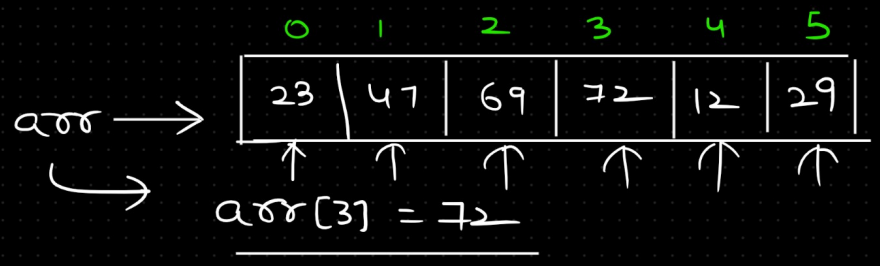
Searching Algorithm



arr[3] = 72

time complexity = O(1)

arr.index(12)

↳ 4

unsorted array

↳ Linear Search

target = 12

output = 4

↳ index number

target = 30

output = -1

time complexity ⇒ O(n)

Linear time Complexity

```
for i in range(n):
    if arr(i) == target:
        return i
return -1
```

Space complexity ⇒ O(1)

** Interviews ← Binary Search

Practical use cases

Logarithmic Time Complexity

→ O(log n)

Constraint

"Sorted data" ??

Right side of mid

Left side of mid

n = 8

0	1	2	3	4	5	6	7
2	4	8	12	27	45	49	52

target = 45

mid = ~~3~~ 5

start = ~~0~~ 4

end = 7

```
while (start <= end):
    mid = (start + end) // 2;
```

O(1)

mid = start + (end - start) // 2 — better

O(1)

```
    if (arr(mid) == target):
        return mid
```

① 12 < 45

```
    else if (arr(mid) < target):
        start = mid + 1
```

②

```
    else
        end = mid - 1
```

③

output: 5

target = 4

mid = ~~3~~ 1

↳ start = 0

↳ end = ~~4~~ 2

1

mid

Note:

↳ Reducing the search space //2 in every iteration

time complexity ⇒ O(log n)

Space complexity ⇒ O(1)

IM → (0 + IM)/2

1st

(start + end) //2

↳ bigger term

Overflow

avoid

Start + (end - start)//2

2nd

Ternary Search → Youtube

Ternary → $\log_3 n$

Binary → $\log_2 n$

→ three parts

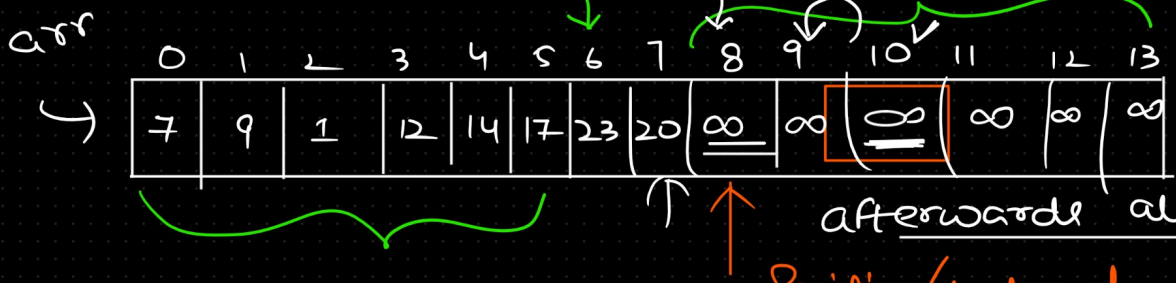
→ $O(\log_3 n)$

Hypothetical Question (Logically)

→ Binary Search

Output = 8

n = 14



afterwards all are infinite

Position/index of first-infinite value

Linear Search

```
for i in range(n):
    if arr(i) == "∞"
        return i
return -1
```

O(n)

Brute force Approach

Binary Search (Modified)

Approach

① Sort ⇒ $n \log n$

② Binary search ⇒ $\log n$

} $O(n \log n)$

7

target = ∞ start = 0 end = ~~13~~ 9

mid = ~~6~~ ~~10~~ 8

Output = 8 ← if (arr(mid) != target)

right side

↳ start = mid+1

arr(mid-1) == target && ↘

elif (arr(mid) == target)

end = mid - 1

Whether it's a first infinite index or not

O(log n)

Interviews → Bit Manipulation

0/1 — Bit

Byte — 8 bits

Competitive Programming

Truth Table

1 — set

0 — unset

AND, OR, NOT, XOR

AND

A	B	O/P
1	0	0
0	0	0
0	1	0
1	1	1

OR

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	1

NOT → NOT 1 = 0

XOR

A	B	O/P
0	0	0
0	1	1
1	0	1
1	1	0

$x \wedge x = 0$

$0 \wedge x = x$

Left Shift & Right shift

Decimal → Binary

5 → 101

$1*2^0+0*2^1+1*2^2$

$1+0+4=5$

2 | 5		
2 | 2	1	
1	0	

$(101)_2$

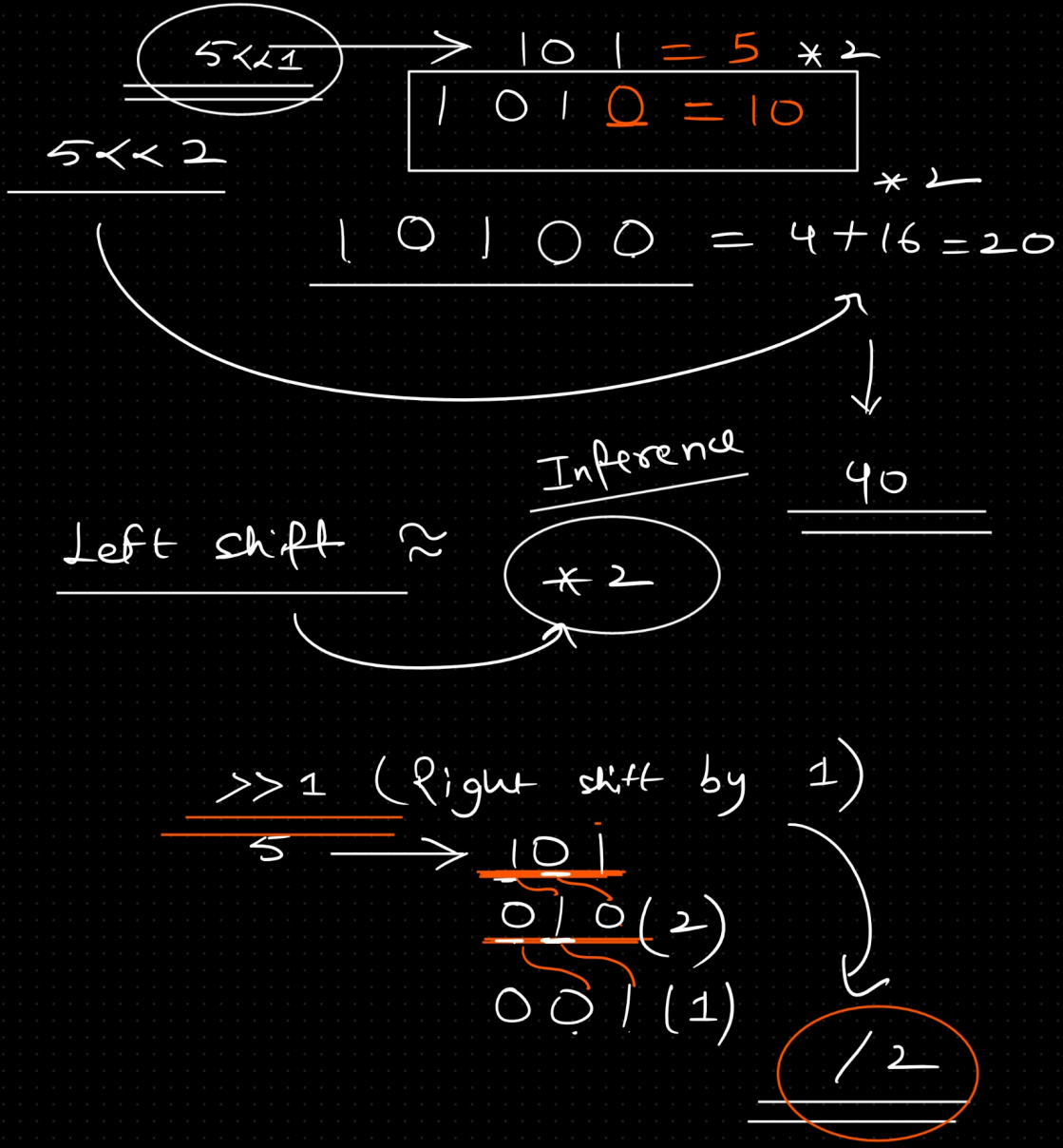
10 →

2 | 10		
2 | 5	0	
2 | 2	1	
1	0	

$(1010)_2$

$0*2^0+1*2^1+0*2^2+1*2^3$

$= 10$


5<<1
101 = 5 *2
1010 = 10
5<<2
*2
10100 = 4+16 = 20
Inference
40
Note: Left shift ≈
*2

>>1 (Right shift by 1)
5
101
010 (2)
001 (1)
/2

$x \wedge x = 0$

$x \wedge 0 = x$

Input:[3 1 2 5 3] n = 5

Output:[3, 4]

Missing Element — Most Optimized Approach

Repeated Element

Bit Manipulation

Application of XOR operation

arr(i) ✓ | i = 1 to n ✓

(3∧1∧2∧5∧3) ∧ (1∧2∧3∧4∧5)

7 ← 3∧4

Repeating / missing

& ~(6)

XOR1 & ~(XOR − 1)

[3, 4]

0 1 1

≈ 001 ∧ 1 0 0

001 → bit No | 1 1 1

set bit (Extreme right)

arr(i) → 3, 1, 2, 5, 3

i = 1 to n → 1, 2, 3, 4, 5

1: Yes → x ✓ | No → y ✓

Yes: 3 ✓, 1, 5, 3, 1, 3, 5 — XOR

No: 2, 2, 4 ✓ — XOR

repeat = 3

missing = 4

met:
if x in a[i]:
  missing = y
  repeat = x
else
  repeat = y

$$
\begin{array}{r}
011 \\
\& \;\; 001 \\
\hline
001
\end{array}
$$

$$
\begin{array}{r}
\text{nisiy} = 8 \\
001 \\
\& \;\; 001 \\
\hline
001
\end{array}
$$

$$
\begin{array}{r}
010 \\
001 \\
\hline
000
\end{array}
\qquad
\begin{array}{r}
101 \\
001 \\
\hline
001
\end{array}
$$

$$
\begin{array}{r}
100 \\
001 \\
\hline
000
\end{array}
$$

Pseudocode (Application of Bit Manipulation)

① XOR — arr(i) ∧ i = 1 to n ⇒ XOR1

missing ∧ repeated ⇒ XOR1 (7)

3 ∧ 4

0 1 1

1 0 0

011

② setbit = XOR1 & ~(XOR-1)

pos: 110

1 1 1

0 0 1

0 0 1 — setbit

&

arr(i) & i = 1 to n

3, 1, 2, 5, 3

③

X	Y
(1)	(0)
~~3~~	~~2~~
~~1~~	~~2~~
~~5~~	4
~~3~~	
~~1~~	
3	
~~5~~	

④ To XOR (x)

XOR (y)

⑤ x is in arr(i)

repeat = x

missing = y

Otherwise

repeat = y

missing = x

num = [7, 8, 6, 5, 4, 8, 2, 1]   n = 8

output = [8, 3]

7 ^ 8 ^ 6 ^ 5 ^ 4 ^ 8 ^ 2 ^ 1 ^ 1 ^ 2 ^ 3 ^ 4 ^ 5

^ 6 ^ 7 ^ 8

(1)

8 ^ 3

(2)

```
  1000 —— 8
^ 0011 —— 3
```

xor1 = 1011 ✓ —— 8 + 2 + 1 = 11

&

~(xor1 − 1)   0101

0001 → setbit (1)

1010

0101

```
  0111 (arr[i])
  0001
  ----
  0001
```

(3)

x	y
7	8
5	6
1	4
1	8
3 ✓	2

5
7

2
4
6
8

(4) missing no. = x

repeated = y