

# Doubt Solving Session

Course on C-Programming & Data Structures: GATE - 2024 & 2025

# Data Structure: Doubts & Array 2

By: Vishvadeep Gothi



---

*Hello!*

# I am Vishvadeep Gothi

I am here because I love to teach

---

# Vishvadeep Gothi: Profile

---

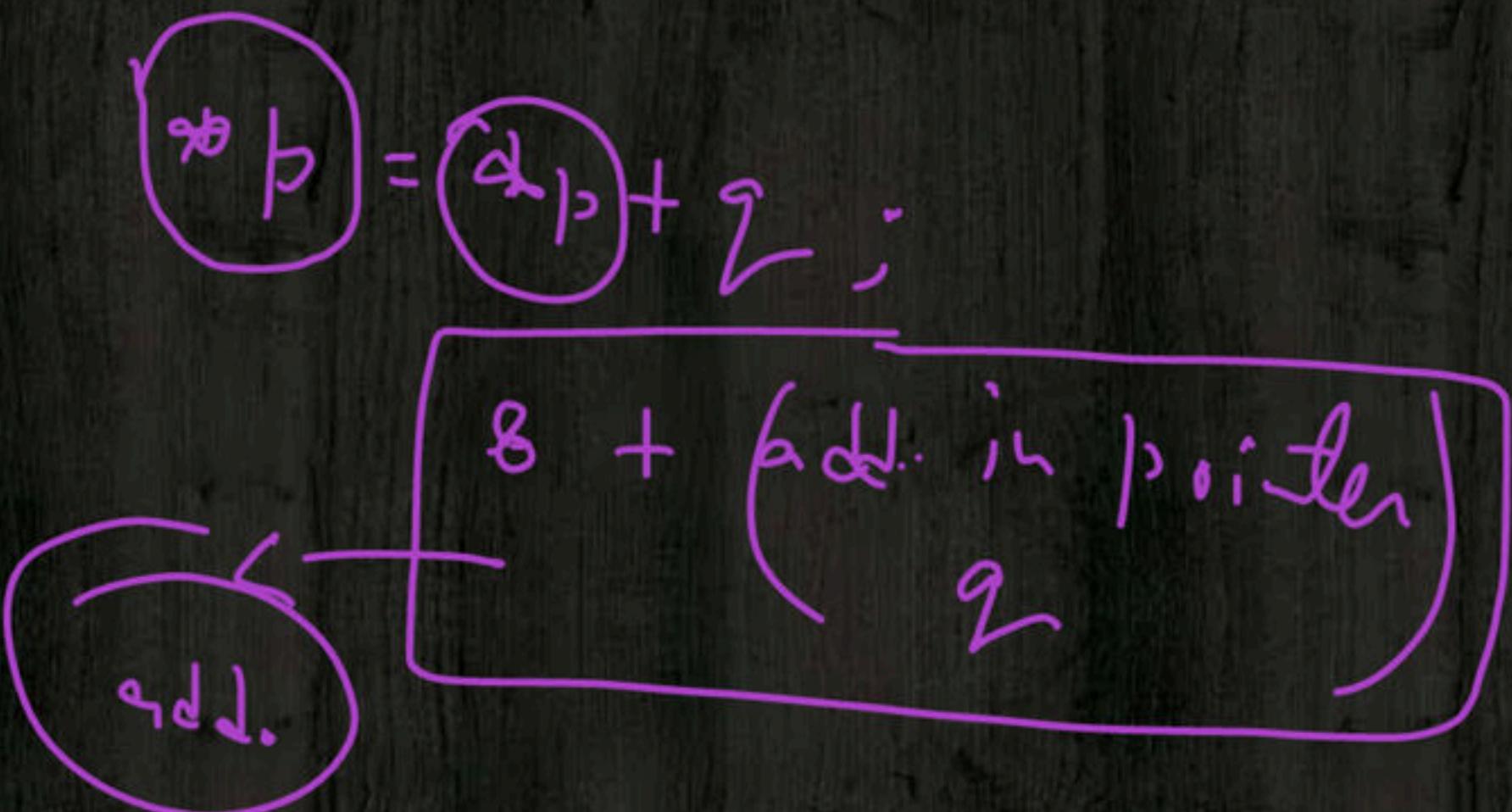
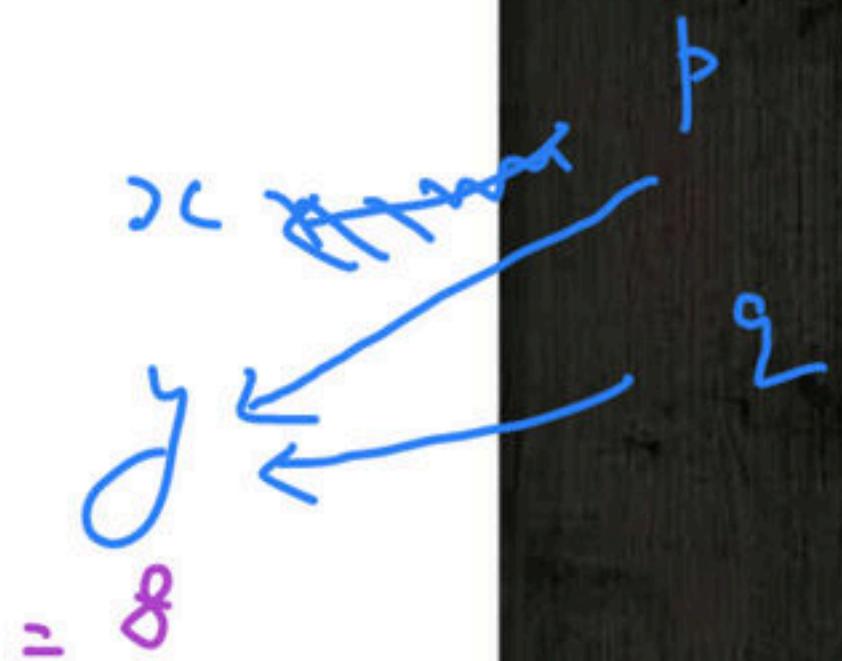
- **GATE Ranks:**
  - 682 (2009) – 3<sup>rd</sup> year
  - 19 (2010) – 4<sup>th</sup> year
  - 119, 440 etc.
- **Education:**
  - ME from IISc Bangalore
  - Mtech from BITS-pilani in Data Science
- **Work:**
  - 13 Year Teaching Experience
  - 9+ in GATE/IES (GateForum, Gate Academy, ACE)
  - Worked in Cisco, Audience Communication
- **Professions:**
  - Freelance S/W developer
  - Educator
  - CrossFit Trainer

# Question

$\& p =$

The value printed by the below program is:

```
#include <stdio.h>
void find (int *p, int *q){
    p=q;
    *p=*p + q; ← error
}
void main (){
    int x=5, y=8;
    find (&x, &y);
    printf("%d", x+y+1);
}
```



$x = 4$

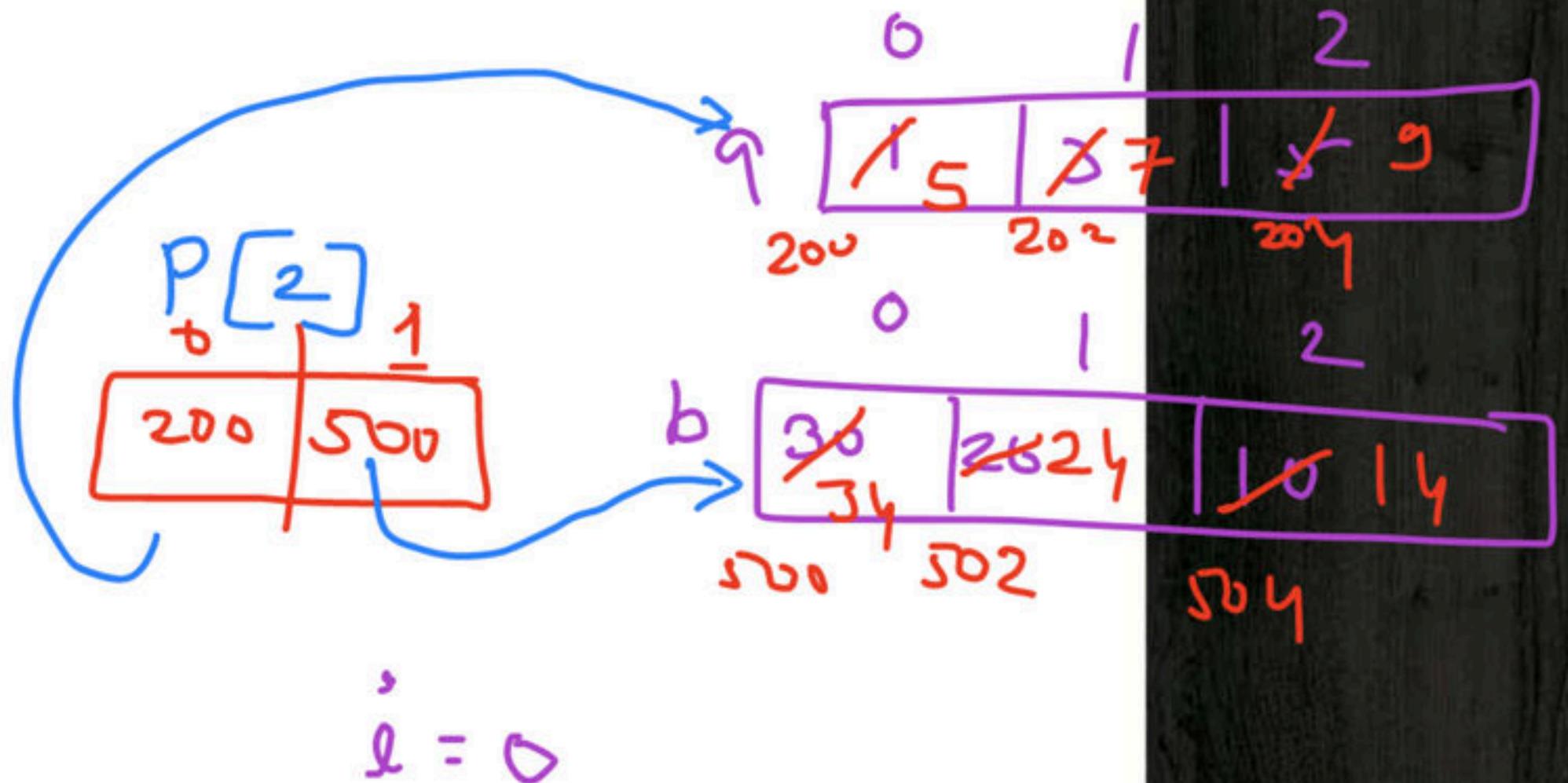
# Question

Value returned by below C-language function when called for f(4)?

```

int f(int x){
int a[3]={1,3,5};
int b[3]={30,20,10};
int i=0;
int *P[2];
P[0]=a;
P[1]=b;
for(i=0;i<3;i++){
P[0][i] += x;
P[1][i] += x;
}
return *P[0]+*P[1];
}

```



$$5 + 34 = 39$$

39

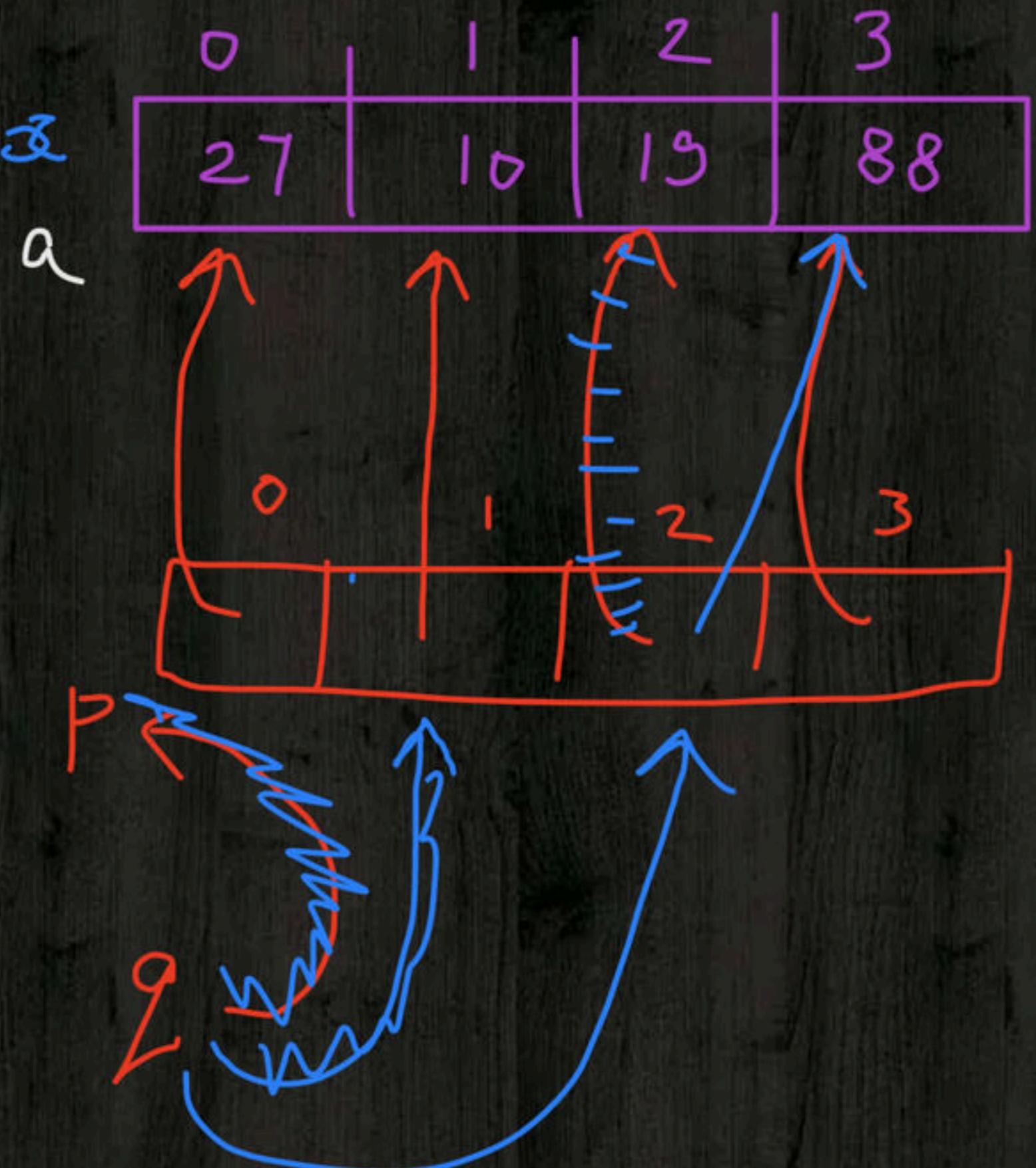
# Question

The output of the following C-program is?

```
#include <stdio.h>

void f(int a[]){
    int *p[]={a,a+1,a+2,a+3};
    int **q;
    q=p;
    q++;
    *(++q);
    ++(*q);
    printf("%d",**q);
}

void main(){
    int x[]={27,10,19,88};
    f(x);
}
```



# Question

The output of the following C-language program:

```
#include<stdio.h>
void fun (char *x, char *y) {
    char *str;
    str=x;
    x=y;
    y=str;
}
void main(){
    char *p="Vishvadeep";
    char *q="Gothi";
    fun(p,&q);
    printf("%s, %s", p, q);
}
```

~~p<sup>q</sup>~~ → "Vishvadeep"  
~~p<sup>q</sup>~~ → "Gothi"

fun ( b , &q );

Gothi, Vishvadeep

# Question

```
struct test{  
    int x;  
    float y;  
};  
void main{  
    struct test *s[10];  
}
```

The variable s here is defined as:

An array, each element of which is a structure of type test

A structure of 2 fields, each field being a pointer to an array of 10 elements

✓ An array of size 10, each element of which is a pointer to a structure of type test

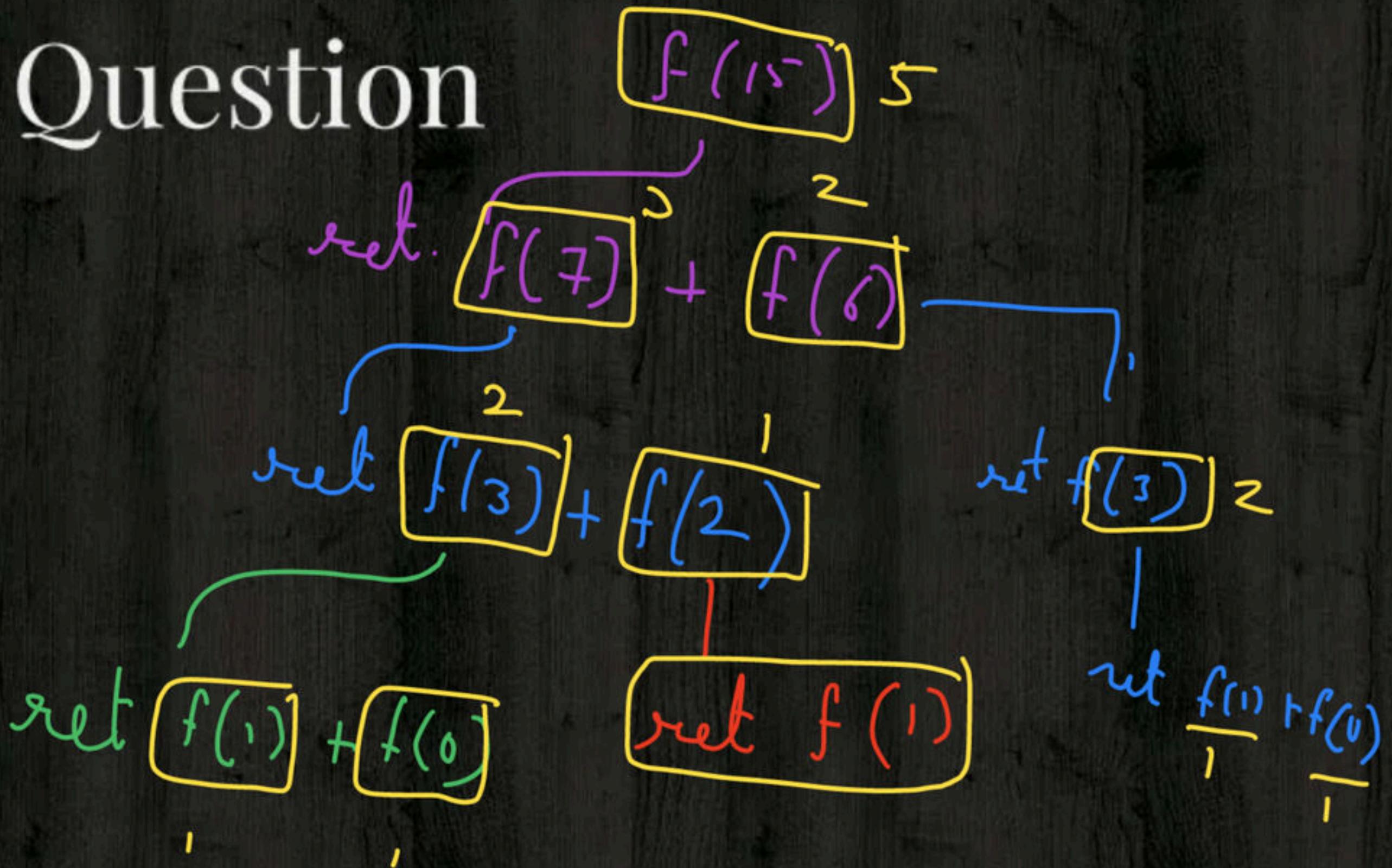
None

Ans = 5

# Question

The output printed by below program is:

```
#include<stdio.h>
int fun(int n){
if(n<=1) return 1;
if(n%2 != 0) return fun(n/2) + fun(n/2-1);
else return fun(n/2);
}
void main()
{
printf("%d", fun(15));
}
```



# Array

- ◆ Collection of homogeneous elements
- ◆ Characteristics:
  1. All elements stored on consecutive memory locations
  2. All elements can be accessed using a set of indexes

Array

$A[LB:UB]$

$A[LB \dots UB]$

$A[3:9]$

Array

$A[1 \dots 5]$

$B[-2:15]$

$B[-2 \dots 12]$

# Location of an Array Element

$$\text{Address}(A[i]) = \text{Base} + w * (i - LB)$$

↓ relative index

# Why Indexing from Zero?

Indexing from zero is a common convention in many programming languages and data structures. It has several advantages:

- It provides a natural way to represent sequences and arrays.
- It allows for more compact and efficient memory representation.
- It simplifies many mathematical operations and algorithms.
- It follows a consistent convention across many different domains and applications.

However, it can also lead to some potential pitfalls and confusion, particularly for new programmers. For example, it requires careful indexing when working with arrays or lists, and it can make it easier to make mistakes when working with large datasets.

In conclusion, while indexing from zero may not always be the most intuitive choice, it is a powerful and widely used convention in computer science and data processing. By understanding its strengths and weaknesses, we can better appreciate its importance and use it effectively in our own work.

# Traversing in Array

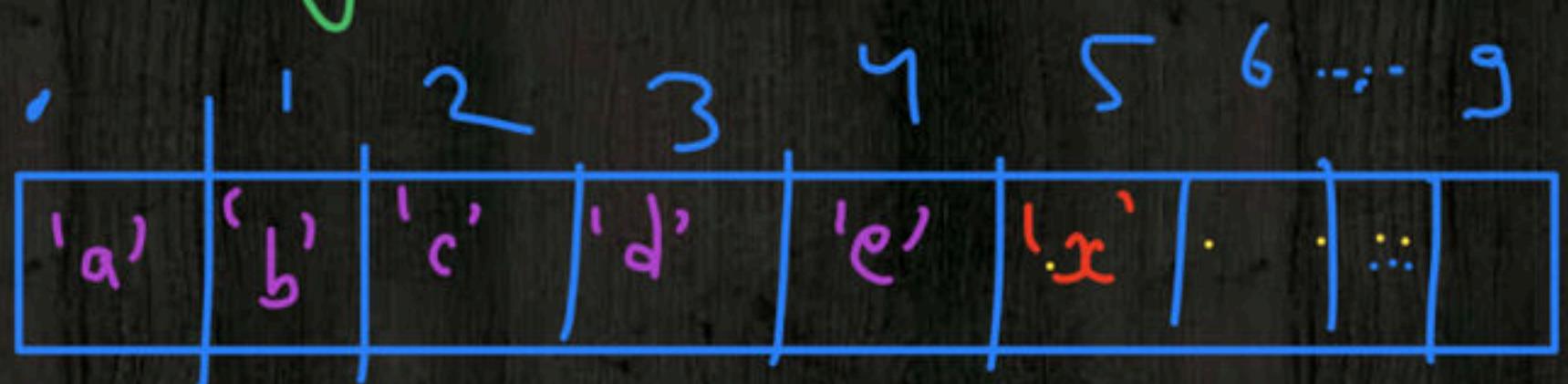
for  $k = LB$  to  $UB$ , step by 1  
process  $A[k]$

R.T. complexity =  $\Theta(n)$

Space complexity = constant

# Insertion in Array

always at the end



LB = 0

UB = 45

n = 56

if ( LB == array - UB )

{ printf (" overflow ");

return;

UB ++;

A[UB] = item; n ++;

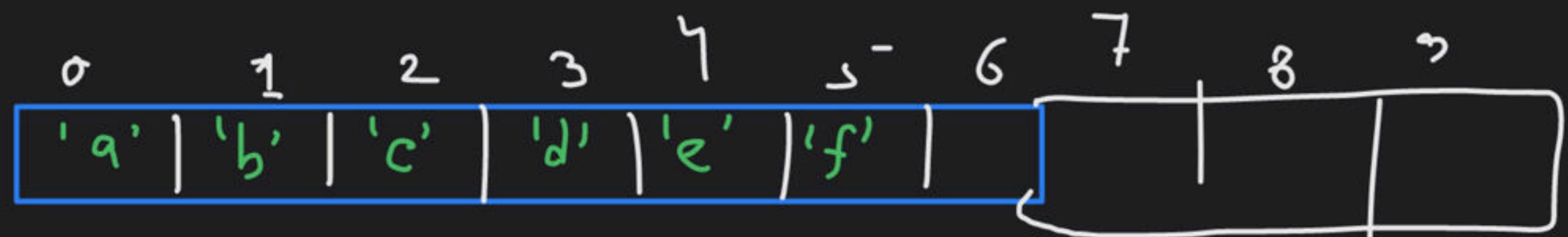
based on index

R.T.  
complexity  
= Constant

Based on index:-

Insertion ( $A[ ]$ , LB, UB, n, item, i)

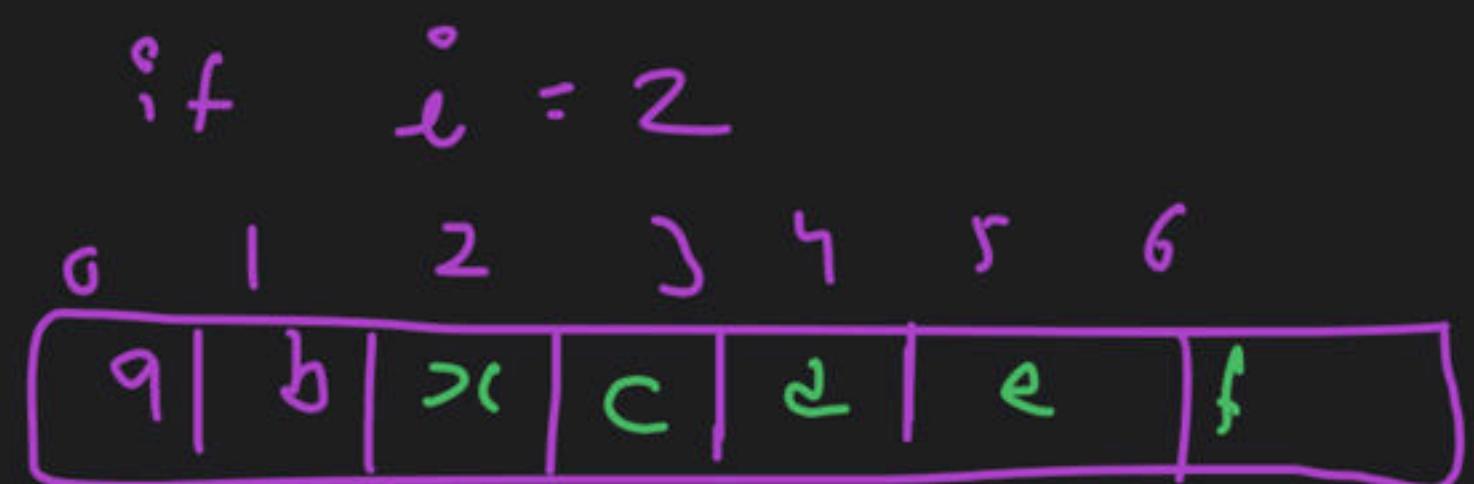
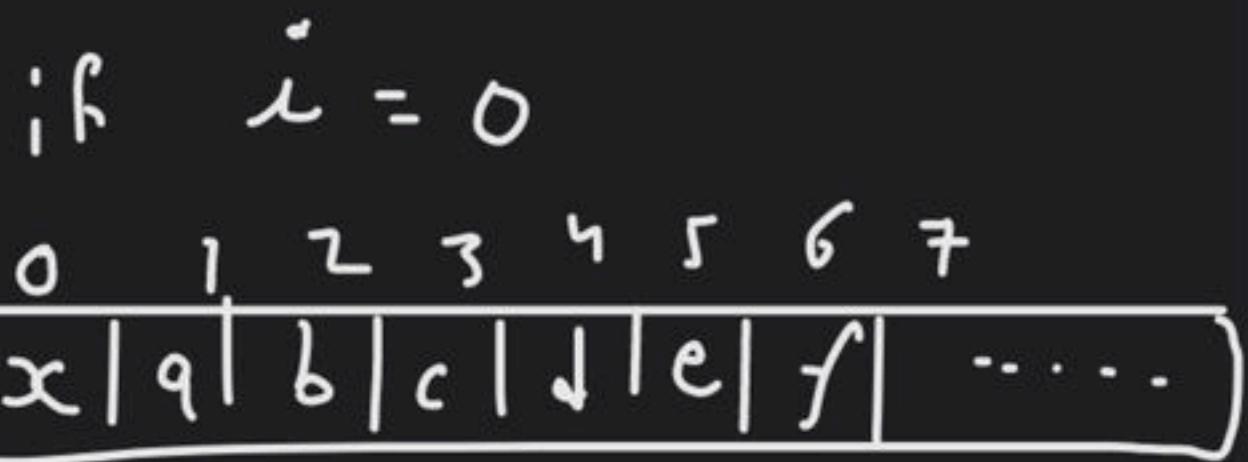
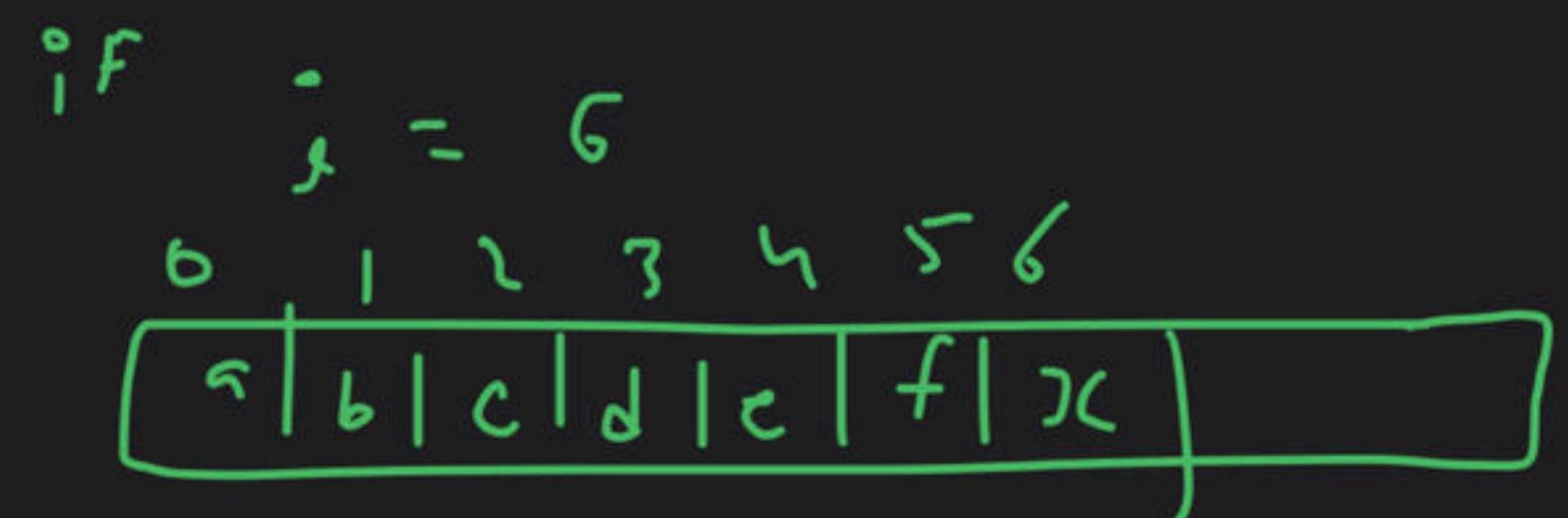
new element to be inserted on index i.



$n = 6$   
 $LB = 0$   
 $UB = 5$

item = 'x'

$LB \leq i \leq UB + 1$



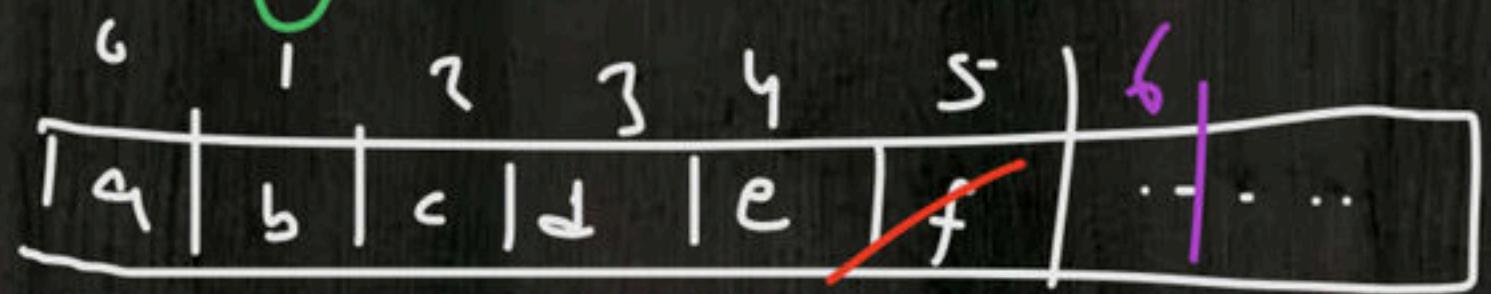
```
if (UB == array-UB)
{
    printf("overflow");
    return;
}
if (i < LB || i > (UB+1))
{
    printf("Invalid index i");
    return;
}
for (k = UB; k >= i; k--)
{
    A[k+1] = n[k];
}
```

```
A[i] = item;
UB++;
n++;
```

Run time complexity =  $O(n)$   
space complexity =  $O(1)$

# Deletion in Array

always last element



$$LB = 0$$

$$UB = 4$$

$$n = 5$$

```
if (UB == -1)
{
    printf("Underflow")
    return;
}
```

UB--;

n--;

based on index

R, T. Complexity =  $\Theta(1)$   
Space -  $\Theta(1)$

Based on index

0	1	2	3	4	5	6	-	-	-
q	b	c	d	e	f	g			

index      i =>

$$LB \leq i \leq UB$$

{ deletion (A[], LB, UB, n, i)

$$i = 2$$

q	b	d	e	f	g	
---	---	---	---	---	---	--

}

```
if (UB == -1)
{
    printf ("Underflow");
    return ;
}
if (i < LB || i > UB)
{
    printf ("invalid index");
    return ;
}
for (k = i; k < UB; k++)
{
    A[k] = A[k+1];
}
UB--;
n--;
```

R.T. Complexity =  $O(n)$

Space - ll =  $\Theta(1)$

# Finding Minimum Element

Given an array of integers, find the minimum element.

Example: Given array [3, 5, 1, 2, 4], the minimum element is 1.

Approach: Traverse the array from left to right and keep track of the minimum element found so far.

Time Complexity: O(n), where n is the number of elements in the array.

Space Complexity: O(1), as we are using only a single variable to store the minimum element.

Implementation:

```
function findMinElement(arr) {
    let minElement = arr[0];
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < minElement) {
            minElement = arr[i];
        }
    }
    return minElement;
}
```

Analysis:

# Finding Maximum Element

# Finding Minimum & Maximum Both

# Question GATE-2014

The minimum number of comparisons required to find the minimum and the maximum of 100 numbers is \_\_\_\_\_?

# Finding Second Minimum

Given a sequence of  $n$  elements, find the second minimum element.

Input: An array of integers  $A = [a_1, a_2, \dots, a_n]$ .

Output: The second minimum element in the array.

Example:

- Input:  $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
- Output:  $2$

Approach:

- Sort the array  $A$  in ascending order.
- Return the second element of the sorted array.

Time Complexity:  $O(n \log n)$  due to sorting.

Space Complexity:  $O(1)$  if we ignore the space required for sorting.

# Searching in Array

1. Linear Search
2. Binary Search

# Linear Search

Search for a value in an array.

Time complexity:  $O(n)$ .

Space complexity:  $O(1)$ .

Implementation:

```
function linearSearch(arr, target) {
```

```
    for (let i = 0; i < arr.length; i++) {
```

```
        if (arr[i] === target) {
```

```
            return i;
```

```
    }
```

```
    return -1;
```

```
}
```

Example:

```
const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

```
const target = 5;
```

```
const index = linearSearch(arr, target);
```

```
console.log(index); // Output: 5
```

# Binary Search

# Question

Consider a sorted array of size  $n$  with duplicate elements. You have been given an element  $k$ , what is the time complexity to find the frequency of element  $k$  in the array?

- (A)  $O(1)$
- (B)  $O(\log n)$
- (C)  $O(n)$
- (D) None

# Question

Consider a sorted array of size  $n$  with duplicate elements. You have been given an element  $k$ , what is the time complexity to find that the element  $k$  is appeared atleast  $n/3$  times or not?

- (A)  $O(1)$
- (B)  $O(\log n)$
- (C)  $O(n)$
- (D) None

# Question

Consider a sorted array of size  $n$  with duplicate elements. Find the time complexity to find whether an element is appeared more than  $n/2$  times or not in the array?

- (A)  $O(1)$
- (B)  $O(\log n)$
- (C)  $O(n)$
- (D) None



*DPP*

# Question 1

---

Consider an array of size  $n-1$  elements which contains elements between 1 to  $n$  (all unique) but 1 number is missing. Find the missing number?

## Question 2

---

Consider an array A[-6 ..... 15], which is stored in memory starting from location 1000. Assume each element in memory is stored on 4 locations. Then the location of element A[2] is?

# Question 3

---

There are  $n$  students in the class and the teacher is going to find the leader of the class. The teacher applies a simple strategy to find the leader as follows:

Teacher asked all students to stand in a single queue then given them numbering from 1 to  $n$ , starting from first student to last student. Then teacher has removed all students with odd numbering i.e. 1, 3, 5 ....

For remaining students teacher given numbering once again from 1 without changing their relative order and then removed the odd number students again from the queue. Teacher repeated the same till there is only 1 student remaining in the queue. That one student becomes leader of the class.

Assume you want to be the leader then in the initial queue of  $n$  students on which position you will stand so that you can become the leader?

---

# Happy Learning



---