Module-3 :- Java + DSA

Collection Frameworks

1. Arraylist
2. Stack
3. Queue
4. Priority Queue {Heap}
5. Hashmap

Object oriented Programming

→ Classes & objects

→ 5 pillars

1. encapsulation
2. datahiding
3. inheritance
4. polymorphism
5. abstraction

Recursion

↓

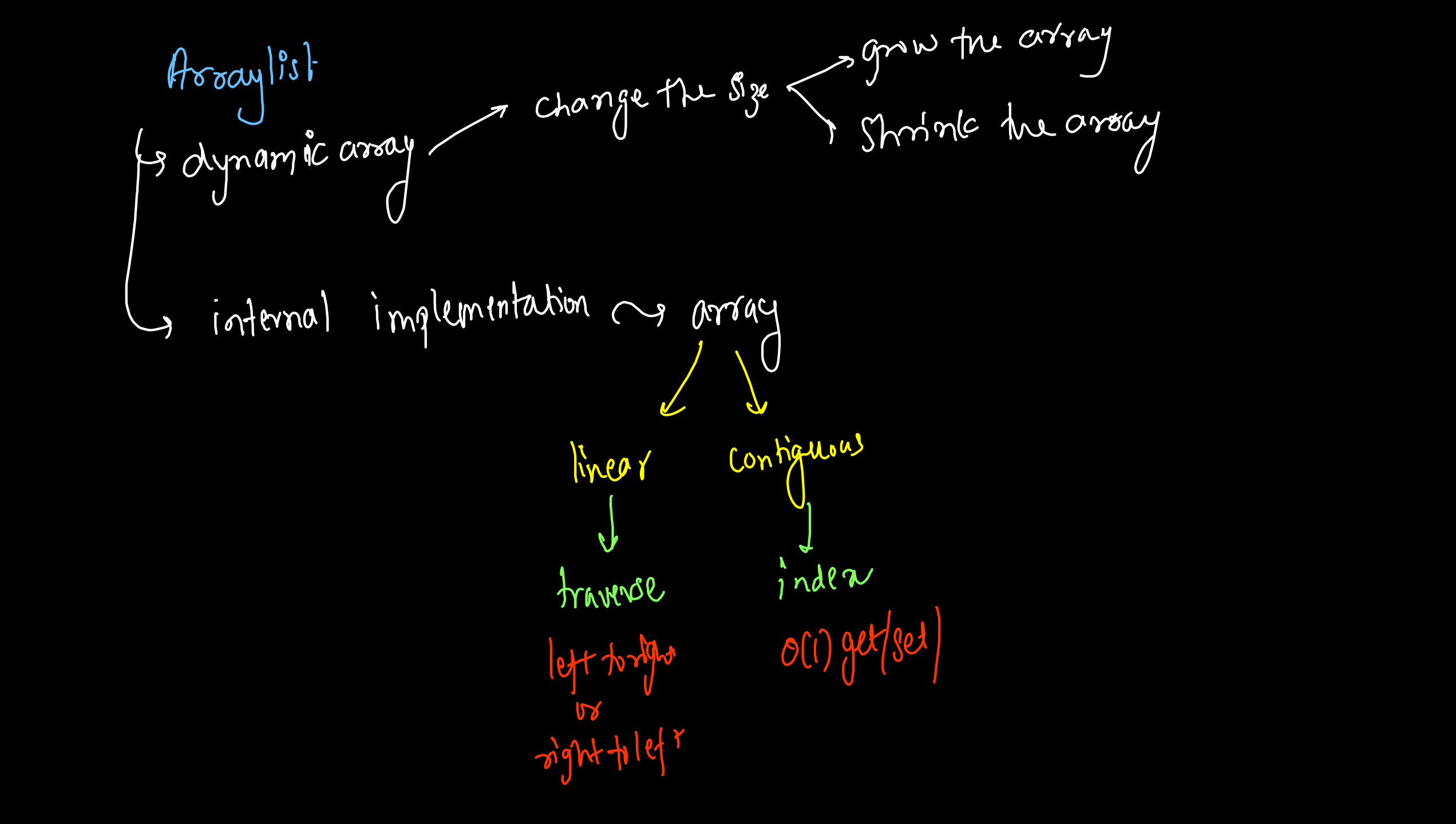
Elite Batch

Arrays

1. linear → forward traversal
2. contiguous memory allocation
   - random index access
   - O(1)
3. similar type of data (primitive array)
   - homogenous
4. Fixed size memory Allocation

```
int[] arr = new int[5]
arr = new int(10)
arr = new int[n];
```

- no addition
- no removal
- it will remain fixed size


ArrayList
dynamic array
change the size
grow the array
shrink the array
internal implementation
array
linear
contiguous
traverse
left to right
or
right to left
index
O(1) get/set

ArrayList<datatype> arr = new ArrayList<>();

↳ Osize arraylist → empty

```
ArrayList<Integer> arr = new ArrayList<>();
ArrayList<String> arr2 = new ArrayList<>();
ArrayList<Character> arr3 = new ArrayList<>();
ArrayList<Boolean> arr4 = new ArrayList<>();
ArrayList<Double> arr5 = new ArrayList<>();
ArrayList<Long> arr6 = new ArrayList<>();
```

Wrapper classes

Object

generics

int → Integer

char → Character

boolean → Boolean

long → Long

double → Double

```java
ArrayList<Integer> arr = new ArrayList<>();
System.out.println(arr);

arr.add(10);
System.out.println(arr);
arr.add(20);
System.out.println(arr);
arr.add(30);
System.out.println(arr);
arr.add(40);
System.out.println(arr);
```

Insert in the last of the arraylist!

→ O(1) constant

```
[]
[10]
[10, 20]
[10, 20, 30]
[10, 20, 30, 40]
```

array → length

String → length()

ArrayList → size()

```java
ArrayList<Integer> arr = new ArrayList<>();
System.out.println(arr);
System.out.println(arr.size());

arr.add(10);
System.out.println(arr);
System.out.println(arr.size());

arr.add(20);
System.out.println(arr);
System.out.println(arr.size());

arr.add(30);
System.out.println(arr);
System.out.println(arr.size());

arr.add(40);
System.out.println(arr);
System.out.println(arr.size());
```

```
Finished in 74 ms
[]
0
[10]
1
[10, 20]
2
[10, 20, 30]
3
[10, 20, 30, 40]
4
```

```java
ArrayList<Integer> arr = new ArrayList<>();
Scanner scn = new Scanner(System.in);
int size = scn.nextInt();

for(int count = 0; count < size; count++){
    int var = scn.nextInt();
    arr.add(var);
    System.out.println(arr);
}
```

User inputted arraylist

```
Finished in 147 ms
[10]
[10, 20]
[10, 20, 30]
[10, 20, 30, 40]
[10, 20, 30, 40, 50]
[10, 20, 30, 40, 50, 60]
[10, 20, 30, 40, 50, 60, 70]
```

```java
ArrayList<Integer> arr = new ArrayList<>();
arr.add(10);
arr.add(20);
arr.add(30);
arr.add(40);
arr.add(50);

// Traversal
for(int idx = 0; idx < arr.size(); idx++){
    System.out.println(arr.get(idx));
}
```

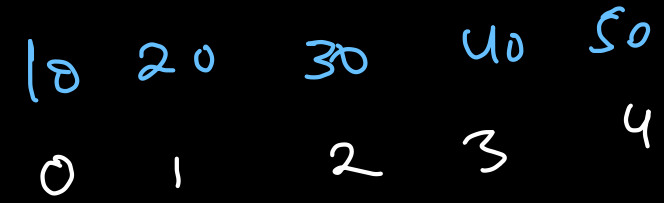
→ left to right traversal

```
Finished in 89 ms
10
20
30
40
50
```

arr.length → arr.size()

arr[idx] → arr.get(idx)

```java
public static void main(String[] args) {
    ArrayList<Integer> arr = new ArrayList<>();

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

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

    // For Loop: Forward Traveral: Indices
    for(int idx = 0; idx < arr.size(); idx++){
        System.out.print(arr.get(idx) + " ");
    }

    System.out.println();

    // For Each Loop: Forward Traversal: Values
    for(int val : arr){
        System.out.print(val + " ");
    }
}
```



index → value = arr.get(idx)

value → ~~index~~

```
// (OPTIONAL: EXTRA GYAAN) For Each Method: Lambda Expression
arr.forEach((val) -> System.out.print(val + " "));
```

```java
public static void main(String[] args) {
    ArrayList<Integer> arr = new ArrayList<>();

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

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

    // For Loop: Backward Traveral: Indices
    for(int idx = arr.size() - 1; idx >= 0; idx--){
        System.out.print(arr.get(idx) + " ");
    }

    System.out.println();

    Collections.reverse(arr);
    // Reverse The ArrayList: Two Pointer

    // For Each Loop: Forward Traversal: Values
    for(int val : arr){
        System.out.print(val + " ");
    }
}
```

Arraylist Reverse Printing

Arraylist if-else

10 20 30 40 50

Format for next T Lines : $(case, x(optional))$

- **case 1**: Print the size of the ArrayList in a separate line. size()
- **case 2**: Print and Remove element from the last index of the ArrayList. remove(arr.size()-1);
- **case 3**: Print x and Add x in last index of the ArrayList. add(x);
- **case 4**: Print and Remove an element from the starting (index = 0) of the ArraList. remove(0);
- **case 5**: Print x and Add x at beginning (index = 0) of the ArrayList. add(x,0);
- **case 6**: Print all the elements from left to right that are there inside the ArrayList. → for(int idx=0 ;idx<n ;idx++) Syso(arr.get(idx));

① add() → insert

② size() → count

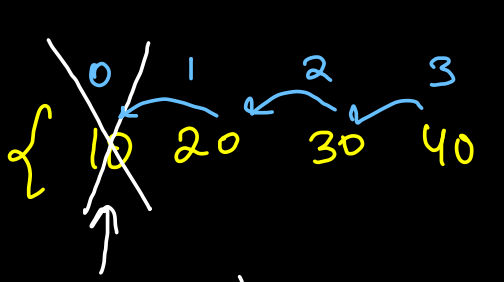
③ get() → read

Sample Input 0

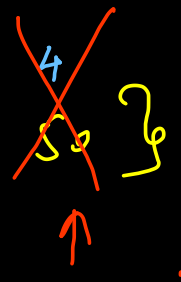
```
8
2 removeLast
6 traversal
3 2 addLast(2)
5 1 addFirst(1)
6 traversal
1 size
3 3 addLast(3)
2 removeLast()
```

{ 1, 2, ~~3~~ }

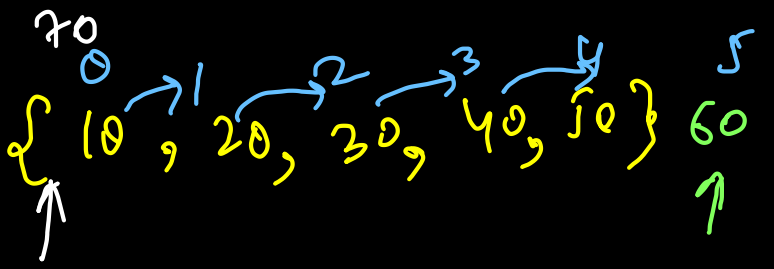
invalid-move
invalid-move
2
1
1 2
2
3
3



removeFirst
: O(n)
shifting
costly



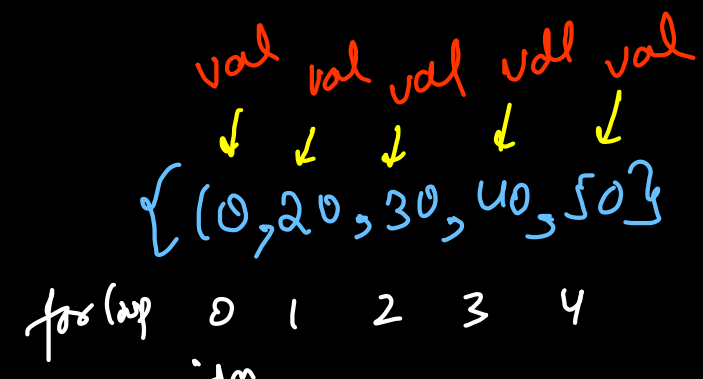
removeLast
O(1)
efficient



insert
at first
arr.add(70, 0)
O(n)

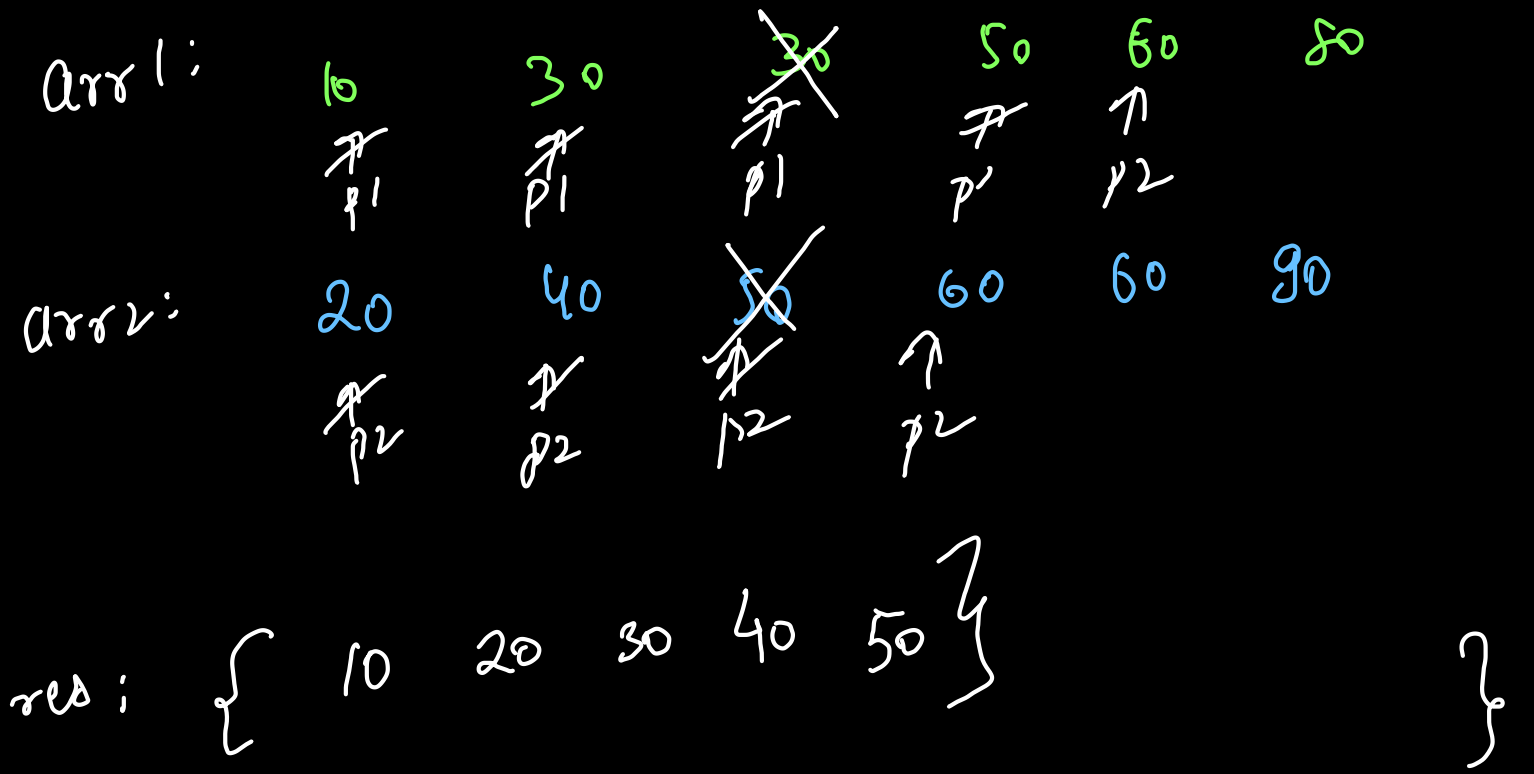
arr.add(60)
O(1)

```java
public static void main(String[] args) {
    ArrayList<Integer> arr = new ArrayList<>();

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

    for(int t = 0; t < tests; t++){
        int caseNo = scn.nextInt();

        if(caseNo == 1){

            System.out.println(arr.size()); // print size : O(1)

        }
        else if(caseNo == 2){

            if(arr.size() == 0) System.out.println("invalid-move"); // empty
            else System.out.println(arr.remove(arr.size() - 1)); // remove from last : O(1)

        }
        else if(caseNo == 3){

            int x = scn.nextInt();
            System.out.println(x);
            arr.add(x); // insert at last : O(1)

        }
```




```java
        else if(caseNo == 4){

            if(arr.size() == 0) System.out.println("invalid-move"); // empty
            else System.out.println(arr.remove(0)); // remove from front : O(n) shifting

        }
        else if(caseNo == 5){

            int x = scn.nextInt();
            System.out.println(x);
            arr.add(0, x); // insert at front : O(n) shifting

        }
        else {

            if(arr.size() == 0) System.out.println("invalid-move"); // empty
            else {
                for(int val: arr){ // for each loop traversal
                    System.out.print(val + " ");
                }
                System.out.println();
            }
        }
    }
```

Merge 2 Sorted Arraylists



```java
public static ArrayList<Integer> merge(int[] a1, int[] a2){
    int p1 = 0, p2 = 0;
    ArrayList<Integer> res = new ArrayList<>();

    while(p1 < a1.length && p2 < a2.length){
        if(a1[p1] <= a2[p2]){
            if(res.size() == 0 || res.get(res.size() - 1) != a1[p1])
                res.add(a1[p1]);
            p1++;
        } else {
            if(res.size() == 0 || res.get(res.size() - 1) != a2[p2])
                res.add(a2[p2]);
            p2++;
        }
    }

    while(p1 < a1.length){
        if(res.size() == 0 || res.get(res.size() - 1) != a1[p1])
             res.add(a1[p1]);
        p1++;
    }

    while(p2 < a2.length){
        if(res.size() == 0 || res.get(res.size() - 1) != a2[p2])
             res.add(a2[p2]);
        p2++;
    }

    return res;
}
```

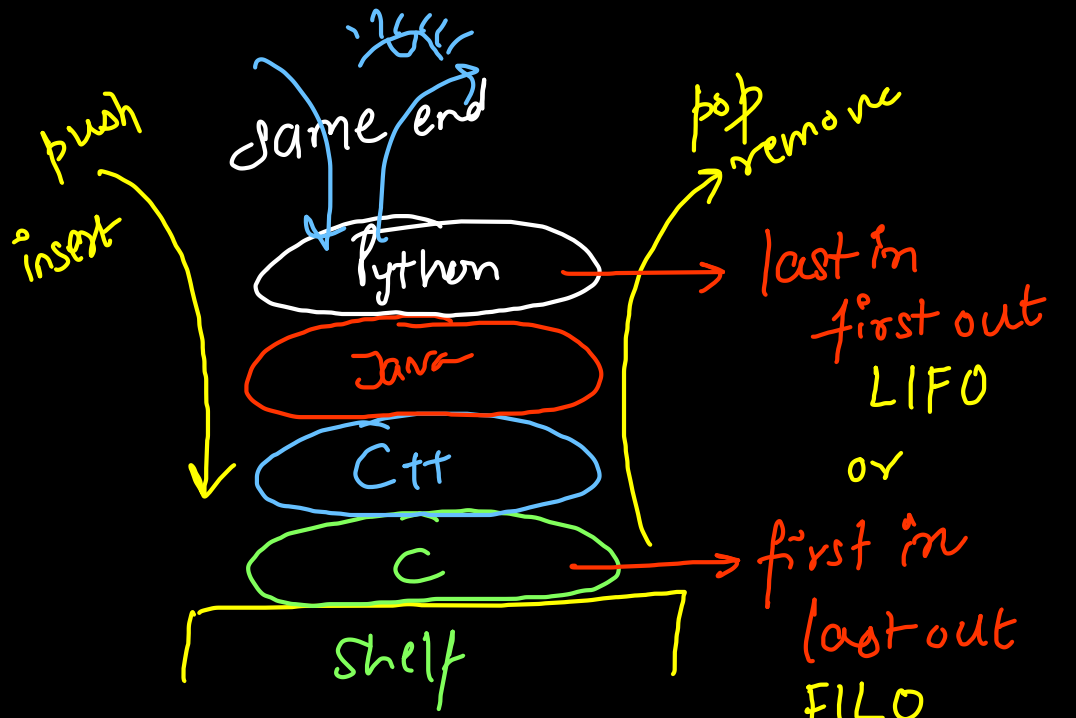
```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    int n = scn.nextInt();
    int[] arr1 = new int[n];

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

    int m = scn.nextInt();
    int[] arr2 = new int[m];

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

    ArrayList<Integer> res = merge(arr1, arr2);
    for(int val: res){
        System.out.print(val + " ");
    }
}
```

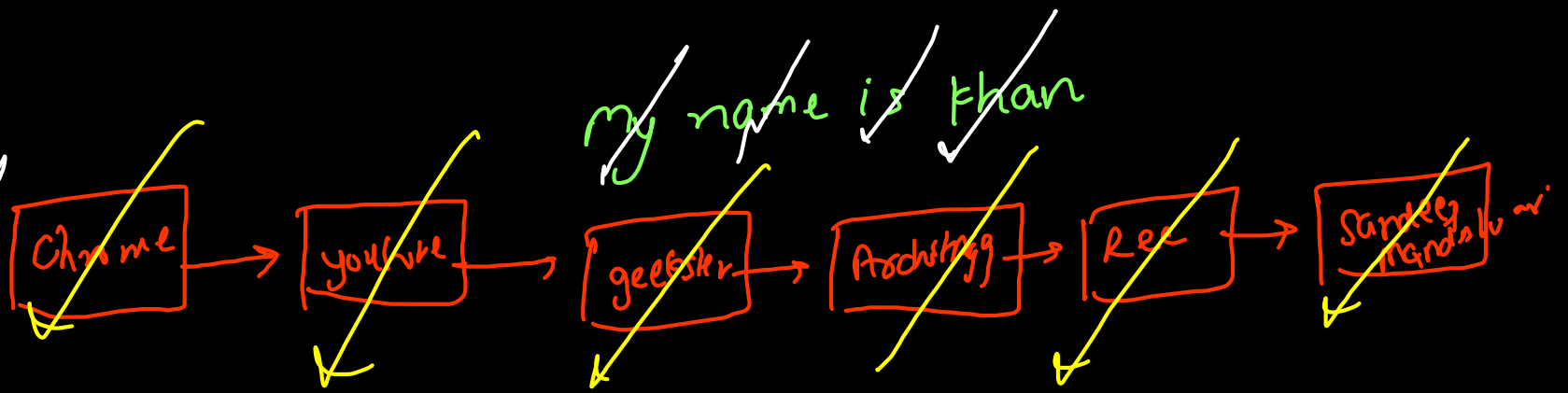
Stack Data Structure

eg

1) pile of plates
2) undo-redo
3) pile of books
4) cd holder
5) webpages
6) function/recursion callstack

~~ArrayList~~ Stack





Stack Data Structure

LIFO or FILO policy

```
Stack<Integer> stk = new Stack<>();
```

(1) push() : insert O(1)

(2) pop() : last O(1)

(3) peek() : get the top element O(1)

(4) empty() ↔ true : 0, false : 0 } → O(1)

(5) size() : count of elements → O(1)

```java
public static void main(String[] args){
    Stack<Integer> stk = new Stack<>();

    System.out.println(stk.empty());
    System.out.println(stk.size());

    stk.push(10);
    stk.push(20);
    stk.push(30);
    stk.push(40);

    System.out.println(stk.empty());
    System.out.println(stk.size());
    System.out.println(stk.peek());
    System.out.println(stk);

    while(stk.size() > 0){
        int val = stk.pop();
        System.out.print(val + " ");
    }
}
```

get ✓ delete ✗

get ✓ delete ✓

```
Finished in 102 ms
true
0
false
4
40
[10, 20, 30, 40]
40 30 20 10
```

1. **Declare an Empty** $stack\ s$.

2. Take Single Integer T as input.

3. For next T Lines format $(case, x(optional))$

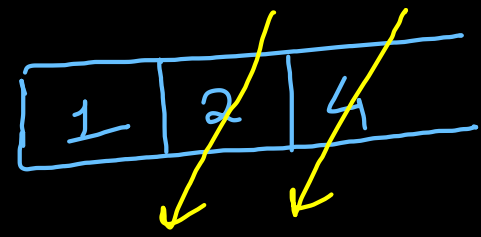
- case 1. $Print$ the $size$ of the $stack$ in a separate line.
- case 2. $Remove$ an element from the stack. If the stack is empty then print -1 in a separate line.
- case 3. Add Integer x to the $stack\ s$.
- case 4. $Print$ an element at the top of the $stack$. If stack is empty print -1 in a seperate line.

size()

empty() ? -1 : stk.pop()

stk.push(x);

empty() ? -1 : stk.peek()




```
10
3 1
3 2
4
4
2
4
3 4
2
4
1
```

stk.add(1)
stk.add(2)
stk.peek() : 2
stk.peek() : 2
stk.pop()
stk.peek() : 1
stk.add(4)
stk.pop()
stk.peek : 1
stk.size() : 1

```java
public class Solution {

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

        Stack<Integer> stk = new Stack<>();

        int tests = scn.nextInt();
        for(int t = 0; t < tests; t++){
            int choice = scn.nextInt();

            if(choice == 1){

                System.out.println(stk.size()); // size

            } else if(choice == 2){

                if(stk.empty()) System.out.println(-1);
                else stk.pop(); // remove

            } else if(choice == 3){

                int x = scn.nextInt();
                stk.push(x); // insert

            } else {

                if(stk.empty()) System.out.println(-1);
                else System.out.println(stk.peek()); // get top

            }
        }
    }
}
```

O(Test cases)

O(1)

O(1)

O(1)

O(1)

O(1)

Reverse Integer using Stack

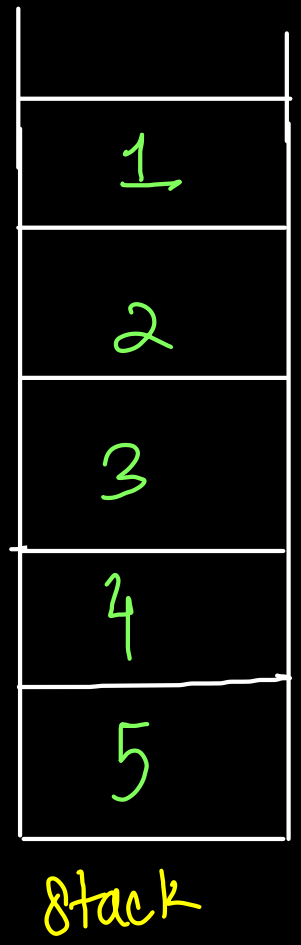
4

int 12345 → 54321

int 70000 → 7

int 70707 → 70707

98765 → 56789



12345
%10→5
/10

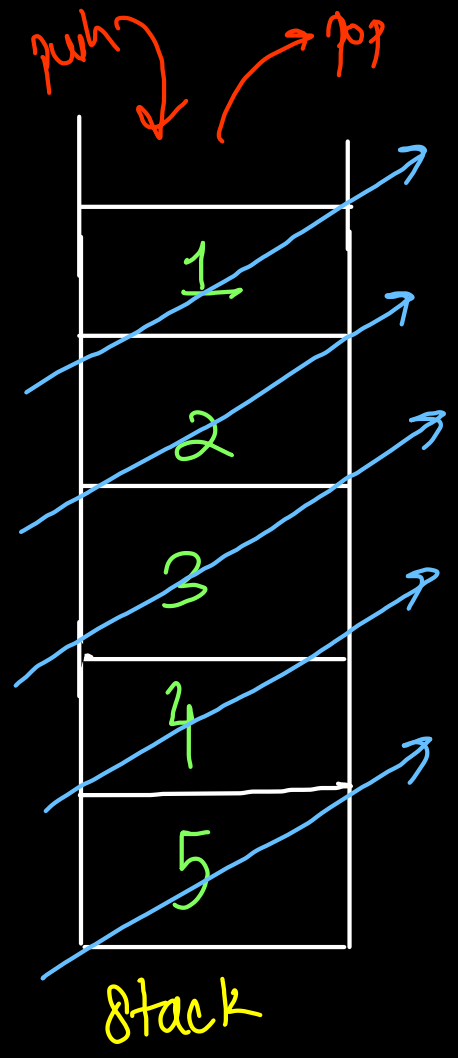
1234
%10→4
/10

123
%10→3
/10

12
%10→2
/10

1
%10→1
/10

0



5432_1

$= 5*10^4 + 4*10^3 + 3*10^2 + 2*10^1 + 1*10^0$

$res = 0 + 1*10^0 = 1 + 2*10 = 21 + 3*100$

$= 321$

$+4*1000$

$= 4321$

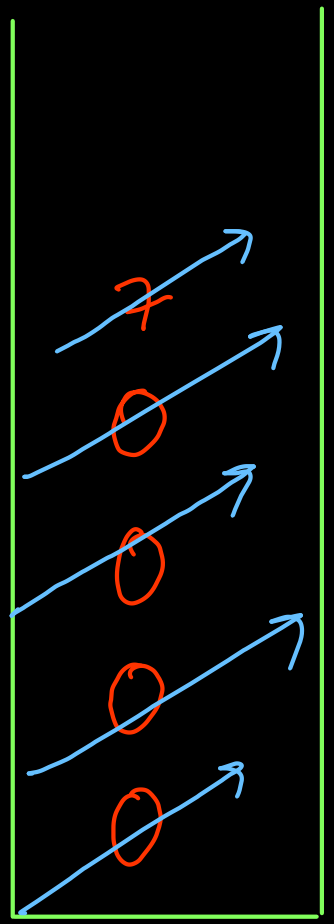
$+5*10000$

$= 54321$

power = $1*10 = 10*10 = 100*10 = 1000$

$*10 = 10000$

required ans



① till the stack

70000

%10 = 0

/10

7000

%10 = 0

/10

700

%10 = 0

/10

70

%10 = 0

/10

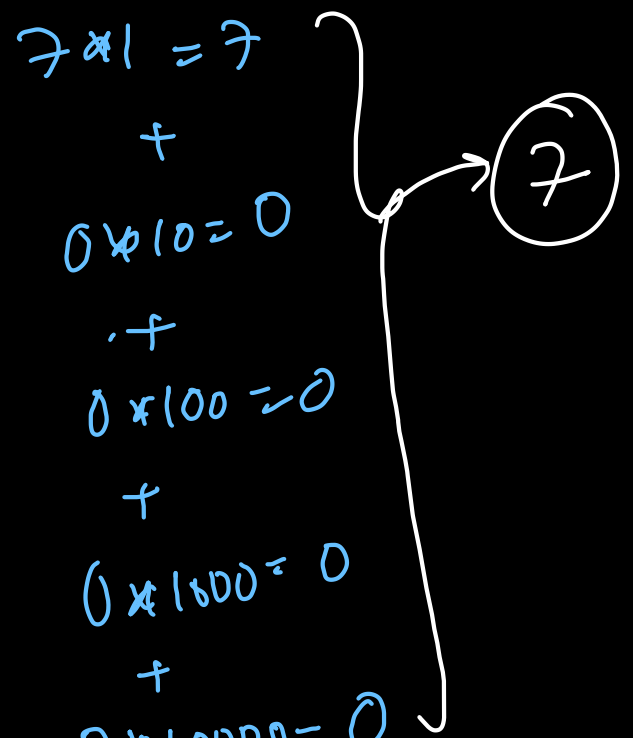
7

%10 = 7

/10

0

power = $10^0 = 1 * 10 * 10 * 10 * 10$



```java
public static int reverse(int num){
    Stack<Integer> stk = new Stack<>();

    // 1. fill the stack
    while(num > 0){
        int digit = num % 10;
        stk.push(digit);
        num = num / 10;
    }

    // 2. clear the stack
    int res = 0, power = 1;

    while(stk.size() > 0){
        int digit = stk.pop();

        res = res + digit * power;
        power = power * 10;
    }

    return res;
}
```

digits

int → log10 n digits

digits

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

    for(int t = 0; t < tests; t++){
        int num = scn.nextInt();
        System.out.println(reverse(num));
    }
}
```

$O(T * (10+10))$

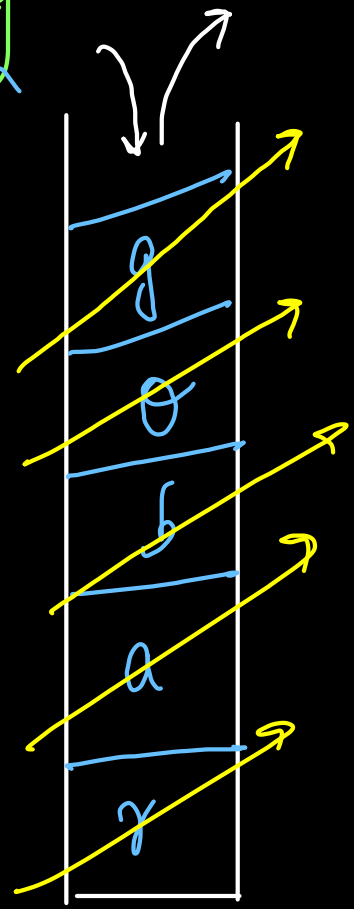
$O(T)$ $O(T * \log_{10} n)$

$\log_{10} n \leq 10$

Reverse Stack Using Stack

String

" r a b o g "



res{StringBuilder}

"" + 'g' → "g"

"g" + 'o' → "go"

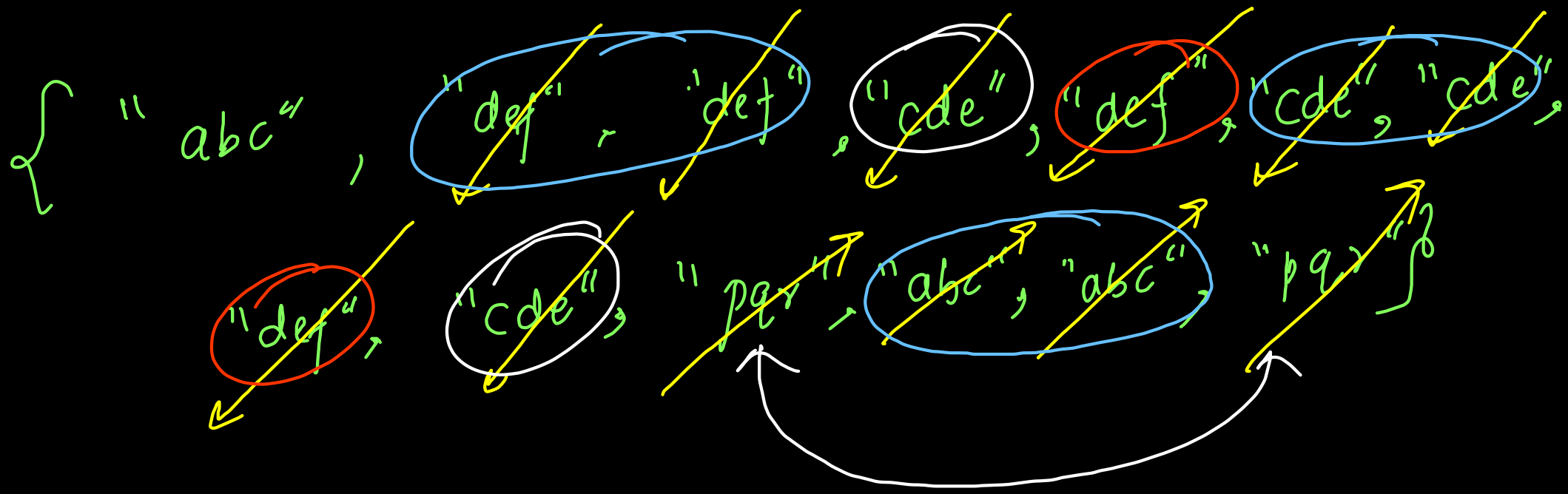
"go" + 'b' → "gob"

"gob" + 'a' → "goba"

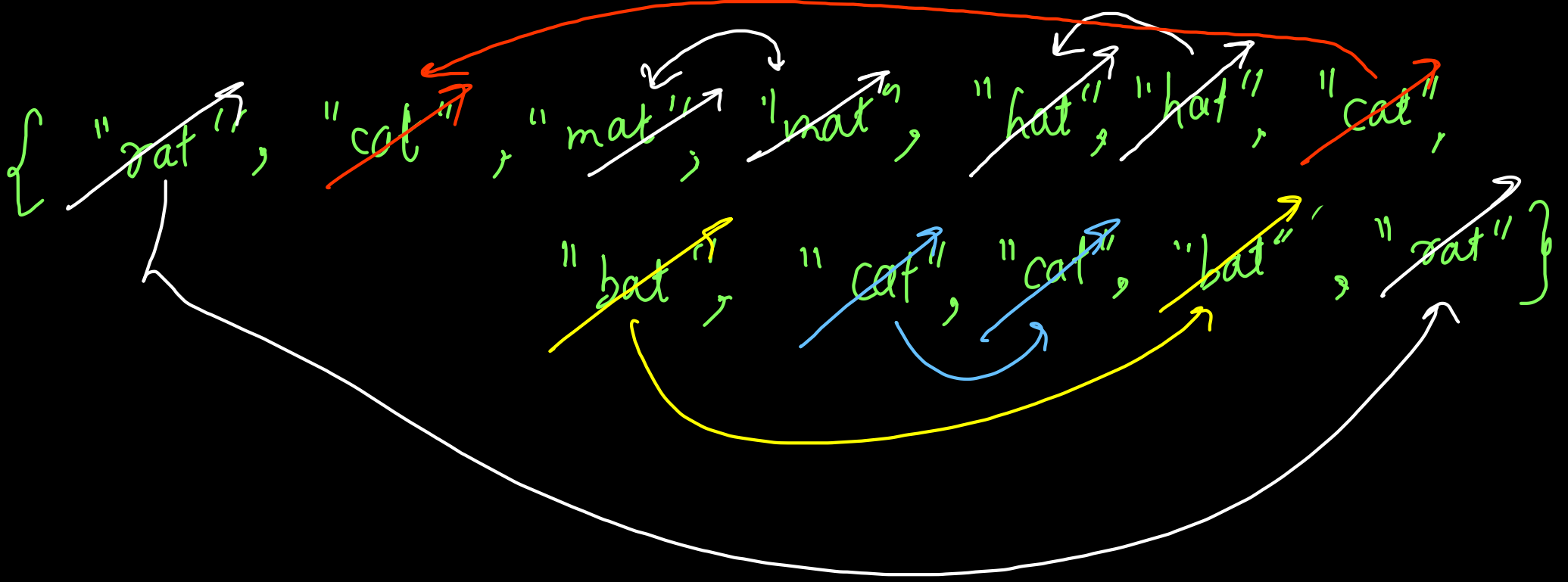
"goba" + 'r' → "gobar"

```java
public static String reverse(String str){
    Stack<Character> stk = new Stack<>();

    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        stk.push(ch);
    }

    StringBuilder res = new StringBuilder();

    while(stk.size() > 0){
        char ch = stk.pop();
        res.append(ch);
    }

    return res.toString();
}
```

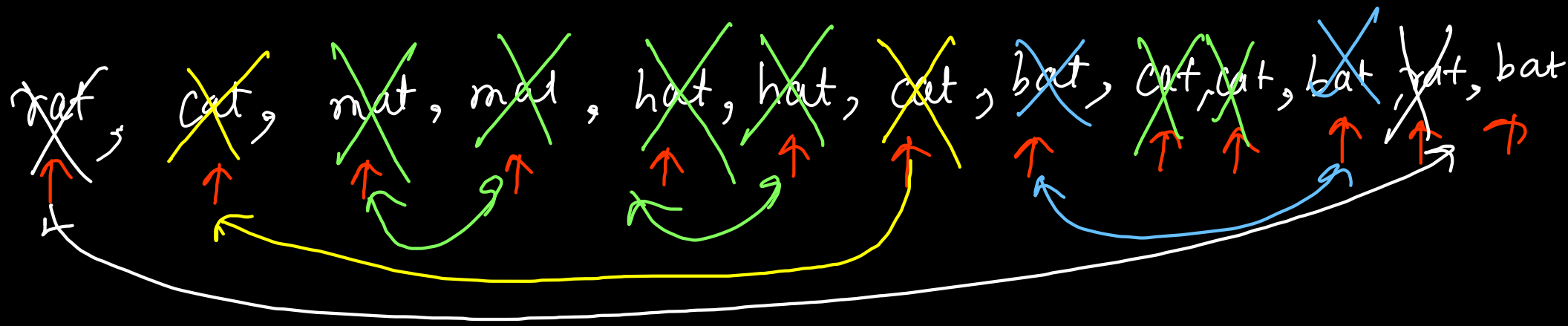
Remove All Adjacent Duplicates Recursively

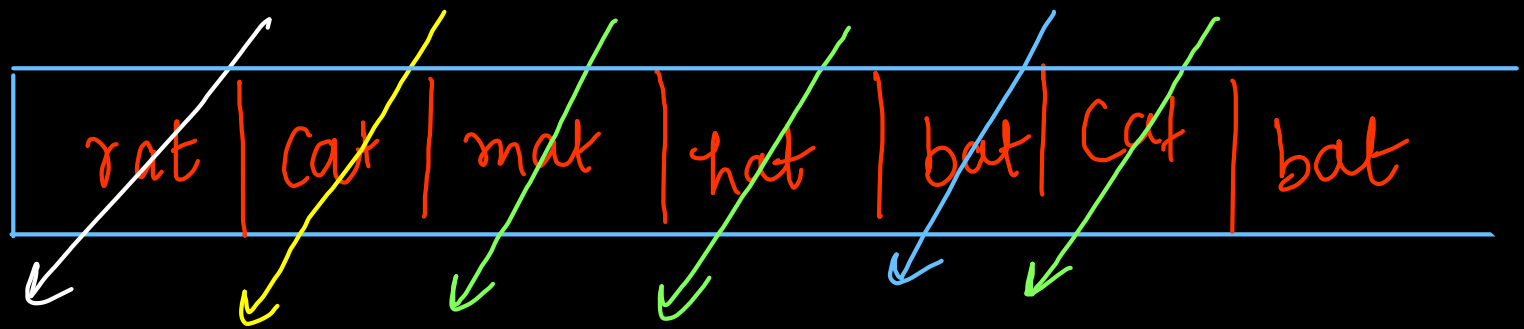


1 remaining



0 string remaining





stk.size() ⇒ 1

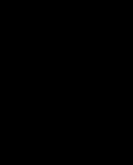
```
if(stk.size()==0 || str.equals(stk.peek())==false)
        stk.push(str);
else    stk.pop();
```

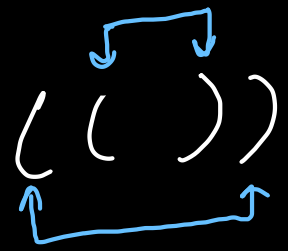
```java
public static void main(String[] args) {
    Stack<String> stk = new Stack<>();

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

    for(int idx = 0; idx < n; idx++){
        String str = scn.next();

        if(stk.size() == 0 || str.equals(stk.peek()) == false){
            stk.push(str);
        } else {
            stk.pop();
        }
    }

    System.out.println(stk.size());
}
```

O(n)

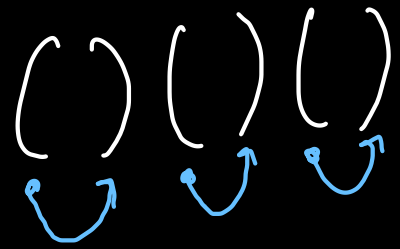
Leetcode ★★☆

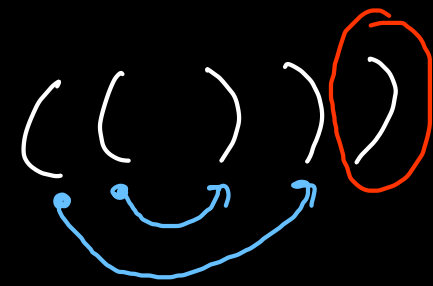
Balanced / Valid Parantheses {Stack}

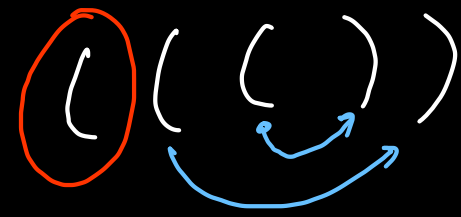


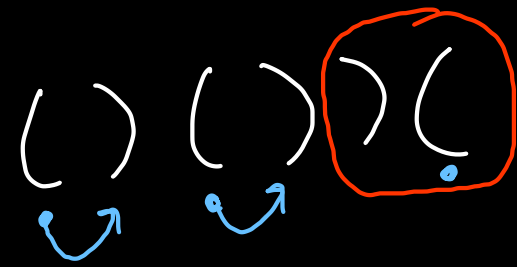
(i)  yes

(ii)  yes

(iii)  yes

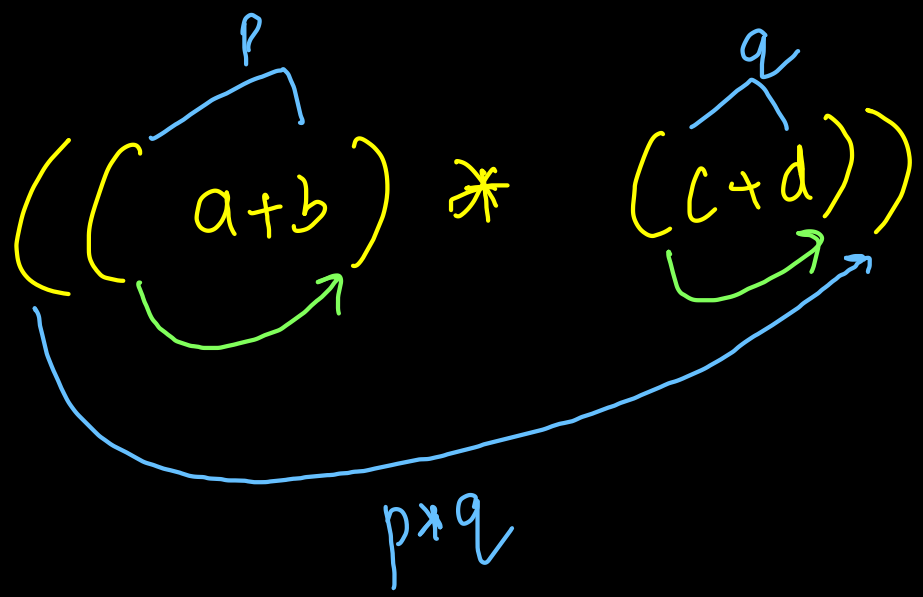
(iv)  no

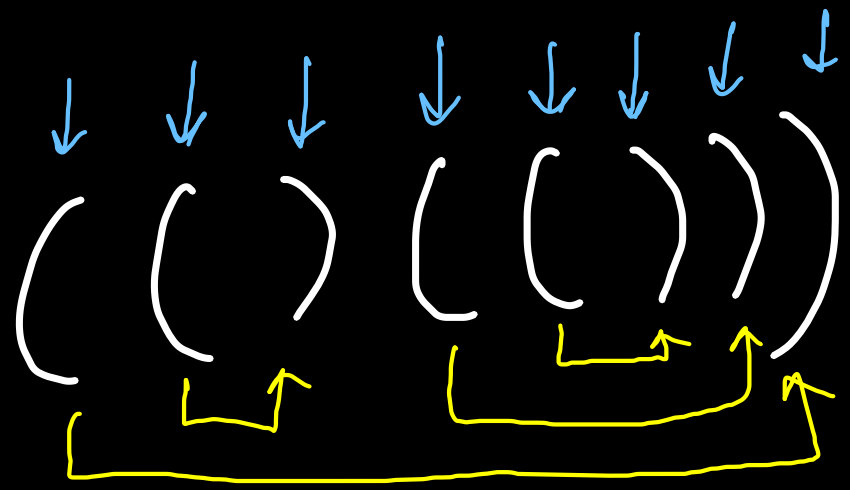
(v)  no

(vi)  no

push → ?

pop → ?

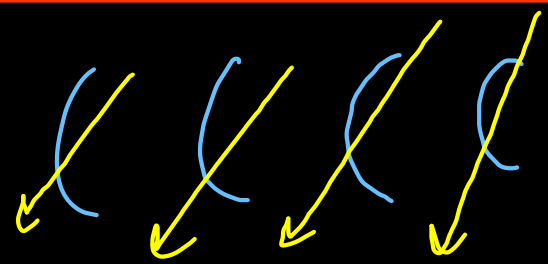

p
q
a+b
*
c+d
p*q

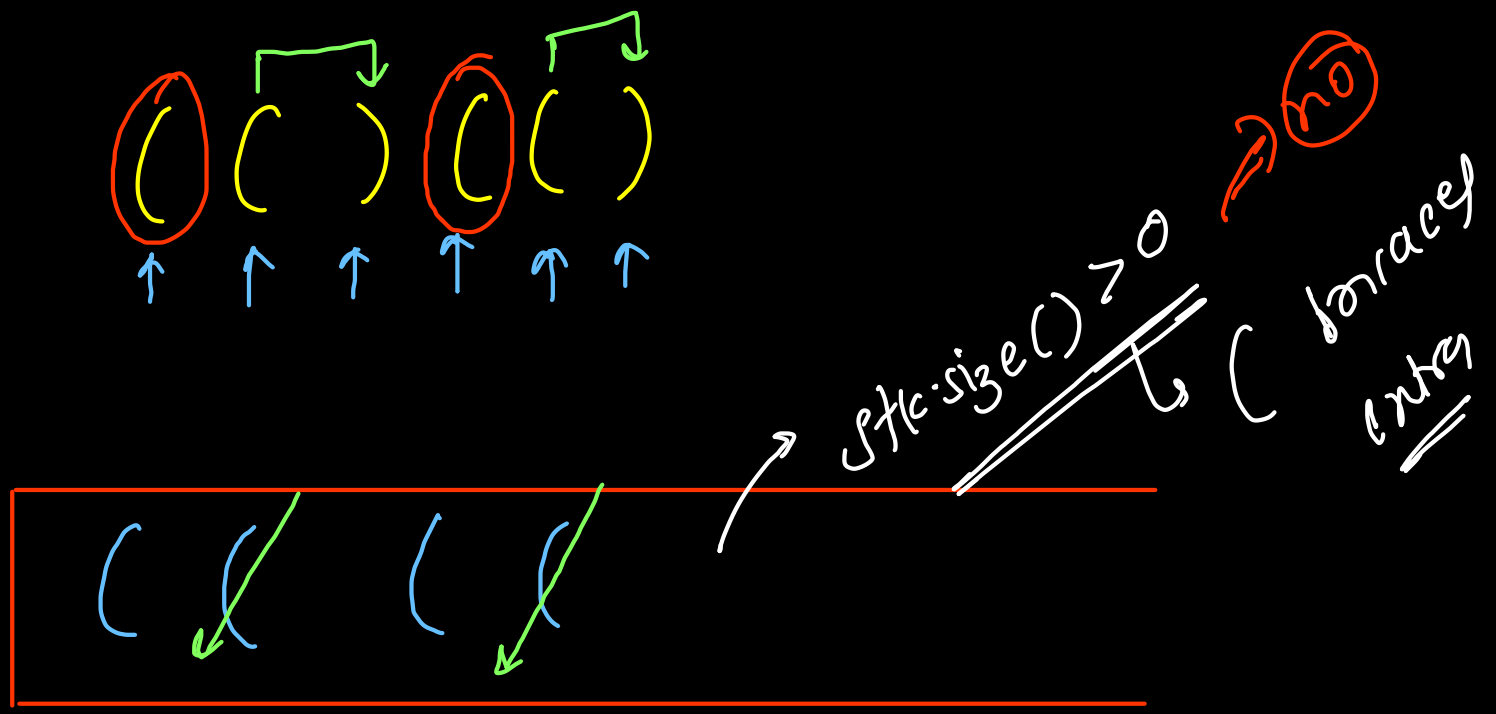


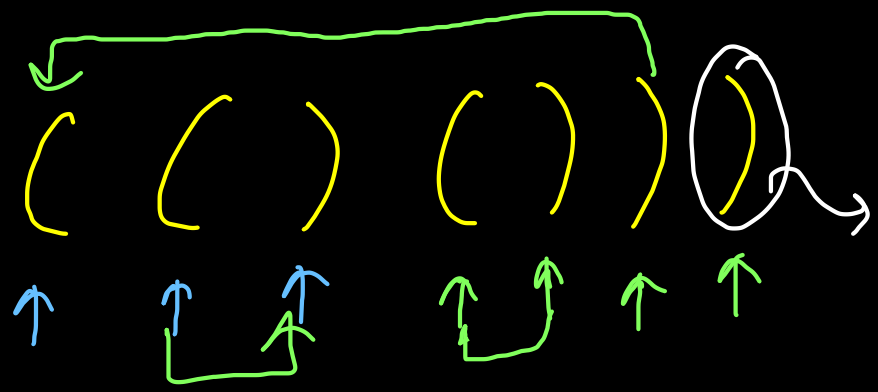
(→ push

) → pop

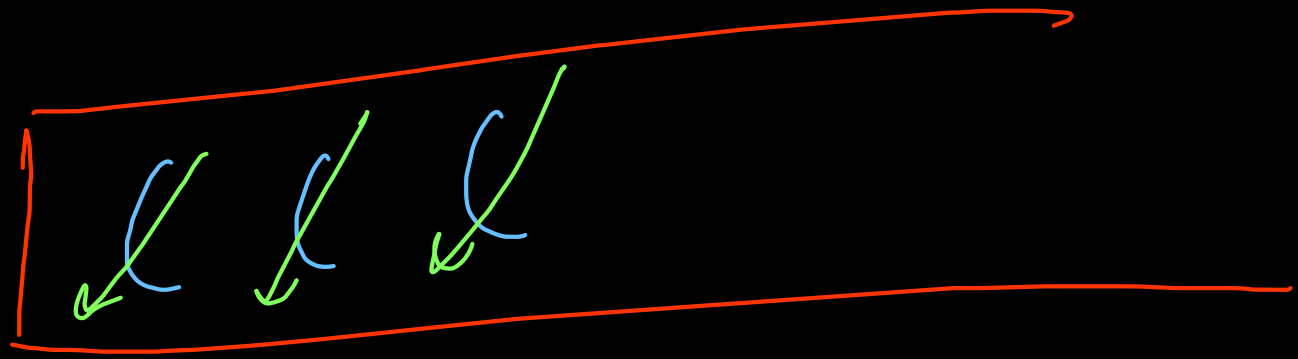
yes




st.size() > 0
no
(for each entry



stk.size() == 0 ⇒ extra closing brac1

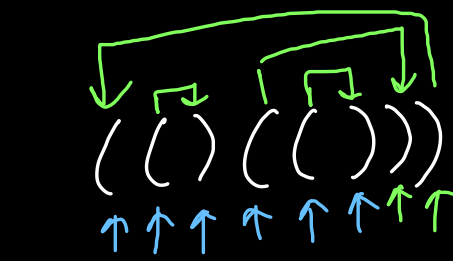


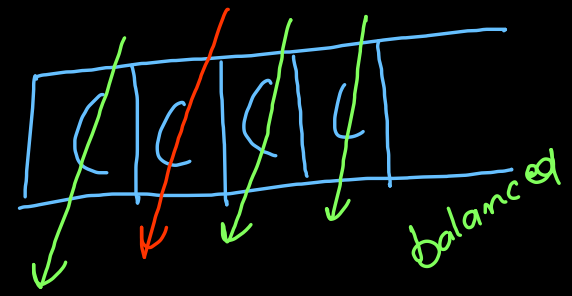
```java
public static boolean validParantheses(String str){
    Stack<Character> stk = new Stack<>();

    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);

        if(ch == '('){
            stk.push('(');
        } else if(ch == ')'){
            if(stk.size() == 0) return false; // closing braces are extra
            stk.pop();
        }
    }

    if(stk.size() > 0) return false; // opening braces are extra
    return true;
}
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();

    System.out.println(validParantheses(str));
}
```

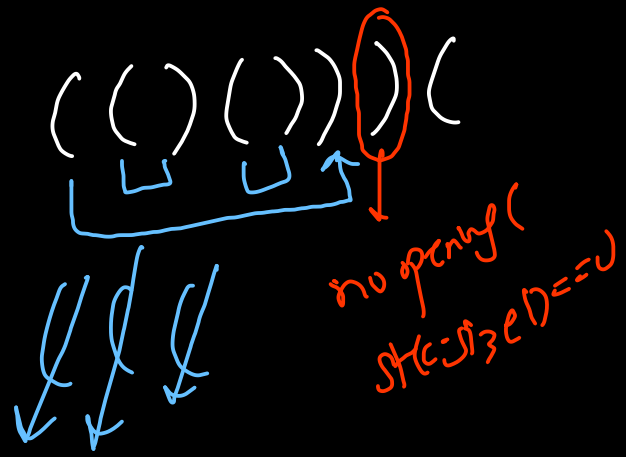
)a+b(

```java
public static boolean validParantheses(String str){
    Stack<Character> stk = new Stack<>();

    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);

        if(ch == '('){
            stk.push('(');
        } else if(ch == ')'){
            if(stk.size() == 0) return false; // closing braces are extra
            stk.pop();
        }
    }

    if(stk.size() > 0) return false; // opening braces are extra
    return true;
}
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();

    System.out.println(validParantheses(str));
}
```

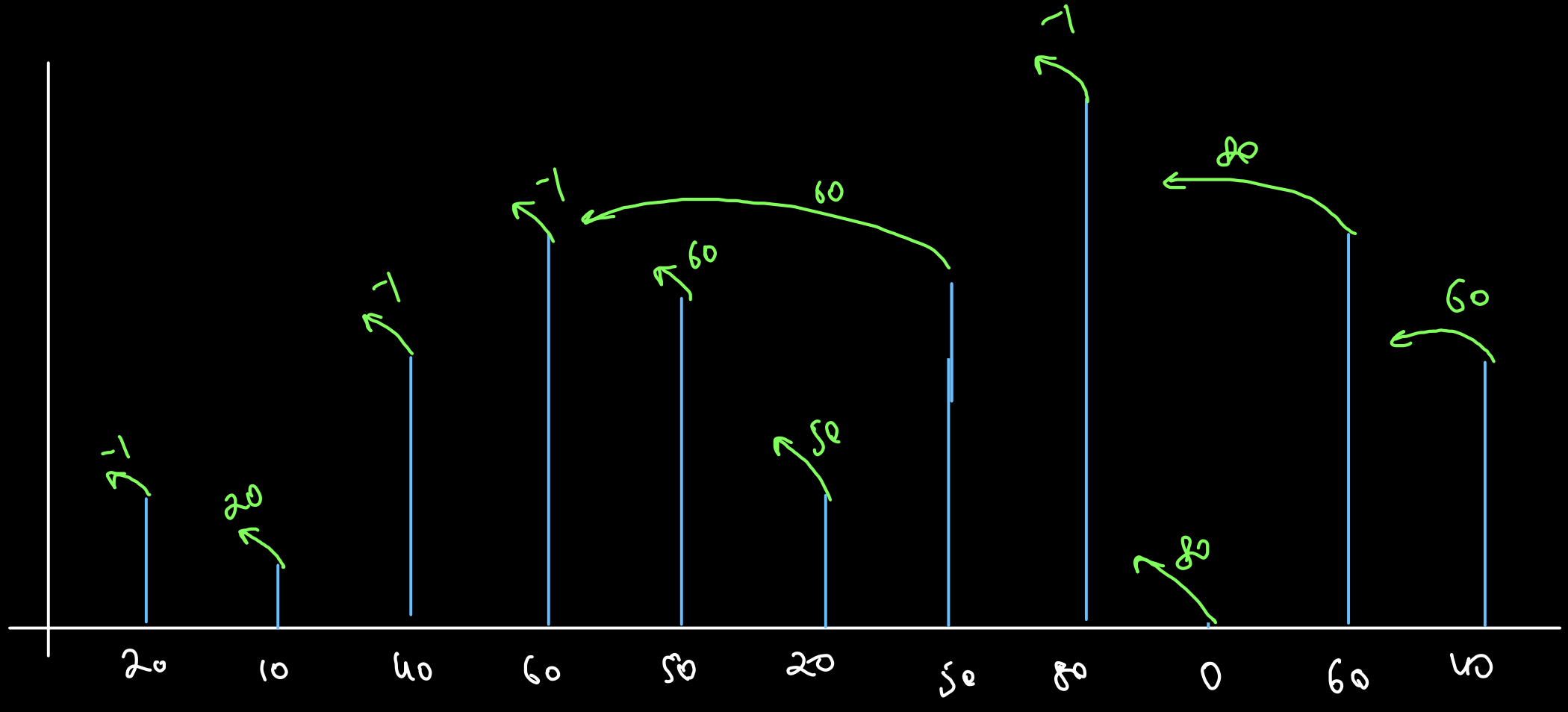
O(N)



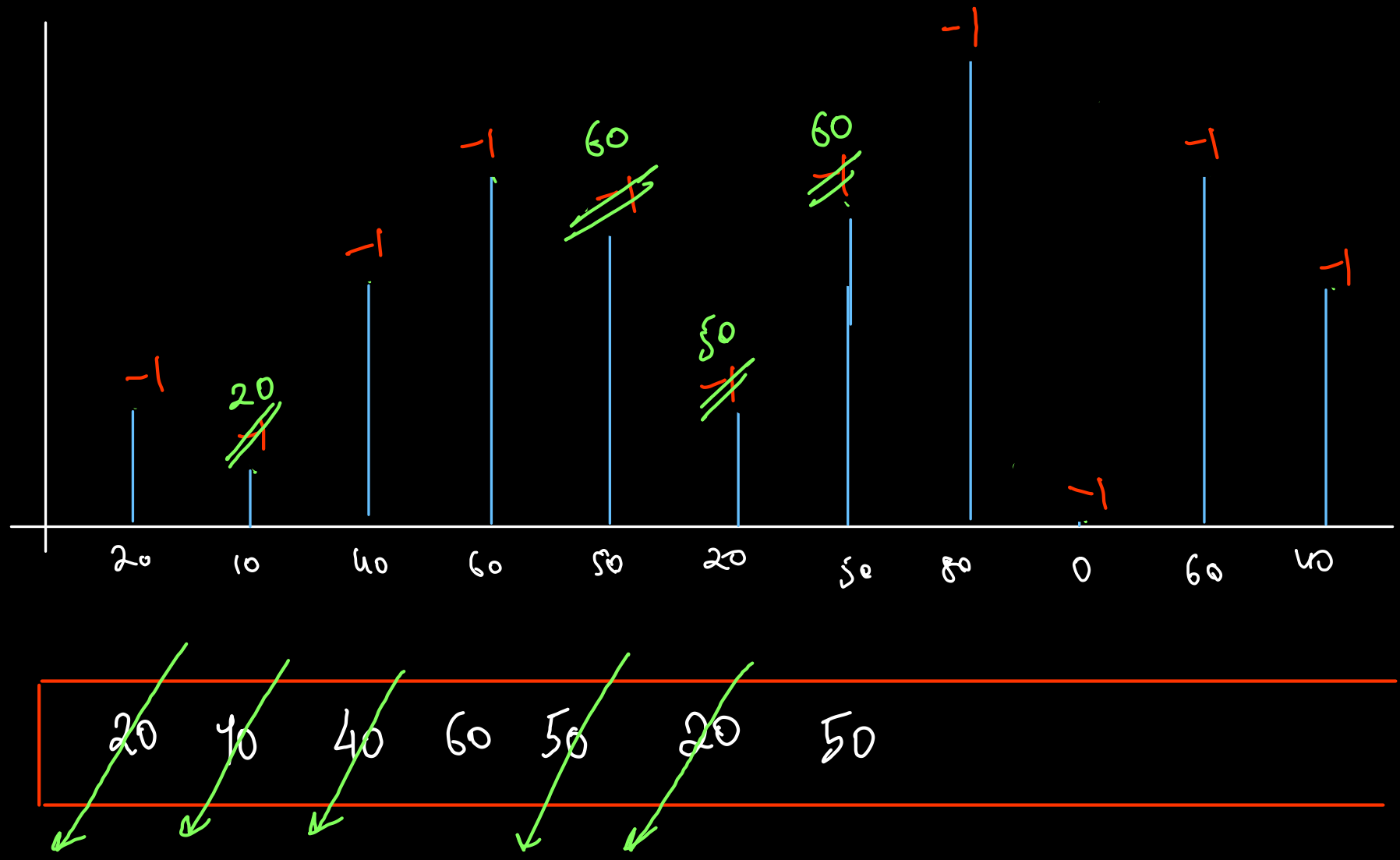




Next Greater Element to Left

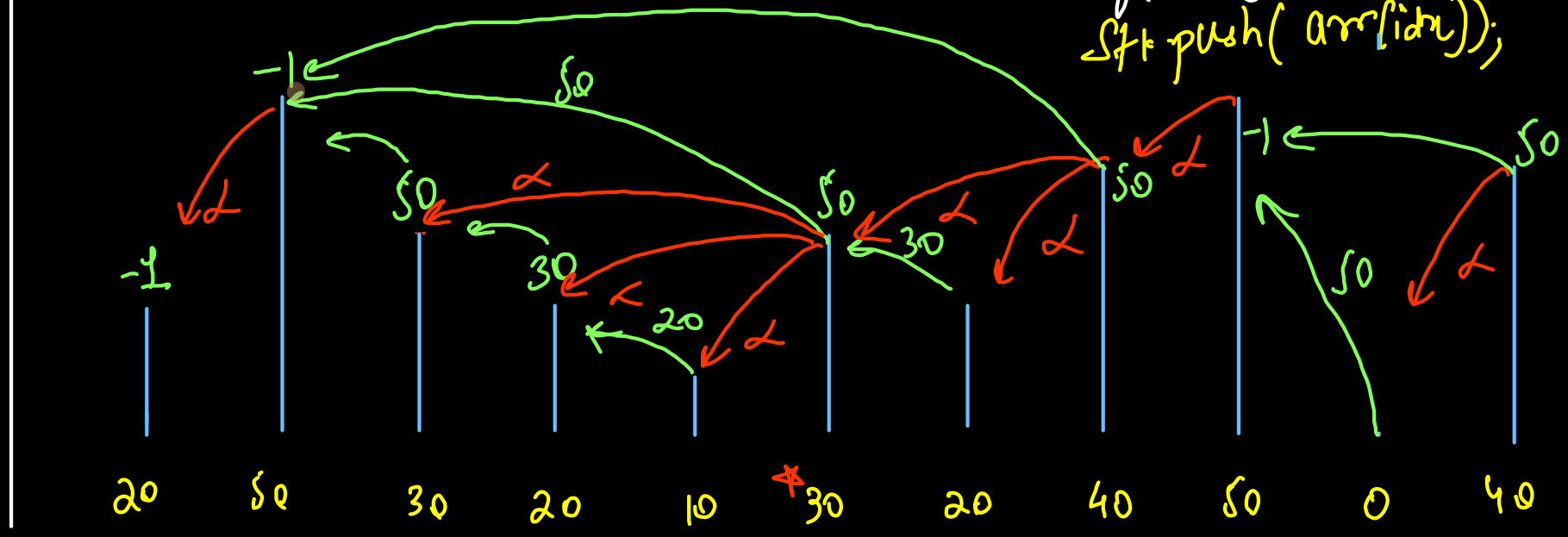


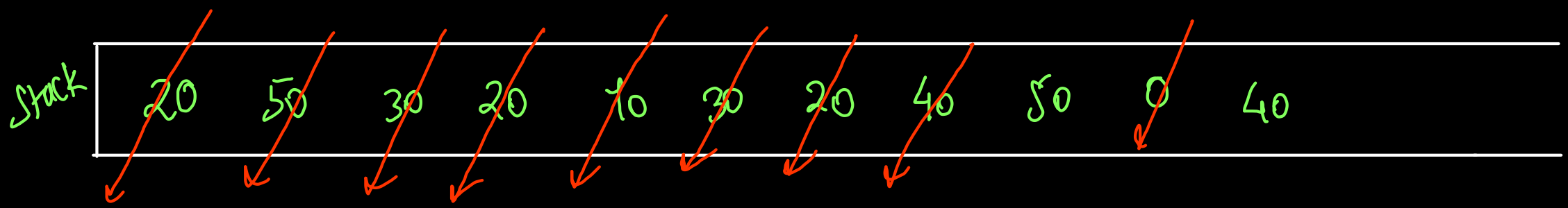
```java
public static int[] nextGreaterElement(int[] arr){
    int[] nge = new int[arr.length];

    // Nested Loop: O(N ^ 2)
    for(int i = 0; i < arr.length; i++){
        nge[i] = -1;
        for(int j = i - 1; j >= 0; j--){
            if(arr[j] > arr[i]){
                nge[i] = arr[j];
                break;
            }
        }
    }

    return nge;
}
```


-1
20
-1
-1
60
60
-1
-1
50
-1
-1
20 10 40 60 50 20 50 80 0 60 45
20 40 40 60 50 20 50

Next Greater Element in Left

```
for(int idx=0; i<n ; i++) {
    while(stk.size()>0 && stk.peek()<=arr[idx]
        stk.pop());
    if(stk.size()==0) res[idx]= -1; else res[idx]
        =stk.peek()
    stk.push( arr[idx]);
```





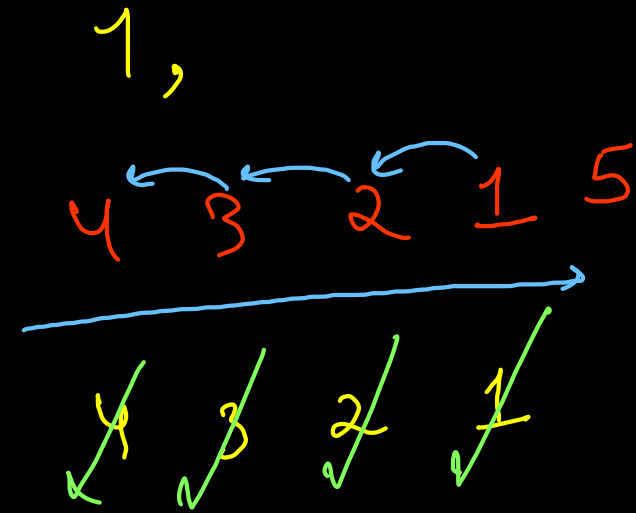
```java
public static int[] nextGreaterElement(int[] arr){
    Stack<Integer> stk = new Stack<>();
    int[] res = new int[arr.length];

    for(int idx = 0; idx < arr.length; idx++){
        // Smaller or Equal Elements in left should be popped
        while(stk.size() > 0 && stk.peek() <= arr[idx]){
            stk.pop();
        }
        // Finding the Next Greater Element
        res[idx] = (stk.size() == 0) ? -1 : stk.peek();

        // Push the current element
        stk.push(arr[idx]);
    }

    return res;
}
```

n

n

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

of linear



Amazon & microsoft

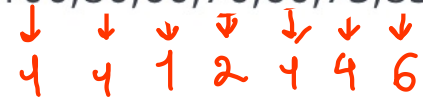
Application of NGE

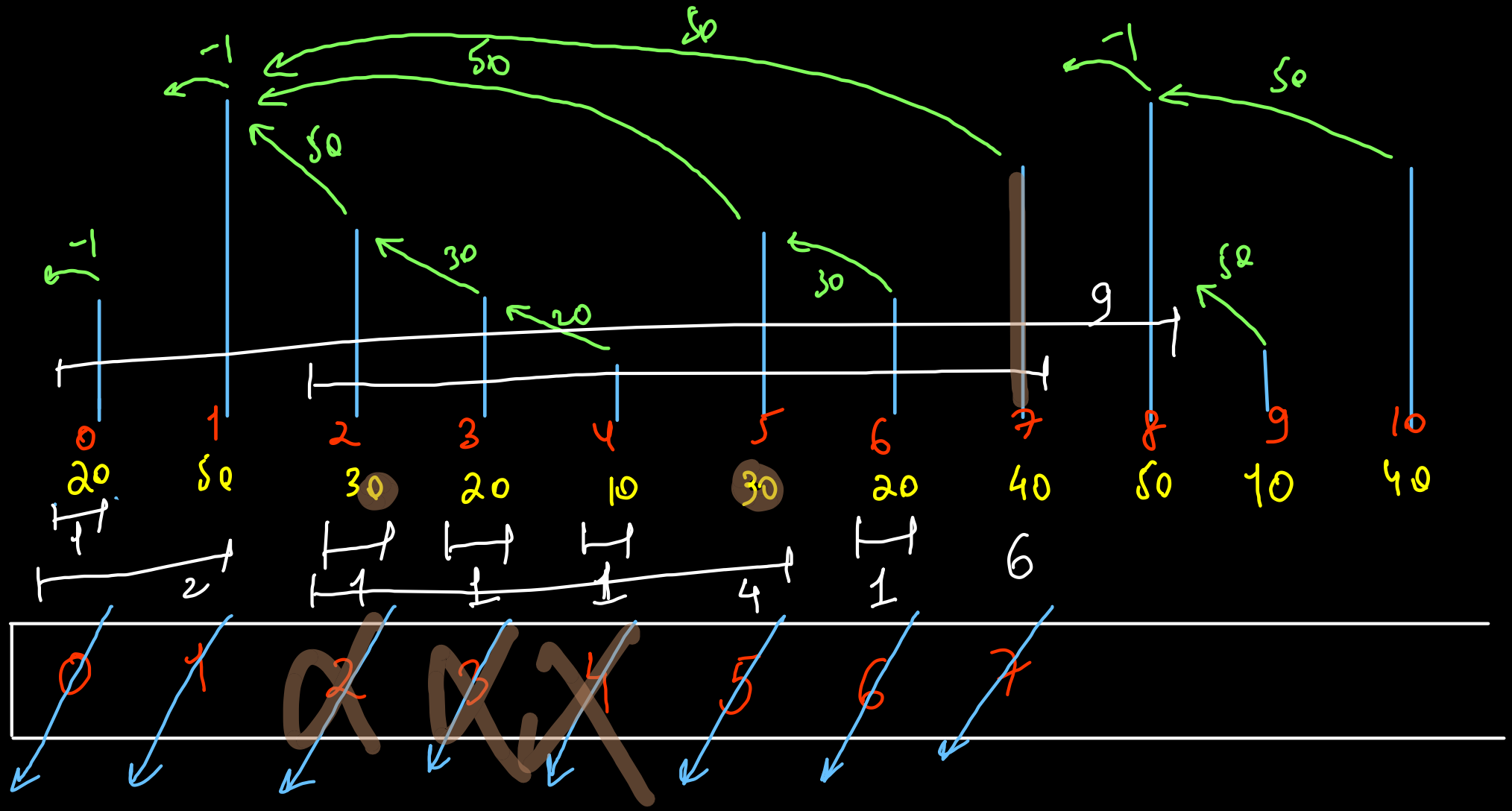
Online Stock Spanner

Design an algorithm that collects daily price quotes for some stock and returns **the span** of that stock's price for the current day.

The **span** of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backward) for which the stock price was less than or equal to today's price.

- For example, if the price of a stock over the next 7 days were **prices** = [100,80,60,70,60,75,85], then the stock spans would be **result** = [1,1,1,2,1,4,6].





Span = no of consecutive days for which current height is max^m

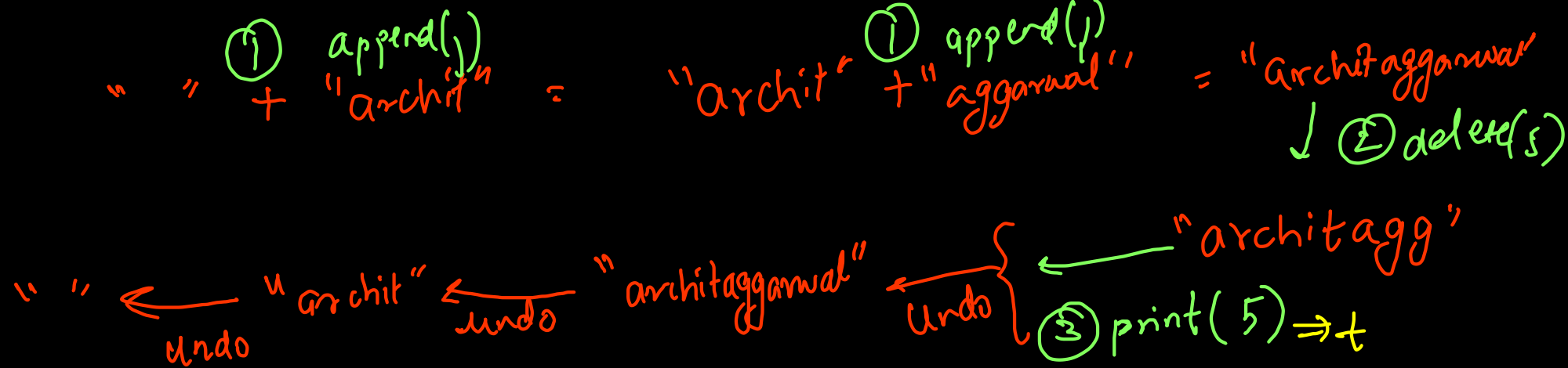
= distance blw current building & the NGE in left.

```java
public static int[] stockSpan(int[] arr){
    Stack<Integer> stk = new Stack<>();
    int[] res = new int[arr.length];

    for(int idx = 0; idx < arr.length; idx++){
        // Pop the smaller or equal elements
        while(stk.size() > 0 && arr[stk.peek()] <= arr[idx]){
            stk.pop();
        }

        // Form the resultant (stock span)
        res[idx] = (stk.size() == 0) ? idx + 1 : idx - stk.peek();

        // Push the current index
        stk.push(idx);
    }

    return res;
}
```

Simple Text Editor

Implement a simple text editor. The editor initially contains an empty string, S. Perform Q operations of the following 4 types:

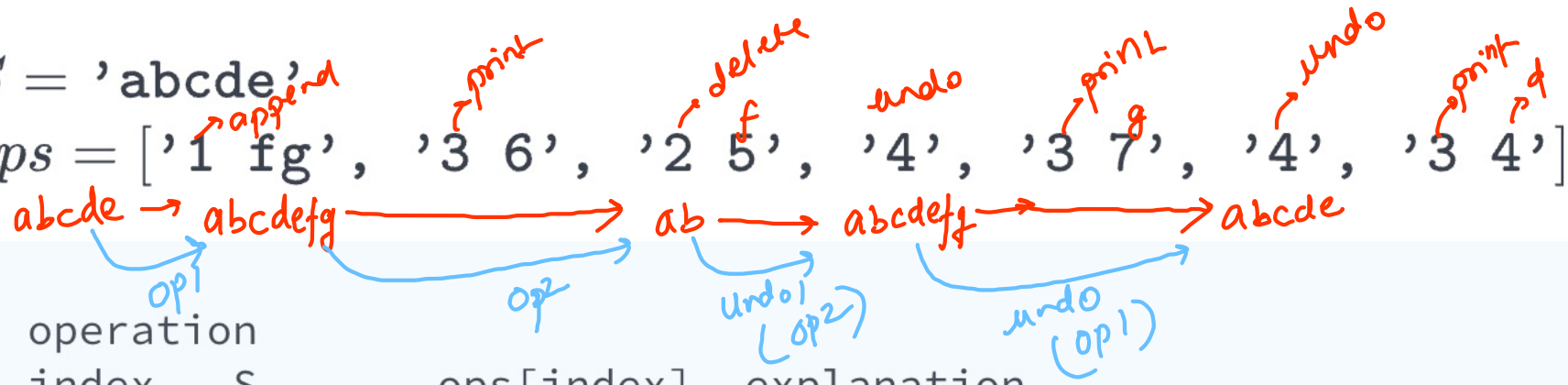
1. *append*(W) - Append string W to the end of S.
2. *delete*(k) - Delete the last k characters of S.
3. *print*(k) - Print the k^{th} character of S.
4. *undo*$()$ - Undo the last (not previously undone) operation of type 1 or 2, reverting S to the state it was in prior to that operation.



Example

$S =$ 'abcde'

$ops = [$'1 fg', '3 6', '2 5', '4', '3 7', '4', '3 4'$]$



operation

index	S	ops[index]	explanation
0	abcde	1 fg	append fg
1	abcdefg	3 6	print the 6th letter - f
2	abcdefg	2 5	delete the last 5 letters
3	ab	4	undo the last operation, index 2
4	abcdefg	3 7	print the 7th characgter - g
5	abcdefg	4	undo the last operation, index 0
6	abcde	3 4	print the 4th character - d

Example

$S =$ 'abcde'

$ops =$ ['1 fg', '3 6', '2 5', '4', '3 7', '4', '3 4'] '4'

print → sb.charAt(idx-1)

delete

undo

print

undo

print

StringBuilder {Stack → delete from last or append at last}

↳ "abcde" + "fg" = "abcdefg" —undo→ "ab" + "cdefg" → "abcdefg"

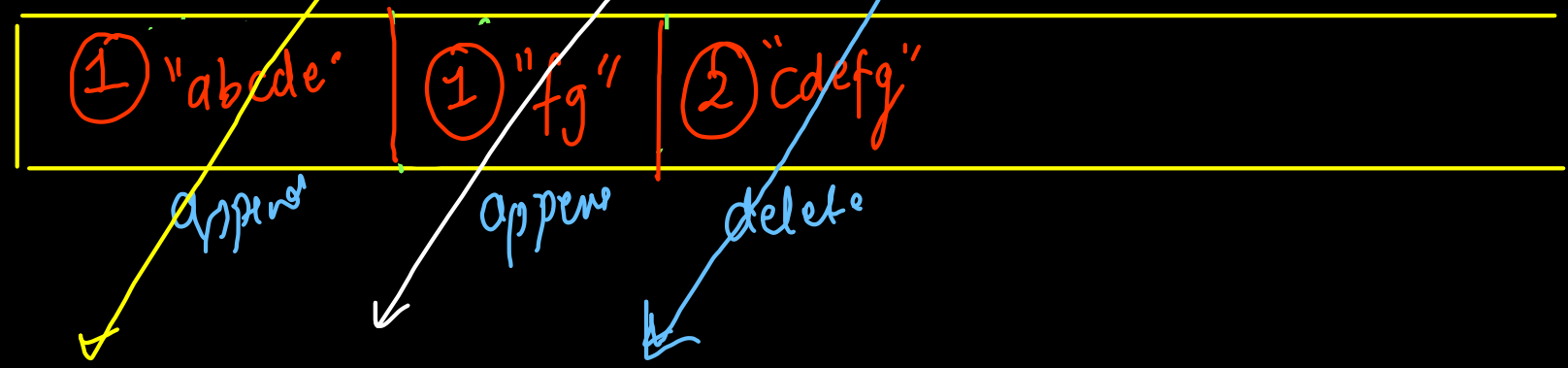
undo

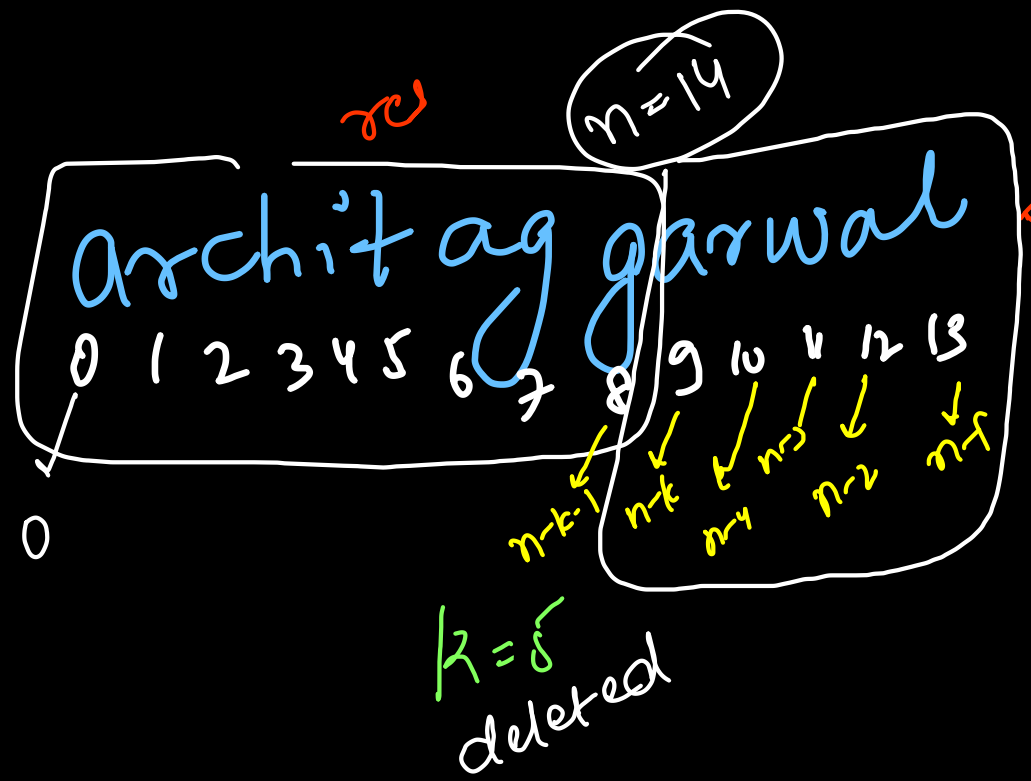
"abcde"

undo

" "

Stack operations :→





res
remaining string

stack
deleted chars.

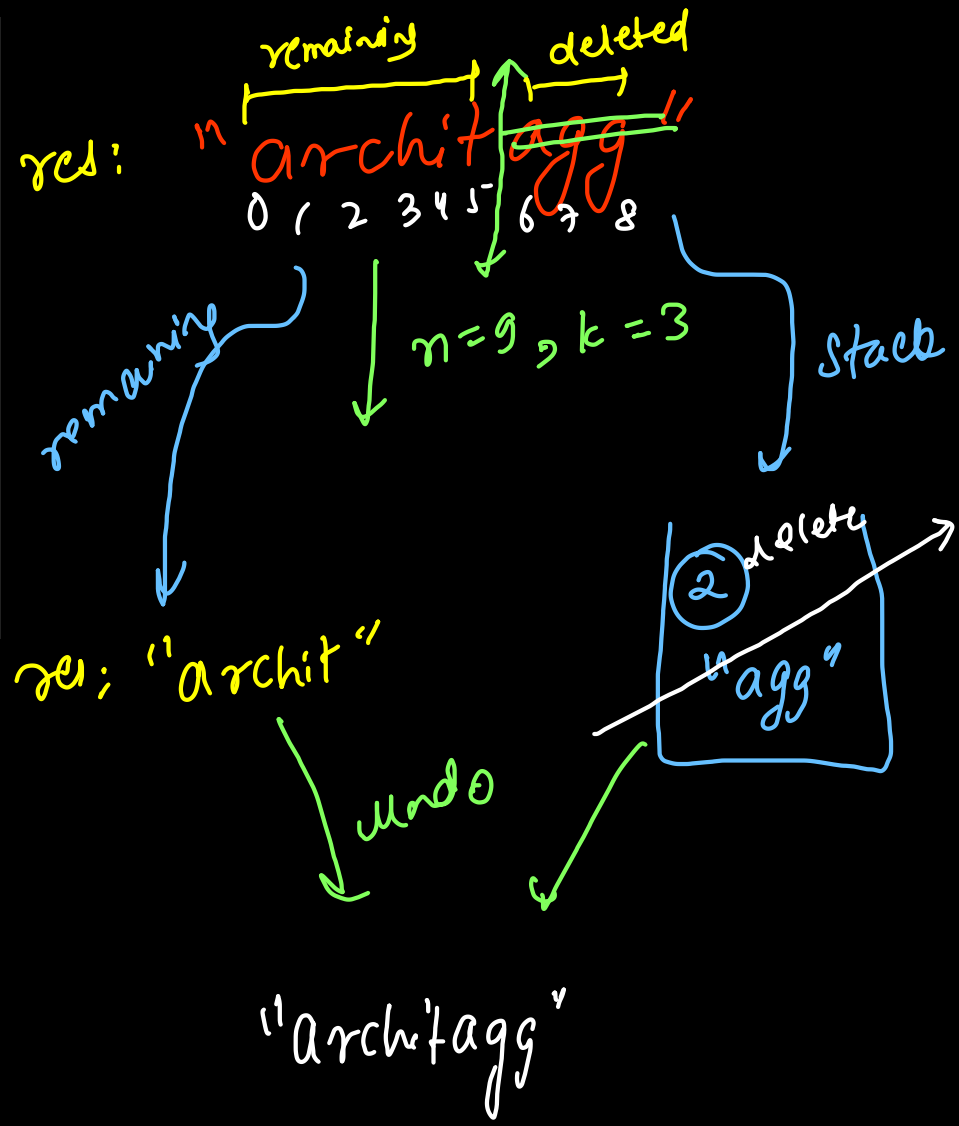
" architagg "

res.substring(0, n-k)

(2) " arwal "

res.substring(n-k, n)

```java
else if(choice == 2){
    // delete
    int k = scn.nextInt();
    int n = res.length();

    type.push(2); // delete operation
    undo.push(res.substring(n - k, n));
    res = res.substring(0, n - k);
```



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

    String res = new String();
    Stack<Integer> type = new Stack<>(); // Type of Operation
    Stack<String> undo = new Stack<>(); // String which we want to do

    for(int t = 0; t < tests; t++){
        int choice = scn.nextInt();

        if(choice == 1){
            // append
            String str = scn.next();
            res += str;
            type.push(1); // insert operation

            undo.push(str);
        }
        else if(choice == 2){
            // delete
            int k = scn.nextInt();
            int n = res.length();

            type.push(2); // delete operation
            undo.push(res.substring(n - k, n));

            res = res.substring(0, n - k);
```

```java
} else if(choice == 3){
    // print
    int k = scn.nextInt(); // 1-based indexing
    System.out.println(res.charAt(k - 1));

} else if(type.pop() == 1){
    // undo insert: delete
    String str = undo.pop();
    int n = res.length();
    int k = str.length();

    res = res.substring(0, n - k);
}
else {
    // undo delete: append
    String str = undo.pop();
    res += str;
}
```

undo

1 "archit" append

1 "agg" append

2 , k=5 delete

undo

undo

undo

+itagg

-"archit"

-"agg"

res "" + "archit" ⇒ "archit" + "agg ⇒ "architagg" ⇒ "arch"

Operations

1 "archit"

1 "agg"

2 "itagg"

str = "archit aggarwal is educator"

↓ str.split(" ");

{ "archit", "aggarwal", "is", "educator" }

str: "////archit-aggarwal////-is////"

↓ str.trim()

"archit_aggarwal_is"

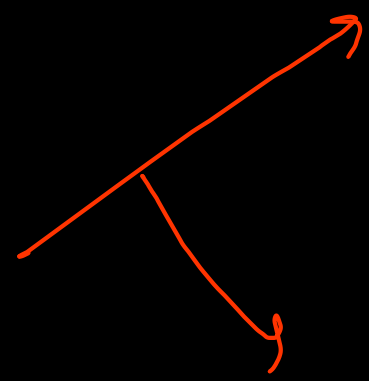
Prefix Infix Postfix Evaluation

$$(((5+(2*8))-((9/4)+7)))$$

infix expression

brackets

Operand 1	operator	operand 2
2	*	8



prefix

operator	op1	op2
*	2	8

postfix

op1	op2	operator
2	8	*

infix

"a+b*(c^d-e)^(f+g*h)-i"

humans

6 + 5 * (7 + 8 − 2) + (1 / 2 * 3) − 5

postfix

abcd^e-fgh*+^*+i-

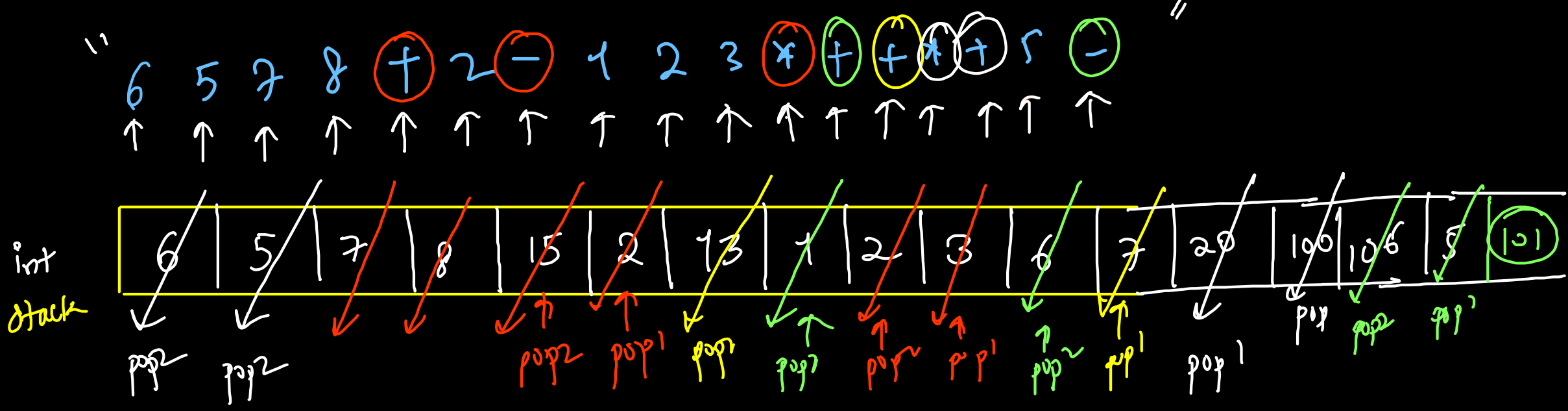
6, 5, 7, 8, +, 2, −, 1, 2, 3, *, /, +, *, +, 5, −

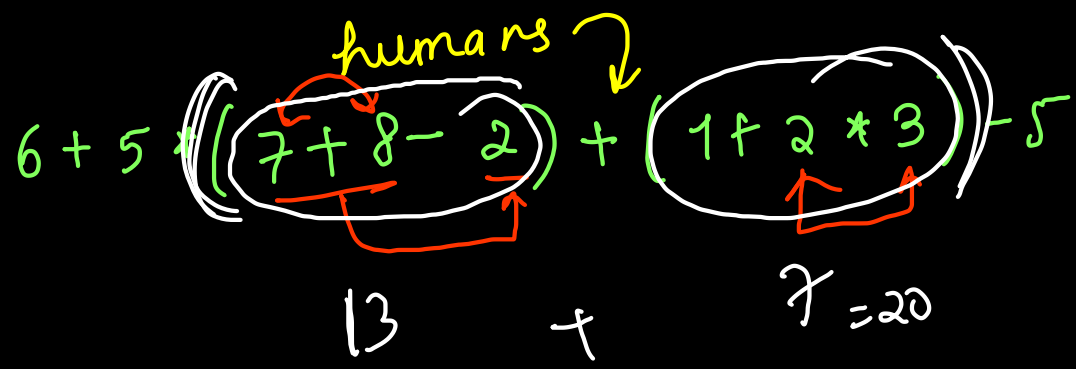
algo (stack)

Reverse Polish Notation

Postfix Expression Evaluation

type → 0-9






```
if(ch >= '0' && ch <= '9')
    stk.push(ch - '0');
else  int pop1 = stk.pop();
      int pop2 = stk.pop();
      stk.push(pop2 op pop1)
```

```java
public static int evaluatePostfix(String str){
    Stack<Integer> stk = new Stack<>();
    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);

        if(ch >= '0' && ch <= '9'){
            stk.push(ch - '0');
        } else {
            int pop1 = stk.pop();
            int pop2 = stk.pop();

            if(ch == '+') stk.push(pop2 + pop1);
            else if(ch == '-') stk.push(pop2 - pop1);
            else if(ch == '/') stk.push(pop2 / pop1);
            else stk.push(pop2 * pop1);
        }
    }

    return stk.peek();
}
```

Time :→ O(n)

Space :→ O(n)

~~Extra space~~

Hashmap Data Structure

key → no multiple values

key : value

String	Integer
IPL Team	IPL Trophies
CSK	4
KKR	2
MI	5
RCB	0
SRH	2

String	String
Country	Capital
India	Delhi
Pakistan	karachi
UK	London
US	Washington
Russia	moscow
Japan	Tokyo

Integer	String
CountryCode	Country
91	India
4	USA
92	Pakistan
97	UAE
60	Malaysia

India : Delhi, Kolkatta

vs

India : Delhi

India : Kolkatta

Not possible

① 2 keys cannot be same

All keys will be unique / distinct keys!

② 2 different keys can have same value!

③ hashmap.get(key)

key → value : O(1) avg ⇒ What is capital time of India?

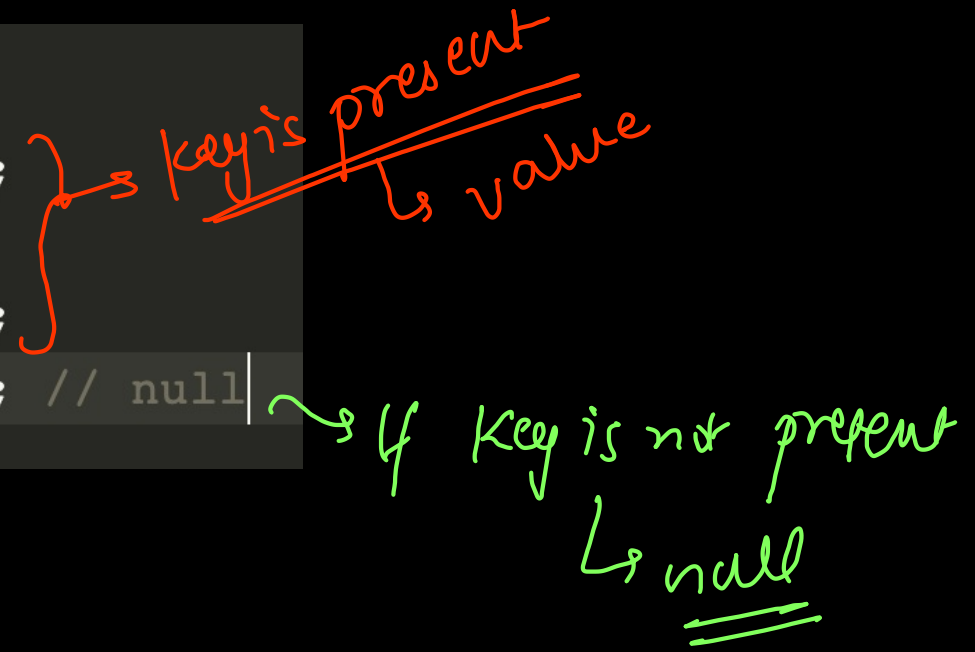
" " " ... Japan?

hashmap.getkey(value)

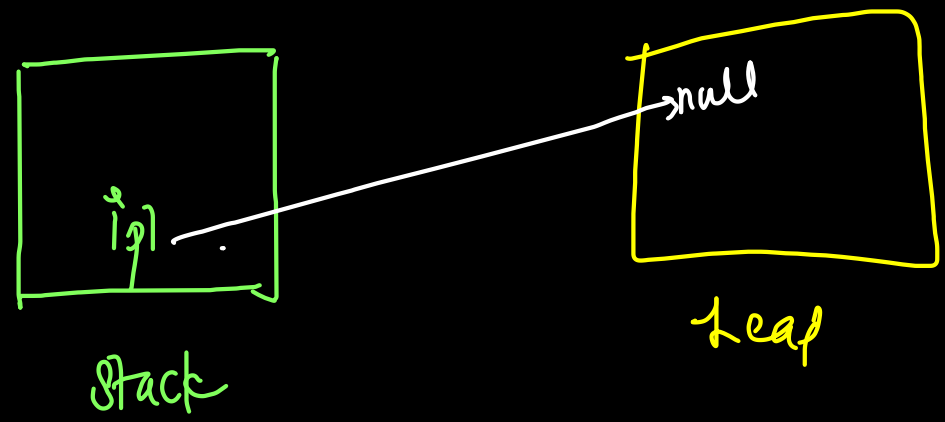
value → key : O(n) ⇒ What is country with capital Karachi?

" ... " " Moscow?

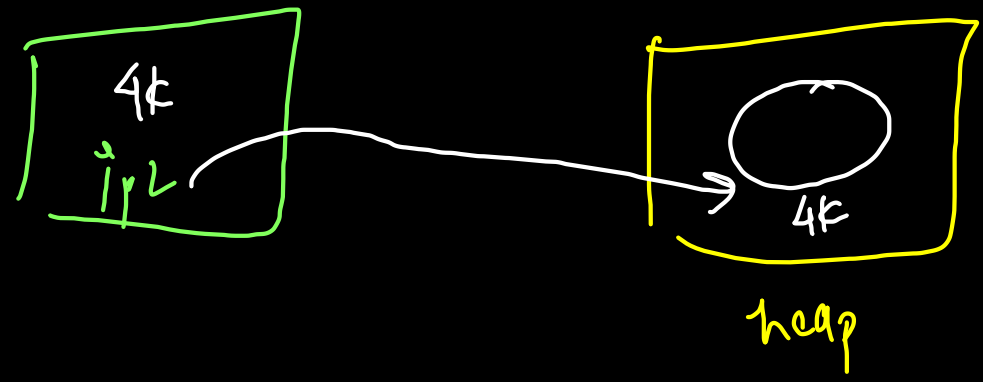
```java
// Read: Get the Value of a Key
System.out.println(ipl.get("CSK"));
System.out.println(ipl.get("MI"));
System.out.println(ipl.get("RCB"));
System.out.println(ipl.get("PWI")); // null
```



```
HashMap<String, Integer> ipl = new HashMap<>();
```




```
HashMap<String, Integer> ipl = new HashMap<>();
```



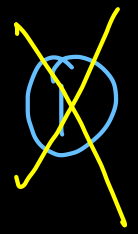
```java
// Creation
HashMap<String, Integer> ipl = new HashMap<>();

System.out.println(ipl.size());
System.out.println(ipl);

// Insertion: Key-Value Pair
ipl.put("RCB", 0);
ipl.put("LSG", 0);
ipl.put("MI", 5);
ipl.put("RR", 1);
ipl.put("CSK", 4);
ipl.put("PK", 0);
ipl.put("DC", 0);
ipl.put("GT", 1);
ipl.put("KKR", 2);
ipl.put("SRH", 2);
```

```java
System.out.println(ipl);
// 1. Insertion Order: Incorrect
// 2. Sorted Order on Keys: Incorrect
// 3. Sorted Order on Values: Incorrect
// 4. I Dont Know: HashCode: Random Order: Correct

System.out.println(ipl);
System.out.println(ipl.size());

// Read: Get the Value of a Key
System.out.println(ipl.get("CSK"));
System.out.println(ipl.get("MI"));
System.out.println(ipl.get("RCB"));
System.out.println(ipl.get("PWI")); // null
System.out.println(ipl.get(null));

// ipl.put(null, 1); // Key: null Value: 1
// System.out.println(ipl.get(null));
```

ipl.put("CSK", 4) ⇒ Insert

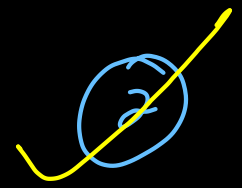
"CSK" : 4

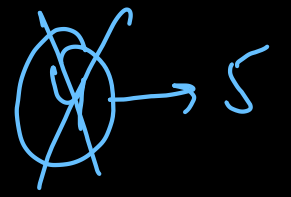
ipl.put("CSK", 5) ⇒ Update



"CSK": 4

"CSK": 5



"CSK" : 

String State	ArrayList<String> Cities
Punjab	Ambala, Ludhiana, Jalandhar, Chandigarh, - -
Haryana	Rohtak, Sirsa, Bhiwani, Faridabad, - - Gurugram,
UP	Gorakhpur, Lucknow, Ayodhya
Hyderabad	Gachibowli, Raidurg, Jubilee Hills Kukatpally,

```
// Traversal: For Each on Key
for(String team : ipl.keySet()){
    System.out.print(team + " : ");
    System.out.println(ipl.get(team));
}
```

Word meaning

You are required to create a **dictionary** consisting of word and its meaning.

Take an integer N as input and **Continue** the process untill **Case 4** is not achieved.

- If **N==1**, take **word** and **meaning** as input from user and **add** it to the dictionary.
- If **N==2**, take a **word** as input from the user and print its **meaning**, if the word is not found print **-1**.
- If **N==3**, take a **word** as input from the user and delete it from the dictionary.
- If **N==4**, **Close** the dictionary(**Exit** the program).

// insert : dict.put(word, meaning)

↳ dict.remove(word);

dict.getOrDefault(word, "-1");

"apple" : "a fruit which is red"

String : String

dict.getOrDefault("apple", "-1");
↳ "a fruit which is red"

dict.getOrDefault("orange", "-1");
↳ "-1";

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

    // EMPTY HASHMAP CREATION
    HashMap<String, String> dict = new HashMap<>();

    while(true){
        int choice = scn.nextInt();

        if(choice == 1){
            // insert
            String word = scn.next();
            String meaning = scn.next();
            dict.put(word, meaning);

        } else if(choice == 2){
            // get or default
            String word = scn.next();
            System.out.println(dict.getOrDefault(word, "-1"));

        } else if(choice == 3){
            // delete
            String word = scn.next();
            dict.remove(word);

        } else break; // exit the while loop or program
    }
}
```

menu chosen

O(1) avg O(n) worst

O(1) avg O(n) worst

O(1) avg O(n) worst

```
1
Geekster
Coding
1
Geek
Coder
2
Geek
3
Geekster
2
Geekster
4
```

String : String

~~Geekster : Coding~~

Geek & Coder

Output:

Coder

-1

get(key)

key present → return value

- ipl.get("CSK") : 4
- ipl.get("MI") : 5

key not present → null

- ipl.get("PWI") : null
- ipl.get("garbage") : null

getOrDefault(key, def)

key present → return value

- ipl.getOrDefault("CSK", def) : 4
- ipl.getOrDefault("MI", def) : 5

Key not present → default value

- ipl.getOrDefault("PWI", 0); → 0

TreeMap : Hashmap with sorted keys
↳ Lexicographical order

Archit Aggarwal @ 2022

Character , Integer

key : value

A : 1+1
r : 1+1
c : 1
h : 1
i : 1
t : 1
g : 1+1
a : 1+1
w : 1
l : 1
@ : 1
'2' : 1+1+1
'0' : 1

if not present
freq.put(ch, 1)

else
freq.put(ch, freq.get(ch)+1);

```java
public static void printFrequency(char[] arr){
    TreeMap<Character, Integer> freq = new TreeMap<>();

    for(char ch: arr){
        if(freq.containsKey(ch) == false){
        // if(freq.get(ch) == null){
            freq.put(ch, 1); //insert
        } else {
            int oldFreq = freq.get(ch);
            freq.put(ch, oldFreq + 1); //update
        }
    }

    for(char ch: freq.keySet()){
        System.out.println(ch + " " + freq.get(ch));
    }
}

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

    int n = scn.nextInt();

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

    printFrequency(arr);
}
```

O(n)

O(1)

O(1)

O(1)

O(unique keys) = O(128)

ascii code

ignore Treemap (assume hashmap)

Time

9:07 - 9:08

Total Time

O(n)

containsKey(ch)

- true: key is present in HM
  ipl. ck("clk") = true
- false: key is not present in HM
  ipl. ck("pwl") = false

```java
public static void printFrequency(char[] arr){
    TreeMap<Character, Integer> freq = new TreeMap<>();

    for(char ch: arr){
        if(freq.containsKey(ch) == false){
        // if(freq.get(ch) == null){
            freq.put(ch, 1);
        } else {
            int oldFreq = freq.get(ch);
            freq.put(ch, oldFreq + 1);
        }
    }

    for(char ch: freq.keySet()){
        System.out.println(ch + " " + freq.get(ch));
    }
}

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

    int n = scn.nextInt();

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

    printFrequency(arr);
}
```

freq.put(ch, freq.getOrDefault(ch, 0) + 1);

A : not present :
'A' => 0 + 1 = 1

A : inserted already
'A' => 1 + 1 => 2

```java
public static void printFrequency(char[] arr){
    TreeMap<Character, Integer> freq = new TreeMap<>();

    for(char ch: arr)
        freq.put(ch, freq.getOrDefault(ch, 0) + 1);

    for(char ch: freq.keySet())
        System.out.println(ch + " " + freq.get(ch));
}

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

    int n = scn.nextInt();

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

    printFrequency(arr);
}
```

O(1) avg

To O(1) avg

Same No Same Frequency

4	-3	2	0	-3	4	2	2	4	-3	4	3

4 : 4 ✓

-3 : 3 ✓

0 : 1 ✗

3 : 1 ✗

2 : 3 ✗

int : int

hashmap → random keys

treemap → sorted keys

↳ Ascending order

```java
public static void printFrequency(int[] arr){
    TreeMap<Integer, Integer> freq = new TreeMap<>();

    for(int val: arr){
        freq.put(val, freq.getOrDefault(val, 0) + 1);
    }
    for(int key: freq.keySet()){
        if(freq.get(key) == Math.abs(key)){
            System.out.println(key);
        }
    }
}
```

// matching the expected output (sorted order)

* Longest Substring w/o Repeating Characters

Salesforce

18 base

25k USD → 4 years

Sprinklr

35 LPA base

Stock α

Longest Substring w/o Repeating Characters

"a r c h i t s h a r m a e d u c a t o r"

↑ ↑
l r

Character → Integer
a → 1

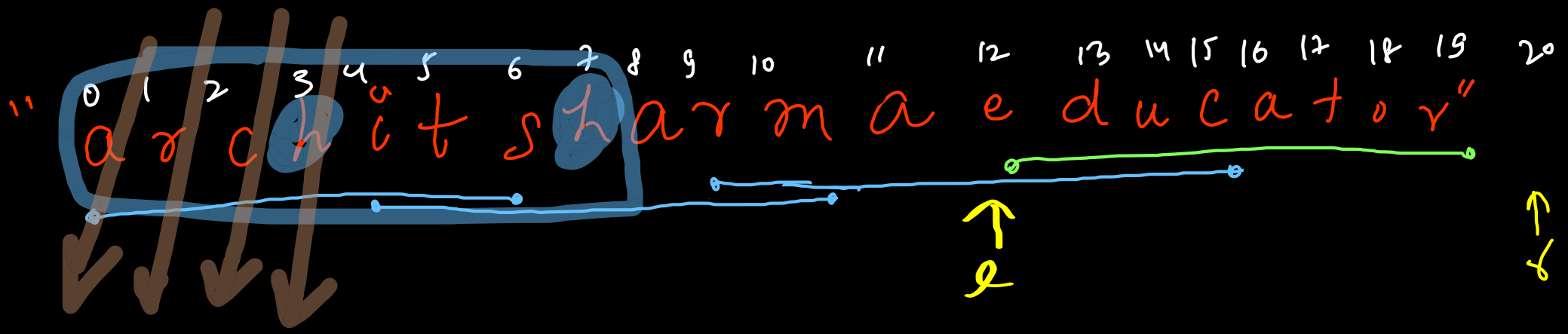
(1) contiguous
(2) max^m length
(3) no duplicate
→ length

to exclude characters ← left ≤ right → to include characters

Sliding window
↳ Two Pointer

+

hashmap



Character ⟶ Integer

a → ~~0~~ ~~1~~ ~~2~~ 1

r → ~~0~~ ~~1~~ 0

c → ~~0~~ 1

h → ~~2~~ 0

i → 0

t → 0

s → 0

m → 0

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

t → 1

e → 1

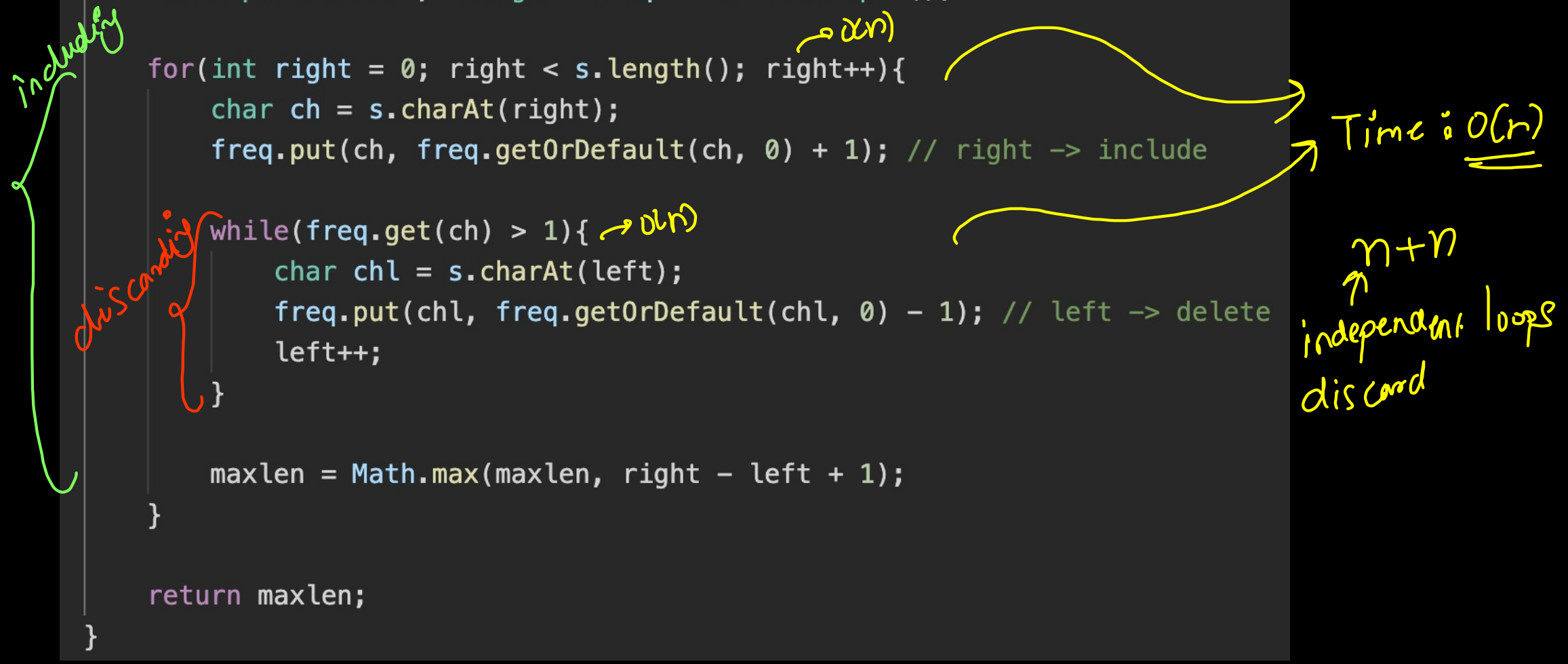
d → 1

u → 1

o → 1

r → 1

```java
public int lengthOfLongestSubstring(String s) {
    int left = 0, maxlen = 0;
    HashMap<Character, Integer> freq = new HashMap<>();

    for(int right = 0; right < s.length(); right++){
        char ch = s.charAt(right);
        freq.put(ch, freq.getOrDefault(ch, 0) + 1); // right -> include

        while(freq.get(ch) > 1){
            char chl = s.charAt(left);
            freq.put(chl, freq.getOrDefault(chl, 0) - 1); // left -> delete
            left++;
        }

        maxlen = Math.max(maxlen, right - left + 1);
    }

    return maxlen;
}
```



Geekster ⇒ Jobs → Indian startups
→ low headcount (≤ 100 employees)

Recession

- US based company
  - hiring
  - freeze
  - fire
- Service based companies
  - Wipro
  - Cognizant

First Unique Character

leftmost index

non-repeating

"geeksforgeeks" : 5th index

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

"geeksgeeks" (-1)

Two Traversal Approach

① Fill frequency Hashmap

g → 2
e → 2
k → 2
s → 2
f → 1
o → 1
r → 1

② Leftmost char: f.get(ch) == 1

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

    HashMap<Character, Integer> freq = new HashMap<>();

    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        freq.put(ch, freq.getOrDefault(ch, 0) + 1);
    }

    for(int idx = 0; idx < str.length(); idx++){ // leftmost
        char ch = str.charAt(idx);
        if(freq.get(ch) == 1) { // unique
            System.out.println(idx);
            return;
        }
    }

    System.out.println(-1);
}
```

extra space

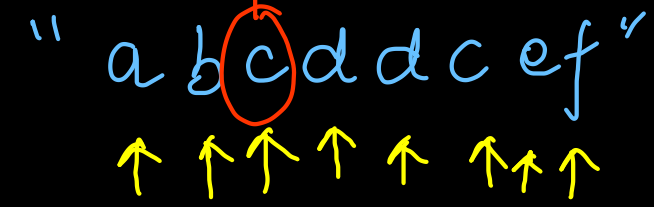
Two Traversal Approach

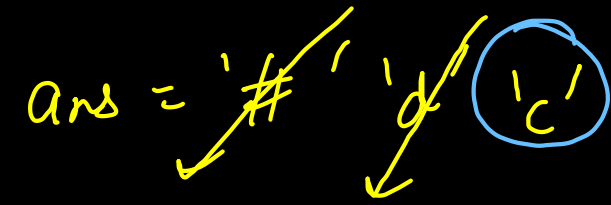
Time ⇒ $O(n+n) = O(n)$

Space ⇒ $O(128)$ hashmap

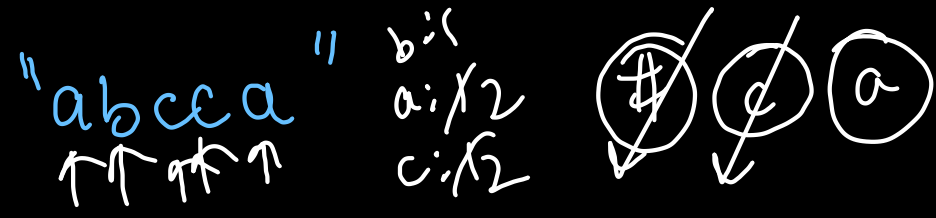
First Repeating Character

```
char firstRep(String str)
{
    HashMap<Character, Integer> freq = new HashMap<>();
    char ans = '#';

    for(int idx = str.length() - 1; idx >= 0; idx--){
        char ch = str.charAt(idx);
        freq.put(ch, freq.getOrDefault(ch, 0) + 1);
        if(freq.get(ch) > 1) ans = ch;
    }

    return ans;
}
```





f : 1
e : 1
c : ✗2
d : ✗2
b : 1
a : 1

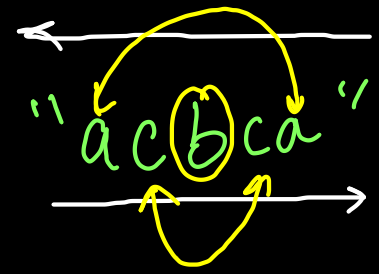


Longest Palindrome Length

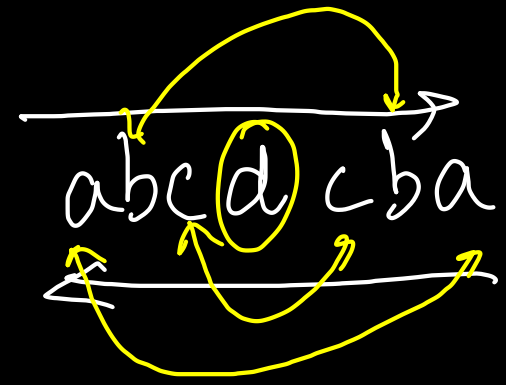
→ pick any character ✓ (need not be contiguous)

→ rearrange characters ✓ (order may not be maintained)

a b a d a c e f



a b c d a c e f b a c



"aaaabbbccccddd"

a: 4 → 2

b: 3 → 2

c: 4 → 1

d: 3 → 2

"aabbccddddd"

a: 2 → 1

b: 2 → 1

c: 2 → 1

d: 4 → 3 → 1

aabcddddcbaa

odd length

len = 0+4+2+4+2+1

abcddddcba

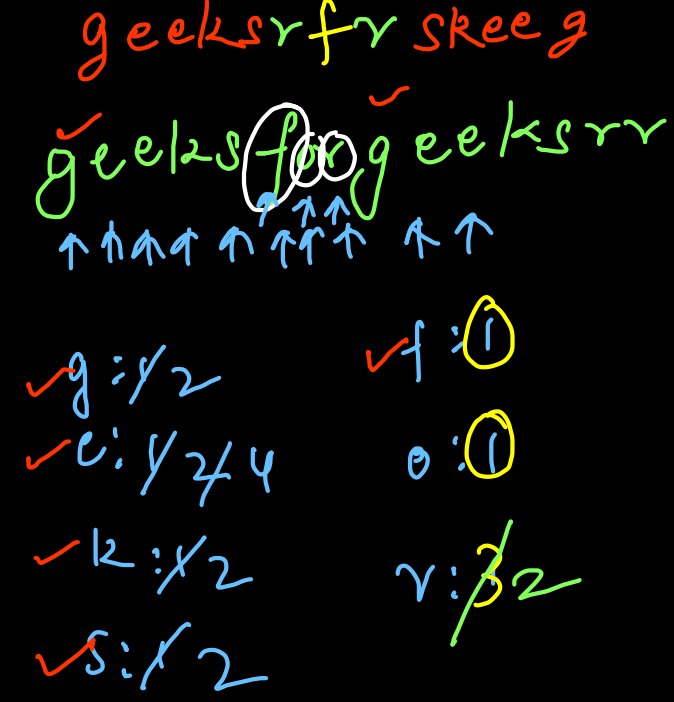
even length

len = 0+2+2+2+4

= 10 == n

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

    int length = 0;
    HashMap<Character, Integer> freq = new HashMap<>();

    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        freq.put(ch, freq.getOrDefault(ch, 0) + 1);
    }

    for(char ch: freq.keySet()){
        int val = freq.get(ch);
        if(val % 2 == 1) val--; //odd:
        length = length + val;
    }

    if(length < str.length())
        length++; // unique character: odd length palindrome

    System.out.println(length);
}
```

fill the hashmap



AVRL Company

& CTC {90% + ⇒ Base}

Front End Engineer (React)

1st Interview

Web Dev → 2 hour

↳ API → CRUD

Javascript

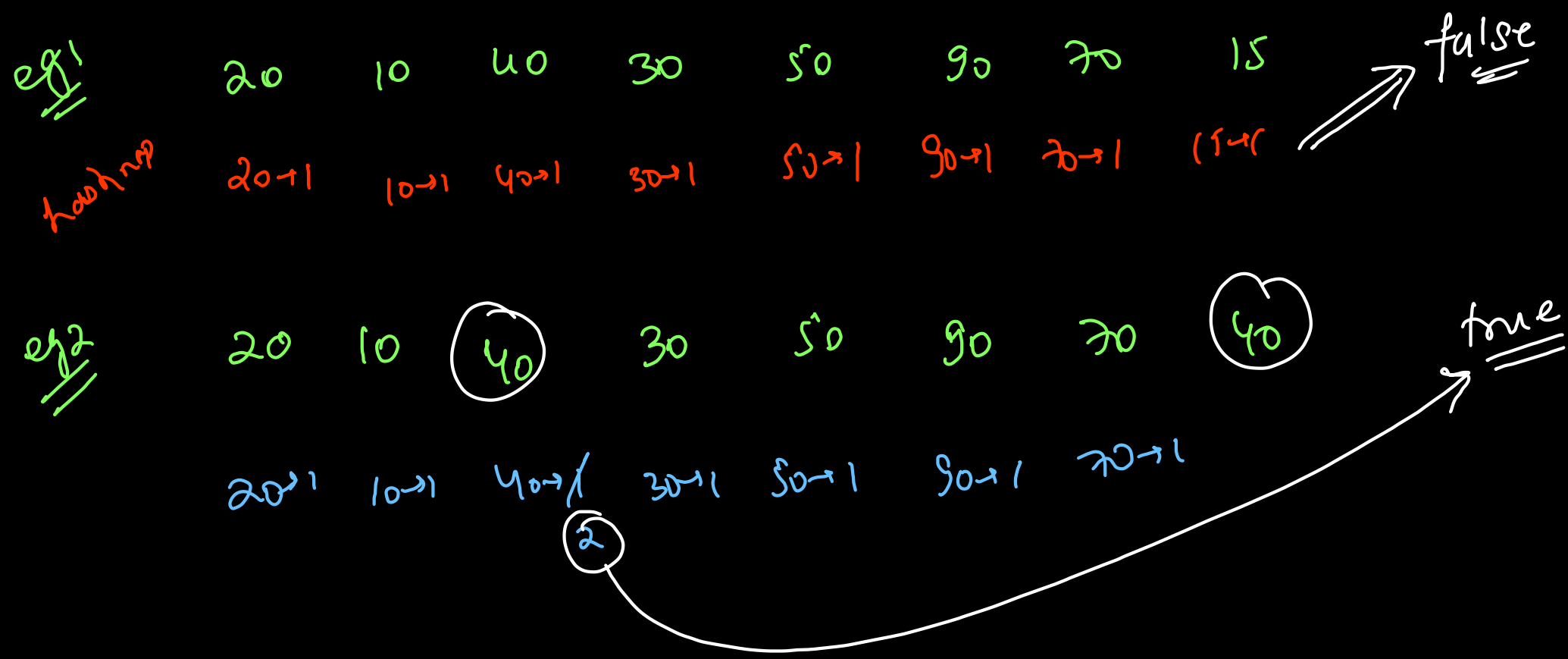
2nd 2 Interviews

→ Reverse words leetcode

→ Merge 2 sorted Array

HR → Interviewbit HR Questions

Contains Duplicate?



```java
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();
    }

    HashMap<Integer, Integer> freq = new HashMap<>();

    for(int key: arr){
        freq.put(key, freq.getOrDefault(key, 0) + 1);
        if(freq.get(key) >= 2){
            System.out.println(true);
            return;
        }
    }
    System.out.println(false);
}
```

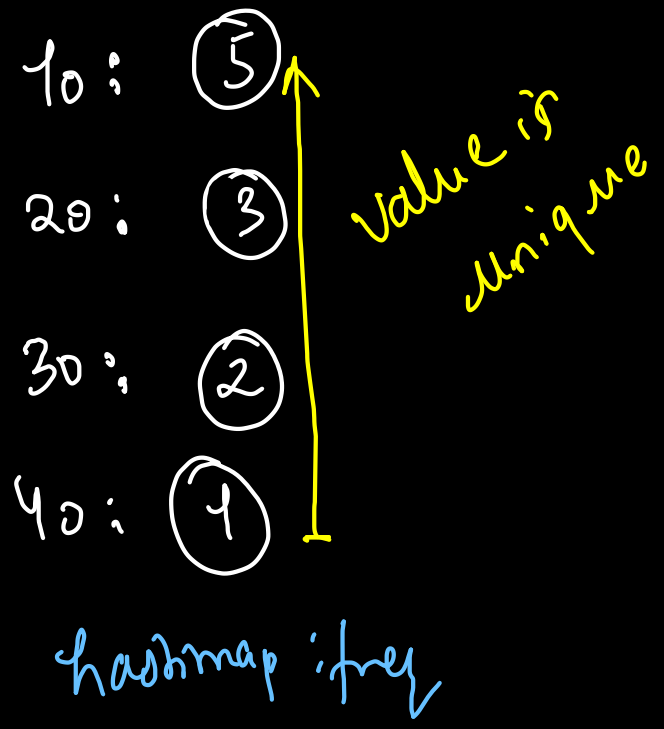
Time ⇒ O(n)
Linear

Space ⇒ O(n)
hashmap

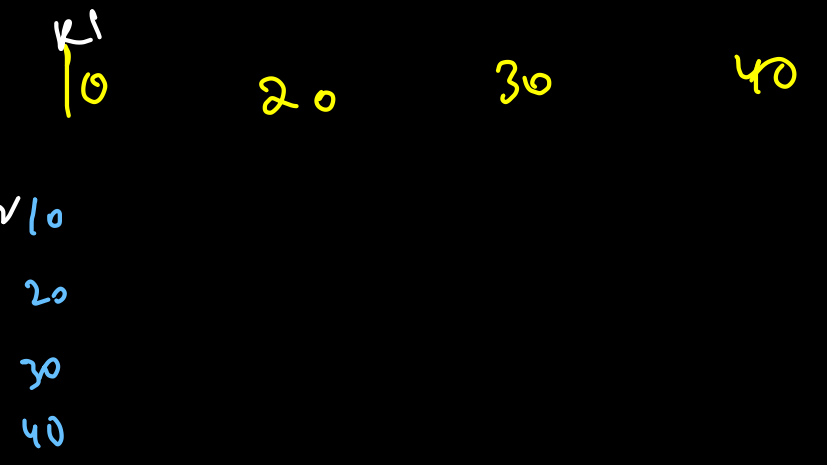
Unique Number of Occurrences

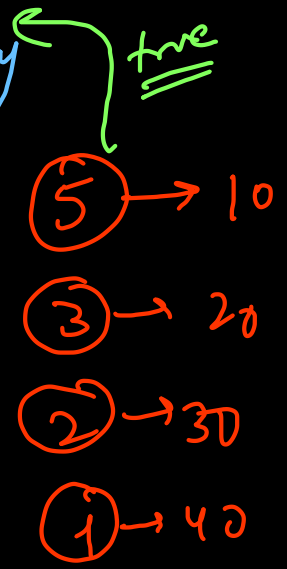
ex 1

10 20 30 10 10 10 10 20 30 40 20



there are no 2 keys having same frequency ⇒ true





Integer Integer

invhm

eg 2

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

10 : 4

20 : 3

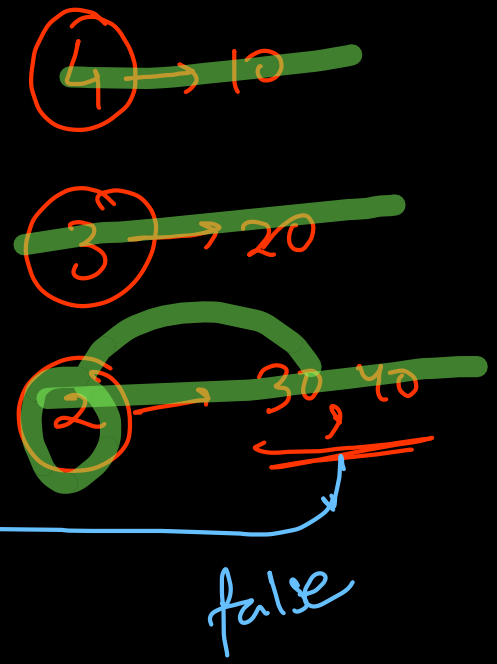
30 : 2

40 : 2

50 : 1

Same frequency

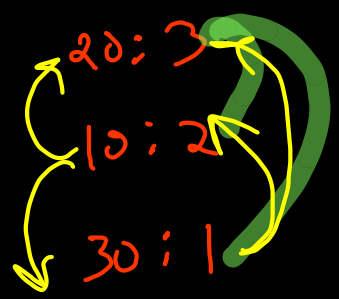
so false



```java
public static boolean isDuplicate(int[] arr){
    HashMap<Integer, Integer> freq = new HashMap<>();
    for(int key: arr){
        freq.put(key, freq.getOrDefault(key, 0) + 1);
    }

    for(int key1: freq.keySet()){
        int val1 = freq.get(key1);

        for(int key2: freq.keySet()){
            int val2 = freq.get(key2);
            if(key1 != key2 && val1 == val2)
                return false;
        }
    }

    return true;
}
```

O(n)

O(n²)



key1 = 20
val1 = 3
k2 10 k2 30
2 1

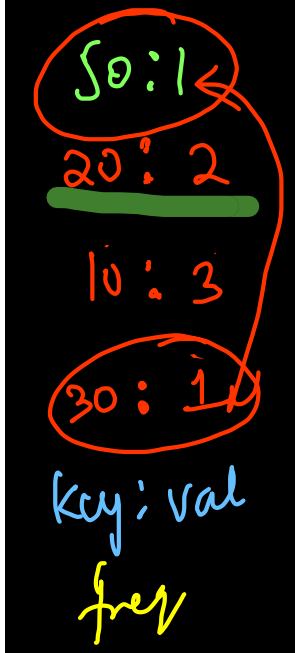
key1 = 10
val = 2
k2 20, 30

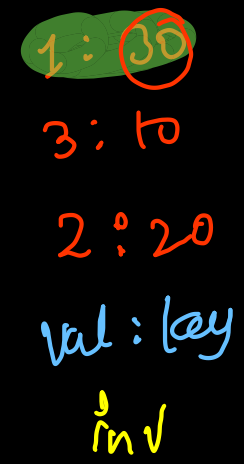
key2 = 30
val1 = 1
key2 = 40
val2 = 1

```java
public static boolean isDuplicate(int[] arr){
    HashMap<Integer, Integer> freq = new HashMap<>();
    for(int key: arr){
        freq.put(key, freq.getOrDefault(key, 0) + 1);
    }

    HashMap<Integer, Integer> inv = new HashMap<>();

    for(int key: freq.keySet()){
        int val = freq.get(key);

        if(inv.get(val) != null) return false;
        inv.put(val, key);
    }

    return true;
}
```

inv.get(1) = 30 != null

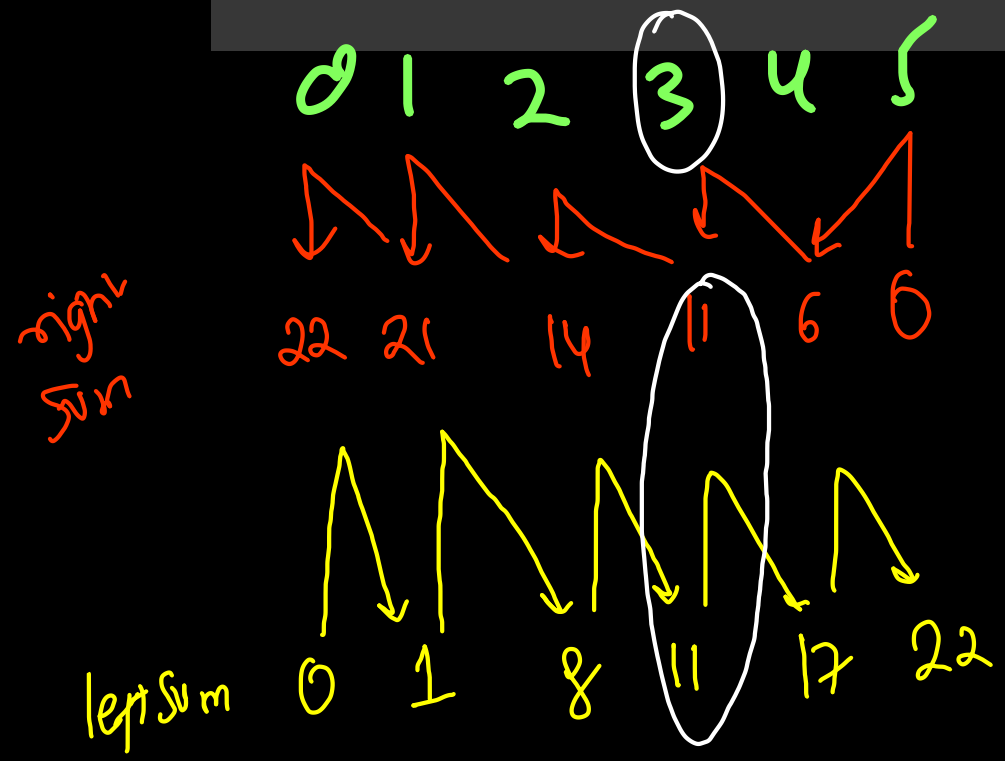
20 10 30 10 20 10 50





Find Pivot Index

```
[1,7,3,6,5,6]
```



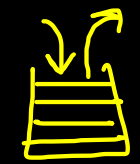
Queue Data Structure

→ Array (1D, 2D) ⇒ similar data in contiguous ⇒ random access (indexes)
⇒ fixed size

→ Strings & StringBuilder ⇒ array of characters

→ ArrayList ⇒ array of dynamic size ⇒ random access

→ Stack ⇒ FILO (First-In Last-Out) or LIFO (Last In - First Out)
⇒ Both insertion & deletion from same end (top)



→ Hashmap & TreeMap ⇒ store key-value pairs
⇒ unique keys ⇒ hashmap (key & value) ⇒ random
⇒ search/insert/remove (key) ⇒ constant avg

Queue Data Structure

→ Circular Queue (Using Array)

→ LinkedList

Sharma

Verma

Removal

⇑

First In First Out (FIFO)

First Come First Serve Basis

Aggarwal

arman

insertion

rishal

⇑

Last In Last Out (LILO)

Insertion & Removal from the different ends!

```java
Queue<String> q = new ArrayDeque<>();

// Insert
q.add("Sharma");
System.out.println(q);

q.add("Verma");
System.out.println(q);

q.add("Arman");
System.out.println(q);

q.add("Vishal");
System.out.println(q);

// Get (Read): Front Element
System.out.println(q.peek());
System.out.println(q);
```

O(1) [Sharma) [S,V] [S,V,A] [S,Vi,A,Vi] O(1) Sharma [S,Ve,A,Vi]

```java
// Remove
System.out.println(q.remove());
System.out.println(q);

System.out.println(q.remove());
System.out.println(q);

System.out.println(q.remove());
System.out.println(q);

System.out.println(q.remove());
System.out.println(q);
```

O(1) Sharma [Ve,A,Vi] Verma [A,Vishal] Arman [Vishal] Vishal []

System.out.println(q.size()) O(1)

1. **Declare an Empty** $queue\ s$.

2. Take Single Integer T as input.

3. For next T Lines format $(case, x(optional))$

- case 1. $Print$ the $size$ of the $queue$ in a separate line.
- case 2. $Remove$ an element from the queue. If the queue is empty then print -1 in a separate line.
- case 3. Add Integer x to the $queue$ s.
- case 4. $Print$ an element at the $front$ of the $queue$. If queue is empty print -1 in a seperate line.

Queue<Integer> q = new ArrayDeque<>();

q.size()

queue underflow

q.remove();

q.add(x);

q.peek()

queue underflow

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    Queue<Integer> q = new ArrayDeque<>();

    int test = scn.nextInt();
    while(test-- > 0){
        int choice = scn.nextInt();

        if(choice == 1){
            // size
            System.out.println(q.size());

        } else if(choice == 2){
            // remove
            if(q.size() == 0) System.out.println(-1);
            else q.remove();

        } else if(choice == 3){
            // insert
            int val = scn.nextInt();
            q.add(val);

        } else {
            // get/peek
            if(q.size() == 0) System.out.println(-1);
            else System.out.println(q.peek());
        }
    }
}
```

test=5 for(int idx=0; idx<5; idx++)

0 → 4

while(test-- > 0)

5 → 1

int[]

char[]

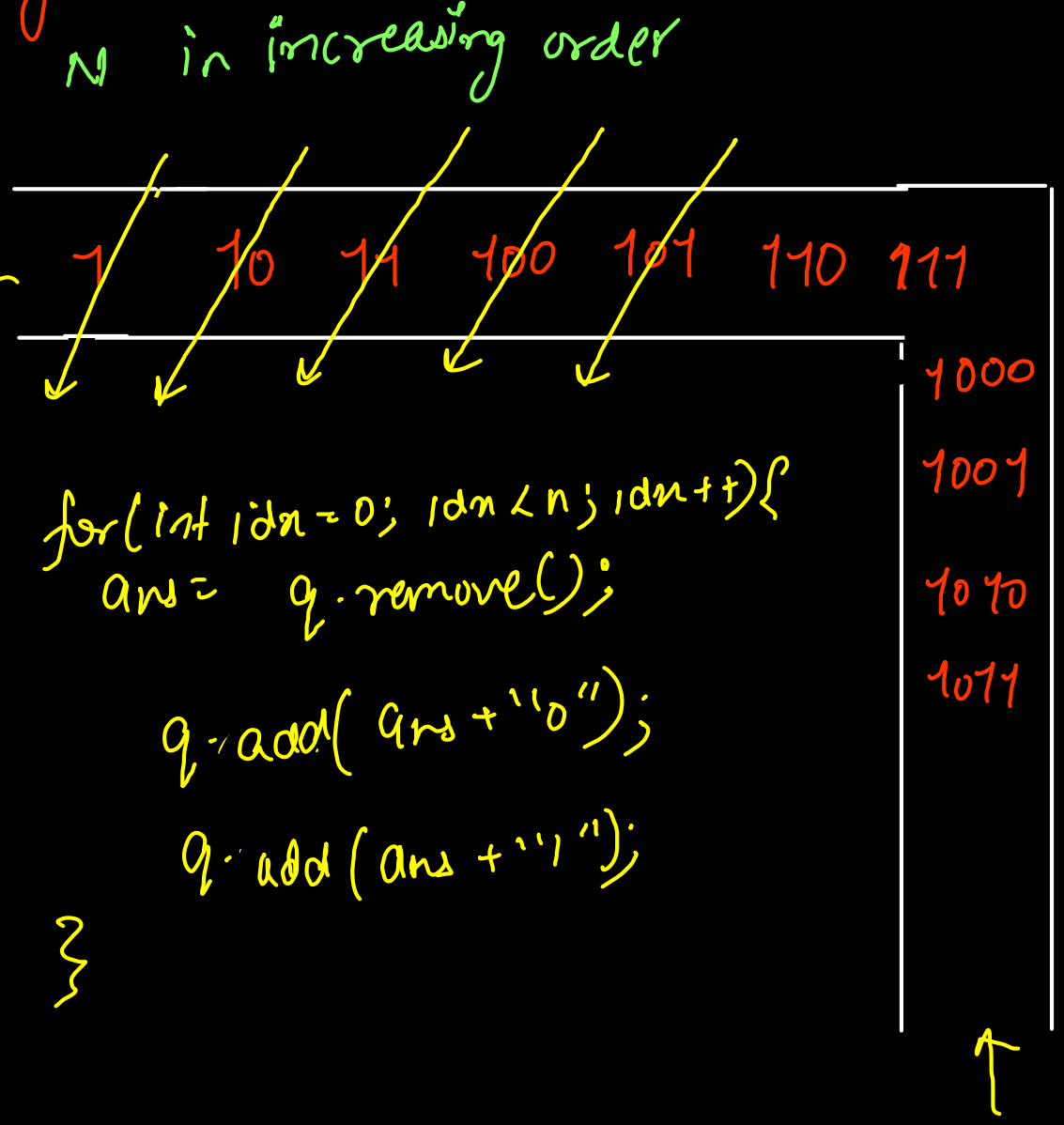
String[]

Queue

Print Binary { BFS }

All from 1 to N in increasing order

1	1 ✓	9	1001
2	10 ✓	10	1010
3	11 ✓	11	1011
4	100 ✓	12	1100
5	101 ✓	13	1101
6	110 -	14	1110
7	111 -	15	1111
8	1000 -	16	10000
		=n	




```
for(int idx = 0; idx < n; idx++){
    ans = q.remove();

    q.add(ans + "0");

    q.add(ans + "1");
}
```

```java
static ArrayList<String> generate(int N)
{
    ArrayList<String> res = new ArrayList<>();
    Queue<String> q = new ArrayDeque<>();
    q.add("1");

    while(N-- > 0){
        String ans = q.remove();
        res.add(ans);

        q.add(ans + "0");
        q.add(ans + "1");
    }

    return res;
}
```

O(1)

O(1)

O(1)

Time $\leq O(n)$ ignoring concatenation

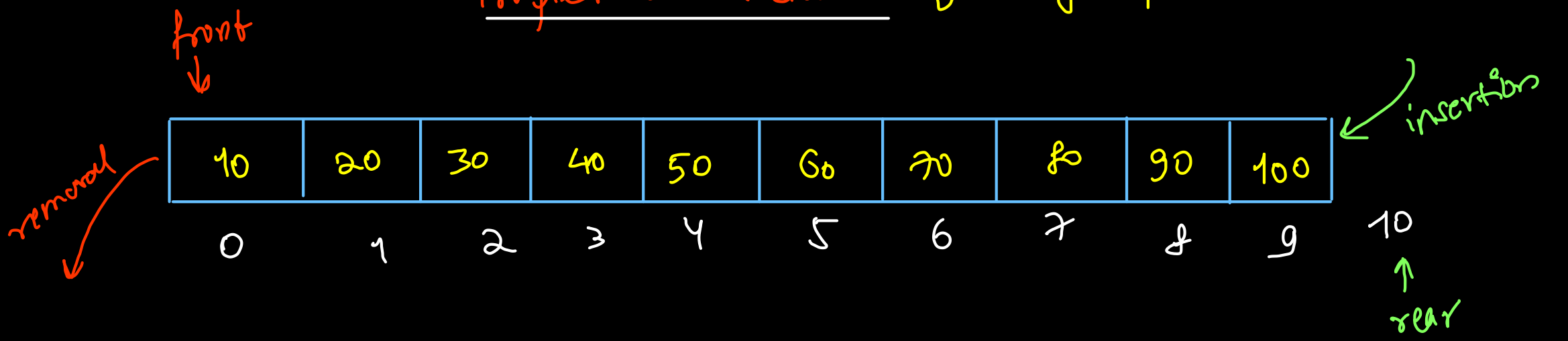
Queue Problem

Generate Binary Nos from 1 to N.

{GFG}

```
[1]
[10, 11]
[11, 100, 101]
[100, 101, 110, 111]
[101, 110, 111, 1000, 1001]
[110, 111, 1000, 1001, 1010, 1011]
[111, 1000, 1001, 1010, 1011, 1100, 1101]
```

Implement a Queue { Array Implementation }



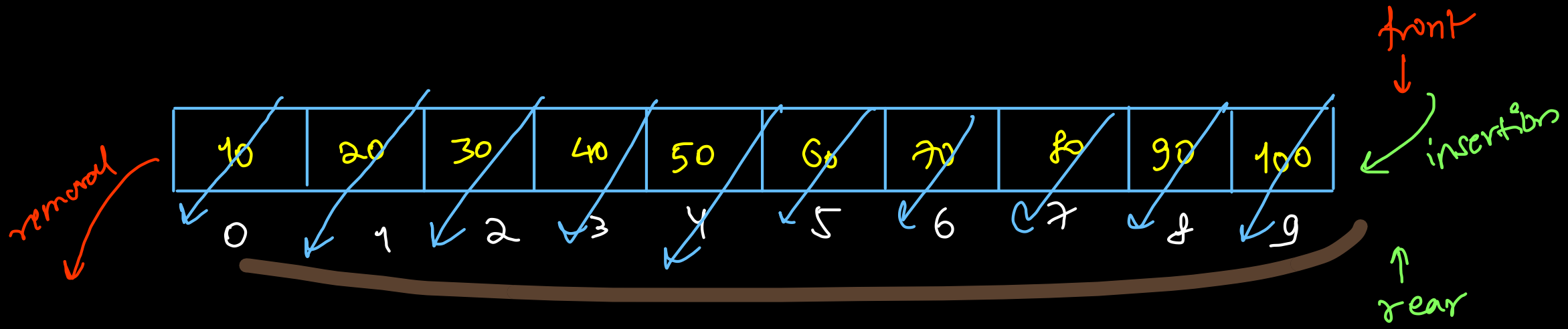

```
q.add(10);
q.add(20);
q.add(30);
```

```
q.add(40);
q.add(50)
q.add(60)
```

```
q.add(70);
q.add(80)
q.add(90);
```

```
q.add(100)
q.add(110){rear == n}
  ↳ Queue overflow}
```

```
void add(int x){
    arr[rear] = x;
    rear++;
}
```



q.remove() → 10

q.remove() → 20

q.remove() → 30

⋮

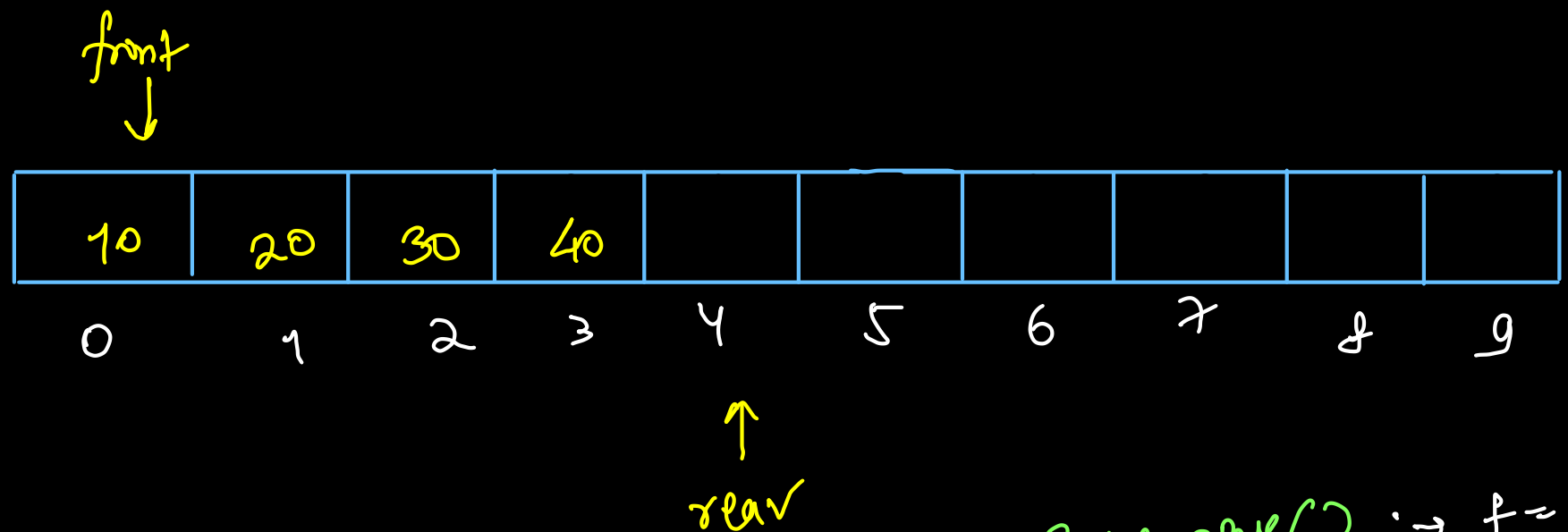
q.remove() → 90

q.remove() → 100

q.remove()

↳ queue underflow

```
int remove() {
    int res = arr[front];
    front++;
    return res;
}
```



q.remove() :→ f == r
queueunderflow

q.add(10)

q.add(20)

q.add(30)

q.add(40)

q.remove() ⇒ 10

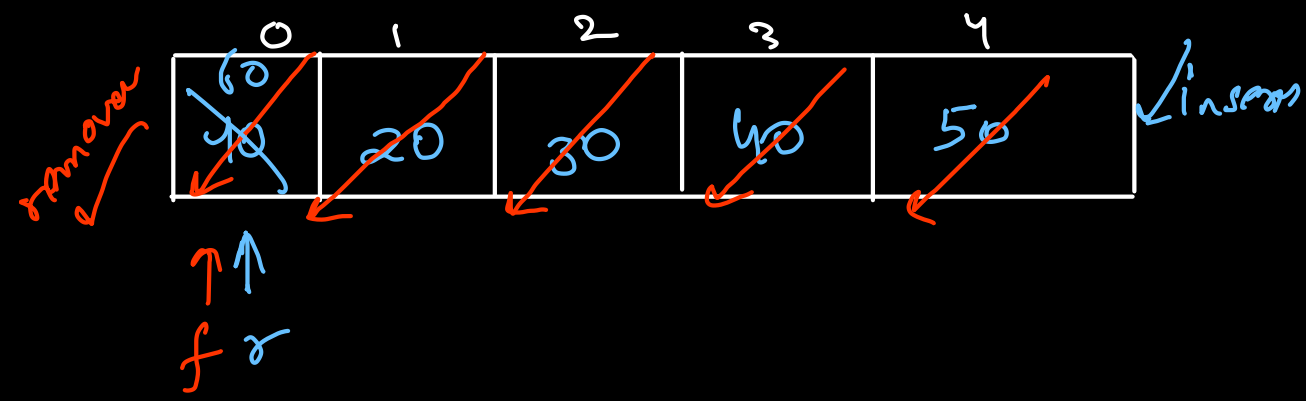
q.remove() ⇒ 20

q.remove() ⇒ 30

q.remove() ⇒ 40

q.remove()
⇒ f == r
queueunderflow

```java
int[] q = new int[100000];
int front = 0, rear = 0;

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

while(tests-- > 0){
    String str = scn.next();

    if(str.equals("enqueue")){
        int x = scn.nextInt();
        if(rear < q.length){  //to ensure queue overflow
            q[rear] = x;
            rear++;
        }

    } else if(str.equals("dequeue")){
        if(front == rear){
            // Queue Underflow: Empty Queue
            front = rear = 0;
        } else {
            front++;
        }
    } else if(str.equals("display")){
        for(int idx = front; idx < rear; idx++){
            System.out.print(q[idx] + " ");
        }
        System.out.println();
    } else if(str.equals("size")){
        System.out.println(rear - front);
    }
}
```



q.add(10)
q.add(20)
q.add(30)
q.add(40)
q.size()
q.remove()
q.remove()

q.add(50)
q.remove()
q.remove()
q.remove()
q.remove() ⇒ f==r
q.add(60);

Implement Queue using Stack

Queue → FIFO

Stack → FILO / LIFO

q.remove()

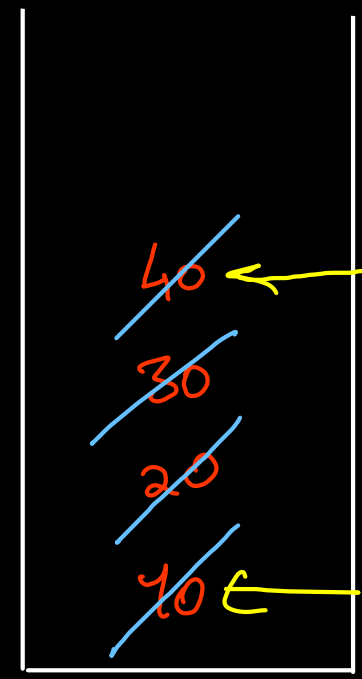
q.add(10) ⇒ stk.push(10)

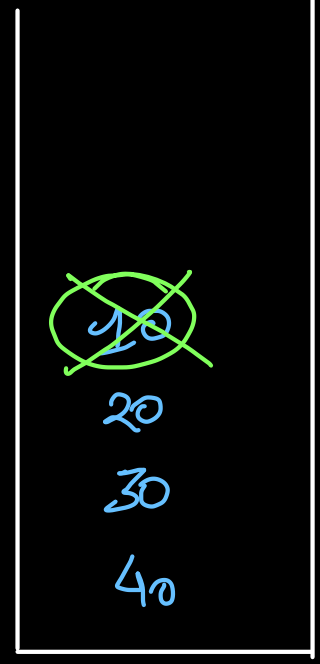
q.add(20) ⇒ stk.push(20)

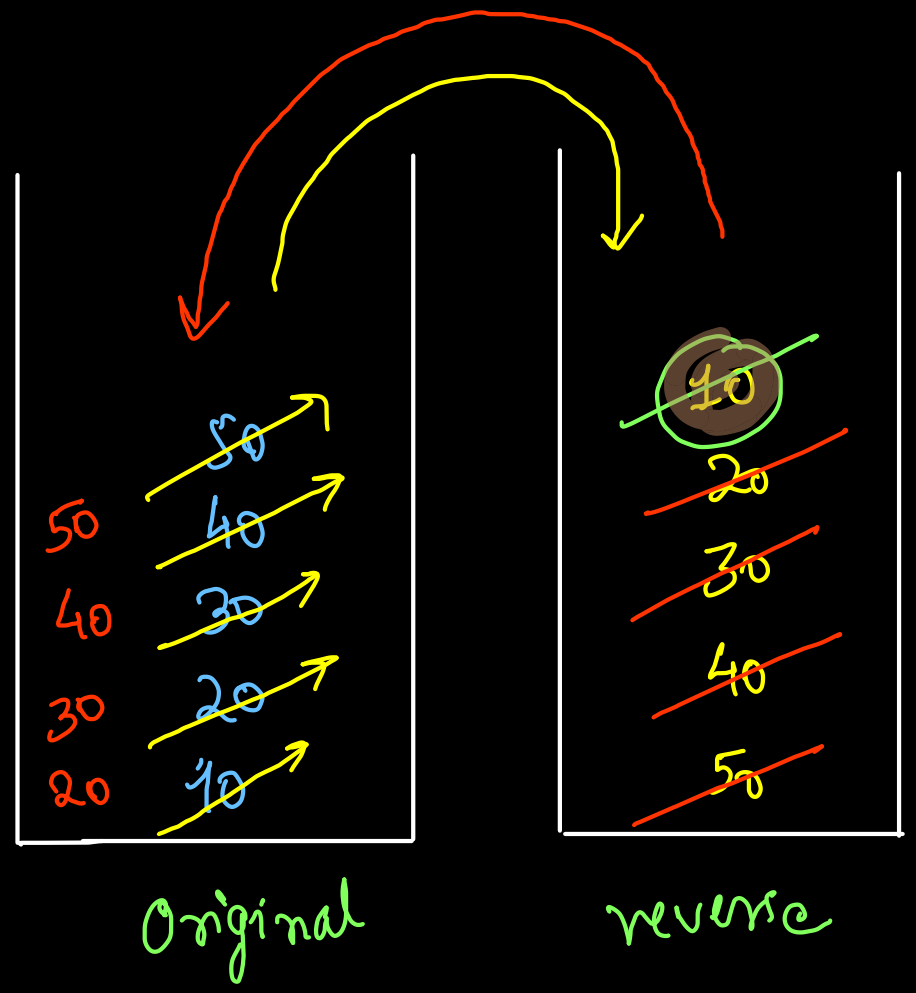
q.add(30) ⇒ stk.push(30)

q.add(40) ⇒ stk.push(40)

} each O(1) operation

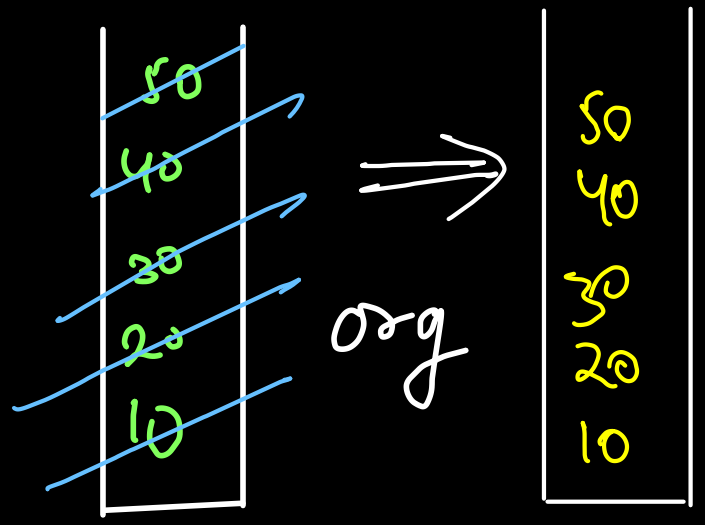







```java
public int pop() {
    Stack<Integer> rev = new Stack<>();
    while(org.size() > 0){
        rev.push(org.pop());
    }

    int res = rev.pop();

    while(rev.size() > 0){
        org.push(rev.pop());
    }

    return res;
}
```

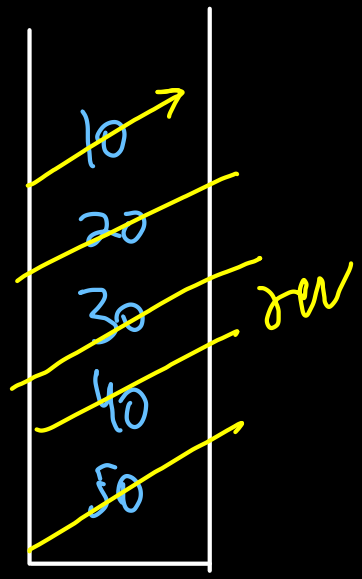
O(n) per remove



q.peek()

↳ front : 10

int res = rev.peek();



```java
Stack<Integer> org = new Stack<>();

public void push(int x) {
    org.push(x); // Insert: O(1)
}

public int pop() {
    Stack<Integer> rev = new Stack<>();
    while(org.size() > 0){
        rev.push(org.pop());
    }

    int res = rev.pop();

    while(rev.size() > 0){
        org.push(rev.pop());
    }

    return res;
}
```

```java
public int peek() {
    Stack<Integer> rev = new Stack<>();
    while(org.size() > 0){
        rev.push(org.pop());
    }

    int res = rev.peek();

    while(rev.size() > 0){
        org.push(rev.pop());
    }

    return res;
}

public boolean empty() {
    return org.empty();
}
```

```java
Stack<Integer> org = new Stack<>();

public void push(int x) {
    org.push(x); // Insert: O(1)
}

public int pop() {
    Stack<Integer> rev = new Stack<>();
    while(org.size() > 0){
        rev.push(org.pop());
    }

    int res = rev.pop();

    while(rev.size() > 0){
        org.push(rev.pop());
    }

    return res; // 10
}
```

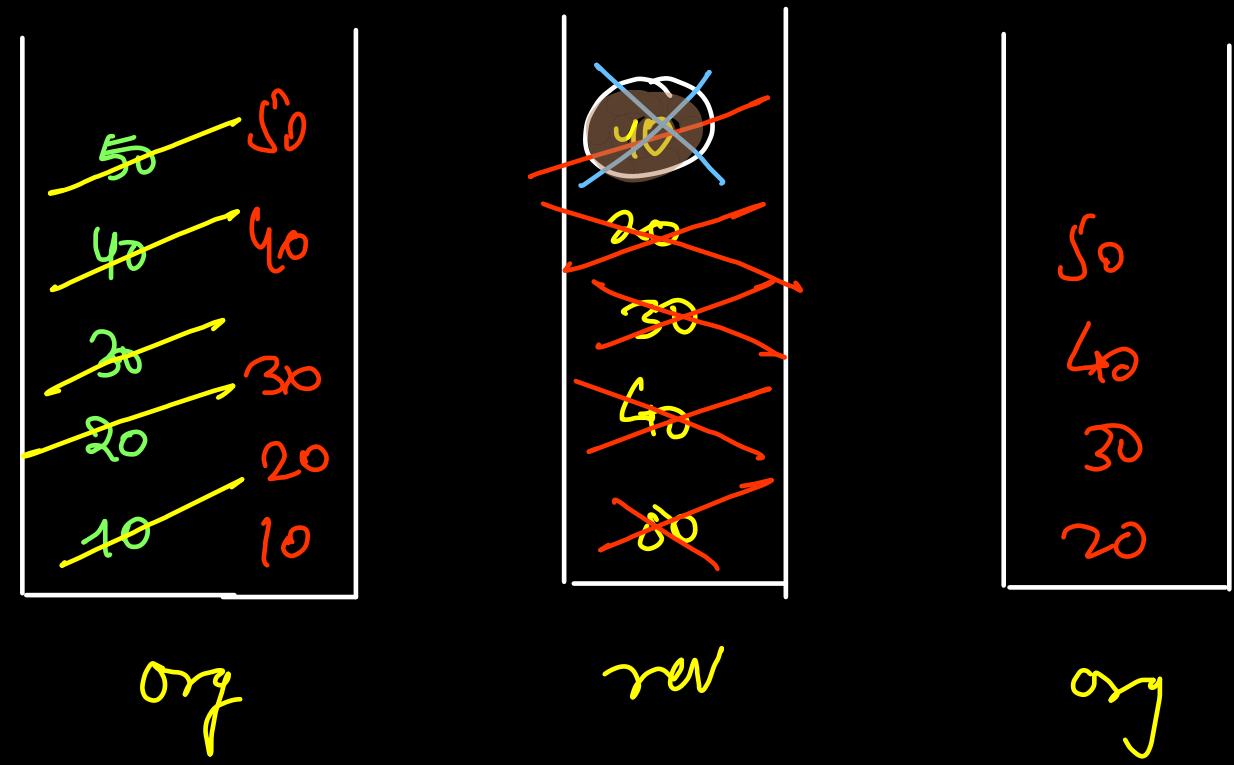
O(1)

O(n)

peek

rev.peek();

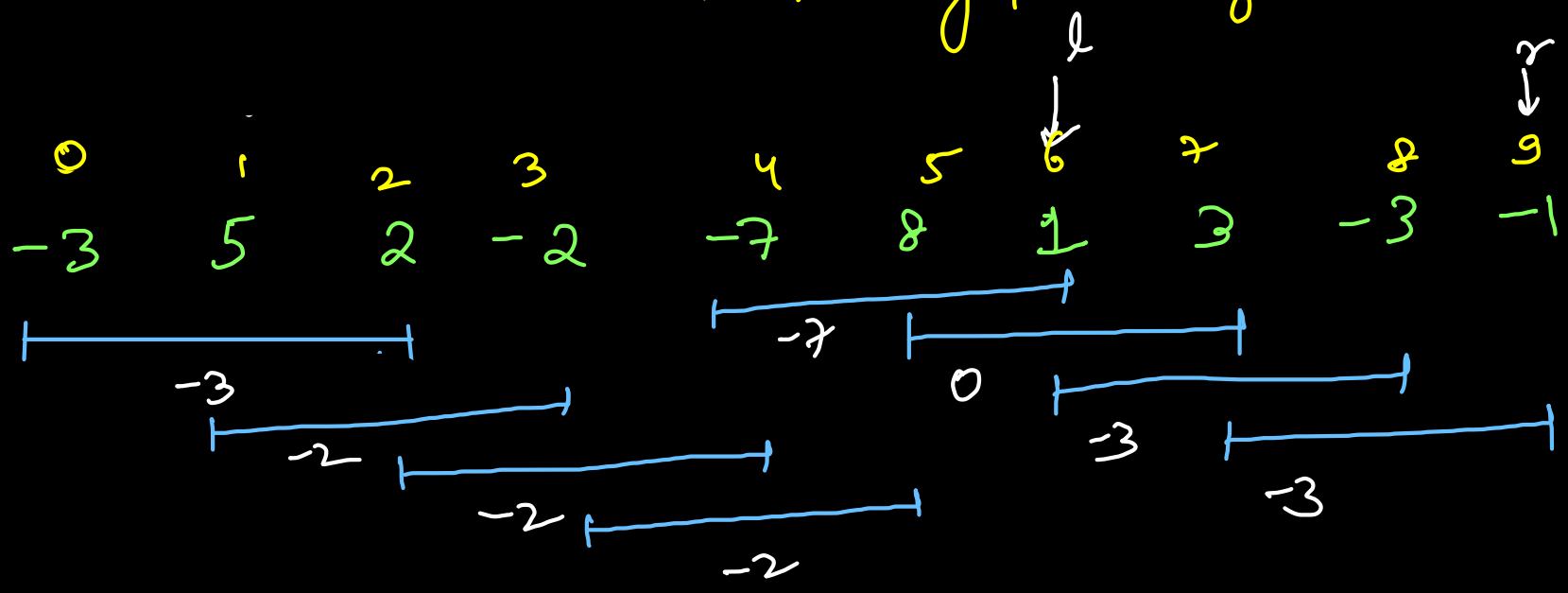
q.add(10)/20/30/40/50);



First Negative Integer

window = 3

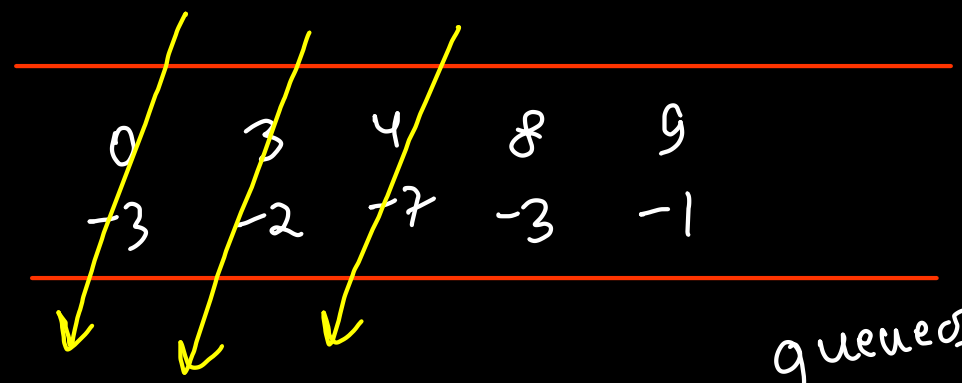
static sliding window



r=5 l=2
r=4 l=1
r=3 l=0
r=0 l=-3

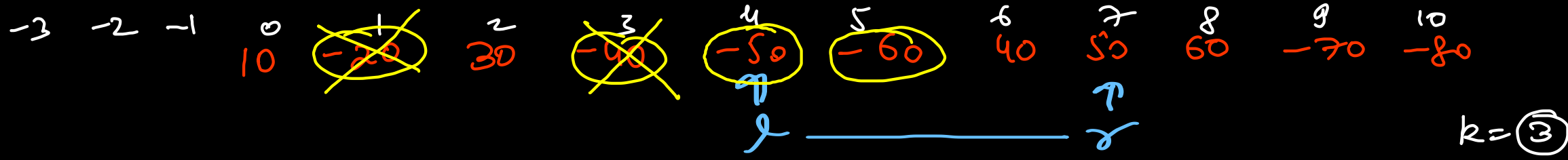
left → exclude

right → include



Queue
First neg In
First Out

queue of negative integer in sliding window




```java
Queue<Integer> q = new ArrayDeque<>();

int right = 0, left = -k;
while(right < n){
    // include the right element
    if(arr[right] < 0)
        q.add(arr[right]);

    // exclude the left element
    if(left >= 0 && arr[left] < 0)
        q.remove();


    left++; right++;
    if(left < 0) continue;
    if(q.size() == 0) System.out.print(0 + " ");
    else System.out.print(q.peek() + " ");
}
```

O(1)

O(1)

-20 , -20 , -40, -40, -50

Time $n * (Include + Exclude)$

$= n * 2 \Rightarrow O(n)$

Linear

Extra space

queue $\rightsquigarrow O(k)$

Priority Queue

preference / importance

→ Ramleela
→ Ram Dwar (VVIP)
→ Laxman Dwar (VIP)
→ Common People

→ Stream BTech
CS
IT
ECE
mech
BioTech

→ windows Task manager
→ chrome ↑cpu
→ timer, disk cleanup, utility tasks
↳ running in background

JEE Rank list

Min-Heap ≈ by default

Rank (1) → min^m no ⇒ highest priority

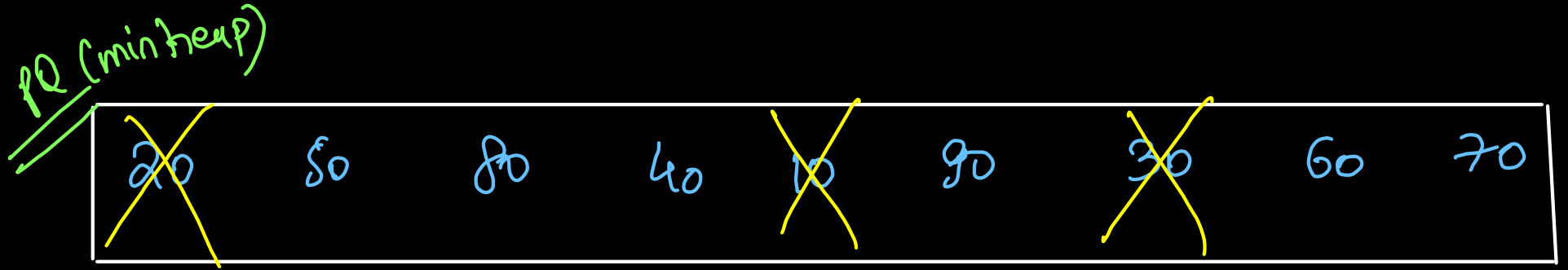
Rank (10 lakh) ⇒ higher no ⇒ lower priority

Marks

max-Heap

0/100 → lowest marks → lowest priority

100/100 → higher marks → highest priority



q.peek() ⇒ 10

q.remove() ⇒ 10

q.peek() ⇒ 20

q.remove()

q.peek() ⇒ 30

Single operations

- insert → O(log n)
- peek → O(1)
- remove → O(log n)

N elements

- insert → NlogN
- peek → N
- remove → NlogN

```java
PriorityQueue<Integer> q = new PriorityQueue<>();
// Min Heap: Highest Priority - Minimum Value

// Single Operation: O(log N)
q.add(20);
q.add(50);
q.add(80);
q.add(40);
q.add(10);
q.add(90);
q.add(30);
q.add(60);
q.add(70);

// Random Order : Unsorted Arrays
System.out.println(q);

// Heap Sort: O(N * log N) Time
while(q.size() > 0){
    System.out.print(q.remove() + " : ");
    System.out.println(q);
}
```

```
Finished in 90 ms
[10, 20, 30, 50, 40, 90, 80, 60, 70]
10 : [20, 40, 30, 50, 70, 90, 80, 60]
20 : [30, 40, 60, 50, 70, 90, 80]
30 : [40, 50, 60, 80, 70, 90]
40 : [50, 70, 60, 80, 90]
50 : [60, 70, 90, 80]
60 : [70, 80, 90]
70 : [80, 90]
80 : [90]
90 : []
```

Min Heap

```java
PriorityQueue<Integer> q = new PriorityQueue<>(Collections.reverseOrder());
// Max Heap: Highest Priority - Maximum Value

// Single Operation: O(log N)
q.add(20);
q.add(50);
q.add(80);
q.add(40);
q.add(10);
q.add(90);
q.add(30);
q.add(60);
q.add(70);

// Random Order : Unsorted Arrays
System.out.println(q);

// Heap Sort in Decreasing Order: O(N * log N) Time
while(q.size() > 0){
    System.out.print(q.remove() + " : ");
    System.out.println(q);
}
```

```
Finished in 86 ms
[90, 70, 80, 60, 10, 50, 30, 20, 40]
90 : [80, 70, 50, 60, 10, 40, 30, 20]
80 : [70, 60, 50, 20, 10, 40, 30]
70 : [60, 30, 50, 20, 10, 40]
60 : [50, 30, 40, 20, 10]
50 : [40, 30, 10, 20]
40 : [30, 20, 10]
30 : [20, 10]
20 : [10]
10 : []
```

1. **Declare an Empty** $PriorityQueue\ q.$

2. Take Single Integer T as input.

3. For next T Lines format $(case, x(optional))$

- case 1. $Print$ the $size$ of the $PriorityQueue$ in a separate line.
- case 2. $Remove$ the smallest element from the $PriorityQueue$. If the queue is empty then print -1 in a separate line.
- case 3. Add Integer x to the $PriorityQueue$ s.
- case 4. $Print$ the smallest element of the $PriorityQueue$. If queue is empty print -1 in a seperate line.

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

    PriorityQueue<Integer> pq = new PriorityQueue<>();

    while(tests-- > 0){
        int choice = scn.nextInt();
        if(choice == 1){
            // size: O(1)
            System.out.println(pq.size());
        } else if(choice == 2){
            // remove: Highest Priority: Minimum Value: O(log N)
            if(pq.size() == 0) System.out.println(-1);
            else pq.remove();
        } else if(choice == 3){
            // Insert: O(log N)
            int x = scn.nextInt();
            pq.add(x);
        } else {
            // Get the Smallest Element: Highest Priority: O(1)
            if(pq.size() == 0) System.out.println(-1);
            else System.out.println(pq.peek());
        }
    }
}
```

Break Stone

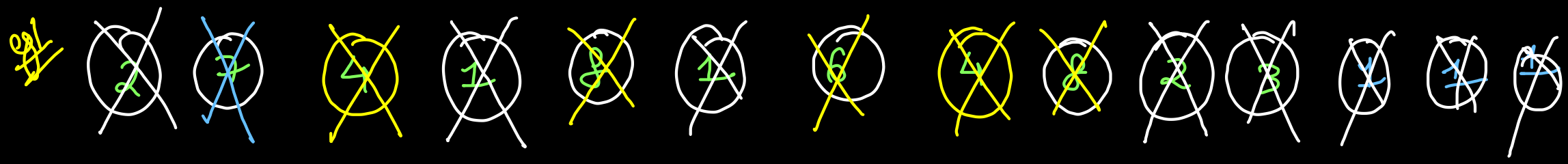
You are given an array of integers **stones** where **stones[i]** is the weight of the i_{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with **x <= y**. The result of this smash is:

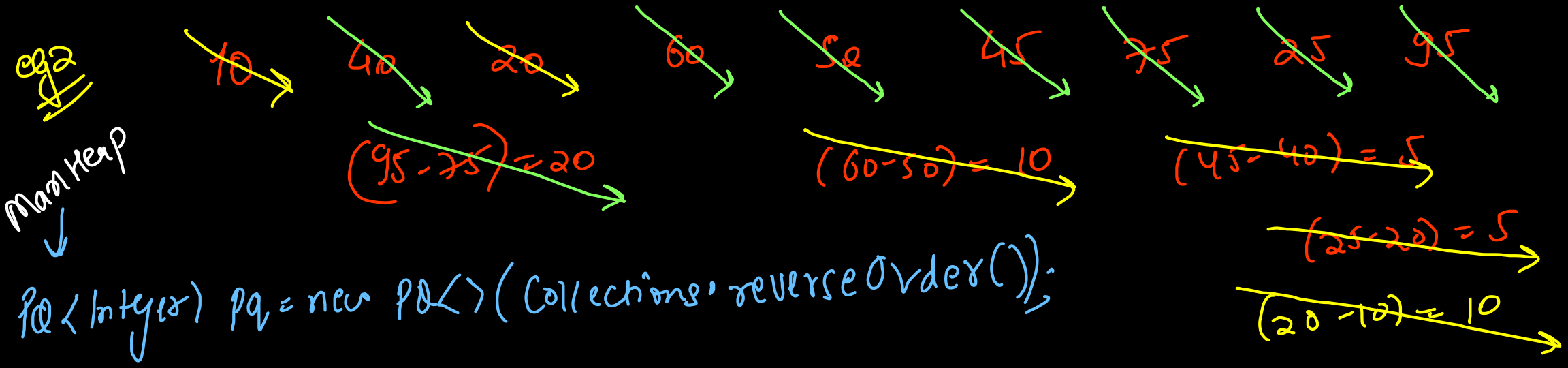
- If **x == y**, both stones are destroyed, and
- If **x != y**, the stone of weight x is destroyed, and the stone of weight y has new weight **y - x**.

At the end of the game, there is **at most one** stone left.

Return the weight of the last remaining stone. If there are no stones left, return 0.



return 0;



MaxHeap

↓

```
PQ<Integer> pq = new PQ<>(Collections.reverseOrder());
```

```java
public static void main(String[] args) {
    PriorityQueue<Integer> pq =
        new PriorityQueue<>(Collections.reverseOrder());
    // max-heap

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

    for(int idx = 0; idx < n; idx++){
        int x = scn.nextInt();
        pq.add(x);
    }

    while(pq.size() > 1){
        int top1 = pq.remove();
        int top2 = pq.remove();
        if(top1 > top2) pq.add(top1 - top2);
    }

    if(pq.size() == 0) System.out.println(0);
    else System.out.println(pq.peek());
}
```

$4n \log n$

$\Rightarrow O(n \log n)$

Problem Statement

You are given an **m x n** binary matrix **mat** of $1's$ (representing soldiers) and $0's$ (representing civilians). The soldiers are positioned **in front** of the civilians. That is, all the $1's$ will appear to the **left** of all the $0's$ in each row.

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

A row i is **weaker** than a row j if one of the following is true:

- The number of soldiers in row i is less than the number of soldiers in row j.
- Both rows have the same number of soldiers and $i < j$.

*Return the indices of the k **weakest** rows in the matrix ordered from weakest to strongest.*

$r_0 < r_1$ $r_2 < r_3$

$r_0 < r_3$ $r_4 > r_3$

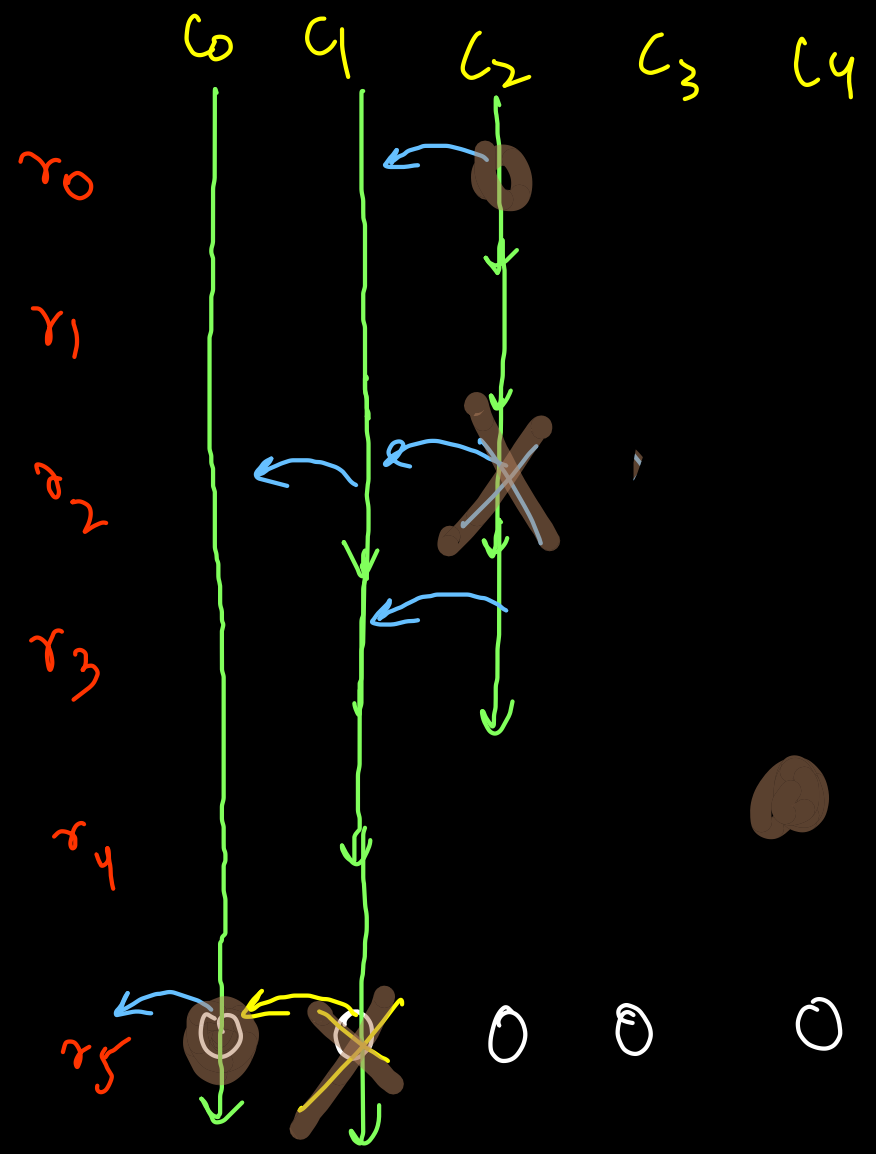
$r_1 < r_4$

$k = 3$

$2^{nd} \rightarrow 1$ soldier

$0^{th} \rightarrow 2$ soldiers

$3^{rd} \rightarrow 2$ soldiers



R = 4 Weakest → Strongest

min^m soldiers top to down

max^m soldiers

r_5 ⇒ 0 soldiers

r_2 ⇒ 1 Soldier

r_0 ⇒ 2 soldiers

r_3 ⇒ 2 soldiers

Time till 8:34 PM to code.

```java
public static void weakestRows(int[][] mat, int rows, int cols, int k){
    for(int c = 0; c < cols; c++){
        for(int r = 0; r < rows; r++){
            if(c - 1 >= 0 && mat[r][c - 1] == 0) continue;
            if(mat[r][c] == 0){
                System.out.print(r + " ");
                k--;
                if(k == 0) return;
            }
        }
    }

    for(int r = 0; r < rows; r++){
        if(mat[r][cols - 1] == 1){
            // entire row consists of 1s: No 0s
            System.out.print(r + " ");
            k--;
            if(k == 0) return;
        }
    }
}
```

if row is already printed ignore it to avoid printing it again!

corner case

Given an array of **digits (values are from 0 to 9)**, find the **minimum possible sum of two numbers** formed from digits of the array. All digits of the given array must be used to form the two numbers.

Any combination of digits may be used to form the two numbers to be summed. Leading zeroes are permitted.

If forming two numbers is **impossible** (i.e. when n==0) then the "sum" is the value of the only number that can be formed.

(123 + 456) = 579

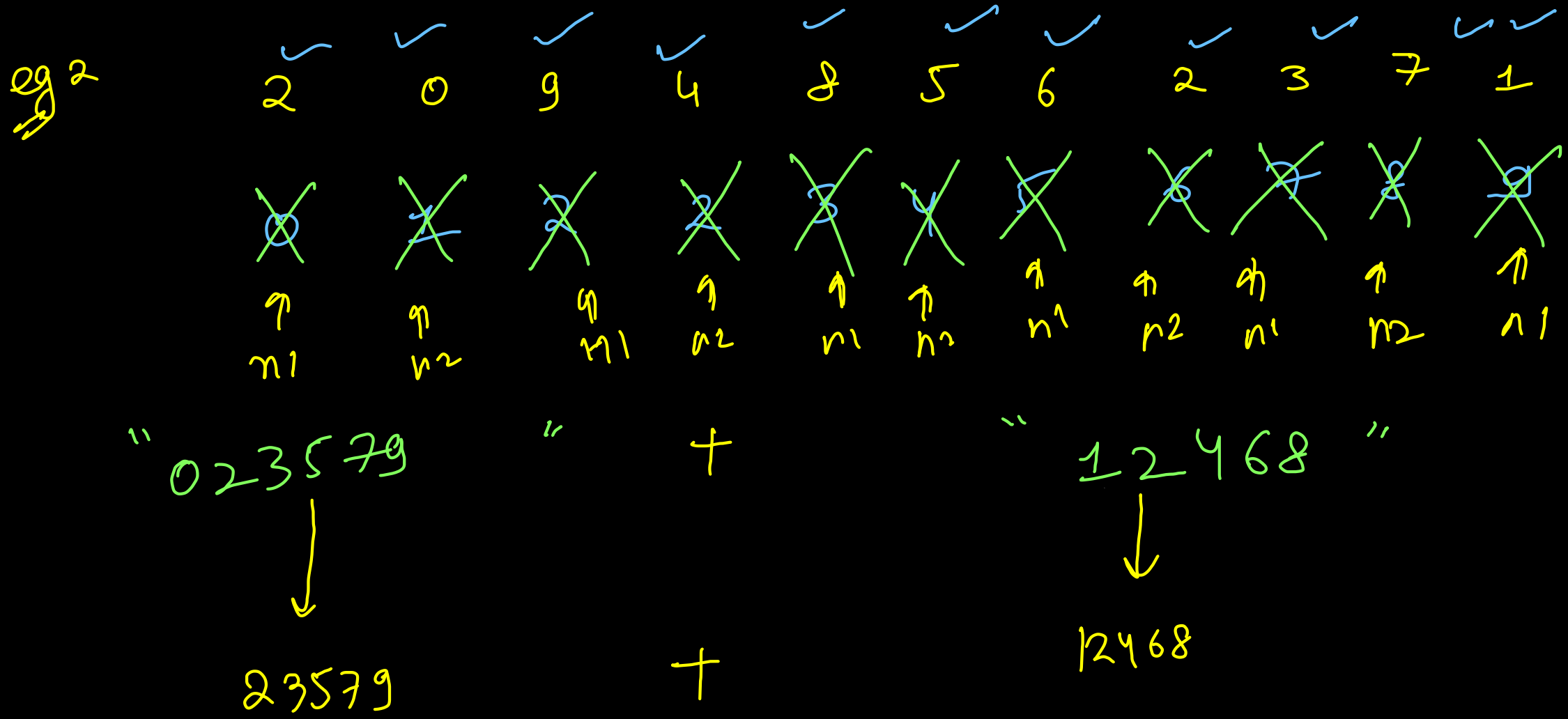
(145 + 236) = 381

(165 + 234) = 399

(135 + 246) = 381

4234 + 56 = 1290

12345 + 6 = 12351


2 0 9 4 8 5 6 2 3 7 1
n1 n2 n1 n2 n1 n2 n1 n2 n1 n2 n1
"023579" + "12468"
23579 + 12468

```java
Scanner scn = new Scanner(System.in);
PriorityQueue<Integer> pq = new PriorityQueue<>();

int n = scn.nextInt();
while(n-- > 0){
    int x = scn.nextInt();
    pq.add(x);
}

int v1 = 0, v2 = 0;

int counter = 0;
while(pq.size() > 0){
    int digit = pq.remove();

    if(counter % 2 == 0)
        v1 = v1 * 10 + digit;
    else v2 = v2 * 10 + digit;

    counter++;
}

System.out.println(v1 + v2);
```

int v1=0; 4 bytes (Stack)

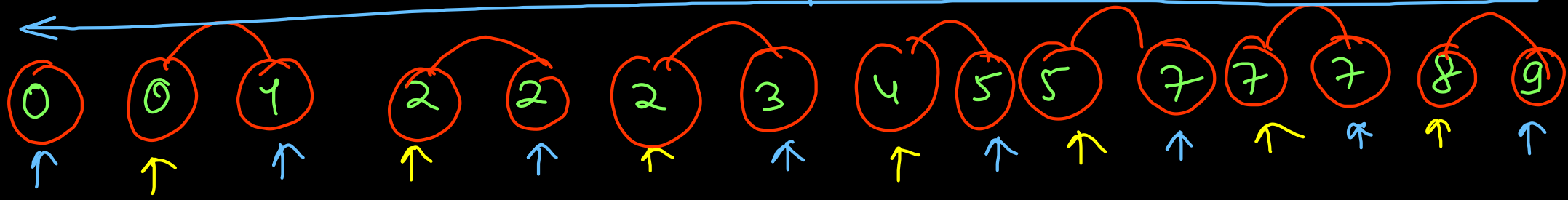
int v2 = 0; 4 bytes (Stack)

uk

Sf

Answer :→ overflowing Integer range

Minm sum of 2 nos



String res = "75306410" reverse ⇒ 01460357

(d1+d2+c)%10

(d1+d2+c)/10 carry = 0 1 1 1 0 0 0 0

"0123579" + "0224578"

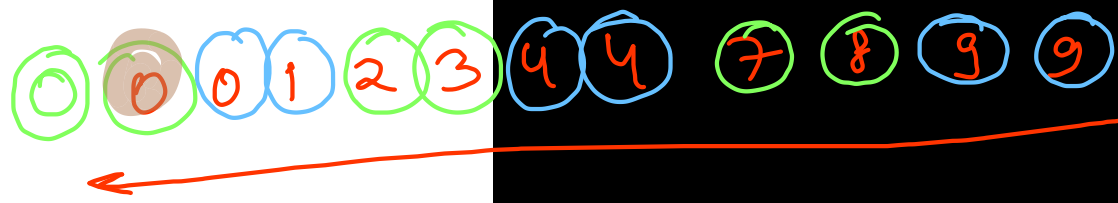
```java
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

int n = scn.nextInt();
while(n-- > 0){
    int x = scn.nextInt();
    pq.add(x);
}

StringBuilder res = new StringBuilder();

int carry = 0;

while(pq.size() > 0 || carry > 0){
    int d1 = (pq.size() == 0) ? 0 : pq.remove();
    int d2 = (pq.size() == 0) ? 0 : pq.remove();

    res.append((d1 + d2 + carry) % 10);
    carry = (d1 + d2 + carry) / 10;
}

// delete leading zeros
for(int idx = res.length() - 1; idx >= 0; idx--){
    if(res.charAt(idx) == '0') res.deleteCharAt(idx);
    else break;
}
res = res.reverse();
System.out.println(res);
```

maxHeap

NlogN

NlogN

arr:



carry = 9 / 1 / 0

"8695100"

reverse

00156 9 8

```java
String solve(int[] arr, int n) {
    PriorityQueue<Integer> pq = new PriorityQueue<>
            (Collections.reverseOrder());

    for(int val: arr)
        pq.add(val);

    StringBuilder res = new StringBuilder();

    int carry = 0;

    while(pq.size() > 0 || carry > 0) {
        int d1 = (pq.size() == 0) ? 0 : pq.remove();
        int d2 = (pq.size() == 0) ? 0 : pq.remove();

        if(d1 == 0 && carry == 0) break;
        // dont add leading zeros

        res.append((d1 + d2 + carry) % 10);
        carry = (d1 + d2 + carry) / 10;
    }

    return res.reverse().toString();
}
```

grocery shop

Fraction

Rice	wheat	Cereals	Salt	Sugar
10 kg	5 kg	2 kg	20 kg	20 kg
700 rs	1000 rs	2000 rs	500 rs	3000 rs
70 rs/kg	200 rs/kg (2)	1000 rs/kg (1)	25 rs/kg	150 rs/kg (3)

thief

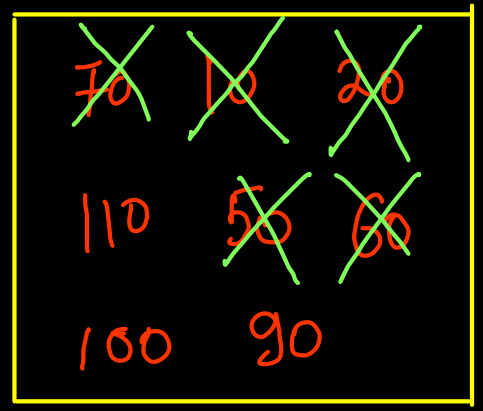
15 kg

Cereals + wheat + sugar

2kg + 5kg + 8kg

K Largest Elements

70 10 20 110 50 60 100 40 90 30 80



k = 3

pq.size = 3

↳ add + remove $\log k$

minHeap (3 largest elements)

k = 3

Time

↳ $N \log k$

Approach 1:

Insert all elements in max Heap
↳ O(NlogN)

+ Pop K ⇒ max K element → O(KlogN)

110, 100, 90

Approach 2:

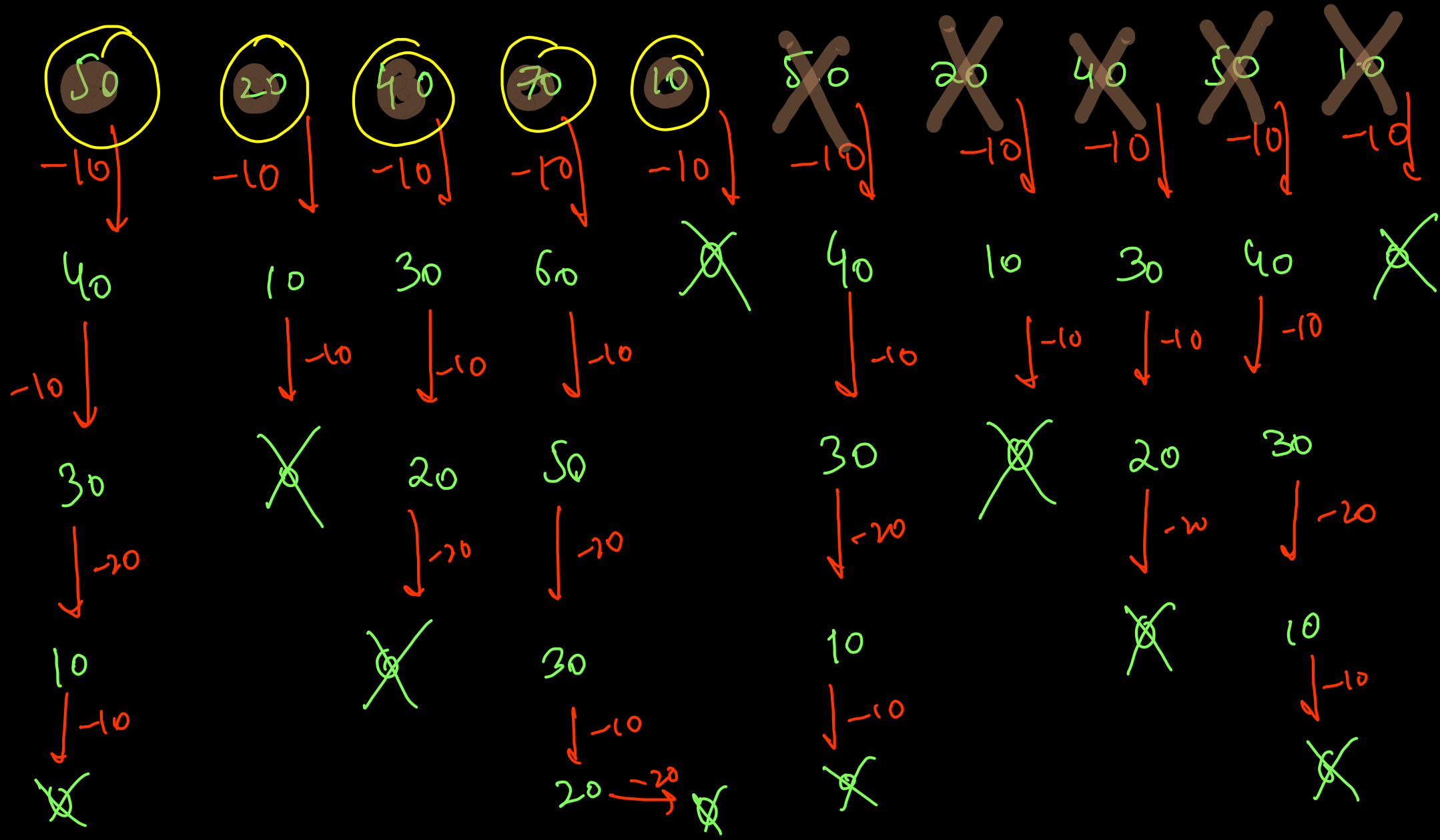
min Heap.

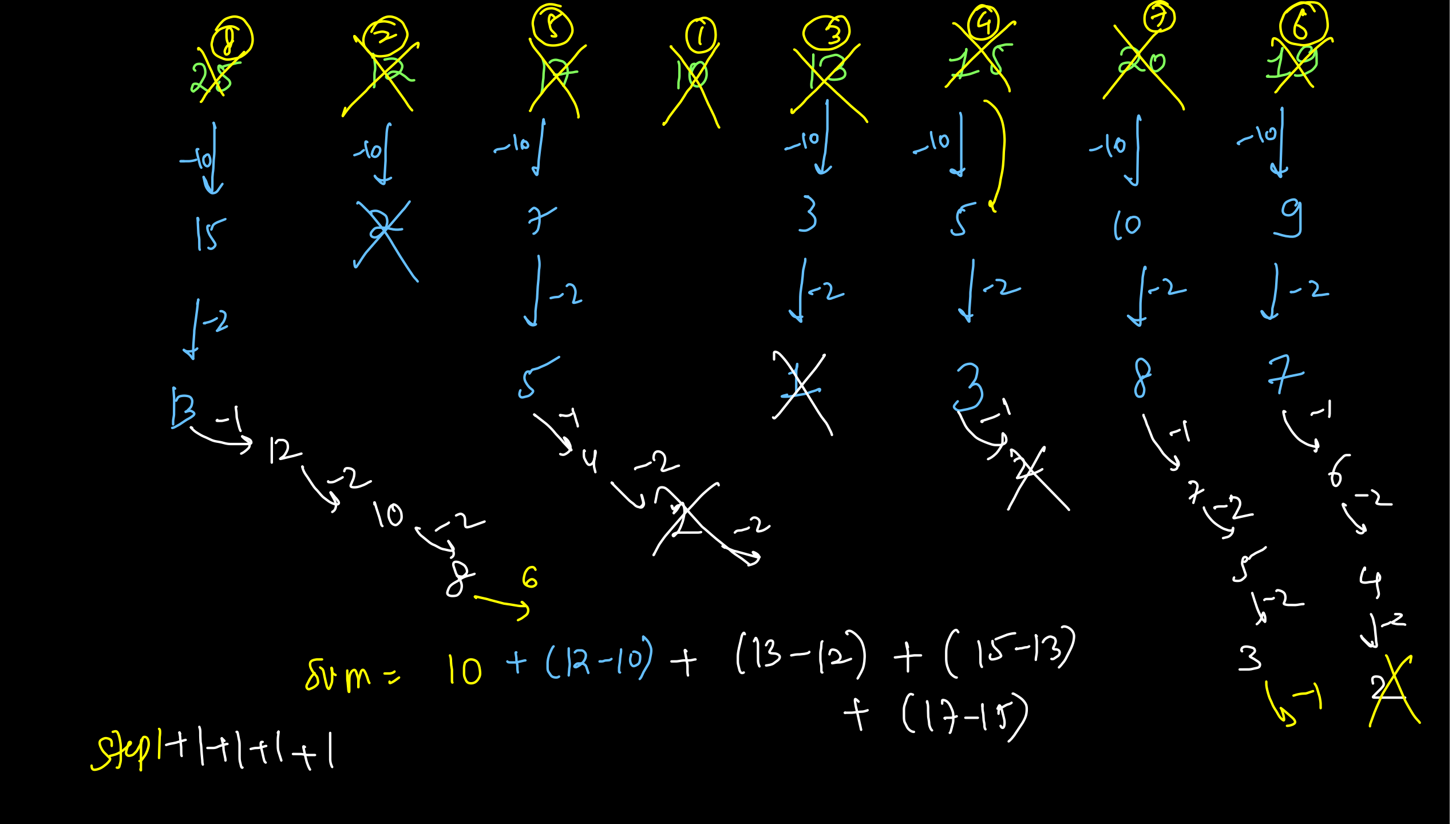
```java
//Function to return k largest elements from an array.
public static ArrayList<Integer> kLargest(int arr[], int n, int k)
{
    PriorityQueue<Integer> pq = new PriorityQueue<>();

    for(int val: arr){
        if(pq.size() < k){
            pq.add(val);
        } else if(pq.peek() < val){
            pq.remove();
            pq.add(val);
        }
    }

    ArrayList<Integer> res = new ArrayList<>();
    while(pq.size() > 0){
        res.add(pq.remove());
    }
    Collections.reverse(res);
    return res;
}
```

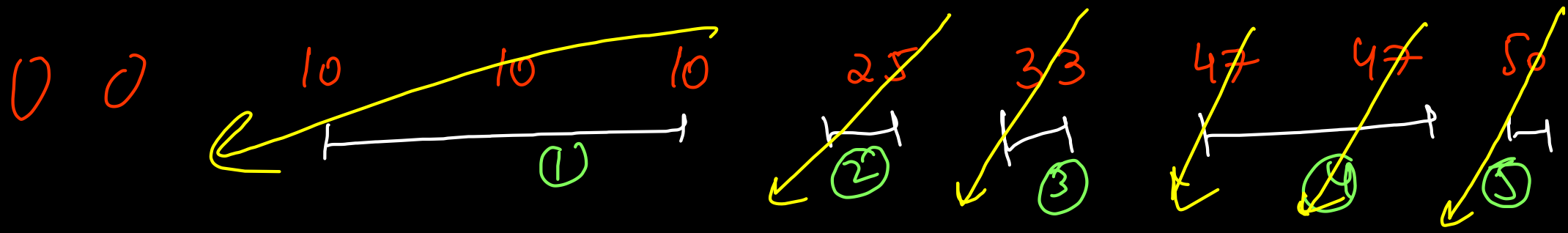
there is vacancy

replace the worst performer!

Subtract Numbers




8 2 5 1 3 4 7 6
25 12 7 10 13 15 20 19
-10 -10 -10 -10 -10 -10 -10 -10
15 2 7 3 5 10 9
-2 -2 -2 -2 -2 -2
13 5 1 3 8 7
-1 -1 -1 -1 -1 -1
12 4 2 2 7 6
-2 -2 -2 -2 -2
10 2 5 7 5 4
-2 -2 -2 -2
8 3 3 5
-1 -1 -1
6 2 2
Sum = 10 + (12-10) + (13-12) + (15-13) + (17-15)
Step 1+1+1+1+1




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

    HashSet<Integer> set = new HashSet<>();

    for(int idx = 0; idx < n; idx++){
        int x = scn.nextInt();
        if(x > 0) set.add(x);
    }

    System.out.println(set.size());
}
```

① 10

② (25-10) = 15

③ (33-25) = 8

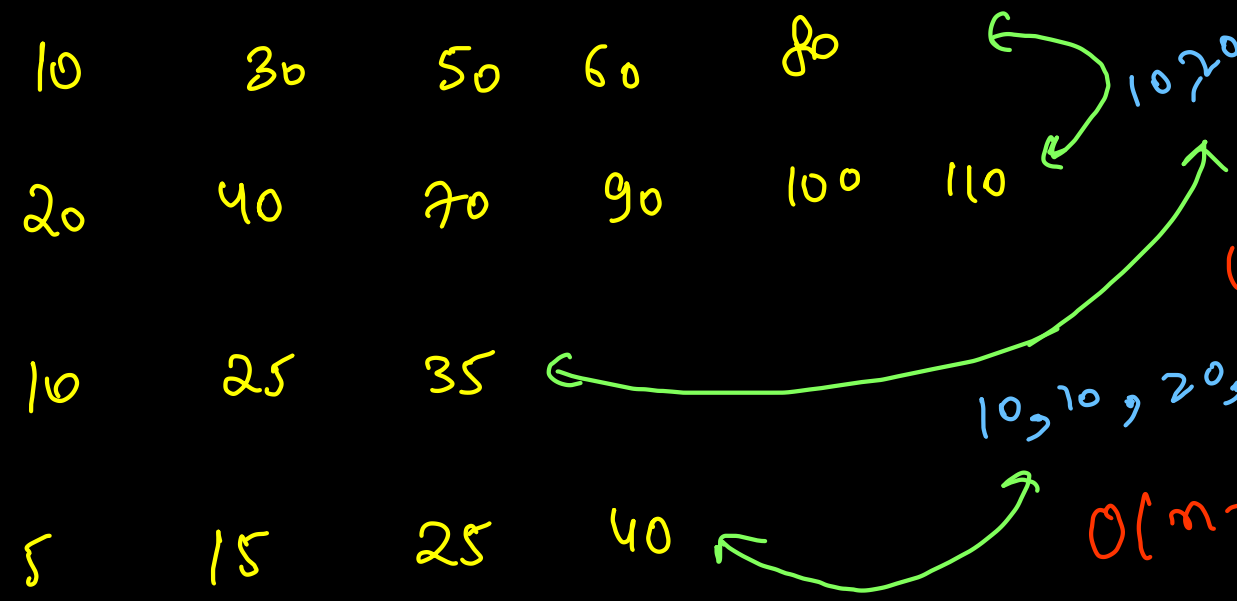
④ (47-33) = 10

⑤ (50-42) = 3

HashSet
↓
Hashmap
w/o value

↳ number of unique non-zero elements

Merge K Sorted Arrays



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

10, 20, 30, 40, 50, 70, 80, 90, 100, 110

$k = 4$

Approach 1: merging 2 at a time

$O(2n+n) = O(3n)$

10, 10, 20, 25, 30, 35, 40 - 110

$O(n+3n) = O(4n)$

5, 10, 10, 15, 20, 25, 25, 30, 35, 40, 40 - 100

Total time Comp'. $\rightarrow$ $O(2n + 3n + 4n + \cdots kn)$

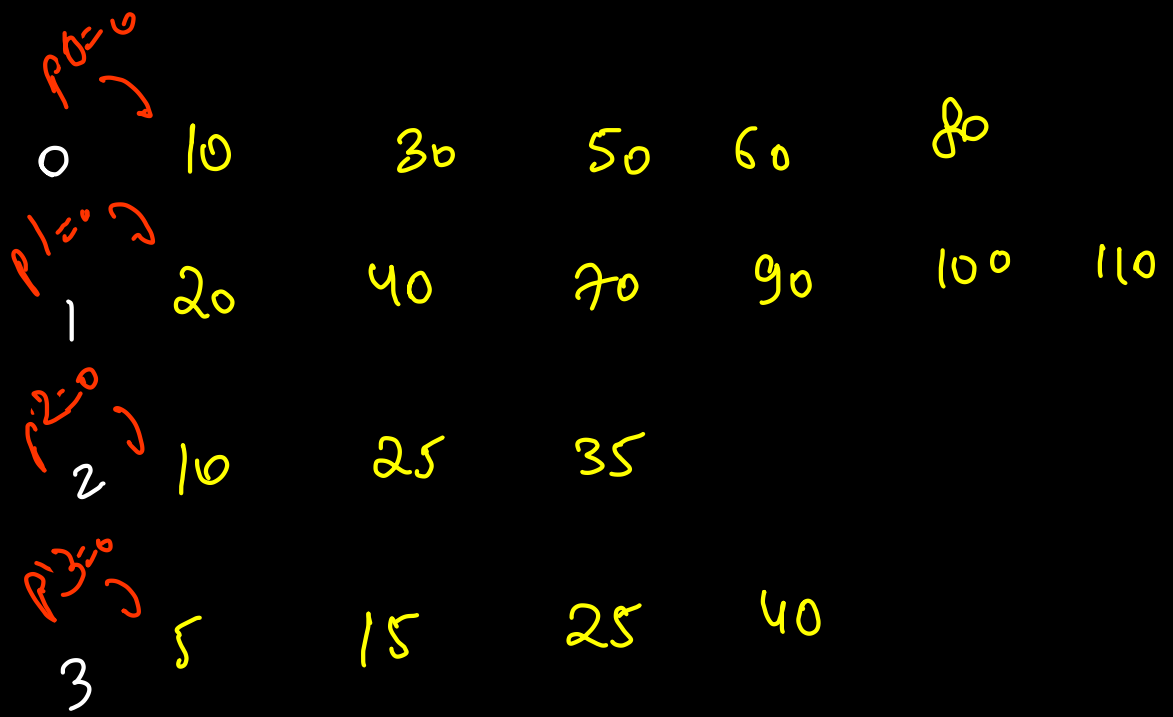
$= O\left(n * \frac{k*(k+1)}{2} \right) = O(nk^2)$

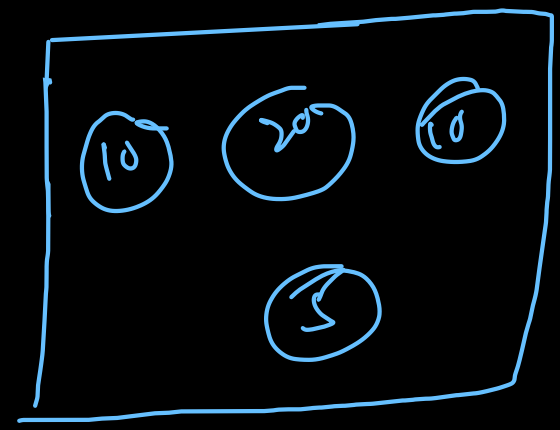
Approach 2: Priority Queue

insert all $(n*k)$ elements in PQ

and sort them $\Rightarrow$ $O(n*k \log n*k)$

$\Rightarrow$ $O(n*k \log n)$





min-Heap

pq < int[] >

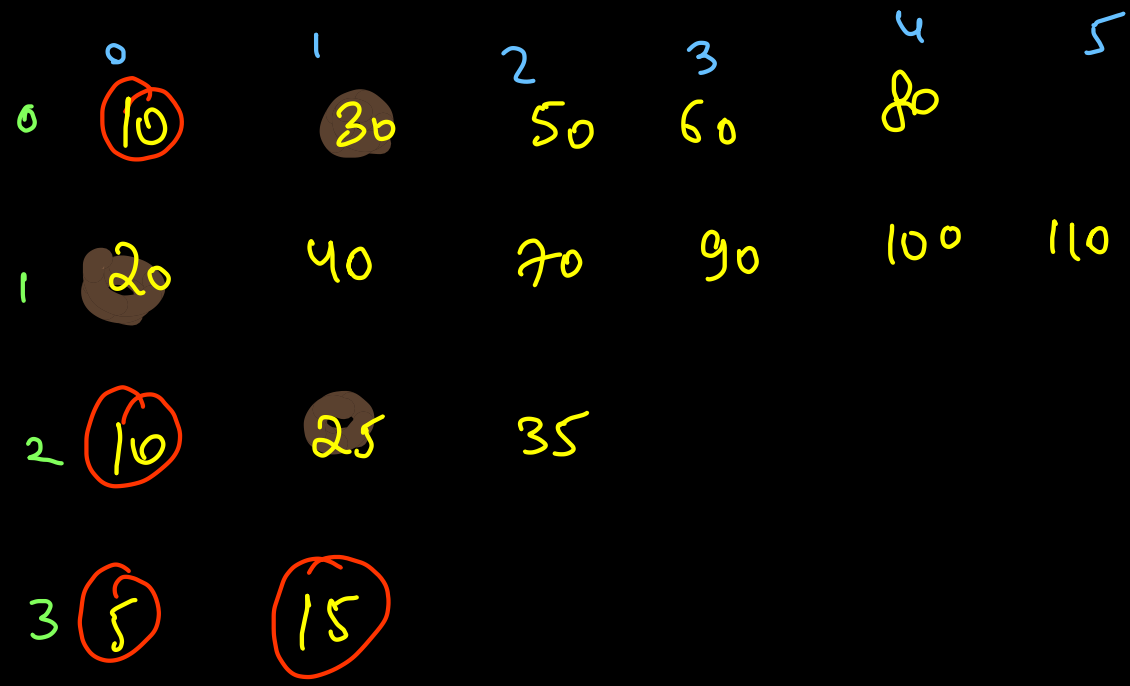
{ row, col, value }

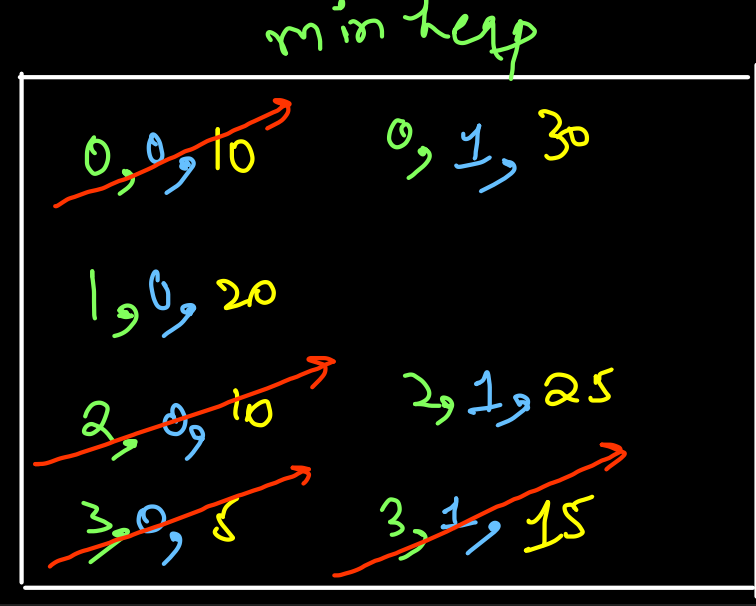
way1: push indices only

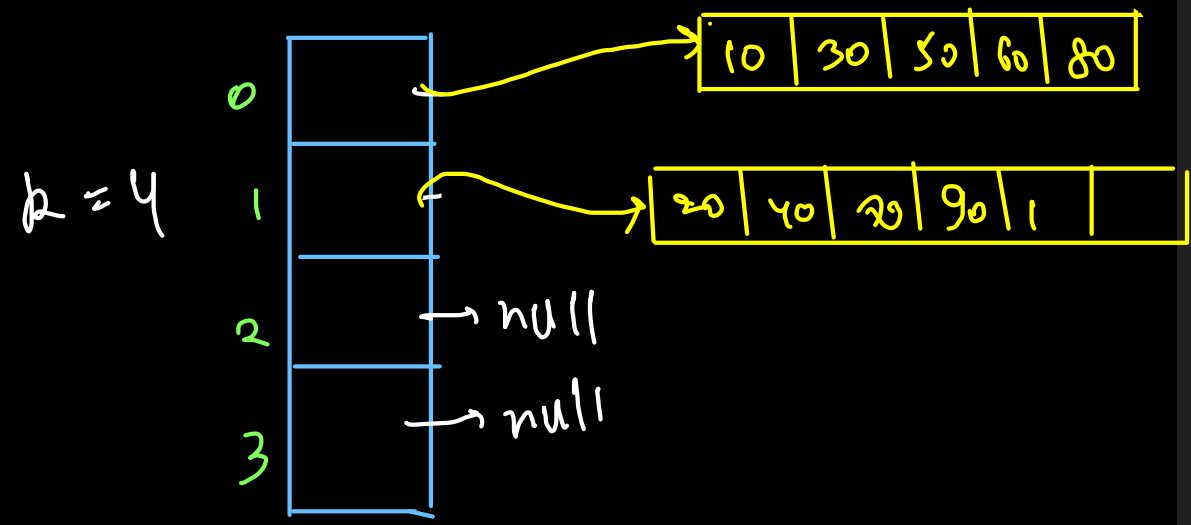
↳ wrong

way2: push values only

↳ wrong

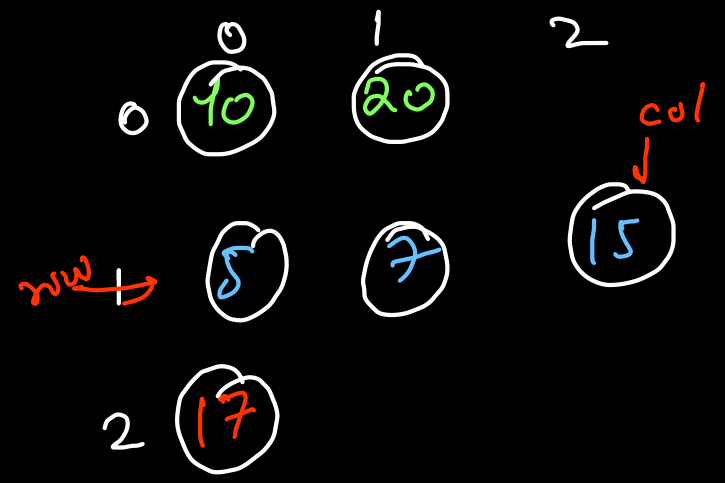


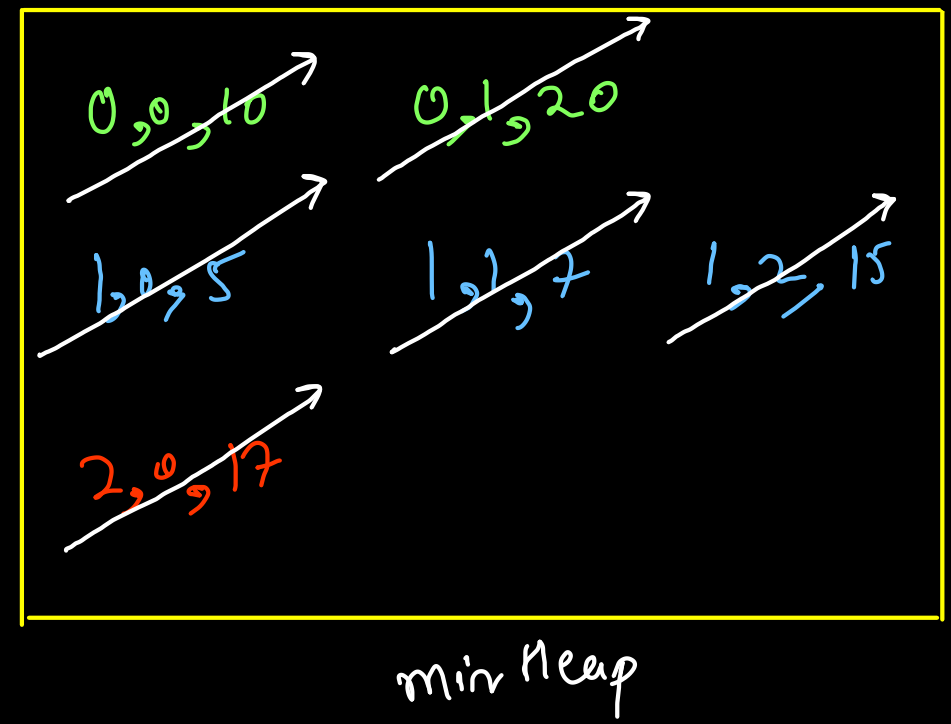





```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    int rows = scn.nextInt();
    int[][] mat = new int[rows][];

    for(int row = 0; row < rows; row++){

        int cols = scn.nextInt();
        mat[row] = new int[cols];

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

    mergeKSorted(mat);
}
```

```java
int rows = mat.length;

PriorityQueue<int[]> pq = new PriorityQueue<>();

for(int row = 0; row < rows; row++){
    int[] ele = new int[3];
    ele[0] = row; ele[1] = 0; ele[2] = mat[row][0];
    pq.add(ele); // {row, col, value}
}

while(pq.size() > 0){
    int[] ele = pq.remove();
    int row = ele[0], col = ele[1], value = mat[row][col];

    System.out.println(value);

    if(col + 1 == mat[row].length) continue;
    col++; value = mat[row][col];
    pq.add(ele);
}
```

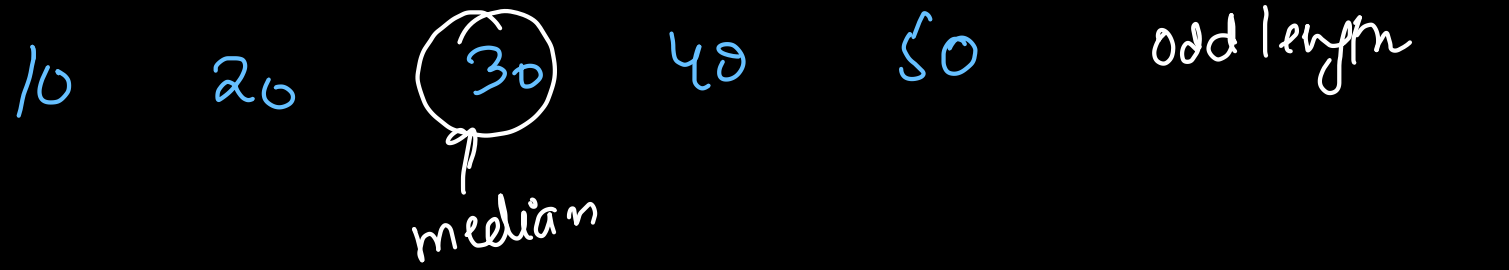


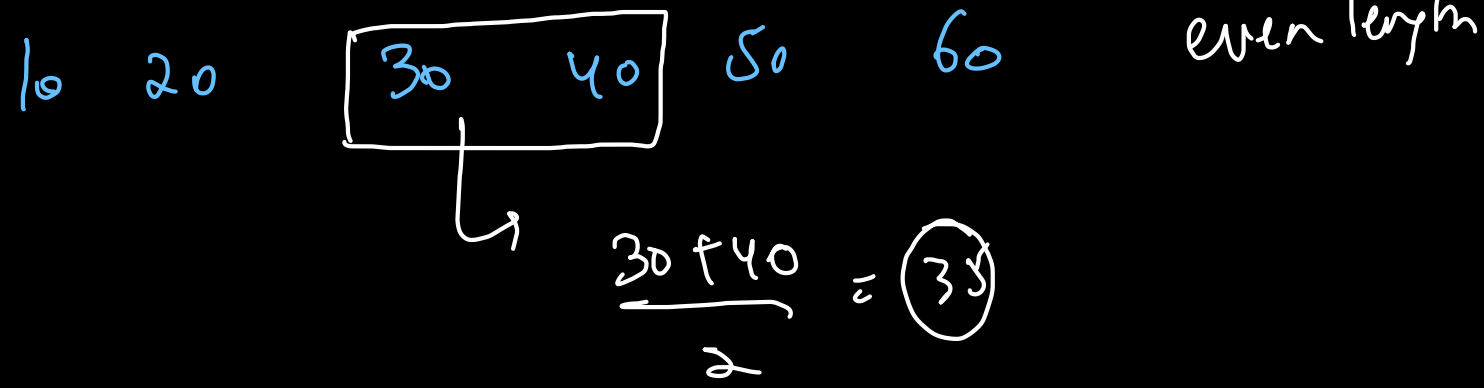


```java
public static void mergeKSorted(int[][] mat){
    int rows = mat.length;

    PriorityQueue<int[]> pq = new PriorityQueue<>((e1, e2) -> e1[2] - e2[2]);

    for(int row = 0; row < rows; row++){
        int[] ele = new int[3];
        ele[0] = row; ele[1] = 0; ele[2] = mat[row][0];
        pq.add(ele); // {row, col, value}
    }

    while(pq.size() > 0){
        int[] ele = pq.remove();
        int row = ele[0], col = ele[1], value = mat[row][col];
        System.out.print(value + " ");

        if(col + 1 == mat[row].length) continue;
        ele[1]++; ele[2] = mat[row][col + 1];
        pq.add(ele);
    }
}
```

comparator (after OOPS)

Running Median





Running Median

[40] → 40

[10, 30, 50, 60, 80] → 50

[40, 50] → (40+50)/2 = 45

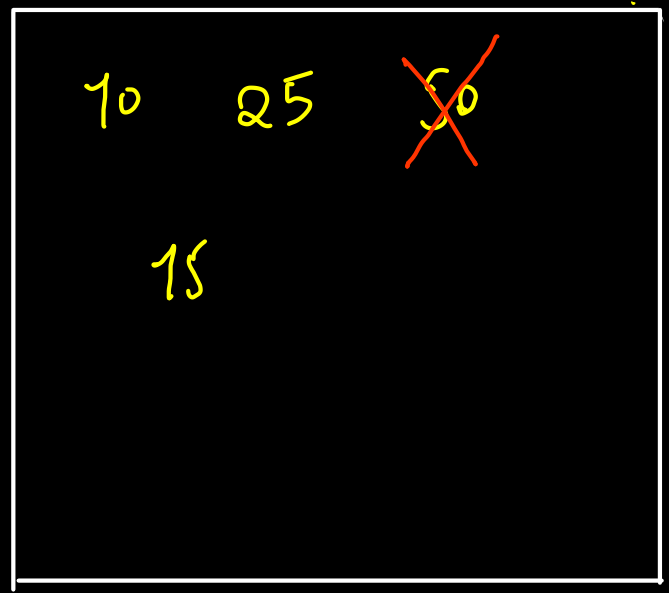
[10, 30, 50, 60, 80, 100] → (50+60)/2 = 55

[10, 50, 80] → 50

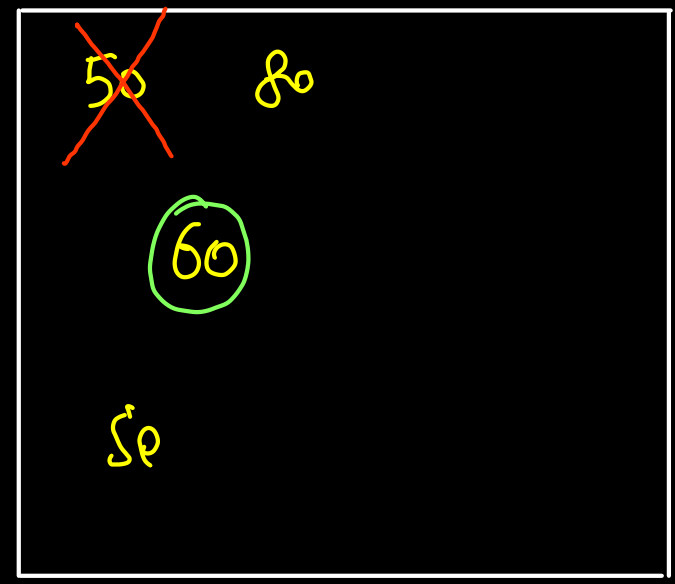
[10, 30, 50, 60, 80, 100, 110]

[10, 50, 60, 80] → (50+60)/2

60



left
{ maxHeap }



right { min Heap }

insert 10 ⇒ (10)

insert 50 ⇒ (10+50)/2 = 30

insert 80 ⇒ (50)

insert 60 ⇒ (50+60)/2 = 55

insert 25 ⇒ 50

insert 15 ⇒

$n = 5$ (odd)

left = 3

right = 2

$l = r + 1$

or

$n = 6$

left = 3

right = 3

$l == r$

balanced

```java
public void addNum(int num) {
    // Normal Insertion
    if(left.size() == 0 || num < left.peek()){
        left.add(num);
    } else right.add(num);

    // Balanced Condition
    if(left.size() == right.size() || left.size() == right.size() + 1)
        return;

    // Rebalancing
    if(right.size() > left.size()) left.add(right.remove());
    else right.add(left.remove());
}
```

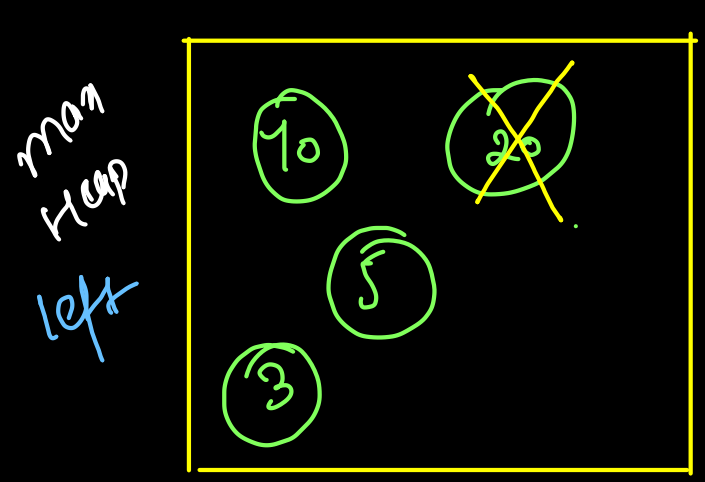
insert 10

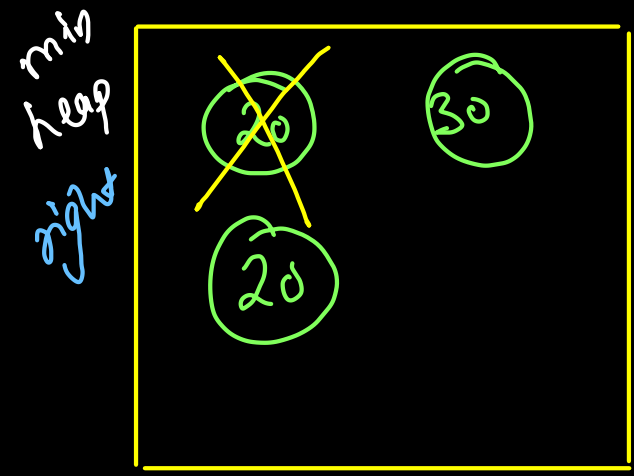
insert 20

insert 30

insert 5

insert 3





```java
class MedianFinder {
    PriorityQueue<Integer> left = new PriorityQueue<>(Collections.reverseOrder());
    PriorityQueue<Integer> right = new PriorityQueue<>();

    public void addNum(int num) {
        // Normal Insertion
        if(left.size() == 0 || num < left.peek()){
            left.add(num);
        } else right.add(num);

        // Balanced Condition
        if(left.size() == right.size() || left.size() == right.size() + 1)
            return;

        // Rebalancing
        if(right.size() > left.size()) left.add(right.remove());
        else right.add(left.remove());
    }

    public double findMedian() {
        if(left.size() == right.size() + 1) return left.peek();
        return (left.peek() + right.peek()) / 2.0;
    }
}
```