Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Search & Sort Patterns	Strings, Arrays OOPs, Exceptions	Stacks, Queues LinkedLists	Trees, BST Heaps	Trie Graph	Graph, Backtracking DP

← Interview Questions →

Binary Search (Sorted Array)

key = 3 key = 9

2	3	4	6	8	9	12
0	1	2	3	4	5	6

s (l) m s e (h)

$mid = \frac{s+e}{2} = \frac{6}{2} = 3$

```
if (arr[mid] == key)        6 == 3
    return mid;
else if (arr[mid] > key) {  6 > 3
    e = m - 1;
}
else {
    s = m + 1;
}
```

s == e   s < e   (s > e) edge case

while (s <= e) {

} return -1;

$k = \log_2 N$ $O(\log N)$

$\frac{N}{2^0}, \frac{N}{2^1}, \frac{N}{2^2}, \frac{N}{2^3}$ … N → $\frac{N}{2}$ → $\frac{N}{4}\ \frac{N}{4}$ → $\frac{N}{8}\ \frac{N}{8}$ ⟹ $\frac{N}{2^K}$

K = 0, 1, 2, 3 …

integer → (-2^{31} INT_MIN $2^{31} - 1$ INT_MAX)

P int → O/C class   JVM   Integer   Wrapper Class

[Boxing]   O — P →

AutoBoxing: char → Character, boolean → Boolean

optimal: $m = \left[s + \frac{e-s}{2}\right] = \frac{2s + e - s}{2} = \frac{s+e}{2}$

$m = \frac{s+e}{2}$ (OOB) $\frac{Mx + Mx}{2}$

Important (Binary Search) Coding Interview Questions (logn)

- * Square Root of a Number *
- * Search In a 2D Matrix
- * Peak in a Mountain
- * First, Last, Total Occurrences in Array *
- * Missing Element in Array *
- * Aggressive Cows
- * Book Allocation Problem
- * Painter's Partition Problem

→ LeetCode → GFG → Coding Ninjas

(450 dsa.com)   (Prepinsta top 100 codes)

Square root of a number using (log(n)) / Binary Search:

n = 36   o/p = 6   x × x = 36

√36 → 0 (s) — 36 (e)

→ e = m − 1 = 18 − 1 = 17 $m = \frac{s+e}{2} = \frac{36}{2} = 18$ 18 × 18 = 324

0 — 17

ans = 0   e = m − 1 = 8 − 1 = 7 $m = \frac{17}{2} = 8$ 8 × 8 = 64

ans = mid (3)   0 — 7 $\frac{11}{2} = 5$ 5 × 5 = 25

ans = mid (5) → s = m + 1   6 — 7 $\frac{7}{2} = 3$ 3 × 3 = 9

4 — 7

→ s = m + 1 $\frac{13}{2} = 6$

XOR / rsb

0001   0011 ^ 0100 = 0111 = 7 (3 ^ 4)   n1 = 0, n2 = 0

1, 2, 3, 4, 1, 2   (n1) 1 ^ 3 ^ 1 = 3

(rsb) rightmost set bit

G I: 1 → 0001 ✓, 3 → 0011 ✓, 1 → 0001 ✓

G II: 2 → 0010, 4 → 0100, 2 → 0010

(n2) 2 ^ 4 ^ 2 = 2

xorAll = 7

rsb = xorAll & −xorAll = 7 & −7

~7 = 0111   −7: 1's 1000, +1 0001 → 1001

7 & −7: 0111 & 1001 = 0001 (rsb)

```
for (i=0; i<n; i++)
    if (arr[i] & rsb == 1)
        n1 ^= arr[i]    (3)
    else
        n2 ^= arr[i]    (4)
```

Finding two unique elements in the array: → Using Bit Masking

int[] arr = {1, 2, 3, 4, 1, 2};   n1 = 0, n2 = 0

0011 ^ 0100 = 0111 = 7

rsb or rightmost set bit

G1 { 1 → 0001, 3 → 0011, 1 → 0001 }   1 ^ 3 ^ 1

G2 { 2 → 0010, 4 → 0100, 2 → 0010 }   2 ^ 4 ^ 2

rsb = xorall & −xorall = 7 & −7 = 0111 & 1001 = 0001

−7: abs(−7) = 7 = 0111   1's = 1000   +1 = 0001 → 1001

0111 & 1001 = 0001

```
for (int i=0; i<n; i++)
    if (arr[i] & rsb == 1)
        n1 ^= arr[i];
        n1 ^= 0^1^3^1
    else
        n2 ^= 0^2^4^2
```

Power of two

(n) 2 = 2^1 true: 0010 & 0001 = 0   8 = 2^3 true: 1000 & 0111 = 0   4 = 2^2 true: 0100 & 0011 = 0

n & n−1

10 × false   16 = 2^4 true: 10000 & 01111 = 0   13 × false

$n = 2^{n-1}$

Power of two: →

```
if (n & (n-1) == 0)
    return true;
else
    return false;
```

Capgemini, IBM

O(n) Important Array Algorithm: (Kadane Algorithm) (Maximum SubArray Sum)

```
int[] arr = {5, -8, 1, 2, -1, 4} = 6
//           0   1  2  3   4  5
int cmax = arr[0]; 5
int gmax = arr[0]; 5
for (int i=1; i<n; i++) {
    cmax = max(arr[i], cmax+arr[i]);
    gmax = max(cmax, gmax);
}
return gmax;
```

arr[i]	cm+arr[i]	max arr[i], cmax+arr[i]	max cmax gmax
−8	5+−8 = −3	−3	5
1	−3+1 = −2	1	5
2	1+2 = 3	3	5
−1	3+−1 = 2	2	5
4	2+4 = 6	6	6