Day 45 → Doubt Class → 21 Oct, 2021.

DAY 46

DSA

Recursion In Arrays

- Programming isn't about what you know, its 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 to 2 types of Thinking

→ इसे 2 Level पर सोचना 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 नाम का

तो displayArray(arr, 0)

arr → given array

0 → starting index

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

Expectation → displayArray(arr, 0):

	index
10	0
20	1
30	2
40	3
50	4

2. Faith :-

Expectation से just छोटा faith build करते है

हम array का size तो छोटा बड़ा नहीं कर सकते क्योंकि array का size तो fixed होता है तो हम array को start index की help से छोटा print करा सकते है, हम startindex को आगे बढ़ाते चलेंगे।

हम faith रखेंगे कि (startingindex + 1) से last index तक का array अपने आप print हो जाएगा।

ये Line अपने मन में बोलो

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

विश्वास रखो कि ये function अपना काम करके देगा।

कैसे करेगा? → बादमें सोचेंगे लेकिन अभी के लिए विश्वास रखो कि होगा ऐसा

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

previous starting index +1 (0+1)

displayArray(arr, 1) :

	index
20	1
30	2
40	3
50	4

array और starting index pass करने है as the function argument तो array length (5) से कम length (4) के element print करा पाएंगे using starting index.

मतलब अगर पहली recursive call 4 element print करने के लिए है तो next recursive call 3 element print करने के लिए होगी & so on, हर बार एक कम element print करने के लिए होगी next recursive call

eg → अगर पहली recursive call (1,2,3,4,5) (all) (last 5 element) print करने के लिए है तो next recursive call (2,3,4,5) (last 4 element) print करने के लिए होगी & so on

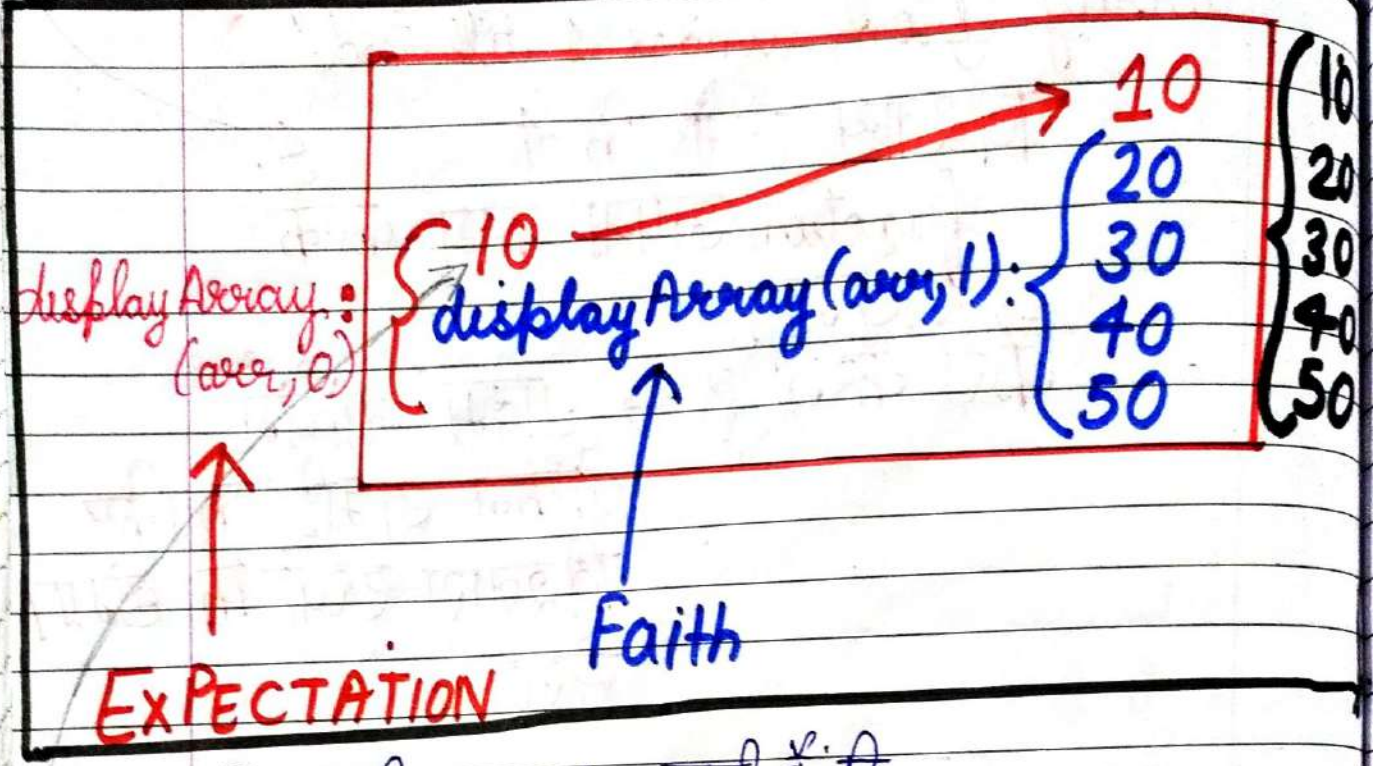
starting index = 0

starting index = 1

ये करने के लिए starting index +1 करते चलो

3. Expectation with Faith

Page No.
Date



→ So, हमारी expectation कहती है कि

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

→ और हमारा faith कहता है.

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 1th 0 the Index. ourselves and leave the rest on the faith.

Basically इस example में हम बस 10 print कर रहे है खुदसे और हम faith रखेंगे कि 20,30,40,50 को function displayArray function अपने आप print करदेगा तो finally we get 10 20 30 40 50

तो हमने High Level Thinking करली है।
अब हमें अपने code की approach
पता चल गई है। **HIGH LEVEL THINKING**

Page No. | Date

Expectation	Faith	Expectation meets faith
displayArray (arr, 0) { arr[0] arr[1] arr[2] ... arr[n-1] } It will print all element from index 0 to last index n = arr.length	displayArray (arr, 1) { arr[1] arr[2] arr[3] arr[4] ... arr[n-1] } It will print all the elements from index 1 to last index here n = arr.length	displayArray (arr, 0) : arr[0] displayArray (arr, 1) we have to print the first element (at 0th index by ourselves, rest will be taken care by displayArray function

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()   ] ये होगा हमारा Base Case
    return
    print arr[idx]           ] Self Work.
    displayArray(arr, idx+1) ] ये recursive call है।
```

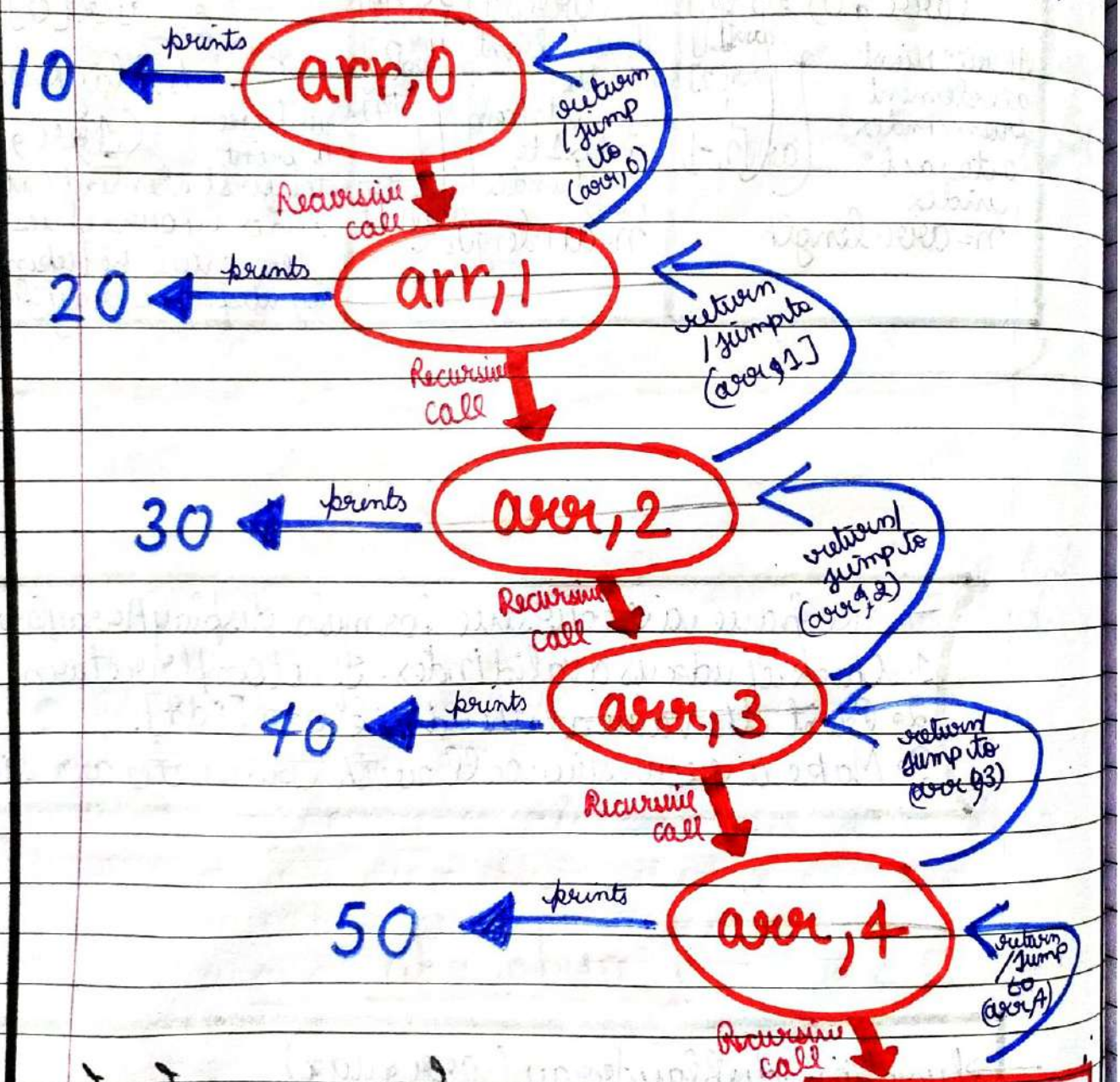
LOW LEVEL THIN KING

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।

↳ 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();                    // Taking the length of array as input
        int arr[] = new int[n];                    // Initializing an array of Integers of length n.
        for (int i=0; i<n; i++)
        {
            arr[i] = scan.nextInt();               // Taking the value at ith 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)
        if (i == arr.length)                       // if base case
        {
            return;
        }
        System.out.println(arr[i]);                // self work
        displayArray(arr, i+1);                    // recursive call
    }
}
```

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

Time Complexity → O(n) → bez n recursive call

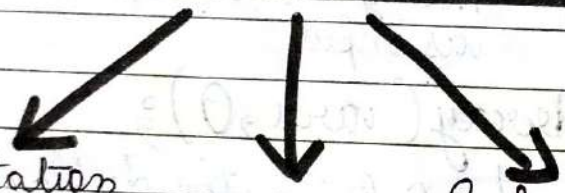
Space Complexity → O(n) → O(n) → for each recursive call parameter store in memory

Ques. 2.

DISPLAY ARRAY IN REVERSE

↳ ये question भी last question के जैसा है थोड़ा इसमें भी array को print करना है (मतलब array के elements को)
लेकिन इस question में एक twist ये है कि इसे array के elements को reverse order में print करना है।
→ हमें Reursive approach से code करना है तो हमें 2 level पर thinking करनी होगी।

1. HIGH LEVEL THINKING



a. Expectation — अगर हमारे पास एक array arr[] है तो हमारी expectation ये होगी कि displayArrayReverse(arr, 0) function arr के सारे elements को display करेगा in the reverse order (ये reverse order उल्टा होगा last index से (n-1) और खत्म होगा 0th index पर)
↳ n मतलब arr.length

index → 0 1 2 3 4

Example:- arr[] = {10, 20, 30, 40, 50};

displayArrayReverse(arr, 0) = 50 40 30 20 10

b) Faith :- ये line मन में बोलो

↓

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). हमे 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 i.e array के element display करेगा from last index to index 0.

→ और हमारा 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 current element i.e arr[0]

Expectation
↓
displayArrayReverse(arr, 0)

Faith
↓
displayArrayReverse(arr, 1)

10 → 10

50
40
30
20
10

HIGH LEVEL THINKING

displayArrayReverse (arr, 0)	displayArrayReverse (arr, 1)	displayArrayReverse (arr, 0)
arr[n-1] arr[n-2] ⋮ arr[0]	arr[n-1] arr[n-2] ⋮ arr[1]	→ displayArrayReverse (arr, 1) → arr[0]
It will print all elements from last index to 0th index	It will print all elements from last index to first index	It will first call the displayArrayReverse (arr, 1) function and then simply print the 0th element.

हमने High Level Thinking करली है तो हमें code की approach का चल गई है।

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].

↳ 2 parameters होते हैं, one will be array & another will be ending index.

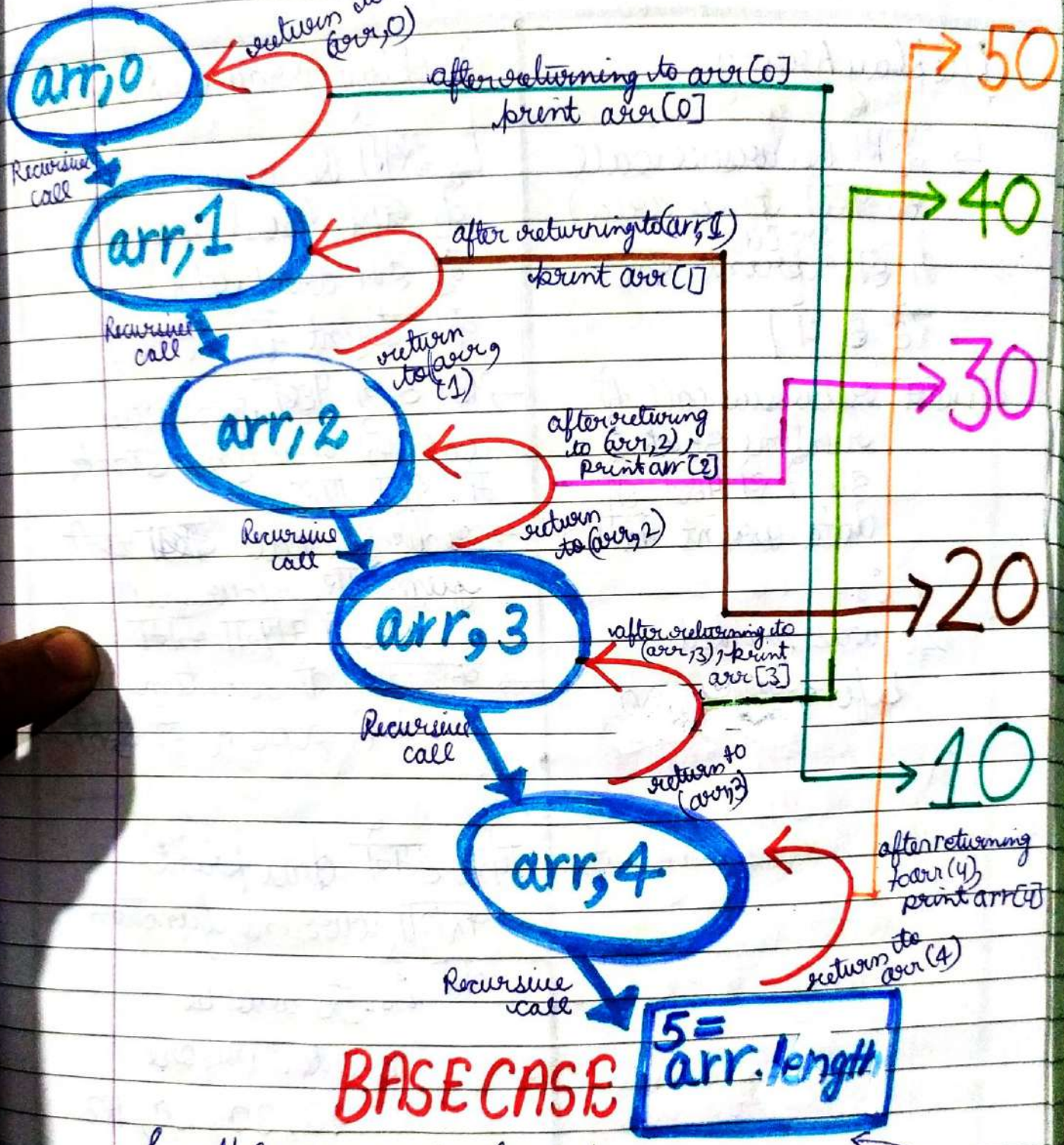
Approach से हम Pseudocode (program का design) लिख सकते हैं।

Pseudocode → Program/code का design होता है।

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

2. LOW LEVEL THINKING

Page No.
Date



So, this recursion tree is created while doing dry run which is done in low level thinking & 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 Recursive call से पहले (pre area) में हम arr[idx] print कर रहे थे।

मतलब recursive call को runtime stack में डालने से पहले ही data print करा दिया

∴ 10 was printed before 20, 30, 40, 50.

Display Array in Reverse

↳ इसमें Recursion call के बाद (post area) में हम arr[idx] को print कर रहे थे।

→ यहाँ हमने पहले recursive call को runtime stack में डाला, फिर वो

→ recursive call यहाँ और recursive call ने अपना काम करा दिया, तो runtime stack से pop out हो चुकी है,

फिर हमने data print कराया current function का ∴ 50 will be printed before 40, 30, 20, 10.

Time Complexity → O(n) → because there are n recursive 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>

```
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) →

```
displayArrayReverse
(arr, i+1);
arr[0];
```

दी index से end तक को उल्टा print करना है।

```
public static void displayArrayReverse
                    (int [] arr, int idx)
{
  if(idx == arr.length)
  {
    return;
  }
  displayArrayReverse(arr, idx+1);
  System.out.println(arr[idx]);
}
}
```

index+1 से end तक को array उल्टा print करते।

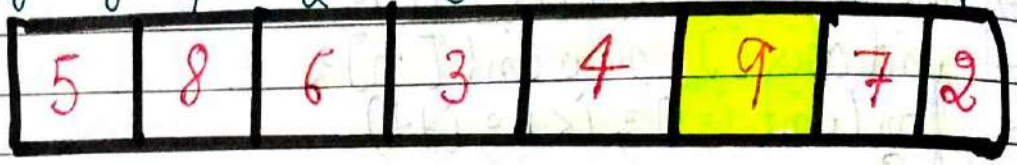
Ques 3

MAX OF AN ARRAY

Page No.
Date

→ हमें एक array given है।
हमें largest Element find करना है array में।

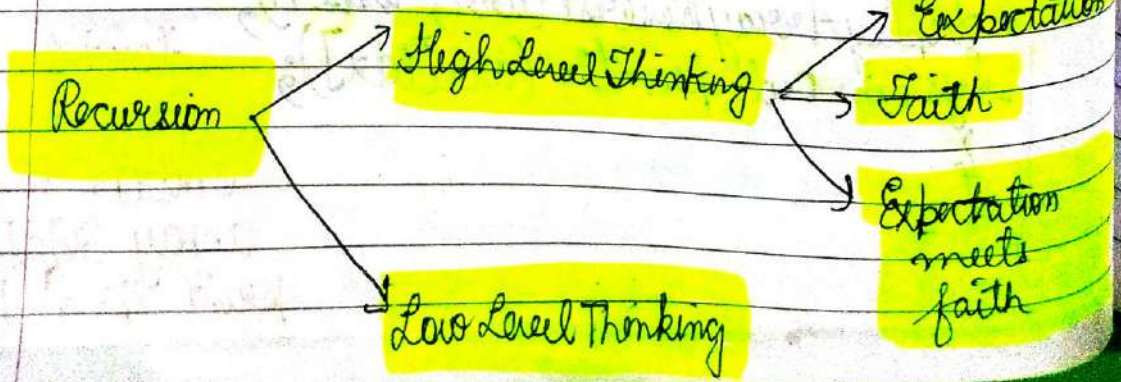
Eg →



max = 9.

✗ इस ques को हम Loops लगाके आसानी से solve कर सकते But वहाँ Time Complexity बहुत ज्यादा होगी So, that is a Bad Approach.

✓ हमें इस ques को recursively solve करना है।
ये approach काफी कम time लेगी as compared to loops.
तो हमें code में Recursive Calls लगानी है instead of Loops.
तो हमें 2 Level पर Thinking करनी होगी
– High Level Thinking & Low Level Thinking



1. HIGH LEVEL THINKING

Page No. | Date

a) Expectation

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

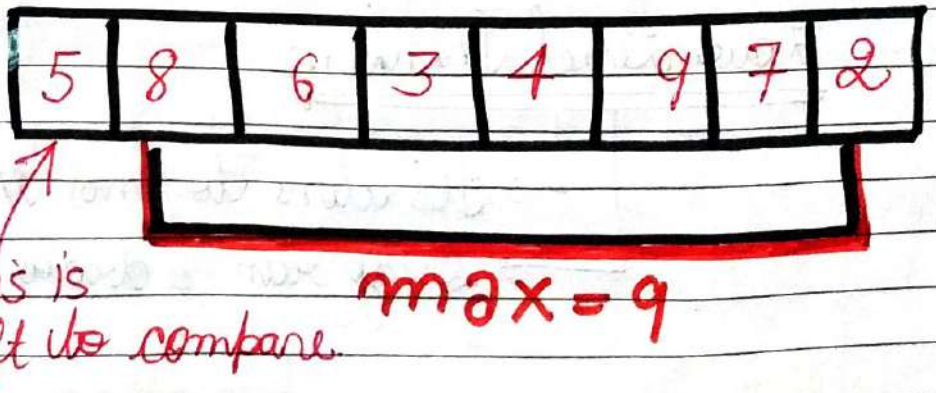
function name — pass the given array & starting index as parameter

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

max(arr, 0) = 9

b) Faith

हम faith रखते हैं कि max(arr, 1). ये function index 1 से last index तक के सारे elements की values compare करके हमें maximum element return करेगा, This is faith. हमें अब बस 0th index वाले element को maximum(arr, 1) element से compare करना है, इन दोनों में से जो overall बड़ा element होगा, return that.

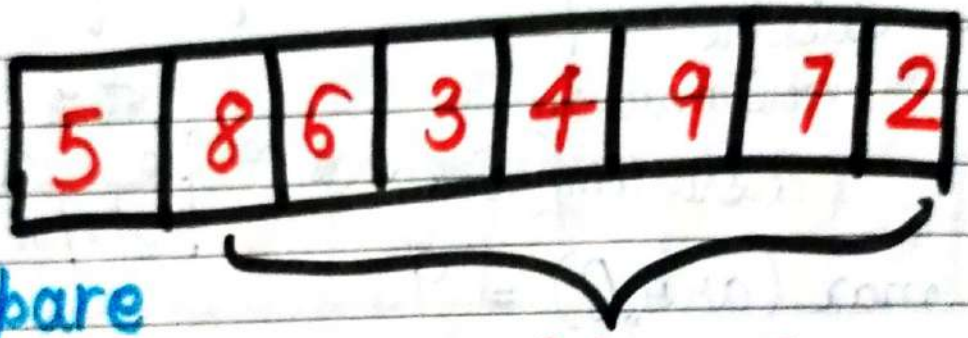


So, our function was able to find the maximum value from index 1 to the end of array i.e of smaller array. Now, just compare it with arr[0] to find out maximum value of the array

c) Expectation meets faith

हमें इस तरह compare करना है

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



Compare with 5

Max=9

```
if (a[0] > 9)
return (a[0]);
else
return 9;
```

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

अगर हम एक ऐसे state में पहुँच जाए जब array में बस एक element बचा हो, तो वो ही max element होगा उस array का → eg → int arr[] = {5};

5 ... will be the max here

So, max(arr, arr.length-1) is the final call and i=(arr.length-1) is the base case.

ये Base Case हमें Low-Level Thinking करने पर मिला

Base Case ये है कि अगर array में एक ही element बचा है तो वो max है

```
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; }
```

Line ①
Line ②
Line ③
Line ④

This code we can write after High Level Thinking

3 calls

Not such index in array so 4 is base case

maxOfArray	4K	4	8	
maxOfArray	4K	3	8 17	123
maxOfArray	4K	2	17	123'
maxOfArray	4K	1	44	123
	4K	0	44	123'
	address	i	max 44	

last index तो यही max होगा

22	44	5	17	8
	1	2	3	4

4K

4K address पर heap memory में ये array है।

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);
}
}
```

// largest index वाला 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)
  { return arr[idx];}

else {return max;}
}
```

// max is the greatest element from idx+1 to end

Comparision of max & arr[idx]

* Time Complexity

Page No.
Date

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

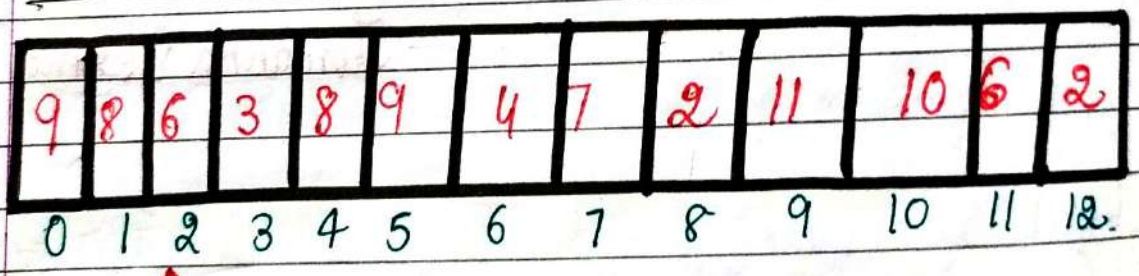
22 Oct, 21

HW1

Ques 1. First INDEX

हमे एक array given होगी और हमे एक element given होगा।
हमे वो index return करना है जिसपर given element सबसे पहले occur हुआ।
और अगर given element पुरा array traverse करने पर कही भी नही मिला तो -1 return करदो।

Example → element = 6



तो 6 सबसे पहले index 2 पर occur हुआ तो हमे 6 return करना है।

return 2

ये question हम easily loops लगाकर iteratively भी कर सकते है लेकिन हमे recursively करना है to have the least time complexity

Recursion

High Level Thinking

Low Level Thinking

Expectation

Faith

Expectation meets faith

Dry run & draw stack

We can form the basic code from High Level Thinking (Won't include base case)

We get the base case after doing the Low-Level Thinking

COMPLETE CODE

1. High Level Thinking

a) Expectation → START expectation & firstIndex(arr, n, key)

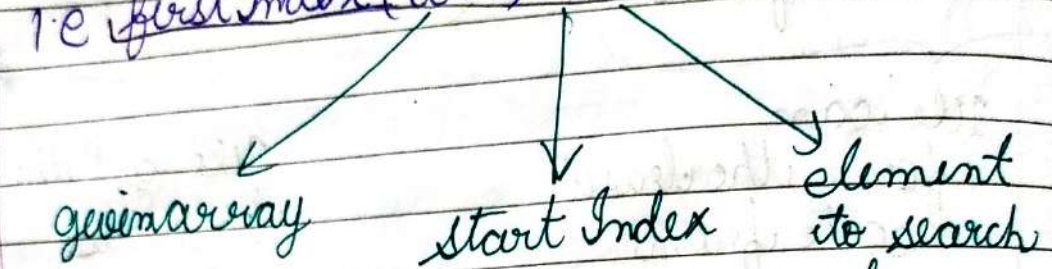
will give us the first index of occurence of element in the array

b) Faith

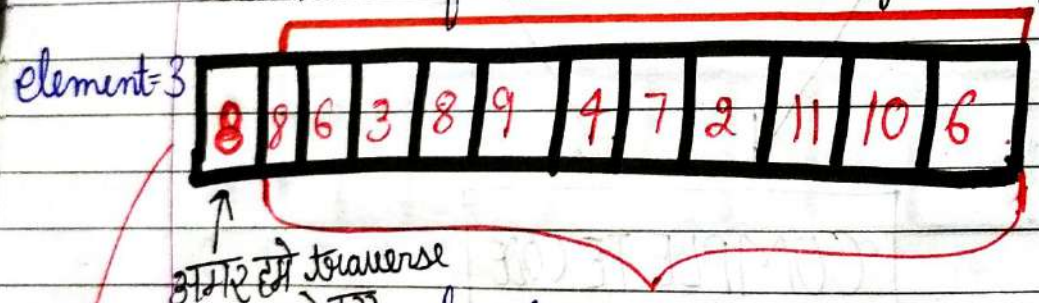
हमारा faith यह है कि recursion will solve the smaller version of the same problem.

हम faith रख रहे हैं कि recursion हमें first index of occurence of element provide करेगा तथा हम smaller array function को pass करेंगे.

i.e firstIndex(arr, 1, data);



So, this should give us the first index of occurence of a particular element from 1 to array.length-1.



अगर हम traverse करते हुए index 0 पर मिलगा element then just return its index.

first occurence of 3 is at index 3 in smaller array

Now, a[0] = 8 ≠ element

पहले arr[0] पर check करो, अगर arr[0] == element then return 0, if not, then make a recursive call firstIndex(arr, 1, data);

↳ ये comparison pre area में करेंगे

∴ 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 faith

* We should compare the value of arr[idx] and element and if they are equal, then stop here only and return the idx, then in this case,

No need to do the recursive call in this case

& if the value of element (data) & arr[idx] are not equal, then make the Recursive call. firstIndex(arr, idx+1, data).

```
if(arr[idx] == x)
{ return idx; }
else { int fiisa = firstIndex(arr, idx+1, x)
       return fiisa;
}
```

So, we are done with High Level Thinking
High Level Thinking में हमें code का approach तो पता चल गई।

APPROACH

```
public static int firstIndex(int arr[], int idx, int x)
{
    if(arr[idx] == x)
    { return idx; }
    else { int fiisa = firstIndex(arr, idx+1, x);
           return fiisa;
    }
}
```

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

Code obtained 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

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

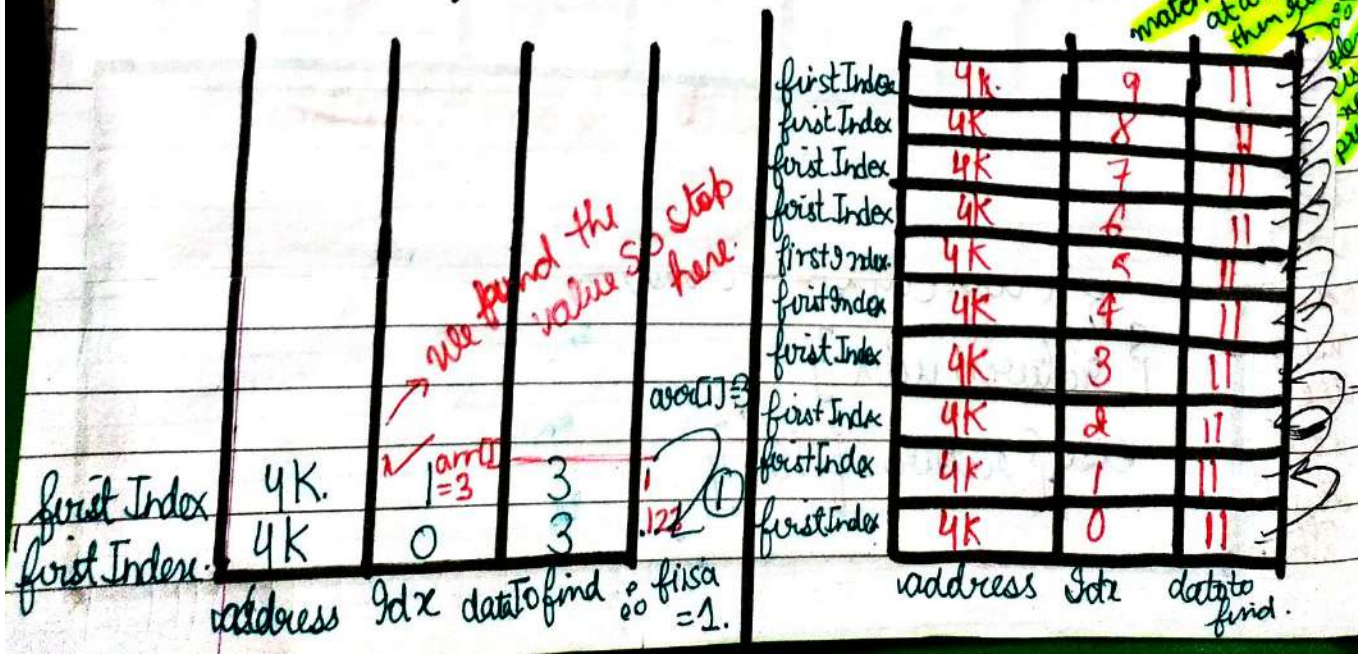
```
public static int firstIndex(int[] arr,
                             int idx,
                             int x)
{
    ① if (arr[idx] == x)
        { return idx; }
      else {
    ② int fiisa = firstIndex(arr, idx+1, x);
    ③ return fiisa;
      }
}
```

No base case found by doing High Level Thinking. We need to find it by doing dry run.

Dry run the code using Recursion stack & find out the base case which will come out to see i = array.length - 1. Write it in the code too to stop the recursion. Else the call stack will overflow leading to RunTimeError.

Dry run the code, try to find the index for element 3 & 11.

If element does not match to the values at any index then return -1 i.e. element is not present



FIRST INDEX <code>

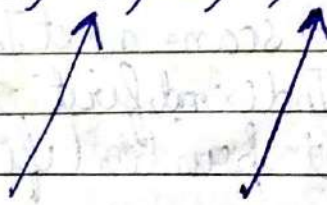
```
import java.io.*;
import java.util.*;
public class Main
{
public static void main (String [] args)
  {
   Scanner scn = new Scanner(System.in);
   int n = scn.nextInt();
   int [] arr = new int[n];
   for(int i=0; i<n; i++)
   {
   arr[i] = scn.nextInt();
   }
   int data = scan.nextInt();
   int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

→ first index in smaller array

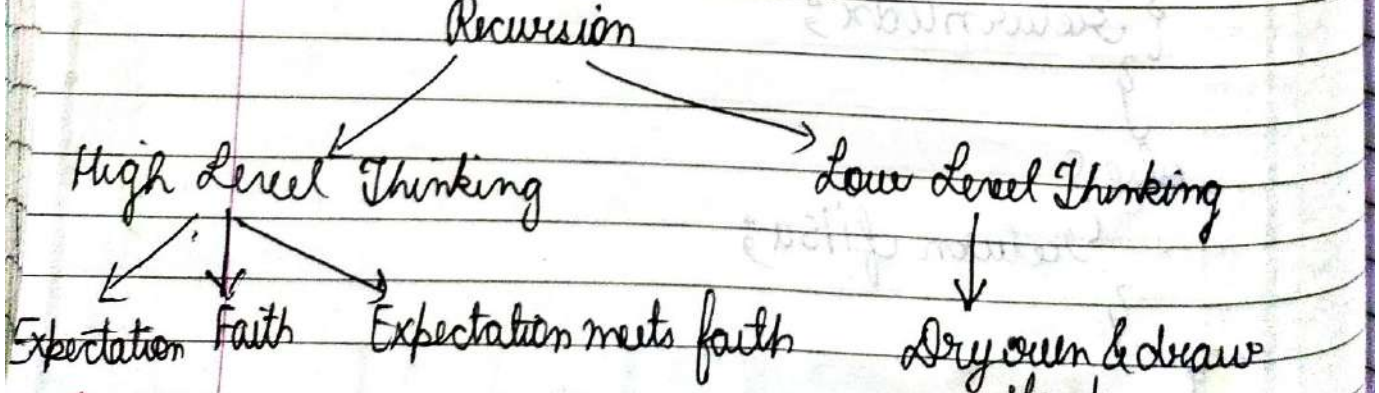
HW 2 Last Index

* हमें एक array given होगा consisting of n number of elements.
हमें एक element x भी given होगा
हमें तो last index find करना है जहाँ x occur (lie) करता है और return करना है तो index.

n = 6 → Size of array
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.



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

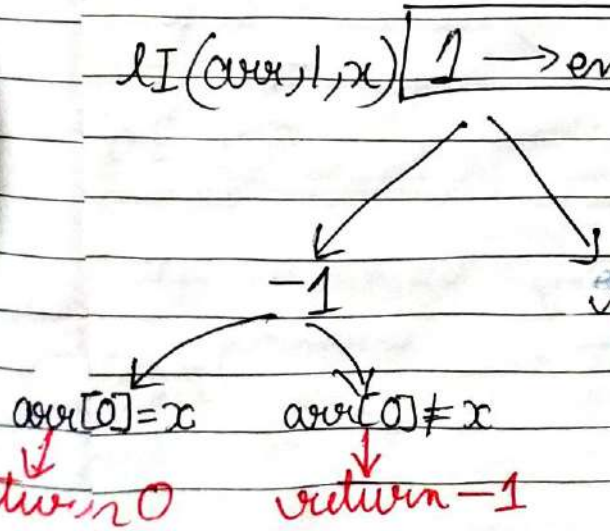
हमारी expectation lI(arr, 0, x) से ये है कि वो 0th index से end index तक check करे x की occurence को और जो x की last occurence मिले, उस index को return करे।

2. Faith

हमारा faith होगा कि lI(arr, 1, x) से ये है कि 1st index से 9th index तक check करो x की occurence & return the last index और अगर 1st index से 9th index के बीच में x की occurence नहीं हुई तो it returns -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 idx i then no need to check at 0th index, bcz its smaller index than i.



2. Low LEVEL THINKING → To find the base case → Need to Dry Run & Draw Stack.

Page No.
Date

हमने High Level Thinking, तो हम अब basic code लिख रखते हैं (code में base case included नहीं होगा तो हमें Low Level Thinking करके देखनी होगी और dry run करके base case find करना होगा :-

```
// 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

	arr	idx	x	
lI	4K	9	3	1,2,2.1,2.3
lI	4K	8	3	1,2,2.1,2.3 → -1
lI	4K	7	3	1,2,2.1,2.2 → -1
lI	4K	6	3	1,2,2.3 → 7
lI	4K	5	3	1,2,3 → 7
lI	4K	4	3	1,2,3 → 7
lI	4K	3	3	1,2,3 → 7
lI	4K	2	3	1,2,3 → 7
lI	4K	1	3	1,2,3 → 7
lI	4K	0	3	1,2,3 → 7

This will be the base case when idx >= 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 idx, int x)
    { if (idx == arr.length)
       {return -1;
       }
int liisa = lastIndex (arr, idx+1, x);   // lastindex in smaller area.
 if (liisa == -1)
    { if (arr[idx] ==x)
      { return idx ;}
      else
      { return -1; }

    } else { return liisa};
}
}
```

– Tamanna Verma