

DAY
46

DSA

Recursion In Arrays

- Programming isn't about what you know,
it's about what you can figure out

Agenda for Today :-

1. Display Array → CW
2. Display Array in Reverse → CW
3. Max of Array → CW
4. First Index in Array → HW
5. Last Index in Array → HW
6. All Indices in Array → pending

ques 1.

DISPLAY ARRAY

→ Here, In this question we need to display the contents of an array.

- This question can be done using a simple loop, but we will do this question using Recursion.
- So, In Recursion, we basically need two types of thinking
- एक 2 Level के रियर्सल होता है।

High Level Thinking

Low Level Thinking

High Level Thinking

Expectation

Faith

Expectation using Faith

First define our expectation, define/build the faith and then derive the expectation using faith.

We need to print the content of an Array

Eg:-

$arr[] = \{10, 20, 30, 40, 50\}$

Index → 0 1 2 3 4

1. Expectation

हमारा function होगा `displayArray` जो `arr` का

तो `displayArray(arr, 0)`

`greenArray` starting index

तो इस function से हमारी expectation है कि ये starting index से last index तक के सभी element print करेगा

Expectation → `displayArray(arr, 0)`:

index
10
20
30
40
50

2. Faith :- Expectation से होता faith, build करते हैं

इस array की size तो होता बड़ा नहीं कर सकते क्योंकि array की size तो fixed होता है तो हम array को start index की help से होता print कर सकते हैं, हम start index की आगे छढ़ते चलेंगे।

हम faith रखेंगे कि $(\text{starting index} + 1)$

से last, तक का array अपने आप print हो जायगा।

if Line अपने मन में लोलो

I have faith that `displayArray(arr, 1)`
will display the contents of the
array from index 1 till end.

विश्वास दरवों कि है

function अपना काम करके

कैसे करेगा।

कैसे करेगा? → बाहर सौचार्ह

लैकिन अभी के लिए

विश्वास दरवों कि होगा
ऐसा

`var arr = [10, 20, 30, 40, 50]`

previous starting
index + 1 (0+1)

0 1 2 3 4

Index

`displayArray(arr, 1) :`

20
30
40
50

Index

array और starting index

pass करें इस function

argument तो array length (5)

से कम length (4) के element

print करायें using starting index.

प्रतिलिपि अगर पहली

recursion call 4 element print करने के लिए होते next
call 3 element print करने के लिए & so on, इसके बारे सकते हैं करने के

eg → अगर पहली recursive call

`(1, 2, 3, 4, 5)(all)` (last 5 element) print कराने के लिए

starting index = 0 तो next recursive call

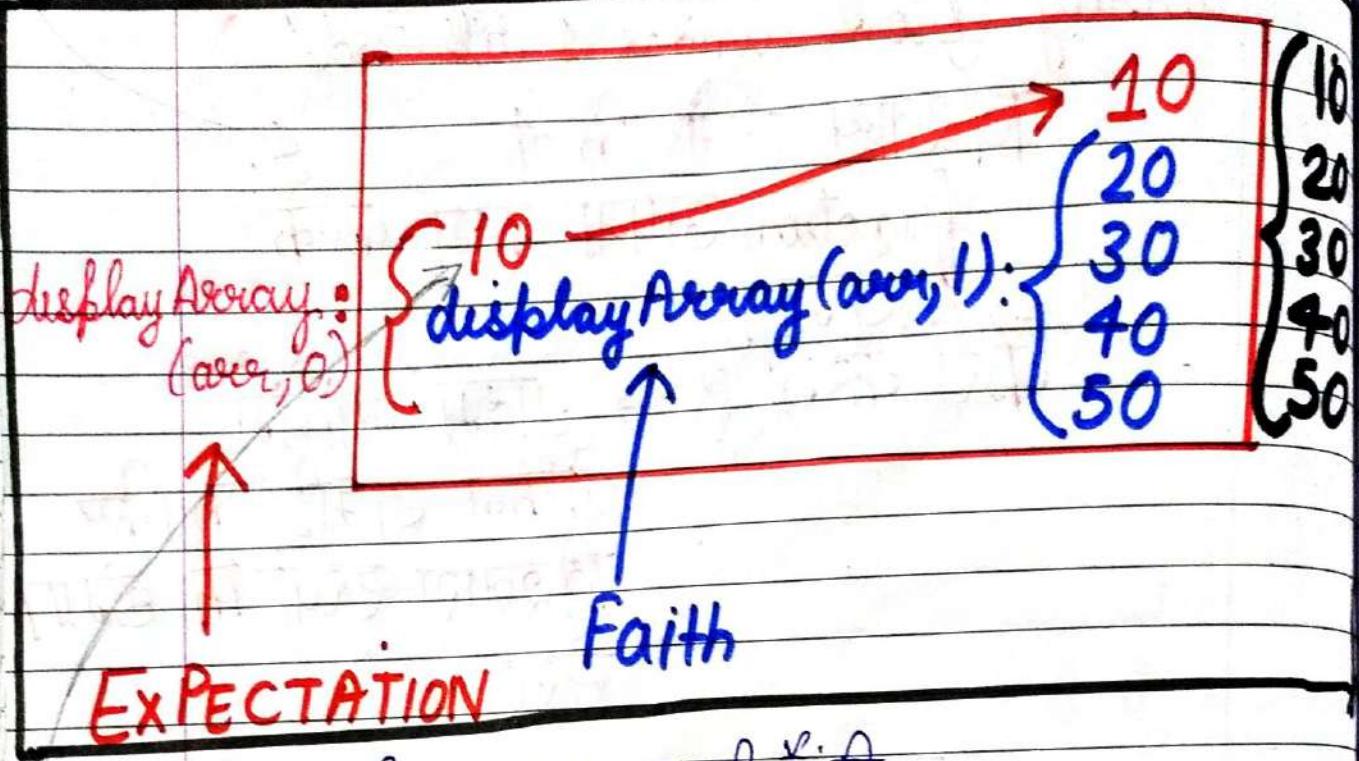
`(2, 3, 4, 5)(last 4 element)` print कराने के लिए होगी।

starting index = 1 & so on

लिए
Starting
index
+1
करते चलो

3. Expectation with Faith

Page No. _____
Date _____



→ So, our expectation कहती है कि

displayArr (arr, 0), will display all the content of array from index 0 till the end.

→ और हमारा faith कहते हैं

displayArr (arr, 1), will display all the content of array from index 1 till the end.

→ तो How to meet Expectation with Faith?

We can just print at 0th Index ourselves and leave the rest on the faith.

v. Basically इस example में हम क्या 10 print कर दें एवं क्या दें और हम faith रखेंगे कि 20, 30, 40, 50, फिर function displayArray function आपने 10 प्रिंट करदेगी और finally we get 10 20 30 40 50

तो हमने High Level Thinking करले हैं।

अब हमें अपने code की approach

पता चल गई है। **HIGH LEVEL THINKING**

Page No.	
Date	

Expectation	Faith	Expectation meets faith
<p>displayArray (arr, 0)</p> <p>It will print all elements from index 0 to last index. $n = arr.length$</p>	<p>display Array (arr, 1)</p> <p>It will print all the elements from index 1 to last index. $n = arr.length$</p>	<p>display Array (arr, 0)</p> <p>display Array (arr, 1)</p> <p>we have to print the first element (at 0th index by ourselves rest will be taken care by displayArray function</p>

Approach.

- We have a recursive formula `displayArray(arr, idx)`
- 1. Check if `idx` is a valid index. If not simply return
- 2. Print the value at the `idx` i.e `arr[idx]`.
- 3. Make a recursive call with parameter `arr & idx+1`

Pseudocode.

```
function displayArray (arr, idx)
    if idx == arr.length () ] If It's Base Case -
    return
    print arr[idx] ] Self Work
    displayArray (arr, idx+1) ] If recursive call
```

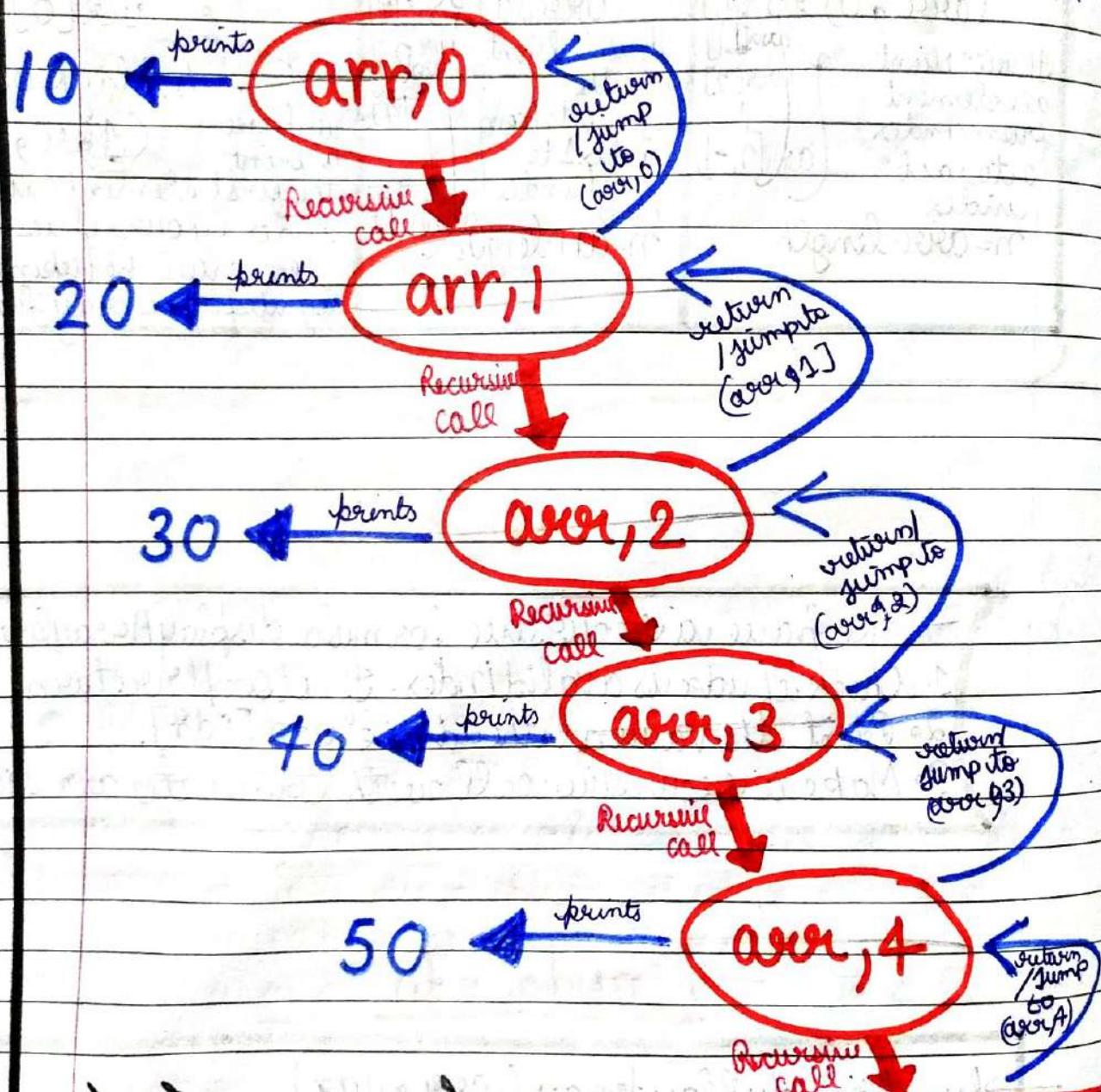
LOW LEVEL THINKING

Page No. _____
Date _____

It is done basically to find the
Base Case

DRY RUN THE CODE

Here is the Recursion Tree



तो हमें dry run करने से पता चलगया कि Base case 5 है।

BASE CASE $5 = \text{arr.length}$

$\hookrightarrow \text{arr.length}$

DISPLAY ARRAY <code>

Page No. _____
Date _____

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main  
→ {  
→   public static void main (String [] args)  
→   {  
→     Scanner scan = new Scanner (System .in);  
→     int n = scan.nextInt(); → int arr [] = new int [n];  
→     for (int i=0; i<n; i++)  
→     {  
→       arr [i] = scan.nextInt();  
→     }  
→   }  
→ }
```

Taking the length of array as input.

→ Taking the value at i th index of array as input.

```
→ displayArray (arr, 0);
```

→ Calling the `displayArray` function to print all the elements from 0th index to last index.

```
→ public static void displayArray (int [] arr, int i)
```

//Expectation → `displayArray (arr, i) = arr [i] — arr [n-1]`
//Faith → `displayArray (arr, i+1) = arr [i+1] — arr [n-1]`.
//Expectation meets Faith → `displayArray (arr, i) = arr [i] displayArray (arr, i+1)`

This function takes 2 parameters. One is `array` (given) & another `input` is the starting index.

```
if (i == arr.length) } if base case  
{ return ; }
```

```
System.out.println (arr [i]); } self work  
displayArray (arr, i+1); } recursive call
```

Time Complexity → $O(n)$ → bcz recursive call

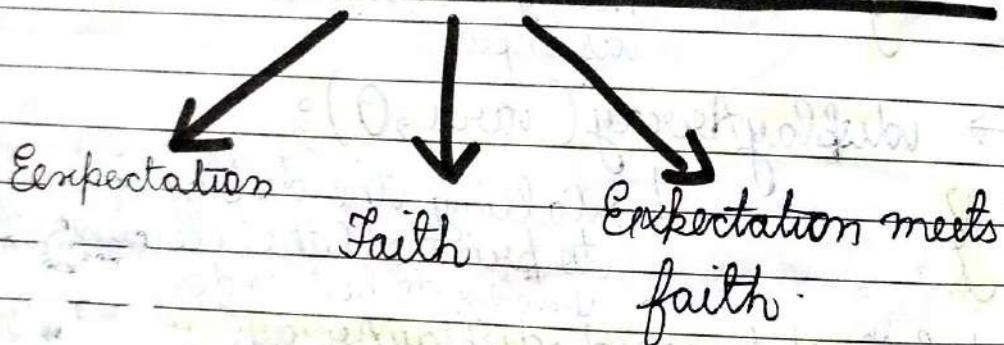
Space Complexity → $O(n)$ → $O(n)$ → for each recursive call parameter store in memory

DISPLAY ARRAY IN REVERSE

पूछा गया कि last question के बीच से यहाँ
 इसमें एक array को print करना है (ट्रिम्पटर array
 के elements को)
 लेकिन यह question में यहाँ twist है कि
 इस array के elements को reverse order
 में print करना है।

→ ऐसी Recursive approach पर code करें।
 तो हमें 2 level of thinking करना होगा।

1. HIGH LEVEL THINKING



a) Expectation अगर हमारे पास एक array arr[] है तो हमारी expectation ये होती है।

b) displayArrayReverse(arr, 0) function arr के सभी elements को display करना है, in the reverse order (if reverse order लिखा होगा)
 last index से ($n-1$) तक रखा होगा 0th index पर।

Example:- $\text{arr}[] = \{10, 20, 30, 40, 50\}$

$\text{displayArrayReverse(arr, 0)} = \begin{cases} 50 \\ 40 \\ 30 \\ 20 \\ 10 \end{cases}$

b) Faith :- ये line मन में बोलो

Page No.	
Date	

I have faith that `displayArrRev(arr, 1)` will display the contents of the array in the reverse order from the last index till the first index.

तो हमने ये मानलिया है, ये विश्वास किया है कि `displayArrRev(arr, 1)` हमें elements display करके देगा from the last index till the index 1

$$arr[] = \{ 10, 20, 30, 40, 50 \}$$

0 1 2 3 4

`displayArrayReverse(arr, 1):` { 50
40
30
20 }

c) Expectation meets Faith :-

→ तो हमारी expectation ये कहती है कि `displayArrayReverse(arr, 0)` function array के element display करेगा from last index to index 0.

→ और हमारी faith ये कहता है कि `displayArrayReverse(arr, 1)` function array के element display करेगा from last index to index 1.

तो How to meet the expectation with faith?

We can first call the `displayArrayReverse(arr, 1)` and then simply print the resultant element i.e `arr[0]`

Expectation

display
Array
Reverse
(arr, 0)

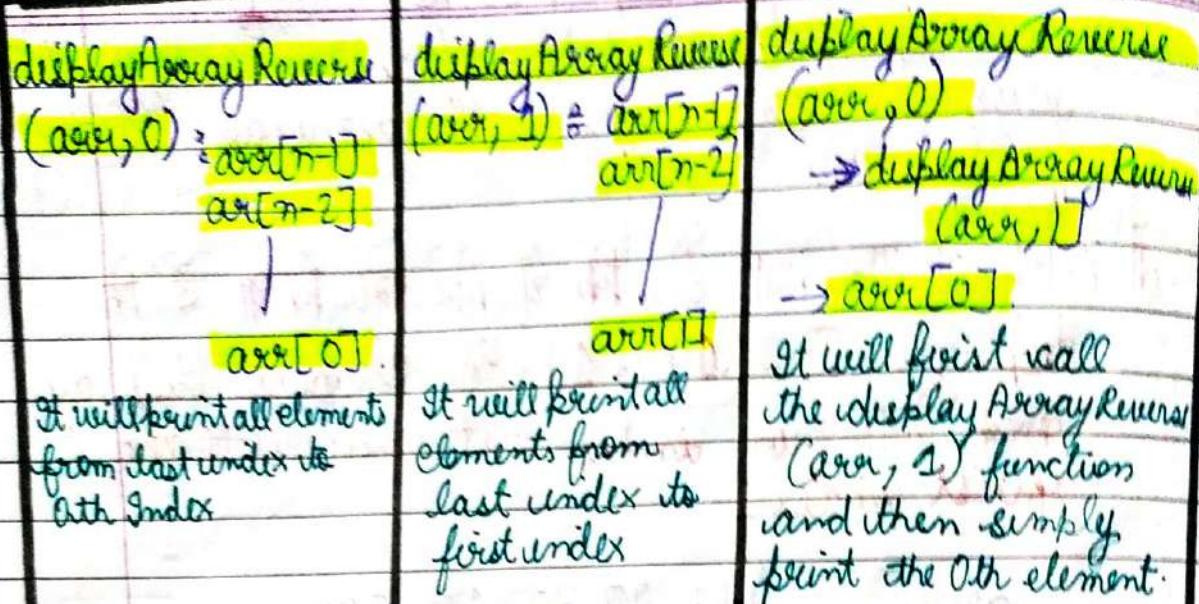
Faith

→ `displayArrayReverse :` { → 10 (arr, 1)

{ 50
40
30
20
10 }

{ 50
40
30
20
10 }

HIGH LEVEL THINKING



ED) High Level Thinking ~~ED~~ & ED code of approach
ED) ED JS & ED

APPROACH

- * We have a recursive formula `displayArrayReverse(arr, idx)`
- 1. Check if `idx` is a valid index.
If not then simply return.
- 2. Make a recursive call with parameters `arr & (idx + 1)`
- 3. Print the value at the index
i.e `arr[idx]`.

(`arr, idx`)

↳ 2 parameters
ED), one will
be array & another
will be ending
index.

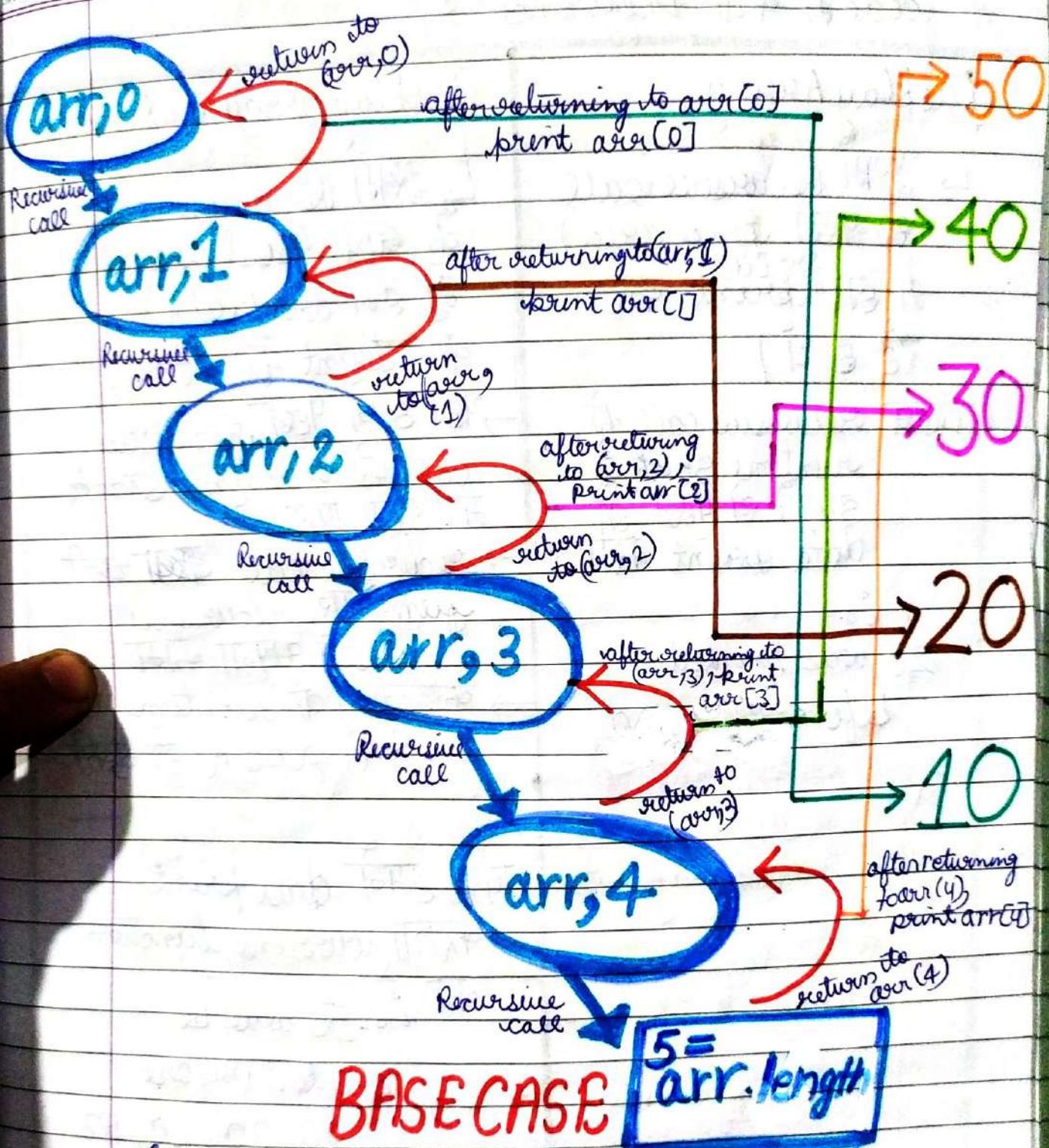
Approach of ED Pseudocode (program of design) ~~ED~~ ~~ED~~

Pseudocode

```
function displayArrayReverse(arr, idx)
if idx == arr.length()
    return
    displayArrayReverse(arr, idx + 1)
    print arr[idx]
```

2. LOW LEVEL THINKING

Page No. _____
Date _____



So, the recursion tree is created while doing low level thinking which is done in low level thinking. So here we found our base case.

So, Low Level Thinking tells us the base case, also gives us clearer understanding of code.

Last Question (Display Array) और इस
Question में (Display Array in Reverse)
के code में क्या Difference है?

Page No.

Date

Display Array

→ इसमें Recursion call
से पहले (pre area)
में arr[0]
में data print कर
रहे थे।

मानो recursion call की
runtime stack पर
stack से पहले ही
data print कर दिया

∴
10 was printed
before 30, 40, 50.

Display Array in Reverse

→ इसमें Recursion call
के बाद (post area)
में arr[0/idx]
की print कर रहे थे।

→ पहले ही recursion
call की runtime stack
में stack, arr की
recursion call की
3TR recursion
call ने 3456 को
कर दिया, तो runtime
stack पर popout हो जाएगा

∴
the main data print
करती ही current function

∴ 50 will be
printed before
40, 30, 20, 10.

Time Complexity → $O(n)$ → because there are n
recursion calls.

Space Complexity → $O(n)$ → for each recursion call,
the parameters, address of the array
are stored in runtime stack & because the
runtime call stack is of n length, it will also take n space in stack.

DISPLAY ARRAY IN REVERSE <Code>

Page No. _____
Date _____

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main {  
→     public static void main (String [ ] args)  
→     {  
→         Scanner scan = new Scanner (System.in);  
→         int n = scan.nextInt();  
→         int arr [] = new int [n];  
→         for (int i = 0; i < n; i++)  
→         {  
→             arr[i] = scan.nextInt();  
→         }  
→         displayArrayReverse (arr, 0);  
→     }  
→ }
```

//Expectation: displayArrayReverse (arr, i) → arr[n-1] — arr[i]
//Faith: displayArrayReverse (arr, i+1) → arr[n-1] — arr[i+1]
//Expectation meets faith: displayArrayReverse (arr, i) →
~~विंडेक्ट एंड एंड ऑट कॉर्ट प्रिंट करो।~~
→ public static void displayArrayReverse
 (int [] arr, int index)
{

```
    if (index == arr.length)  
    {  
        return;  
    }
```

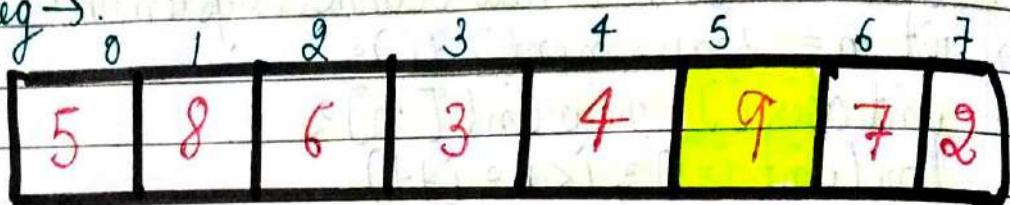
```
    displayArrayReverse (arr, index + 1);  
    System.out.println (arr[index]);  
}
```

underbrace $\frac{\text{index}+1}{\text{end (कहां की)}} \text{array object print object}$

MAX OF AN ARRAY

→ हमें एक array दिया गया है।
हमें largest element find करना है array में।

Eg →



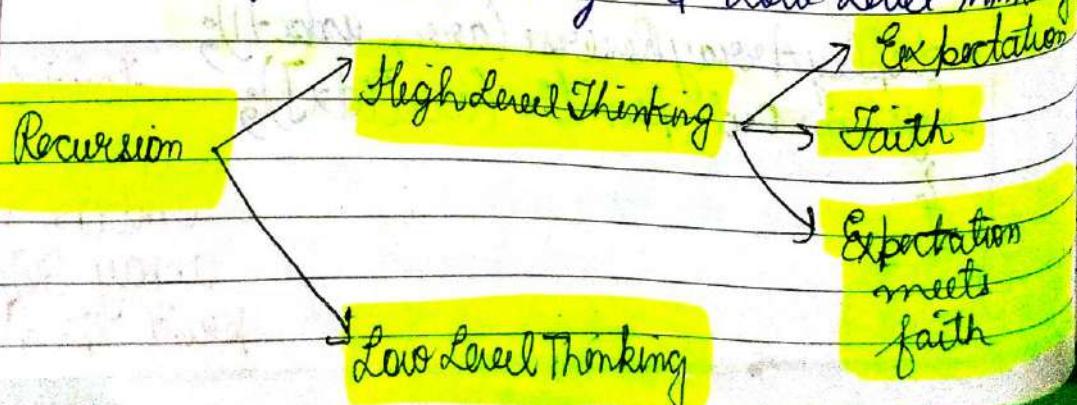
$$\max = 9$$

X इस ques को एक loops के आलानी से solve कर सकते हैं। But वहाँ Time Complexity बहुत ज्यादा हो गी। So, that is a Bad Approach.

✓ हमें इस ques को recursively solve करना है।
इस approach की कम time लगती है as compared to loops.

तो हमें Code में Recursive Calls नहीं हैं।
instead of Loops.

तो हमें 2 Level of Thinking करनी होती है।
- High Level Thinking & Low Level Thinking



1. HIGH LEVEL THINKING

Page No.	
Date	

a) Expectation

We expect that we have to find the $\max(\text{arr}, 0)$ meaning the largest element of the array starting from position 0.

function name base the given array & starting index as parameter
 $\max(\text{arr}, 0) = 9$

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

b) Faith

मेरी faith रखते हैं कि $\max(\text{arr}, 1)$ ।
यदि function index 1 से last index तक के सभी elements की values compare करते हैं तो maximum element return करता है, यह दोनों में से जो overall largest element होगा, return करता है।

मेरी अब क्या 0th index का element का maximum element का compare करना है, इस दोनों में से जो overall largest element होगा, return करता है।



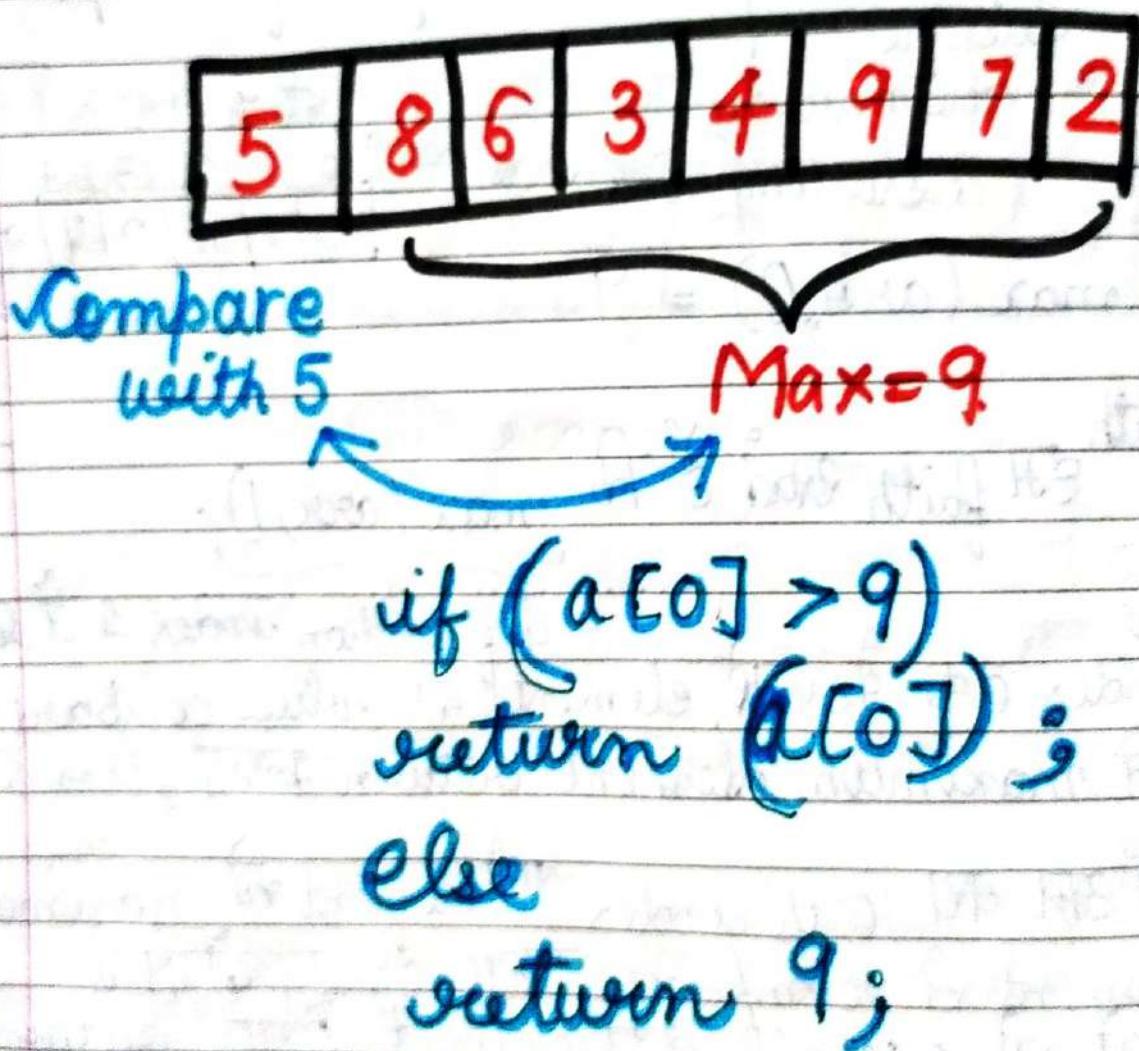
This is $\max = 9$
left to compare.

So, our function was able to find the maximum value from index 1 to the end of array i.e. of smaller array. Now, just compare it with $\text{arr}[0]$ to find out maximum value of the array.

c) Expectation meets path

ईसे लिए अब compare करो।

the maximum element obtained from
the smaller/remaining part of the
array with the first element of array



2. Low Level Thinking

- It's done to find the base case
- Dry run & draw the stack.

Now, we will try to dry run the recursive code.

and then see the levels inside the call stack, then we will try to find the base case for the recursion.

जहां में एक कैसे state में पहुंच पाया जा सकता है अरे में वह स्कैप element था हो, तो वो ही max element होगा अरे array की → eg - $\text{arr}[j = 5]$,

5 will
be the max here

So, $\max(\text{arr}, \text{arr.length}-1)$ is the final call and $i = (\text{arr.length}-1)$ is the base case.

∴ Base Case इह

Low Level Thinking

कैसे प्रैक्टिस
base case इह कि
अरे array में एक ही
element था हो तो
उसका max हो

```
public static int maxOfArray(int[] arr, int idx)
{
    if(idx == arr.length - 1)
        return arr[idx];
    int max = maxOfArray(arr, idx + 1);
    if(arr[idx] > max)
        return arr[idx];
    else
        return max;
}
```

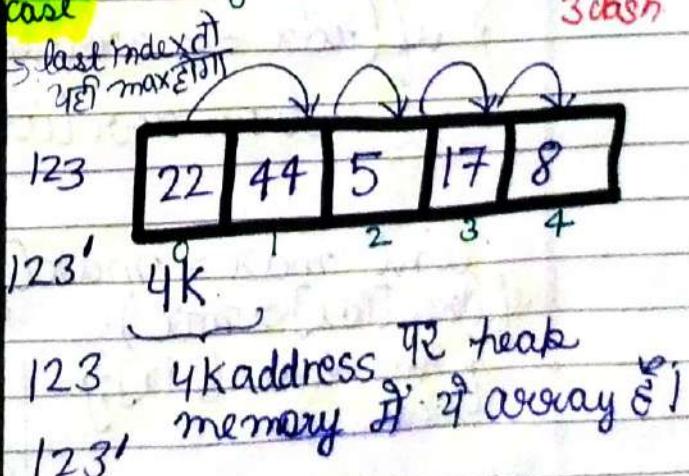
Not such index
in array
e.g. 4 is base case

This code we can write after High

Low Level Thinking

3 dash

maxOfArray	4K	4	8	
maxOfArray	4K	3	8 17	
maxOfArray	4K	2	1 7	
maxOfArray	4K	1	1 4 4	
address	4K	0	44	max.



MAX OF ARRAY <code>

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main  
→ {  
→     public static void main (String [ ] args)  
→     {  
→         Scanner scan = new Scanner (System.in);  
→         int n = scan.nextInt();  
→         int arr [ ] = new int [n];  
→         for (int i = 0; i < n; i++)  
→         {  
→             arr [i] = scan.nextInt();  
→         }  
→         int max = maxOfArray (arr, 0);  
→         System.out.println (max);  
→     }  
→ }
```

Find largest index & get from given starting index till last index

```
→ public static int maxOfArray (int [ ] arr,  
→                               int idx)  
→ {  
→     if (idx == arr.length - 1)  
→     {  
→         return arr [idx];  
→     }  
→ }
```

```
→     int max = maxOfArray (arr, idx + 1);  
→     if (arr [idx] > max) // max is the greatest  
→     {  
→         return arr [idx];  
→     }  
→     else  
→     {  
→         return max;  
→     }  
→ }
```

Comparison of max & arr [idx]

* Time Complexity

The Time Complexity of this solution is $O(n)$. This is because we have iterated the array twice.

Once while going into the recursion and the other time while coming out of it. So, n elements were visited twice.

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

* Space Complexity

The Space Complexity will be $O(1)$ as we have not used any extra space.

If we consider the runtime stack/recursion stack, it will be $O(n)$ i.e. maximum height of the recursion stack.

— Tamanna Verma

HW1

22 Oct, 21

Ques. 1. First INDEX

- हमें एक array given होगा और हमें उसे एक element give होगा।
- हमें वो index return करना है जिसपर give element सबसे पहले occurs हुआ।
- और अगर give element पुरा array traverse करने पर कही भी नहीं मिलता तो -1 return करदा।

Example → element = 6

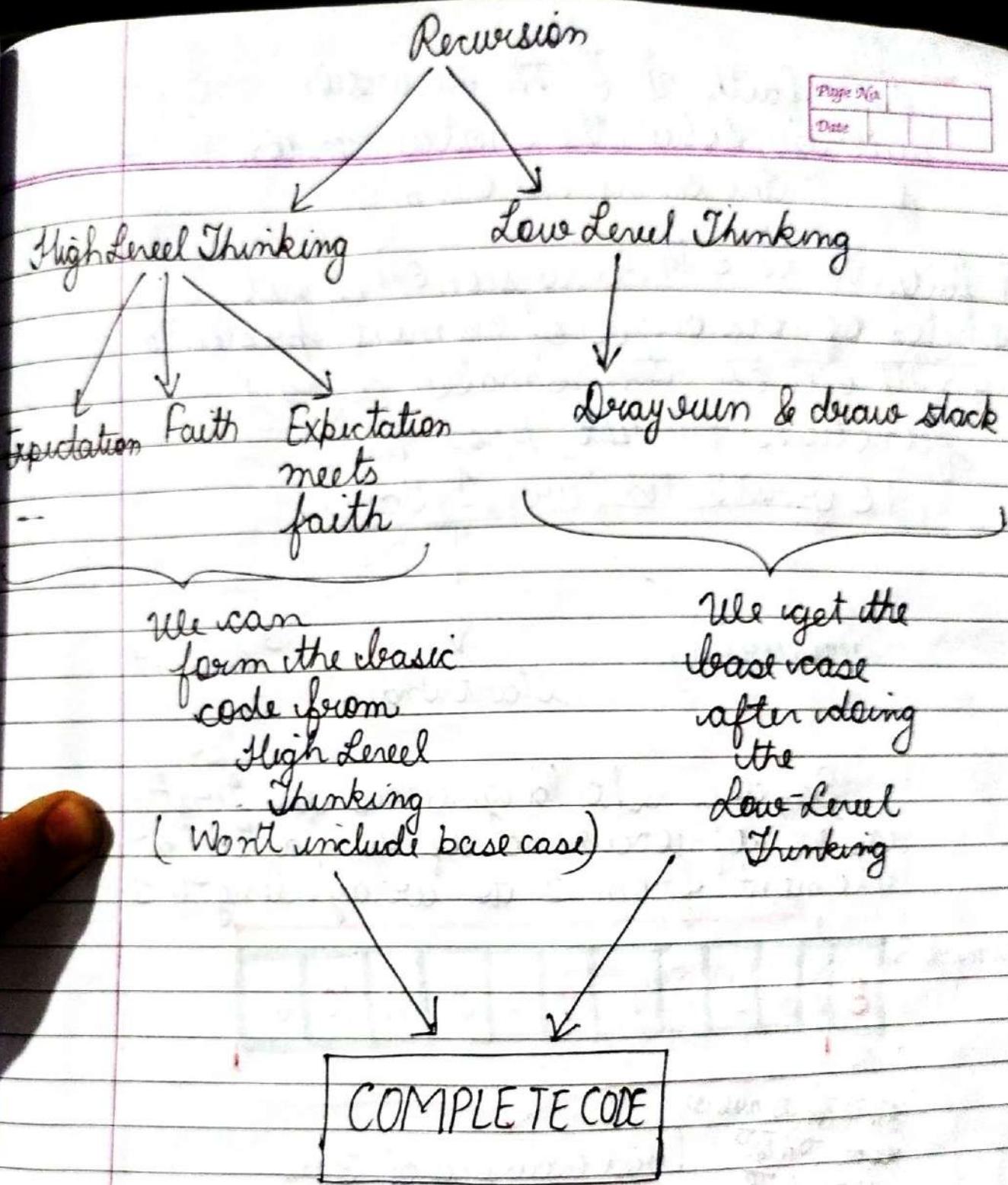
9	8	6	3	8	9	4	7	2	11	10	6	2
0	1	2	3	4	5	6	7	8	9	10	11	12



तो 6 सबसे पहले index 2 पर occurs हुआ। तो हमें 6 return करना है।

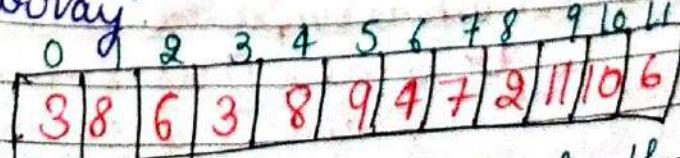
return 2

ये question हमें easily looks लगाकर iteratively भी कर सकते हैं लेकिन हमें recursively करना है जो have the least time complexity.



1. High Level Thinking

- a) Expectation → ~~HT~~ expectations & firstIndex (array)
 will give us the first index of occurrence of element in the array



Start index element return

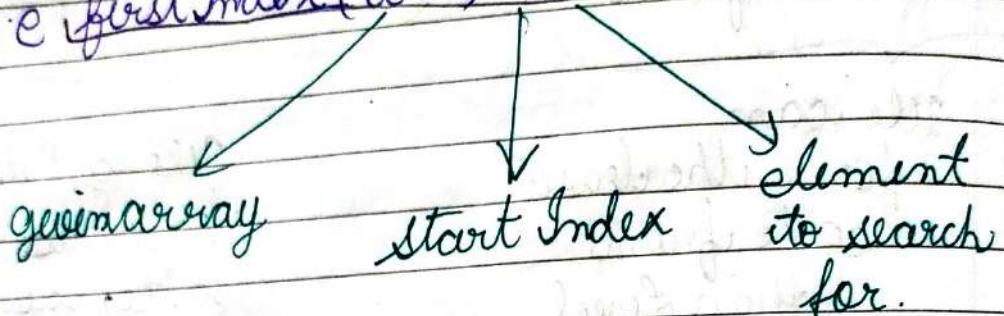
↑ 6 is at index 2 for the first time occurrence

b) Faith

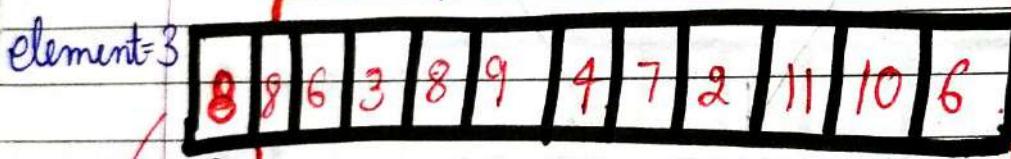
with
EHR faith it is the recursion
will solve the smaller version
of the same problem.
→ it's recursive first

<i>Page No.</i>	
<i>Date</i>	

of the same.
In fact यह है कि Recursion इसके first
index of occurrence of element provide
करना वो है जो smaller array
function को pass करता है।
i.e. `firstIndex(arr, 1, data)`



So, this should give us the first index of occurrence of a particular element from 1 to $\text{array.length} - 1$.



अमरद्वे Transverse
→ अमरद्वे

करते हुए
index 0 पर

first occurrence of 3 is at index 3

first occurrence of 3 is
index 0 पर
मिलने वाला element + at index 3 in smaller array
then return its - 1.

Now, $a[0] = 8$ element

\Rightarrow if comparison

Rec) arr[0] QR check arr, If arr[0] == element then return 0, if not, then make a recursive call for arr[1:n].

In Pre area(before Recursive call) check if the element at current index is equal to given element & then make the recursive call.

(C) Expectation meets fact

* We should compare the value of $\text{arr}[\text{id}x]$ and element and if they are equal, then stop here only and return the $\text{id}x$, otherwise this case,

No need to do the recursive call in this case (data) & if the value of element & $\text{arr}[\text{id}x]$ are not equal, then make the Recursive call. $\text{firstIndex}(\text{arr}, \text{id}x + 1, \text{data})$.

$\text{if}(\text{arr}[\text{id}x] == x)$
 { return $\text{id}x$; }

else { int $\text{fisa} = \text{firstIndex}(\text{arr}, \text{id}x + 1, x)$
 return fisa ; }

So, we are done with High Level Thinking
High Level Thinking from EII code of approach at TGT CTG JSI

APPROACH

We need to find the base case using the Low Level Thinking

$\text{if}(\text{arr}[\text{id}x] == x)$
 { return $\text{id}x$; }

else {

 int $\text{fisa} = \text{firstIndex}(\text{arr}, \text{id}x + 1, x)$;
 return fisa ;

). Code

\rightarrow public static int $\text{firstIndex}(\text{int} \text{arr}[], \text{int} \text{id}x, \text{int} x)$ after

High Level Thinking

This code doesn't consist the base code yet.

2. LOW LEVEL THINKING

→ It helps us to find the base case
 ↳ Dry run & draw the stack

Page No.	
Date	

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

public static int firstIndex (int arr,
 int idx,
 int x).

① [if (arr[idx] == x)
 ↳ return idx;]

else {

② [int fli = firstIndex (arr, idx + 1, x);

③ [return fli;]

base case which will come out ↳ its lee
 i = array.length - 1. Write it in the
 code too to stop the recursion. Else the
 call stack will overflow leading to Run time Error.
 Dry run the code, try to find the index for
 element 3 & 11.

Dry run the code using
 Recursion stack & find out the

base case which will come out ↳ its lee

If element does not
 match to the values
 at any index
 then return -1

firstIndex	4K	arr[1]	3	1	122
firstIndex	4K	0	3	1	122
address	9dx	data to find	∴ fli = 1.		
firstIndex	4K	0	3	1	122

firstIndex	4K	9	11
firstIndex	4K	8	11
firstIndex	4K	7	11
firstIndex	4K	6	11
firstIndex	4K	5	11
firstIndex	4K	4	11
firstIndex	4K	3	11
firstIndex	4K	2	11
firstIndex	4K	1	11
firstIndex	4K	0	11

FIRST INDEX <code>

```
import java.io.*;
import java.util.*;
public class Main
{
    public static void main (String [] args)
    {
        Scanner scn = new Scanner (System.in);
        int n = scn.nextInt();
        int [] arr = new int [n];
        for (int i=0; i<n; i++)
        {
            arr[i] = scn.nextInt();
        }
        int data = scn.nextInt();
        int fi = firstIndex (arr, 0, data);
        System.out.println (fi);
    }
    public static int firstIndex (int [] arr, int index, int x)
    {
        if (index == arr.length)
        {
            return -1;
        }
        if (arr[index] == x)
        {
            return index;
        }
        else
        {
            int fisa = firstIndex (arr, index+1, x);
            return fisa;
        }
    }
}
```

Last Index

- * दो एक array given होता consisting of n
number of elements.
दो एक element x का given होता
दो वो last index find करना है जहाँ x
occur (lie) करता है और return करना है उस
index.

$n = 6 \rightarrow$ Size of array.
 $\text{arr}[] = \{ 5, 7, 4, 2, 1, 4 \}$
 $x = 4$

4 occur at 2 index / position
i.e Index 2 & index 5.

So, because we need to return the last index, so 5 will be the answer & we need to return 5.

Recursion

Low Level Thinking

Expectation Faith Expectation meets faith Dry own & draw
1. HIGH LEVEL THINKING

1. HIGH LEVEL THINKING

1. Expectation

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

Page No.	
Date	

स्थिरी expectation $lI(arr, 0, x)$ से यह है कि यह

0th index से end index तक check करें x की occurrence को और यदि x की last occurrence नहीं होती तो उस index का return करें।

2. Faith

स्थिरी faith, दोनों लाइब्रेरी $lI(arr, i, x)$ से यह है कि

1st index से i th index तक check करें x की occurrence & return the last index

और यदि 1st index से i th index के बीच में x की occurrence नहीं होती तो उस return -1.

3) Expectation meets faith

If the output of $lI(arr, 1, x)$ is -1, then
check at 0th index & if its x , then return 0
else return -1.

If the output of $lI(arr, 1, x)$ is an index i
then no need to check at 0th index, bcz its smaller
index than i .

$lI(arr, 1, x) \boxed{1 \rightarrow \text{end}}$

-1

i

→ return i

$arr[0] = x$

return 0

$arr[0] \neq x$

return -1

2. Low LEVEL THINKING

→ To find the base case
→ Need to

इसे High Level Thinking, नहीं

Day Run & Date

इसे अब basic code form करेंगे

Draw & Stack

* (code में base case included)

जहाँ दोगा तो हमें Low Level Thinking

करके कैसी होगी ऐसे day run

करके base case find करते हैं :-

→ last index in smaller array.

→ int liisa = lastIndex(arr, idx + 1, x); → line 1

→ if (liisa == -1) { → line 2.

 if (arr[idx] == x) → line 2.1

 { return idx; } → line 2.2

 else { return -1; } → line 2.3

}

else {

 return liisa;

}

→ line 3

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

lI	4K	9	1,2,2,1,2,3
lI	4K	8	1,2,2,1,2,3
lI	4K	7	1,2,2,1,2,3
lI	4K	6	1,2,2,1,2,3
lI	4K	5	1,2,2,1,2,3
lI	4K	4	1,2,2,1,2,3
lI	4K	3	1,2,2,1,2,3
lI	4K	2	1,2,2,1,2,3
lI	4K	1	1,2,2,1,2,3
lI	4K	0	1,2,2,1,2,3

This will be the base case when $idx \geq arr.length$

Then we need to return -1

Element to find = 3

∴ 7 is the last index.

LAST INDEX <CODE>

Page No. _____
Date _____

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main {  
    public static void main (String [] args) {  
        Scanner scan = new Scanner (System.in);  
        int n = scan.nextInt();  
  
        int arr = new int [n];  
        for (int i=0; i<arr.length; i++) {  
            arr[i] = scan.nextInt();  
        }  
  
        int x = scan.nextInt();  
        int li = lastIndex (arr, 0, x);  
        System.out.println(li);  
    }  
  
    public static int lastIndex (int [] arr, int udx, int x) {  
        if (udx == arr.length) {  
            return -1;  
        }  
        else {  
            if (lastIndex (arr, udx+1, x) != -1) {  
                return udx; }  
            else {  
                return -1; }  
        }  
    }  
}
```

- Tamanna Verma