| | | | $3*3+0$ 100 | $3*3+1$ 110 | $3*3+2$ 120 | |

| | | |
|----|-----------------|-----------------------------|
| 1D | $3 \rightarrow$ | $2D$ |
| | | $(1, 0) \rightarrow (1, 0)$ |
| | | $(r, c) \rightarrow ?$ |
| | | $2D$ |
| | | $(1, 0) \rightarrow 3$ |

| 1d idx | 2d coord |
|--------|----------|
| 10 | 0,0 |
| 20 | 0,1 |
| 30 | 0,2 |
| 40 | 1,0 |
| | 1,1 |
| | 1,2 |
| 50 | 2,0 |
| 60 | 2,1 |
| 70 | 2,2 |
| 80 | 3,0 |
| | 3,1 |
| | 3,2 |
| 90 | 4,0 |
| 100 | 4,1 |
| 110 | 4,2 |
| 120 | 5,0 |

cols = 5

(x, y) \Rightarrow

2d

idx \rightarrow

1d

0 *

0 * 3 + 1

0 * 3 + 0

$n * \text{cols} + y$

1D coordinate

(idx / cols, idx % cols)

x, c

2d coordinate

Good News / Bad News

recommended
students!

TA

Mock Interview

Confident ↑ nervous ↓

Learning ↑

Referrals → Companies

Elite Batch (advanced dsa)

(1-2 weeks) (most probably - Archit)

→ Backlog's
→ Revise (2-3-4 days)
→ Get for Interview

weekends
morning/afternoon
1/2 weekdays

→ HTML/CSS/Javascript:

notes → handwritten
Code/Notes → Educator
Search for YT

→ Programming language (Module 1) → hackerrank
1-1 problem
from each category

Module 2
Arrays &
Strings

① 1D array

normal single loop
sorting (nested loops) → two pointers
prefix array

② 2D Arrays

③ Time & Space

```

public static int[][] convert(int[] arr, int rows, int cols){
    int[][] mat = new int[rows][cols];

    // Approach 1:
    int idx = 0;
    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = arr[idx++];
        }
    }

    return mat;
}

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int size = scn.nextInt();
    int[] arr = new int[size];
    for(int idx = 0; idx < size; idx++) {
        arr[idx] = scn.nextInt();
    }

    int rows = scn.nextInt();
    int cols = scn.nextInt();

    int[][] mat = convert(arr, rows, cols);
    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            System.out.print(mat[row][col] + " ");
        }
    }
}

```

$\Theta(r \times c)$
 $= O(n)$

```

public static int[][] convert(int[] arr, int rows, int cols){
    int[][] mat = new int[rows][cols];

    // Approach 1:
    // int idx = 0;
    // for(int row = 0; row < rows; row++){
    //     for(int col = 0; col < cols; col++){
    //         mat[row][col] = arr[idx++];
    //     }
    // }

    // Approach 2:
    for(int idx = 0; idx < arr.length; idx++){
        int row = idx / cols, col = idx % cols;
        mat[row][col] = arr[idx];
    }

    return mat;
}

```

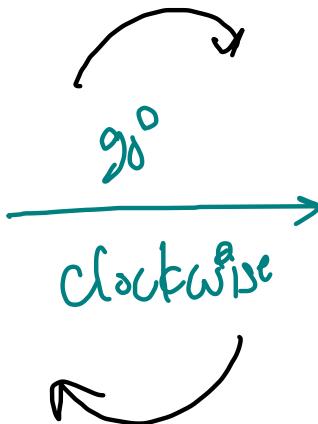
n = 1d array length

$O(n)$

Rotate matrix by 90° clockwise

| | | | | |
|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |
| 30 | 31 | 32 | 33 | 34 |
| 40 | 41 | 42 | 43 | 44 |

Transpose



| | | | | |
|----|----|----|----|----|
| 00 | 10 | 20 | 30 | 40 |
| 01 | 11 | 21 | 31 | 41 |
| 02 | 12 | 22 | 32 | 42 |
| 03 | 13 | 23 | 33 | 43 |
| 04 | 14 | 24 | 34 | 44 |

| | | | | |
|----|----|----|----|----|
| 40 | 30 | 20 | 10 | 00 |
| 41 | 31 | 21 | 11 | 01 |
| 42 | 32 | 22 | 12 | 02 |
| 43 | 33 | 23 | 13 | 03 |
| 44 | 34 | 24 | 14 | 04 |

All arrays
should be
reversed

```

public static void transpose(int[][] mat, int n){
    for(int row = 0; row < n; row++){
        for(int col = 0; col < row; col++){
            int temp = mat[row][col];
            mat[row][col] = mat[col][row];
            mat[col][row] = temp;
        }
    }
}

```

$\mathcal{O}(n^2)$

```

public static void reverseRows(int[][] mat, int n){

    // Iterate on all rows .
    for(int row = 0; row < n; row++){
        int left = 0, right = n - 1;

        // reverse the given row
        while(left < right){
            // Swap (row, left) with (row, right)
            int temp = mat[row][left];
            mat[row][left] = mat[row][right];
            mat[row][right] = temp;

            left++; right--;
        }
    }
}

```

$\mathcal{O}(n^2)$

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    int n = scn.nextInt();
    int[][] mat = new int[n][n];
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            mat[row][col] = scn.nextInt();
        }
    }
}

```

✓ transpose(mat, n); $\rightarrow n^2 +$ ✓ reverseRows(mat, n); $\rightarrow n^2 = \mathcal{O}(n^2)$

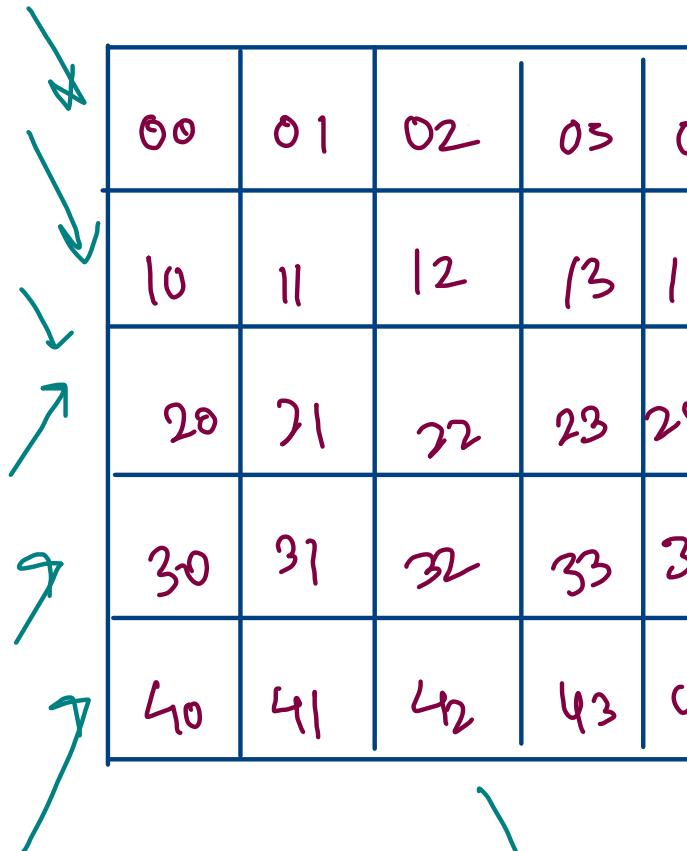
```

    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            System.out.print(mat[row][col] + " ");
        }
        System.out.println();
    }
}

```

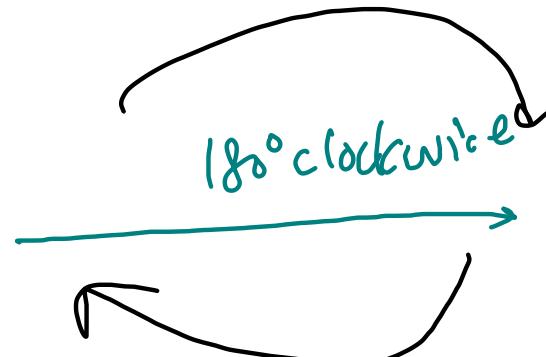
Rotate 180° clockwise

| | | | | |
|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |
| 30 | 31 | 32 | 33 | 34 |
| 40 | 41 | 42 | 43 | 44 |



Horizontally
Flip
(Columns)

| | | | | |
|----|----|----|----|----|
| 40 | 41 | 42 | 43 | 44 |
| 30 | 31 | 32 | 33 | 34 |
| 20 | 21 | 22 | 23 | 24 |
| 10 | 11 | 12 | 13 | 14 |
| 00 | 01 | 02 | 03 | 04 |



| | | | | |
|----|----|----|----|----|
| 44 | 43 | 42 | 41 | 40 |
| 34 | 33 | 32 | 31 | 30 |
| 24 | 23 | 22 | 21 | 20 |
| 14 | 13 | 12 | 11 | 10 |
| 04 | 03 | 02 | 01 | 00 |

Reverse
Each rows

```

public static void interchange(int[][] mat, int n){
    int top = 0, bottom = n - 1;
    while(top < bottom){
        for(int col = 0; col < n; col++){
            // Swap Elements From First Row with Last Row for Every Column
            int temp = mat[top][col];
            mat[top][col] = mat[bottom][col];
            mat[bottom][col] = temp;
        }
        top++; bottom--;
    }
}

public static void reverseRows(int[][] mat, int n){

    // Iterate on all rows
    for(int row = 0; row < n; row++){
        int left = 0, right = n - 1;

        // reverse the given row
        while(left < right){
            // Swap (row, left) with (row, right)
            int temp = mat[row][left];
            mat[row][left] = mat[row][right];
            mat[row][right] = temp;

            left++; right--;
        }
    }
}

```

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    int n = scn.nextInt();
    int[][] mat = new int[n][n];
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            mat[row][col] = scn.nextInt();
        }
    }

    interchange(mat, n);  $\Rightarrow O(n^2)$ 
    reverseRows(mat, n);  $\Rightarrow O(n^2)$ 

    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            System.out.print(mat[row][col] + " ");
        }
        System.out.println();
    }
}

```

| | | | | |
|----|----|----|----|----|
| 40 | 41 | 42 | 43 | 44 |
| 00 | 01 | 02 | 03 | 04 |
| 30 | 31 | 32 | 33 | 34 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |

Matrix Problems

- 1) Shift matrix row-wise
- 2) Modify the target
- 3) Rotation check in matrix

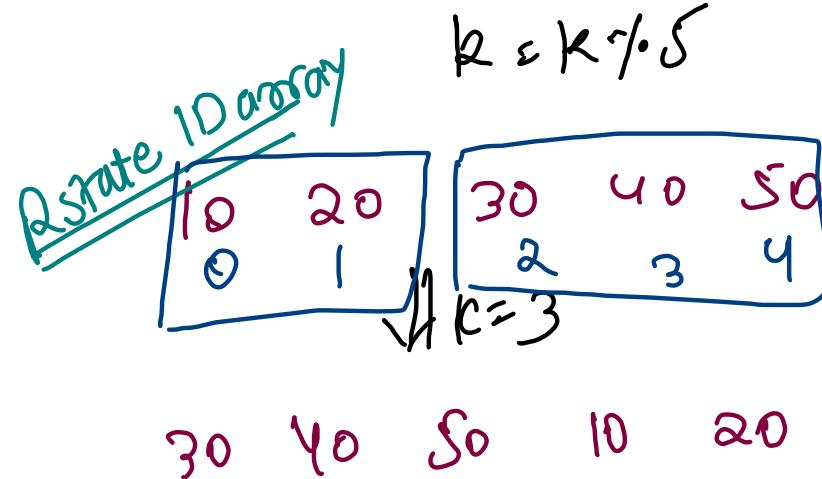
Shift Matrix Row wise

| $k \leq 3$ | 0^2 | 0^3 | 0^4 | 0^0 | 0^1 |
|------------|---------------|---------------|---------------|---------------|---------------|
| | 00 | 01 | 02 | 03 | 04 |
| $k \leq 3$ | 10 | 11 | 12 | 13 | 14 |
| $k \leq 3$ | 20 | 21 | 22 | 23 | 24 |
| $k \leq 3$ | 30 | 31 | 32 | 33 | 34 |
| $k \leq 3$ | 40 | 41 | 42 | 43 | 44 |

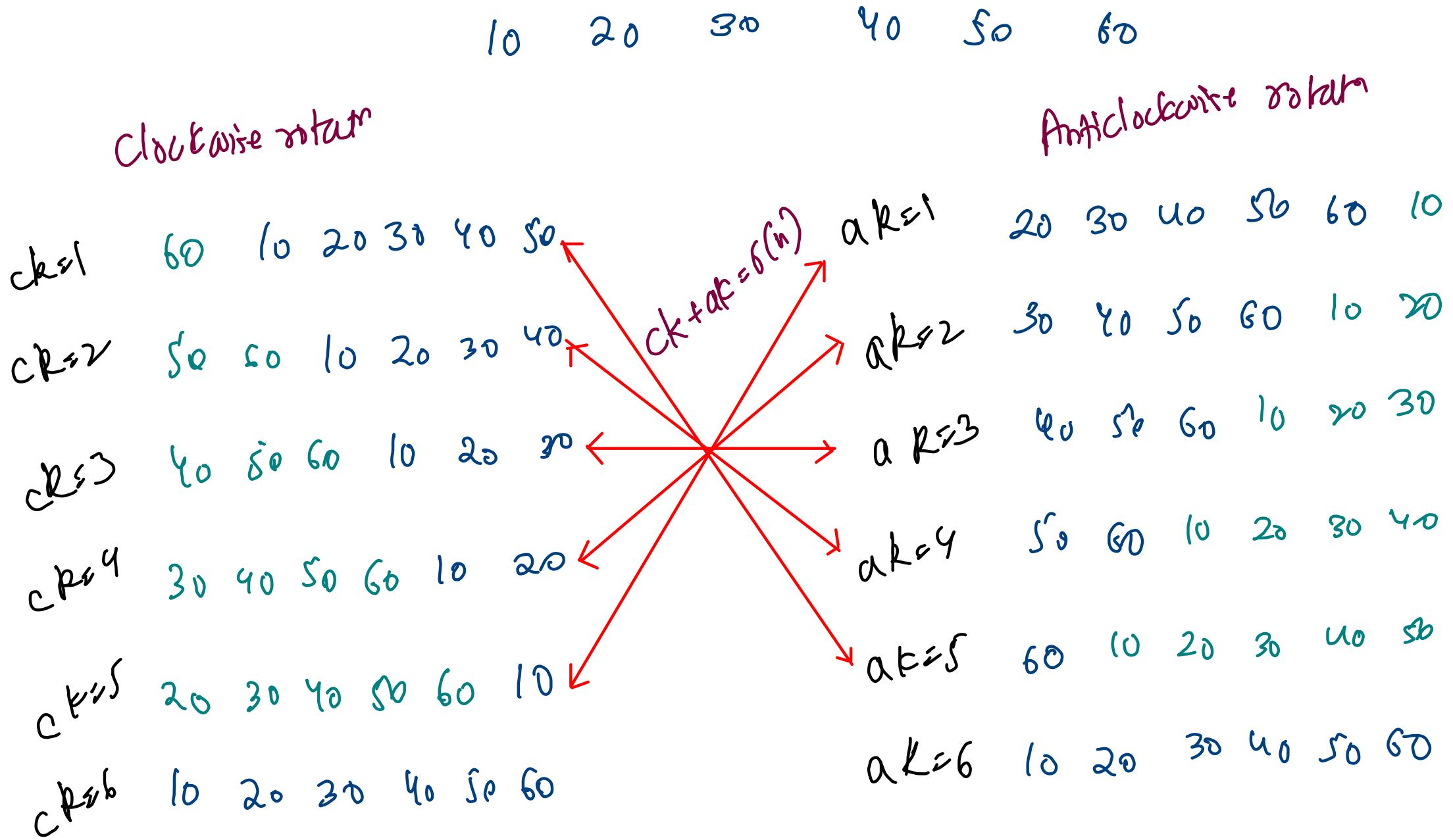
~~in place~~

- Square \Rightarrow Row = Col

• $k = 3 \leftarrow 103$



- $\left\{ \begin{array}{l} \text{1) Reverse } (0, n-k-1) \\ \text{2) Reverse } (n-k, n-1) \\ \text{3) Reverse } (0, n-1) \end{array} \right.$



```

// Reverse A Row from left column till right column
public static void reverse(int[][] mat, int row, int left, int right){
    while(left < right){
        int temp = mat[row][left];
        mat[row][left] = mat[row][right];
        mat[row][right] = temp;
        left++; right--;
    }
}

```

$\Theta(n)$

Space complexity
 $\Theta(1)$

constant extra space

```

public static void rotateRows(int[][] mat, int k){
    int n = mat.length;
    k = k % n; → to reduce the k
    k = n - k; // anticlockwise into clockwise

    // Rotate All Rows by K Steps
    for(int row = 0; row < n; row++){
        reverse(mat, row, 0, n - k - 1);
        reverse(mat, row, n - k, n - 1);
        reverse(mat, row, 0, n - 1);
    }
}

```

$2 \times \Theta(n)$

clockwise rotation (k)
for each row

$N \times 2N$

$= 2N^2$

$\Rightarrow \Theta(n^2)$

quadratic
time

Set matrix Zeros (Modify the matrix)

(FAANG)

| | | | | |
|-----|-----|-----|-----|-----|
| 10 | 20 | 30 | 40 | 50 |
| 60 | 70 | 80 | 0 | 90 |
| 100 | 110 | 120 | 130 | 140 |
| 150 | 0 | 160 | 170 | 0 |
| 210 | 180 | 190 | 0 | 200 |

Copy
for the
output

if row contains 0 \Rightarrow make
entire row 0.

if column contains 0 \Rightarrow make
entire Col 0

| | | | | |
|------------------|------------------|------------------|------------------|------------------|
| 10 | 20 0 | 30 | 40 0 | 50 0 |
| 0 | 60 0 | 70 0 | 80 0 | 90 0 |
| 100 | 0 | 110 0 | 120 | 130 0 |
| 150 0 | 0 | 160 0 | 170 0 | 0 |
| 210 0 | 180 0 | 190 0 | 0 | 200 0 |

```

public boolean isRow0(int[][] mat, int row){
    for(int col = 0; col < mat[0].length; col++){
        if(mat[row][col] == 0) return true;
    }
    return false;
}

public boolean isCol0(int[][] mat, int col){
    for(int row = 0; row < mat.length; row++){
        if(mat[row][col] == 0) return true;
    }
    return false;
}

public void setZeroes(int[][] mat) {
    int rows = mat.length, cols = mat[0].length;

    int[][] copy = new int[rows][cols];

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            copy[row][col] = mat[row][col];
            if(isRow0(mat, row) == true || isCol0(mat, col) == true){
                copy[row][col] = 0;
            }
        }
    }

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            mat[row][col] = copy[row][col];
        }
    }
}

```

Brute force Approach

~~8:53~~

Time complexity: $O(n^3)$

make output matrix
copied into input
matrix (void return type)

```

public void makeRow0(int[][] mat, int row){
    for(int col = 0; col < mat[0].length; col++){
        mat[row][col] = 0;
    }
}

public void makeCol0(int[][] mat, int col){
    for(int row = 0; row < mat.length; row++){
        mat[row][col] = 0;
    }
}

public void setZeroes(int[][] mat) {
    int rows = mat.length, cols = mat[0].length;

    int[][] copy = new int[rows][cols];
    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++)
            copy[row][col] = mat[row][col];
    }

    for(int row = 0; row < rows; row++){
        for(int col = 0; col < cols; col++){
            if(copy[row][col] == 0){
                makeRow0(mat, row);
                makeCol0(mat, col);
            }
        }
    }
}

```

Better Approach

| | | | | |
|-----|-----|-----|-----|-----|
| 10 | 20 | 30 | 40 | 50 |
| 60 | 70 | 80 | 90 | 100 |
| 110 | 120 | 130 | 140 | 150 |
| 150 | 160 | 170 | 180 | 190 |
| 210 | 180 | 190 | 100 | 200 |

mat

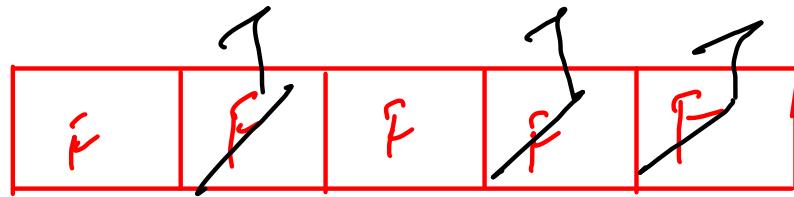
| | | | | |
|-----|-----|-----|-----|-----|
| 10 | 20 | 30 | 40 | 50 |
| 60 | 70 | 80 | 90 | 100 |
| 110 | 120 | 130 | 140 | 150 |
| 150 | 160 | 170 | 180 | 190 |
| 210 | 180 | 190 | 100 | 200 |

copy

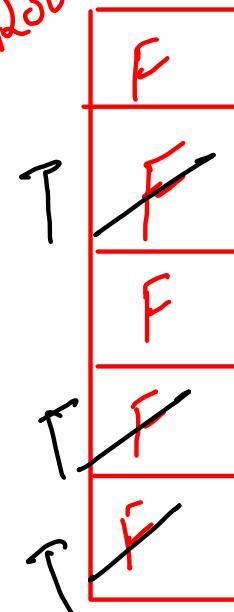
$O(n^3)$
in worst case
(All elements
in original
are 0s)

bad Approach

(O)



Row



| | | | | |
|-----|-----|-----|-----|-----|
| 10 | 20 | 30 | 40 | 50 |
| 60 | 70 | 80 | 90 | 100 |
| 110 | 120 | 130 | 140 | |
| 150 | 0 | 160 | 170 | 0 |
| 210 | 180 | 190 | 0 | 200 |

```

public void setZeroes(int[][] mat) {
    boolean[] row = new boolean[mat.length];
    boolean[] col = new boolean[mat[0].length];

    for(int i = 0; i < mat.length; i++){
        for(int j = 0; j < mat[0].length; j++){
            if(mat[i][j] == 0){
                row[i] = col[j] = true;
            }
        }
    }

    for(int i = 0; i < mat.length; i++){
        for(int j = 0; j < mat[0].length; j++){
            if(row[i] == true || col[j] == true){
                mat[i][j] = 0;
            }
        }
    }
}

```

Space
Time
 $\Rightarrow O(n^2)$

$O(n^2)$ time

```
public static void modifyMatrix(int[][] mat) {  
    boolean[] row = new boolean[mat.length];  
    boolean[] col = new boolean[mat[0].length];  
  
    for(int i = 0; i < mat.length; i++){  
        for(int j = 0; j < mat[0].length; j++){  
            if(mat[i][j] == 1){  
                row[i] = col[j] = true;  
            }  
        }  
    }  
  
    for(int i = 0; i < mat.length; i++){  
        for(int j = 0; j < mat[0].length; j++){  
            if(row[i] == true || col[j] == true){  
                mat[i][j] = 1;  
            }  
        }  
    }  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int rows = scn.nextInt();  
    int cols = scn.nextInt();  
  
    int[][] mat = new int[rows][cols];  
    for(int row = 0; row < rows; row++){  
        for(int col = 0; col < cols; col++){  
            mat[row][col] = scn.nextInt();  
        }  
    }  
  
    modifyMatrix(mat);  
    for(int row = 0; row < rows; row++){  
        for(int col = 0; col < cols; col++){  
            System.out.print(mat[row][col] + " ");  
        }  
        System.out.println();  
    }  
}
```

Check Rotate matrix

10 20 30 40 50

20 30 40 50 10

30 40 50 10 20

40 50 10 20 30

50 10 20 30 40

```

public static boolean compare(int[][] mat, int row1, int row2){
    for(int col = 0; col < mat[0].length; col++){
        if(mat[row1][col] != mat[row2][col]) return false;
    }
    return true;
}

public static void rotate(int[][] mat, int row){
    int temp = mat[row][mat[0].length - 1];
    for(int col = mat[0].length - 1; col > 0; col--){
        mat[row][col] = mat[row][col - 1];
    }
    mat[row][0] = temp;
}

```

0) 10 20 30 40 50

1) 10 20 30 40 50 10

2) 10 20 30 40 50 10 20

3) 10 20 30 40 50 10 20 30

4) 10 20 30 40 50 10 20 30 40

*equal=false
true*

*equal=false
true*

equal=false

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    int[][] mat = new int[n][n];
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            mat[row][col] = scn.nextInt();
        }
    }

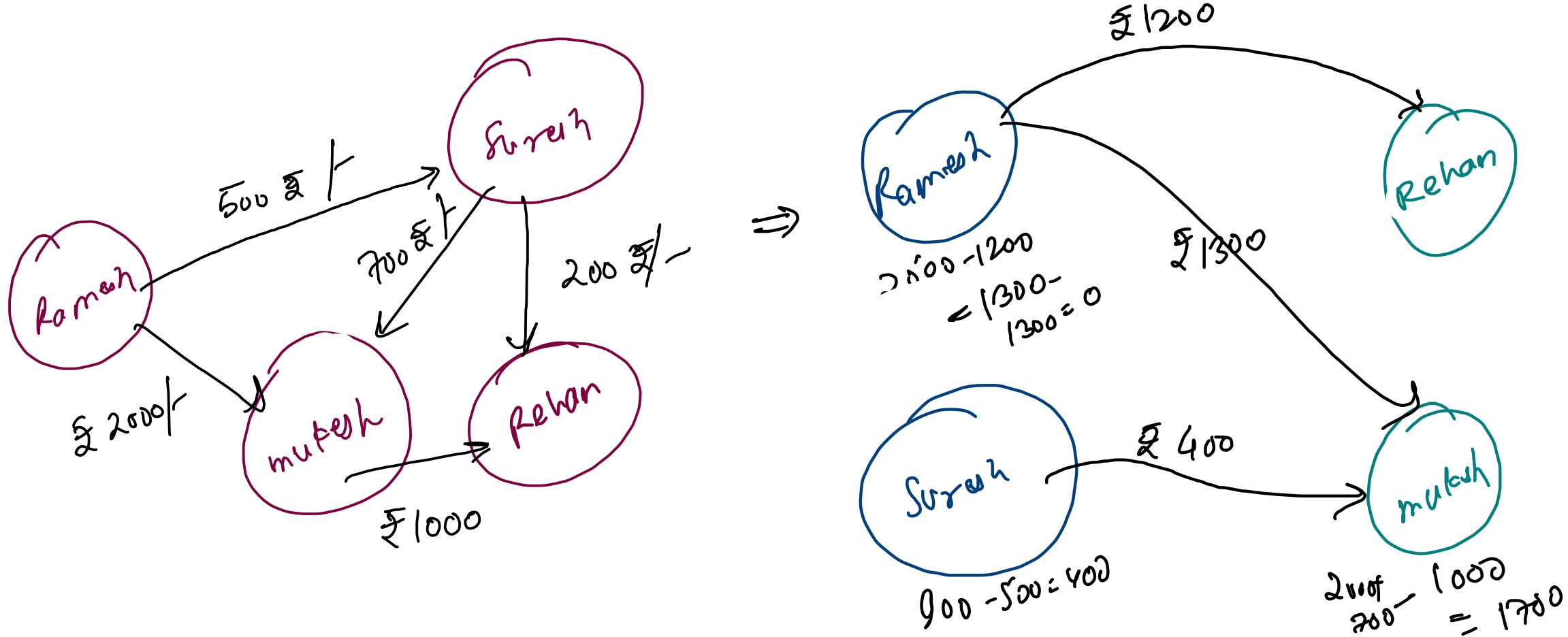
    for(int row = 1; row < n; row++){
        boolean equal = false;
        for(int k = 0; k <= n; k++){
            rotate(mat, row);
            if(compare(mat, 0, row) == true){
                equal = true;
                break;
            }
        }
        if(equal == false){
            System.out.println("NO");
            return;
        }
    }
    System.out.println("YES");
}

```

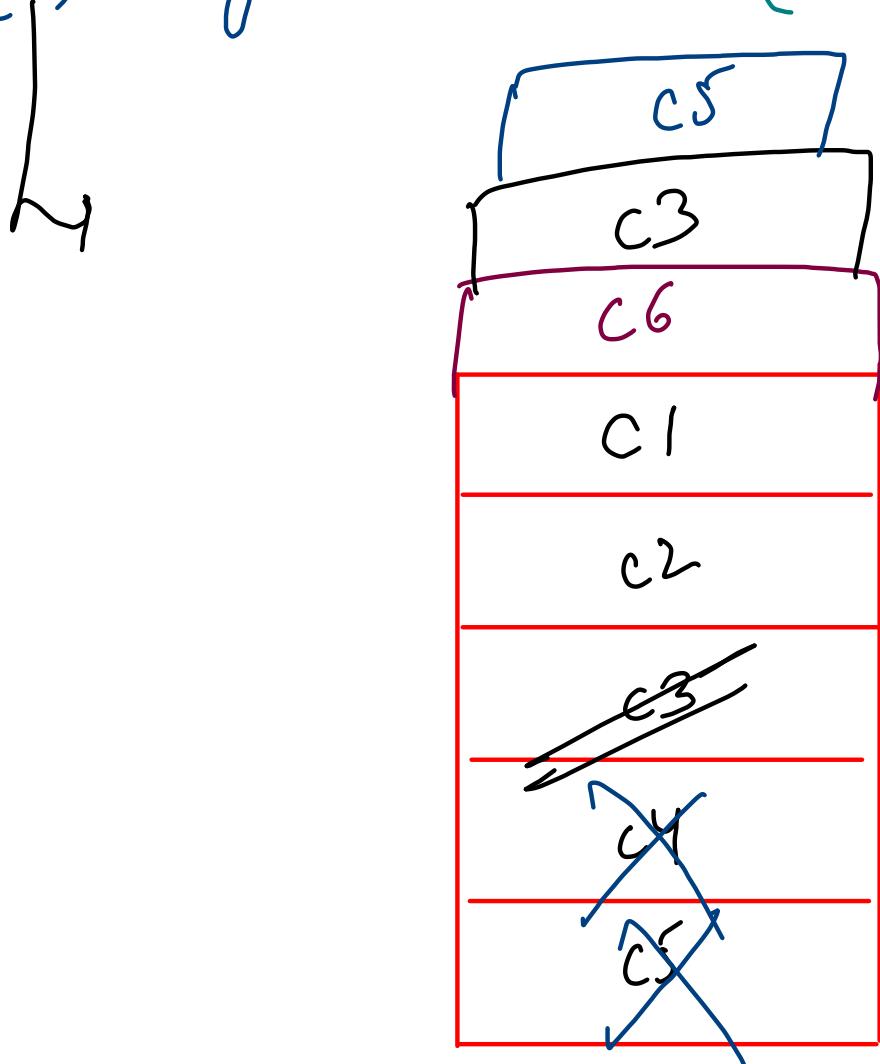
Medium
Hard

DSA based Project

(1) Splitwise App (Zen-Pen) (Python → split the payment)



~~Medium~~ (2) Design ~~LRU~~ Cache (whatsapp)



~~easy~~

(3) Design → ① Tic Tac Toe



~~graph~~ (2) Snake & Ladder

③ Ludo

④ Monopoly (DP)

~~Advanced~~

(5) Implement
advanced tree

↳ B-tree → DBMS

↓
SQL

menu-driven
program
(Java)
→ do while loop

Hucky No / Saddlepoint in Matrix

| | | | | |
|------|------|-------------------------------|------|------|
| 35 | 110 | 200 | 40 | 60 |
| 100 | 70 | 500 | 700 | 600 |
| 200 | 90 | 800 | 10 | 30 |
| 4000 | 3000 | 1500 ^{C²} | 5000 | 6000 |

Saddle Point

$\min^m \Rightarrow$ rows

$\max^m \Rightarrow$ cols

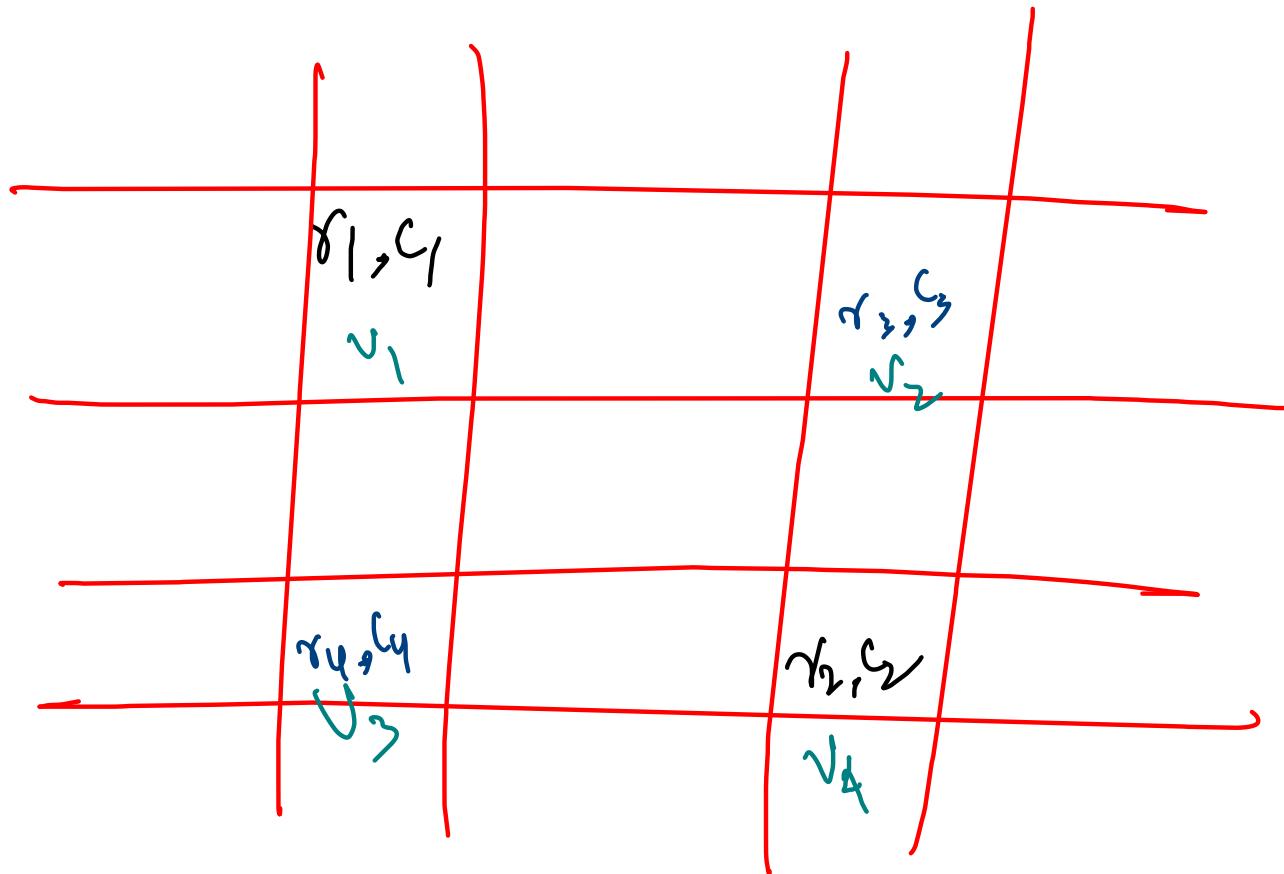
~~No~~

Q1) Can there be
or more than
2 saddle point?

Q2) Can there be
no saddle
point?

~~Yes~~

Assume 2 saddle points



v_1 is saddle point

$$v_1 < v_2 < v_4$$

$$v_1 > v_3 > v_4$$

contradiction

v_4 is saddle point

$$v_4 < v_3$$

$$v_4 > v_2$$

```

public int getMinInRow(int[][] mat, int row){
    int minIdx = 0;

    for(int col = 0; col < mat[0].length; col++){
        if(mat[row][col] < mat[row][minIdx]){
            minIdx = col;
        }
    }

    return minIdx;
}

```

Time $\rightarrow O(n^2)$

```

public int getMaxInCol(int[][] mat, int col){
    int maxIdx = 0;

    for(int row = 0; row < mat.length; row++){
        if(mat[row][col] > mat[maxIdx][col]){
            maxIdx = row;
        }
    }

    return maxIdx;
}

```

```

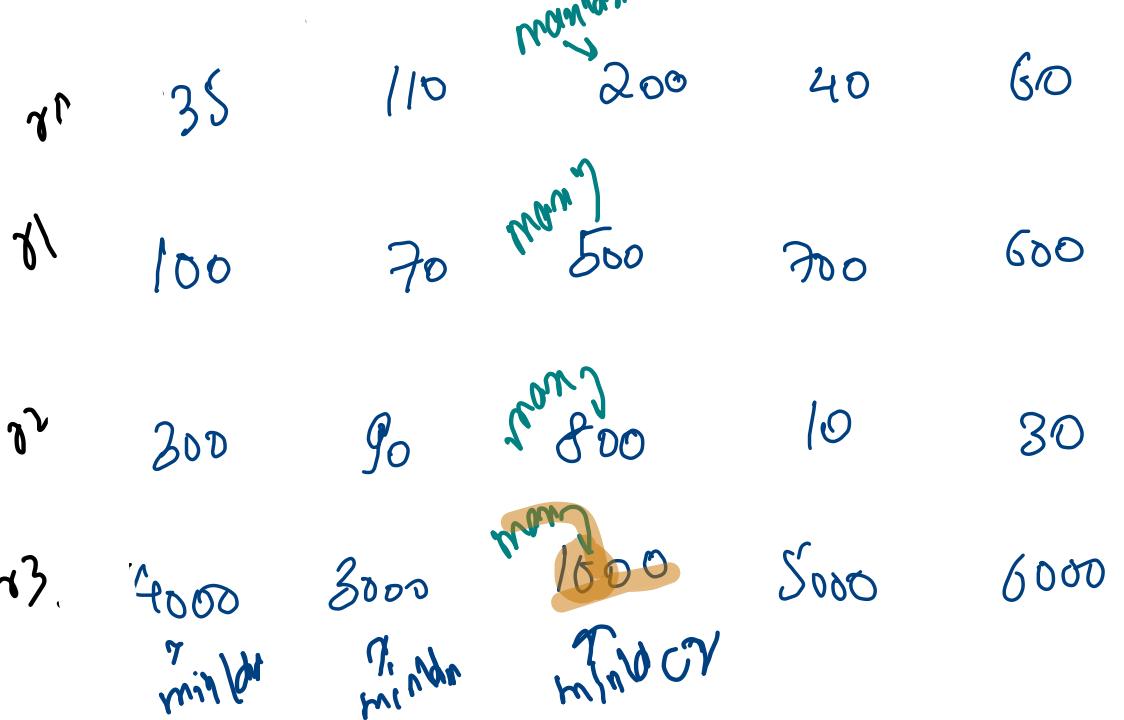
public List<Integer> luckyNumbers (int[][] mat) {
    for(int row = 0; row < mat.length; row++){
        int minCol = getMinInRow(mat, row);
        int maxRow = getMaxInCol(mat, minCol);

        if(maxRow == row){
            List<Integer> res = new ArrayList<>();
            res.add(mat[row][minCol]);
            return res;
        }
    }

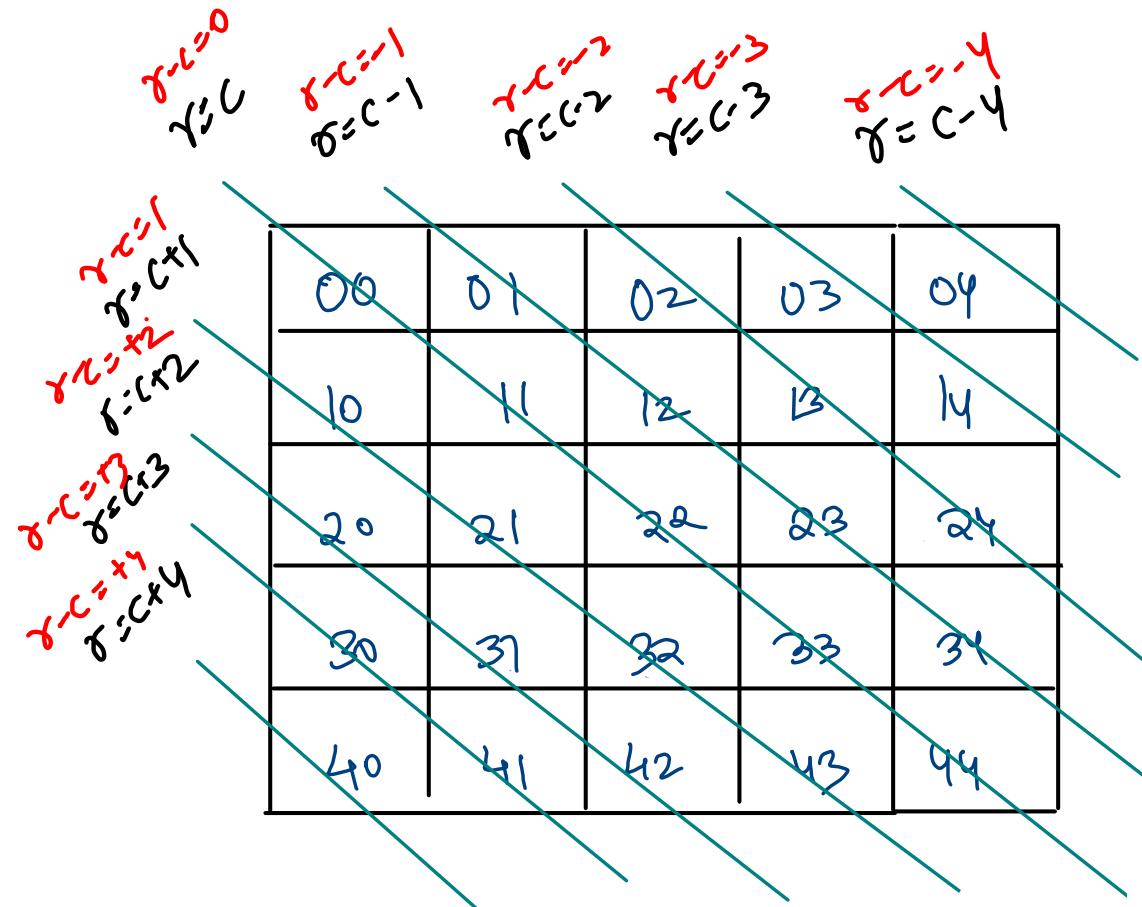
    return new ArrayList<>(); } no saddle point

```

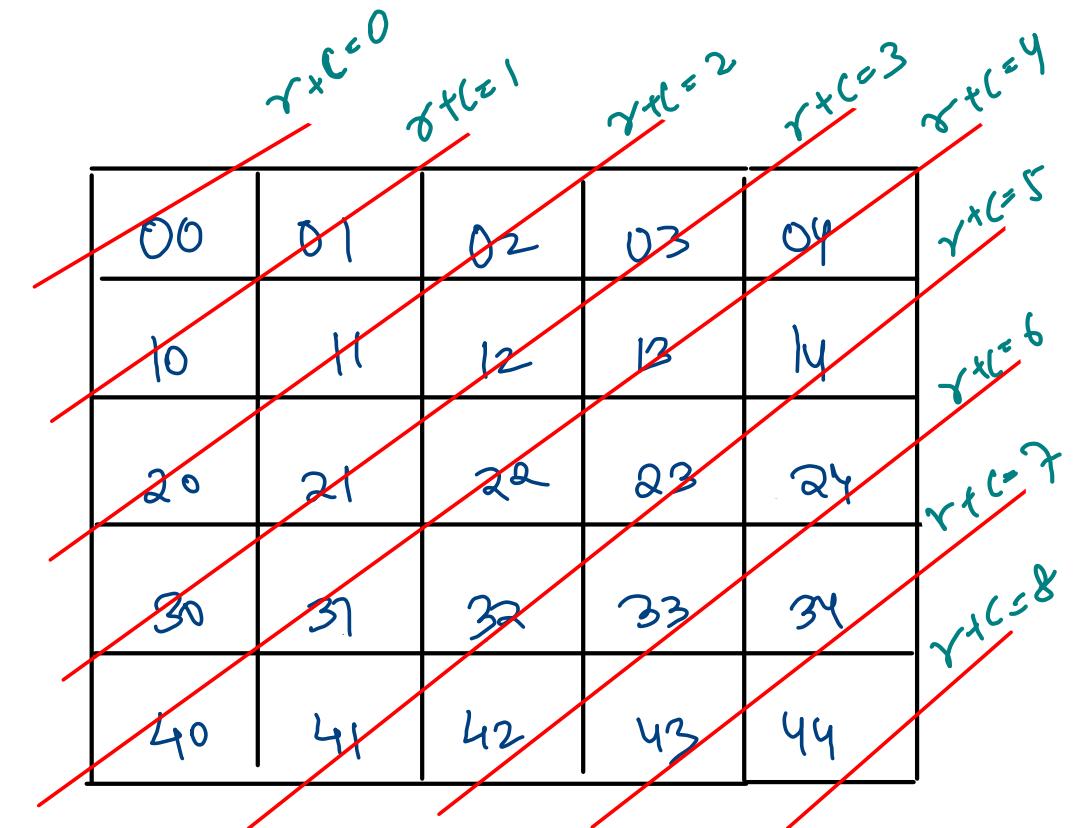
1 saddle point



Diagonal Traversal



Top left to bottom right
Diagonals



Bottom left to Top right
Anti-diagonals

| $r+c=0$ | $r+c=1$ | $r+c=2$ | $r+c=3$ | $r+c=4$ | |
|---------|---------|---------|---------|---------|----|
| $c=0$ | 00 | 01 | 02 | 03 | 04 |
| $c=1$ | 10 | 11 | 12 | 13 | 14 |
| $c=2$ | 20 | 21 | 22 | 23 | 24 |
| $c=3$ | 30 | 31 | 32 | 33 | 34 |
| $c=4$ | 40 | 41 | 42 | 43 | 44 |

```

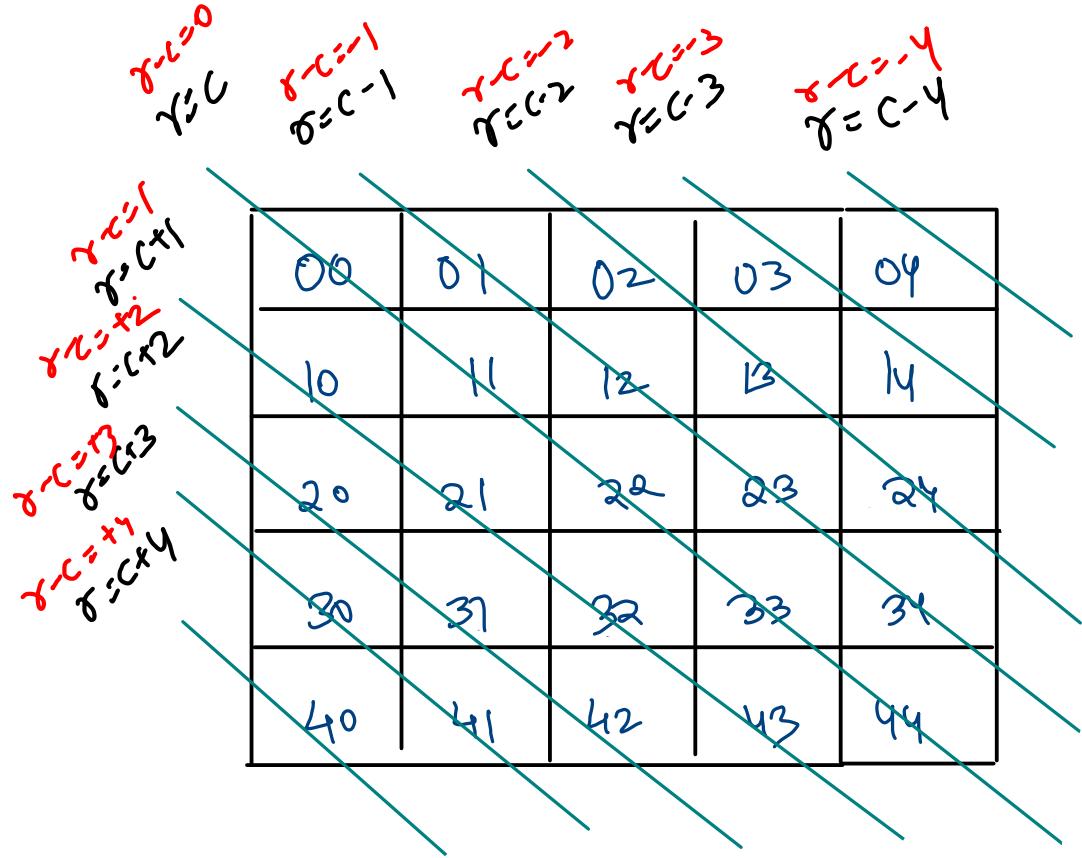
for(int sum = 0; sum <= 2 * n - 2; sum++){
    int row = 0, col = 0;

    if(sum < N){
        row = sum; col=0;
    } else {
        row = n - 1;
        col = sum - row;
    }

    while(row >= 0 && col < n){
        System.out.print(A[row][col]);
        row--; col++;
    }
}

```

Bottom Left to Top Right Diagonal



```

for(int diff = (n - 1); diff >= -(n - 1); diff--){
    int row = 0, col = 0;

    if(diff >= 0){
        row = diff; col = 0;
    } else {
        row = 0; col = -diff;
    }

    while(row < n && col < n){
        System.out.print(A[row][col] + " ");
        row++; col++;
    }
}

return new ArrayList<>();

```

| | | | | |
|----|----|----|----|----|
| 00 | 01 | 02 | 03 | 04 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |
| 30 | 31 | 32 | 33 | 34 |
| 40 | 41 | 42 | 43 | 44 |

row by row

outer loop \rightarrow row (fixing)

inner loop \rightarrow col (dynamic)

col by col

outer loop \rightarrow col (fixing)

inner loop \rightarrow row (dynamic)

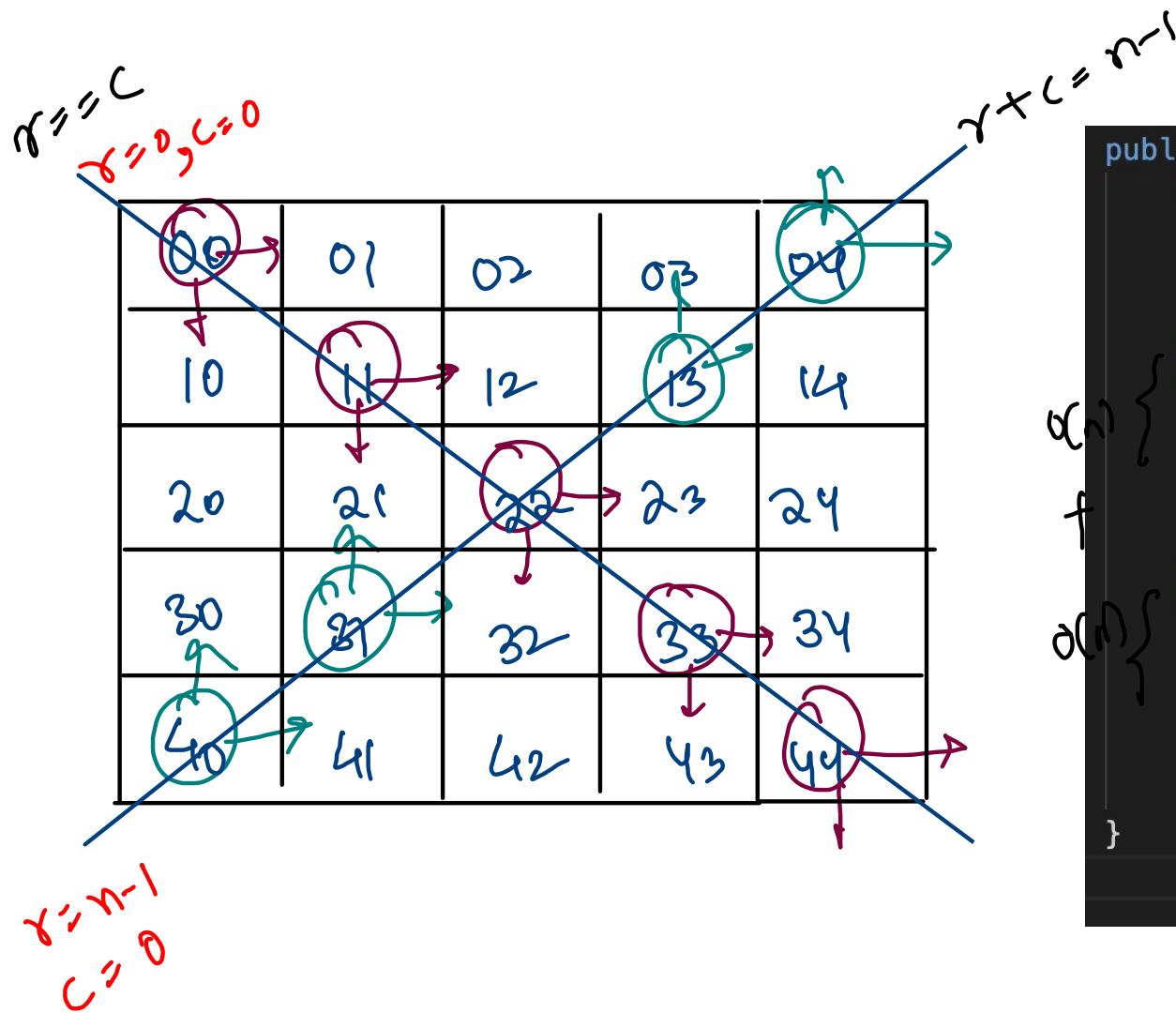
To split Matrix
 $\overbrace{\quad\quad}$ Tree

| $\gamma = c^0$ | $\gamma = c^{-1}$ | $\gamma = c^{-2}$ | $\gamma = c^{-3}$ | $\gamma = c^{-4}$ |
|-------------------|-------------------|-------------------|-------------------|-------------------|
| $\gamma = c^1$ | $\delta = c^{-1}$ | $\gamma = c^{-2}$ | $\gamma = c^{-3}$ | $\gamma = c^{-4}$ |
| $\gamma = c^{-1}$ | $\delta = c^{-2}$ | $\gamma = c^{-3}$ | $\gamma = c^{-4}$ | |
| $\gamma = c^{-2}$ | $\delta = c^{-3}$ | $\gamma = c^{-4}$ | | |
| $\gamma = c^{-3}$ | $\delta = c^{-4}$ | | | |
| $\delta = c^0$ | $\delta = c^1$ | $\delta = c^{-1}$ | $\delta = c^{-2}$ | $\delta = c^{-3}$ |
| $\delta = c^1$ | $\delta = c^{-1}$ | $\delta = c^{-2}$ | $\delta = c^{-3}$ | |
| $\delta = c^{-1}$ | $\delta = c^{-2}$ | $\delta = c^{-3}$ | | |
| $\delta = c^{-2}$ | $\delta = c^{-3}$ | | | |
| $\delta = c^{-3}$ | | | | |

Handwritten notes:

- Top row: $\gamma = c^0, \gamma = c^{-1}, \gamma = c^{-2}, \gamma = c^{-3}, \gamma = c^{-4}$
- Second row: $\gamma = c^1, \delta = c^{-1}, \gamma = c^{-2}, \gamma = c^{-3}, \gamma = c^{-4}$
- Third row: $\gamma = c^{-1}, \delta = c^{-2}, \gamma = c^{-3}, \gamma = c^{-4}$
- Fourth row: $\gamma = c^{-2}, \delta = c^{-3}, \gamma = c^{-4}$
- Fifth row: $\gamma = c^{-3}, \delta = c^{-4}$
- Bottom row: $\delta = c^0, \delta = c^1, \delta = c^{-1}, \delta = c^{-2}, \delta = c^{-3}$

Matrix Diagonal Sum



```

public int diagonalSum(int[][] mat) {
    int n = mat.length;
    int sum = 0;

    // top left to bottom right
    for(int row = 0, col = 0; row < n; row++, col++){
        sum = sum + mat[row][col];
    }

    // bottom left to top right
    for(int row = n - 1, col = 0; row >= 0; row--, col++){
        if(row != col) sum = sum + mat[row][col];
    }

    return sum;
}
    => O/n

```

Leetcode 74 Search 2D Matrix

| | | | | |
|----|----|----|----|----|
| 5 | 7 | 8 | 10 | 15 |
| 19 | 21 | 28 | 30 | 33 |
| 36 | 37 | 38 | 40 | 42 |
| 43 | 47 | 49 | 52 | 53 |
| 55 | 58 | 60 | 62 | 68 |

→ Completely sorted

→ All rows are sorted individually

→ All columns are sorted individually

StepCase Search Algorithm

search(58)

Approach 1
Linear Search
 $\Rightarrow O(n^2)$

Approach 2
Binary Search
 $\Rightarrow O(\log n^2)$

Approach 3
Step Case Search

$\Rightarrow O(n+m)$

Leetcode 240 Search 2D Matrix - II

| | | | | |
|----|----|----|----|----|
| 1 | 4 | 7 | 11 | 15 |
| 2 | 5 | 8 | 12 | 19 |
| 3 | 6 | 9 | 16 | 22 |
| 10 | 13 | 14 | 17 | 24 |
| 18 | 21 | 23 | 26 | 30 |

- Each row sorted individually
- Each col sorted individually

Approach 1

LinearSearch

$\Rightarrow O(n^2)$

Approach 2

BinarySearch

$\Rightarrow O(n \log n)$

Approach 3

Step Case Search

(Divide & Conquer)

search (13)

$15 > 13$

$g < 13$

$11 < 13$

$m > 13$

$12 < 13$

~~$13 = 13$~~

$16 > 13$

| | | | | |
|----|----|----|----|----|
| 1 | 4 | 7 | 11 | 15 |
| 2 | 5 | 8 | 12 | 19 |
| 3 | 6 | 9 | 16 | 22 |
| 10 | 13 | 14 | 17 | 24 |
| 18 | 21 | 23 | 26 | 30 |

search(20)

$15 < 20$

$19 < 20$

$22 > 20$

$16 < 20$

$12 < 20$

$26 > 20$

$23 > 20$

$$\gamma = 0, c = m-1 = 4$$

$$\gamma = 1, c = 4$$

$$\gamma = 2, c = 4$$

$$\gamma = 2, c = 3$$

$$\gamma = 3, c = 3$$

$$\gamma = 4, c = 3$$

$$\gamma = 4, c = 2$$

$\text{now } == n \quad (1 \text{ col } == -1)$

 unsuccessful search

```

public boolean searchMatrix(int[][] matrix, int target) {
    int row = 0, col = matrix[0].length - 1; // top right corner

    while(row < matrix.length && col >= 0){
        if(matrix[row][col] == target){
            return true; // successful search
        } else if(matrix[row][col] < target){
            row++; // reject row
        } else {
            col--; // reject column
        }
    }
    return false; // unsuccessful search
}

```

~~Time²~~

$\rightarrow O(\text{down} + \text{left})$

$= O(\text{rows} + \text{cols})$

$= O(n \text{frn})$

Linear

Winner of Tic Tac Toe

$(A \rightarrow X, B \rightarrow O)$

| | | |
|---|---|---|
| X | | |
| | X | |
| O | O | X |

A is winner

left diagonal is X

right diagonal is X

entire row is X

entire col is X

| | | |
|---|---|---|
| X | X | O |
| X | O | |
| O | | |

B is winner

left diagonal is O

right diagonal is O

entire row is O

entire col is O

| | | |
|---|---|---|
| X | X | O |
| O | O | X |
| X | O | X |

"Draw"

Count of
empty = 0

or
Count of
moves

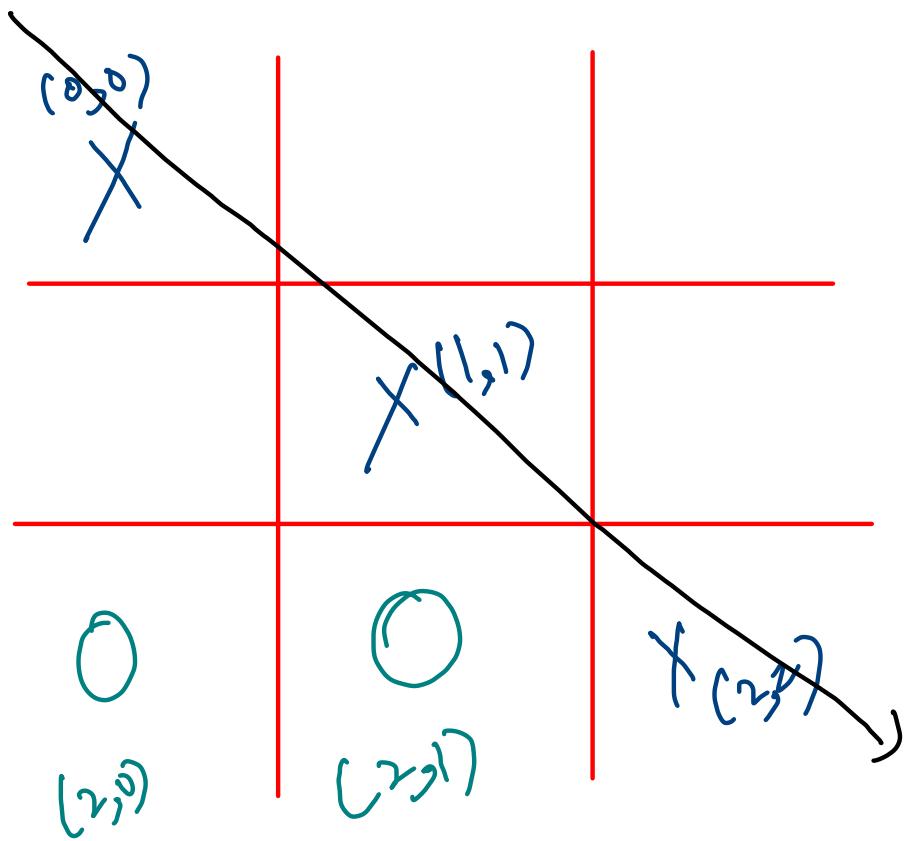
| | | |
|---|----|----|
| X | .. | O |
| O | .. | X |
| X | O | .. |

"Pending"

Count of
empty > 0

Count of
moves < 9

Input: moves = [[0, 0], [2, 0], [1, 1], [2, 1], [2, 2]]



```
public boolean checkWinner(char[][] mat, char ch){  
    if(mat[0][0] == ch && mat[1][1] == ch && mat[2][2] == ch) return true;  
    // left diagonal  
    if(mat[0][2] == ch && mat[1][1] == ch && mat[2][0] == ch) return true;  
    // right diagonal  
    if(mat[0][0] == ch && mat[0][1] == ch && mat[0][2] == ch) return true;  
    // 0th row  
    if(mat[1][0] == ch && mat[1][1] == ch && mat[1][2] == ch) return true;  
    // 1st row  
    if(mat[2][0] == ch && mat[2][1] == ch && mat[2][2] == ch) return true;  
    // 2nd row  
    if(mat[0][0] == ch && mat[1][0] == ch && mat[2][0] == ch) return true;  
    // 0th col  
    if(mat[0][1] == ch && mat[1][1] == ch && mat[2][1] == ch) return true;  
    // 1st col  
    if(mat[0][2] == ch && mat[1][2] == ch && mat[2][2] == ch) return true;  
    // 2nd col  
    return false;  
}
```

O(8)

O(9)

Constant

```
public String tictactoe(int[][] moves) {  
    char[][] mat = new char[3][3];  
    char turn = 'X';  
  
    for(int idx = 0; idx < moves.length; idx++){  
        int row = moves[idx][0];  
        int col = moves[idx][1];  
  
        mat[row][col] = turn;  
  
        if(checkWinner(mat, turn) == true){  
            if(turn == 'X') return "A";  
            else return "B";  
        }  
  
        if(turn == 'X') turn = 'O';  
        else turn = 'X';  
    }  
  
    if(moves.length == 9) return "Draw";  
    else return "Pending";  
}
```

N Queens

4×4

| | c_0 | c_1 | c_2 | c_3 |
|-------|-------------|-------------|-------------|-------------|
| r_0 | 0 | $\alpha(1)$ | 0 | 0 |
| r_1 | 0 | 0 | 0 | $\alpha(1)$ |
| r_2 | $\alpha(1)$ | 0 | 0 | 0 |
| r_3 | 0 | 0 | $\alpha(1)$ | 0 |

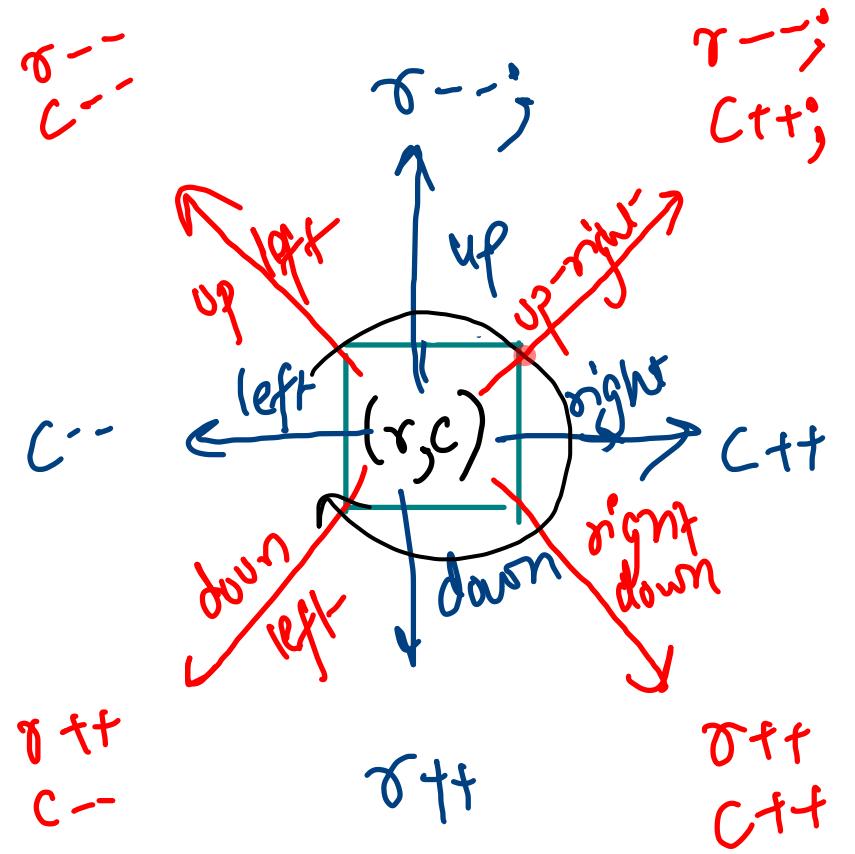
Answer : True
 : "N Queens"

No pair of queens should kill each other

4×4

| | c_0 | c_1 | c_2 | c_3 |
|-------|-------------|-------------|-------|-------------|
| r_0 | 0 | $\alpha(1)$ | 0 | 0 |
| r_1 | 0 | 0 | 0 | $\alpha(1)$ |
| r_2 | $\alpha(1)$ | 0 | 0 | 0 |
| r_3 | 0 | 0 | 0 | $\alpha(1)$ |

Answer: False
 : "Danger"

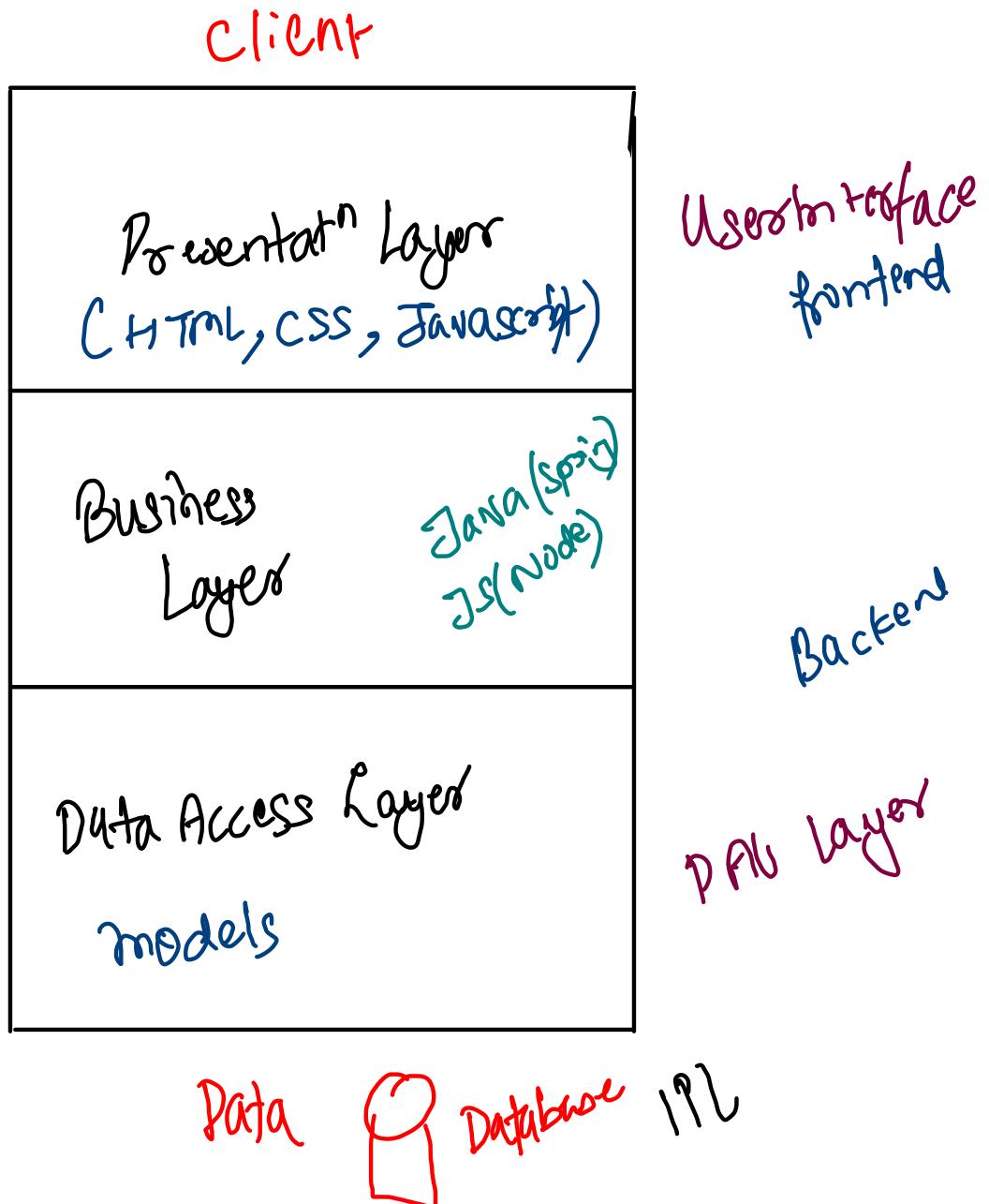


8:24

```
public static boolean safeState(int[][] chess, int row, int col){  
    // left  
    for(int r = row, c = col - 1; c >= 0; c--){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // right  
    for(int r = row, c = col + 1; c < chess[0].length; c++){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // up  
    for(int r = row - 1, c = col; r >= 0; r--){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // down  
    for(int r = row + 1, c = col; r < chess.length; r++){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // up-right  
    for(int r = row - 1, c = col + 1; r >= 0 && c < chess[0].length; r--, c++){  
        if(chess[r][c] == 1) return false;  
    }  
  
    .  
  
    // up-left  
    for(int r = row - 1, c = col - 1; r >= 0 && c >= 0; r--, c--){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // down-right  
    for(int r = row + 1, c = col + 1; r < chess.length && c < chess[0].length; r++, c++){  
        if(chess[r][c] == 1) return false;  
    }  
  
    // down-left  
    for(int r = row + 1, c = col - 1; r < chess.length && c >= 0; r++, c--){  
        if(chess[r][c] == 1) return false;  
    }  
  
    return true;  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
    int[][] chess = new int[n][n];  
    for(int row = 0; row < n; row++){  
        for(int col = 0; col < n; col++){  
            chess[row][col] = scn.nextInt();  
        }  
    }  
  
    for(int row = 0; row < n; row++){  
        for(int col = 0; col < n; col++){  
            if(chess[row][col] == 1 && safeState(chess, row, col) == false){  
                System.out.println("Danger");  
                return;  
            }  
        }  
    }  
    .  
    System.out.println("N Queens");  
}
```

3 Tier Architecture



Module ② (Strings, Binary Search, Modular Arithmetic)

30th October — MCT-②

Module ③ (OOPS in Java, Collection framework)

30 days - 45 days (mid December)

→ Stack, Queue,
LinkedList,
HashMap, Map,
Recursion

10



Crossword

START

- ① Green → horizontal (Right)
- ② Brown → vertical (Down)
- ③ CAT → diagonal (Down right)
- ④ SIT → anti-diagonal (Down left)

```
int n = matrix.length;
int idx = 0;

// horizontal - right
for(int r = row, c = col; c < n && idx < word.length(), c++, idx++){
    if(matrix[r][c] != word.charAt(idx)) break;
}

if(idx == word.length()) return true;
```

```
public static boolean isWordPresent(char[][] matrix, int row, int col, String word){  
    // horizontal - right  
  
    // vertical - down  
  
    // down - right  
  
    // down - left  
  
    return false;|  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int n = scn.nextInt();  
    char[][] mat = new int[n][n];  
  
    for(int row = 0; row < n; row++){  
        for(int col = 0; col < n; col++){  
            mat[row][col] = scn.next().charAt(0);  
        }  
    }  
  
    String word = scn.next();  
  
    for(int row = 0; row < n; row++){  
        for(int col = 0; col < n; col++){  
            if(isWordPresent(matrix, row, col, word) == true){  
                System.out.println(true);  
                return;  
            }  
        }  
    }  
  
    System.out.println(false);  
}
```

```

public static boolean isWordPresent(char[][] matrix, int row, int col, String word){
    int n = matrix.length;
    int idx = 0;

    // horizontal - right
    for(int r = row, c = col; c < n && idx < word.length(); c++, idx++){
        if(matrix[r][c] != word.charAt(idx)) break;
    }

    if(idx == word.length()) return true;

    // vertical - down

    idx = 0;
    for(int r = row, c = col; r < n && idx < word.length(); r++, idx++){
        if(matrix[r][c] != word.charAt(idx)) break;
    }

    if(idx == word.length()) return true;

    // down - right
    idx = 0;
    for(int r = row, c = col; c < n && r < n && idx < word.length(); r++, c++, idx++){
        if(matrix[r][c] != word.charAt(idx)) break;
    }

    if(idx == word.length()) return true;

    // down - left
    idx = 0;
    for(int r = row, c = col; c >= 0 && r < n && idx < word.length(); r++, c--, idx++){
        if(matrix[r][c] != word.charAt(idx)) break;
    }

    if(idx == word.length()) return true;

    return false;
}

```

```

public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    char[][] mat = new char[n][n];

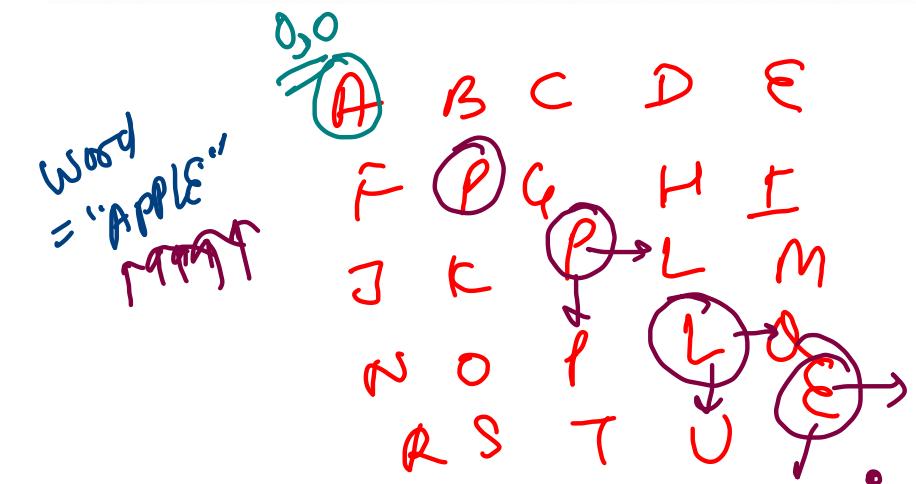
    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            mat[row][col] = scn.next().charAt(0);
        }
    }

    String word = scn.next();

    for(int row = 0; row < n; row++){
        for(int col = 0; col < n; col++){
            if(isWordPresent(mat, row, col, word) == true){
                System.out.println(true);
                return;
            }
        }
    }

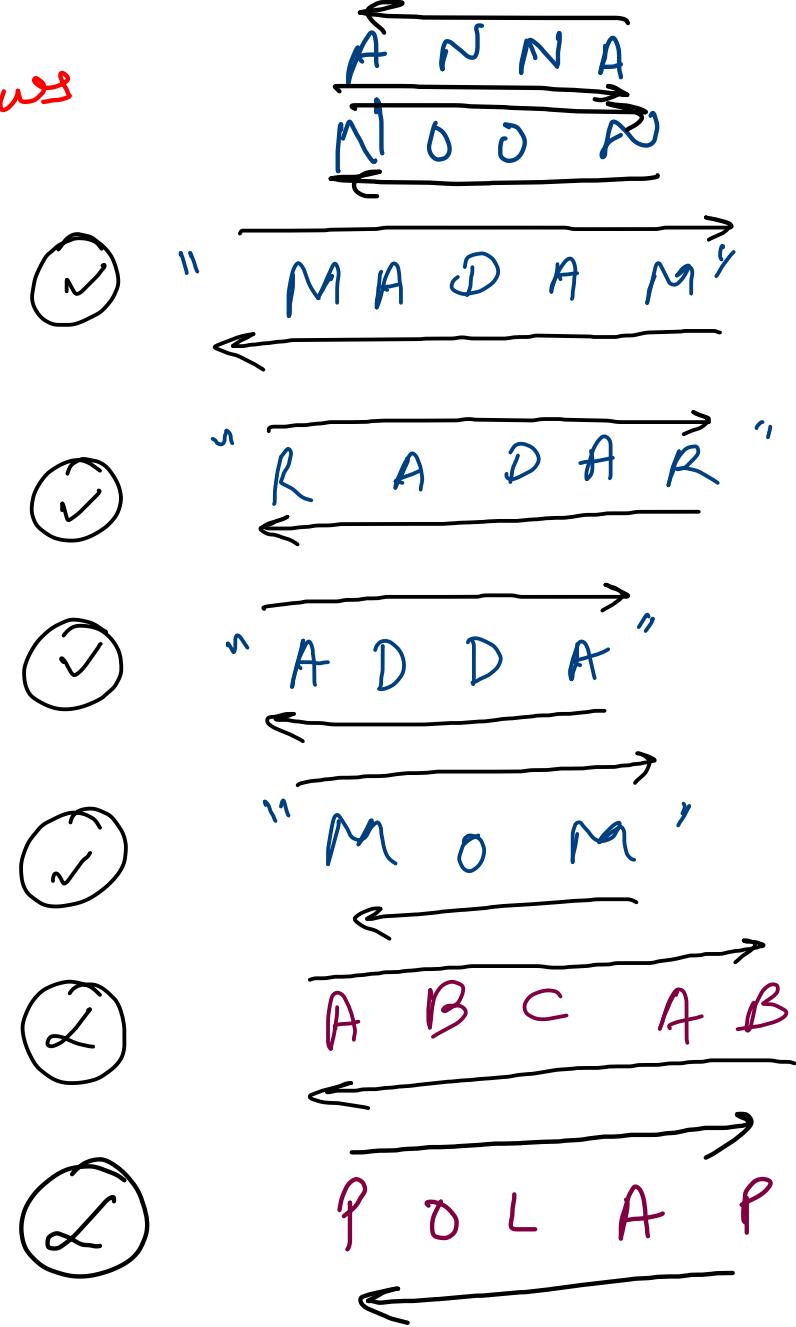
    System.out.println(false);
}

```



Count all Palindromic Rows

| | | | | |
|---|---|---|---|---|
| m | A | D | A | m |
| P | O | L | A | P |
| R | A | D | A | R |
| N | A | M | A | N |
| A | B | C | D | E |
| L | E | V | E | L |
| A | B | C | A | B |



odd length
characters

M A ~ L ~ A Y A ~ L ~ A ~ M

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
left left L R f f i Y right right

$l = \frac{n}{2}$

even length
palindrome

P Ü L ~ L ~ U P

↑ ↑ ↑ ↑ ↑ ↑
e L R f f i Y

$l > \frac{n}{2}$

~~false~~ A B C A B
↑ L

~~false~~ A B C C A
↑ L ↑ L
Y Y

```
public static boolean isPalindrome(int[][] matrix, int row){  
    int left = 0, right = matrix[0].length - 1;  
  
    while(left < right){  
        if(matrix[row][left] != matrix[row][right]) return false;  
        left++; right--;  
    }  
  
    return true;  
}
```

$O(n)$

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int rows = scn.nextInt();  
    int cols = scn.nextInt();  
  
    int[][] matrix = new int[rows][cols];  
    for(int row = 0; row < rows; row++){  
        for(int col = 0; col < cols; col++){  
            matrix[row][col] = scn.nextInt();  
        }  
    }  
  
    int count = 0;  
    for(int row = 0; row < rows; row++){  
        if(isPalindrome(matrix, row) == true) count++;  
    }  
    System.out.println(count);  
}
```

$n * O(n) = O(n^2)$

"Palindrome String"

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();

    int left = 0, right = str.length() - 1;
    while(left < right){
        if(str.charAt(left) != str.charAt(right)) {
            System.out.println("Not a Palindrome");
            return;
        }
        left++; right--;
    }

    System.out.println("Palindrome");
}
```

Matrix Addition/Subtraction

$$\begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \\ 100 & 110 & 120 \end{bmatrix}$$

4×3

mat 1
 $r_1 \times C_1$

+

$$\begin{bmatrix} 5 & 15 & 25 \\ 8 & 18 & 28 \\ 36 & 26 & 6 \\ 40 & 10 & 30 \end{bmatrix}$$

4×3

mat 2
 $r_2 \times C_2$

=

$$\begin{bmatrix} 10+5 & 20+15 & 30+25 \\ 40+8 & 50+18 & 60+28 \\ 70+36 & 80+26 & 90+6 \\ 100+40 & 110+10 & 120+30 \end{bmatrix}$$

4×3

mat
 $r_3 \times C_3$

constraint
 $\rightarrow r_1 = r_2 = r_3$
 $\rightarrow c_1 = c_2 = c_3$

Matrix Multiplication

→ Constraint: $\begin{cases} A_{r_1 \times c_1} \\ B_{r_2 \times c_2} \\ c_1 = r_2 \end{cases}$

| | | |
|--|--|--|
| | | |
| | | |
| | | |
| | | |
| | | |

*

| | | | |
|--|--|--|--|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

=

| | | | |
|--|--|--|--|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

$$A \ 5 \times 3$$

$$r_1 \times c_1$$

$$B \ 3 \times 4$$

$$r_2 \times c_2$$

$$C \ 5 \times 4$$

$$r_1 \times c_2$$

| | | |
|----|----|----|
| 01 | 02 | 03 |
| 10 | 11 | 12 |
| 20 | 21 | 22 |
| 30 | 31 | 32 |

τ_2^2

$$4 \times 3 \\ \tau_1 \wedge C_1$$

$$\begin{array}{ccc}
 & C_1 & \\
 \begin{array}{c} 4_0 \\ 5_0 \\ 6_0 \end{array} & \begin{array}{c} 4_1 \\ 5_1 \\ 6_1 \end{array} & \begin{array}{c} 4_2 \\ 5_2 \\ 6_2 \end{array} \\
 = & & \\
 \end{array}$$

$$3 \times 3 \\ \tau_1 \wedge C_2$$

| | C0 | C1 | C2 |
|----------|---|---|---|
| τ_0 | $01 \times 4_0 + f$ $02 \times 5_0 + f$ $03 \times 6_0 + f$ | $01 \times 4_1 + f$ $02 \times 5_1 + f$ $03 \times 6_1 + f$ | $01 \times 4_2 + f$ $02 \times 5_2 + f$ $03 \times 6_2 + f$ |
| τ_1 | $10 \times 4_0 + f$ $11 \times 5_0 + f$ $12 \times 6_0 + f$ | $10 \times 4_1 + f$ $11 \times 5_1 + f$ $12 \times 6_1 + f$ | $10 \times 4_2 + f$ $11 \times 5_2 + f$ $12 \times 6_2 + f$ |
| τ_2 | $20 \times 4_0 + f$ $21 \times 5_0 + f$ $22 \times 6_0 + f$ | $20 \times 4_1 + f$ $21 \times 5_1 + f$ $22 \times 6_1 + f$ | $20 \times 4_2 + f$ $21 \times 5_2 + f$ $22 \times 6_2 + f$ |
| τ_3 | $30 \times 4_0 + f$ $31 \times 5_0 + f$ $32 \times 6_0 + f$ | $30 \times 4_1 + f$ $31 \times 5_1 + f$ $32 \times 6_1 + f$ | $30 \times 4_2 + f$ $31 \times 5_2 + f$ $32 \times 6_2 + f$ |

$$4 \times 3$$

$$\tau_1 \wedge C_2$$

```
public static void matrixMultiplication(int[][] m1, int[][] m2){  
    int r1 = m1.length, c1 = m1[0].length;  
    int r2 = m2.length, c2 = m2[0].length;  
  
    if(c1 != r2) {  
        System.out.println(-1);  
        // Matrix Multiplication Not Possible  
        return;  
    }  
  
    int[][] m3 = new int[r1][c2];  
  
    for(int row = 0; row < r1; row++){  
        for(int col = 0; col < c2; col++){  
            for(int k = 0; k < r2; k++){  
                m3[row][col] = m3[row][col] + (m1[row][k] * m2[k][col]);  
            }  
        }  
    }  
  
    for(int row = 0; row < r1; row++){  
        for(int col = 0; col < c2; col++){  
            System.out.print(m3[row][col] + " ");  
        }  
        System.out.println();  
    }  
}
```

```
public static void main(String[] args) {  
    Scanner scn = new Scanner(System.in);  
    int r1 = scn.nextInt();  
    int c1 = scn.nextInt();  
  
    int[][] m1 = new int[r1][c1];  
    for(int row = 0; row < r1; row++){  
        for(int col = 0; col < c1; col++){  
            m1[row][col] = scn.nextInt();  
        }  
    }  
  
    int r2 = scn.nextInt();  
    int c2 = scn.nextInt();  
  
    int[][] m2 = new int[r2][c2];  
    for(int row = 0; row < r2; row++){  
        for(int col = 0; col < c2; col++){  
            m2[row][col] = scn.nextInt();  
        }  
    }  
  
    matrixMultiplication(m1, m2);  
}
```

| | | |
|----|----|----|
| 01 | 02 | 03 |
| 10 | 11 | 12 |
| 20 | 21 | 22 |
| 30 | 31 | 32 |

```

public static void matrixMultiplication(int[][] m1, int[][] m2){
    int r1 = m1.length, c1 = m1[0].length;
    int r2 = m2.length, c2 = m2[0].length;

    if(c1 != r2) {
        System.out.println(-1);
        // Matrix Multiplication Not Possible
        return;
    }

    int[][] m3 = new int[r1][c2];
    for(int row = 0; row < r1; row++){
        for(int col = 0; col < c2; col++){
            for(int k = 0; k < r2; k++){
                m3[row][col] = m3[row][col] + (m1[row][k] * m2[k][col]);
            }
        }
    }

    for(int row = 0; row < r1; row++){
        for(int col = 0; col < c2; col++){
            System.out.print(m3[row][col] + " ");
        }
        System.out.println();
    }
}

```

| | | |
|----|----|----|
| 40 | 41 | 42 |
| 50 | 51 | 52 |
| 60 | 61 | 62 |

3x3
x 3x2

3x3

$$mat3[1][2] = 0$$

$$0 + 1 \times 42 = 420$$

$$420 + 11 \times 52 = x$$

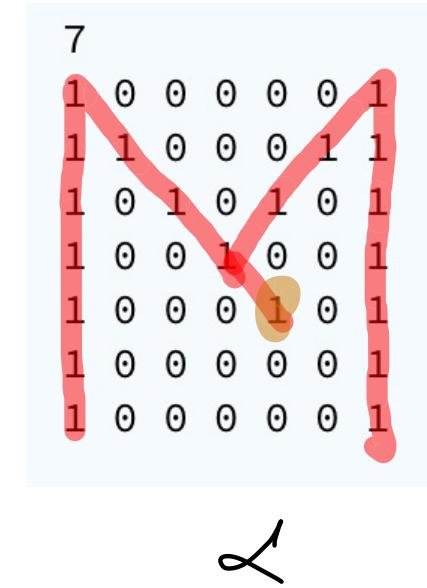
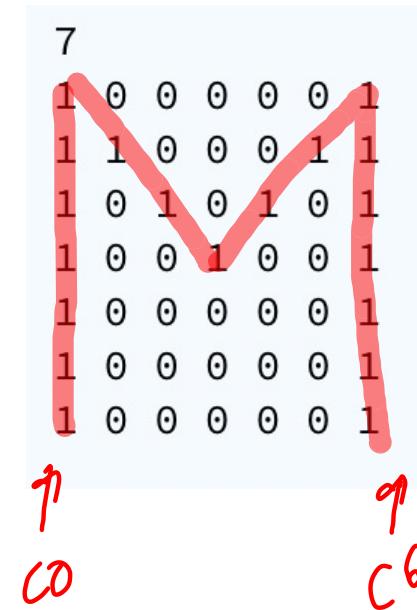
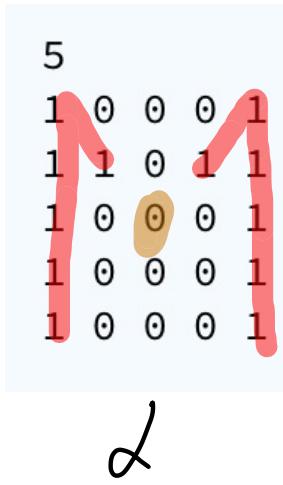
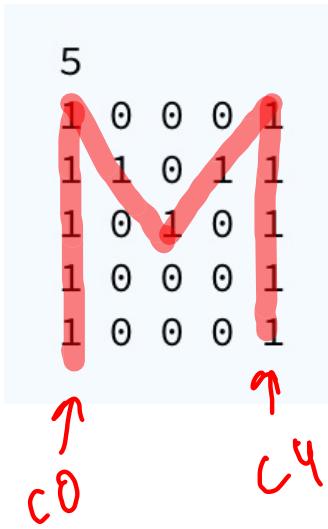
$$x + 12 + 62 = y$$

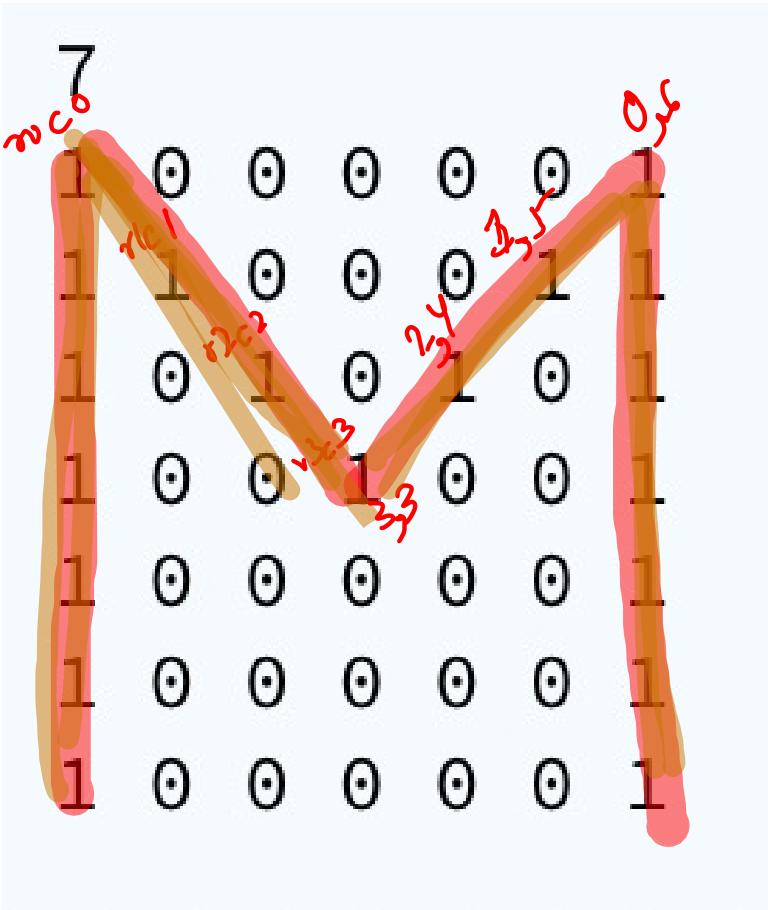
c0

c1

c2

| 00 | 01 | 02 |
|-----------------------------------|-----------------------------------|-----------------------------------|
| 01 * 40 + 02 * 50 + 03 * 60 | 01 * 41 + 02 * 51 + 03 * 61 | 01 * 42 + 02 * 52 + 03 * 62 |
| 10 | 11 | 12 |
| 10 * 40 + 11 * 50 + 12 * 60 | 10 * 41 + 11 * 51 + 12 * 61 | 10 * 42 + 11 * 52 + 12 * 62 |
| 20 | 21 | 22 |
| 20 * 40 + 21 * 50 + 22 * 60 | 20 * 41 + 21 * 51 + 22 * 61 | 20 * 42 + 21 * 52 + 22 * 62 |
| 30 | 31 | 32 |
| 30 * 40 + 31 * 50 + 32 * 60 | 30 * 41 + 31 * 51 + 32 * 61 | 30 * 42 + 31 * 52 + 32 * 62 |





```

    col == 0 || col == n-1 ||
    || (row == col && row <= n/2)
    || (row + col == n-1 && row <= n/2)
  
```

1

else 0

```
public class Solution {  
    public static boolean MMatrix(int[][] mat){  
        int n = mat.length;  
        for(int row = 0; row < n; row++){  
            for(int col = 0; col < n; col++){  
                if(col == 0 || col == n - 1 || (row == col && row <= n/2)  
                || (row + col == n - 1 && row <= n/2)){  
                    // M Element: Should be 1  
                    if(mat[row][col] == 0) return false;  
                } else {  
                    // Not on M: Should be 0  
                    if(mat[row][col] == 1) return false;  
                }  
            }  
        }  
  
        return true;  
    }  
    public static void main(String[] args) {  
        Scanner scn = new Scanner(System.in);  
        int n = scn.nextInt();  
  
        int[][] mat = new int[n][n];  
        for(int row = 0; row < n; row++){  
            for(int col = 0; col < n; col++){  
                mat[row][col] = scn.nextInt();  
            }  
        }  
  
        System.out.println(MMatrix(mat));  
    }  
}
```

Annotations on the code:

- anti** **diagonal**: A red bracket and arrow point to the condition `(row == col && row <= n/2)`.
- upper half**: A red bracket and arrow point to the condition `(row + col == n - 1 && row <= n/2)`.
- last col**: A red bracket and arrow point to the condition `col == n - 1`.
- upper half of diagonal**: A red bracket and arrow point to the condition `row == col && row <= n/2`.