Search in a 2D Matrix :→

Imp *** TCS / Capgemini / Pure Storage
*** Deutsche Bank → ***
Amazon / Infosys.
→ Leet Code

r0	1	3	5	7
r1	10	11	16	20
r2	23	30	34	60
	c0	c1	c2	c3

int row = matrix.size(); target
2D matrix,
r×c = 3×4
int col = matrix[0].size();

0	1	2	3	4	5	6	7	8	9	10	11
1	3	5	7	10	11	16	20	23	30	34	60

s = 0 e
[1][1] 5/4 = 1 5 % 4 = 1
m = (s+e)/2 = 11/2 = 5 row col

⇒ int element = matrix[mid/col][mid % col]
e = r×c − 1; if (element == target)
return true;
else if ele < target e = m − 1
else s = m+1;

if n == 0 ⟶ 0000
↳ return −1;

otherwise, we need to check whether the number has only one '1' or not.

5 ⟶ 0101 ⟶ return −1
6 ⟶ 0110 ⟶
power of 2 ← 8 ⟶ 1000
← 16 ⟶ 10000

5 & 4: 0101 & 0100 = 0100 = 4
6 & 5: 0110 & 0101
8 & 7: 1000 & 0111 = 0

while(n&1 == 0)

pos = [1, 2, 3, 4]

Power of X to N x^n

x → base, n → exp

$5^2 = 25$
$9^3 = 729$
(Last week)

result = 1
base = X
→ while (N > 0) {
if (N % 2 == 1) {
result *= base;
}
base *= base;
N = N/2;
}
return result;

$x^0 = 1$
base 5×5 = 25 →
2/2 = 1
1 % 2 = 1
1/2 = 0

* Sorting Algorithms :→

1. Bubble sort → O(n^2) → repeated swap
2. Selection " → O(n^2) → swap
3. Insertion " → O(n^2) → shift
4. Merge " → O(n log n) → D&C
5. Count " → O(n + max) } NCA
6. Radix " → O(n + max) }
7. Wave " → O(n) + n log n → sort
8. Heap " → n log n (Trees) (CBT)
9. Quick " → n log n → O(1)
↳ in place sorting

Wave Sort :→

int arr[] = 10, 90, 49, 2, 1, 5, 23
Step 1 → sort 1, 2, 5, 10, 23, 49, 90
2 1 10 5 49 23 90

n log n + O(n)

```
for (int i = 0; i < n-1; i = i+2)
    Swap (arr[i], arr[i+1]);
```

O/P :→ 2 ↘ 1 ↗ 10 ↘ 5 ↗ 49 ↘ 23 ↗ 90

Quick Sort Algorithm :→

In-place sorting algorithm :→ → pivot
recursion

int arr = | 2 | 1 | 3 | 5 | 8 | 7 | 6 |
s pivot index e

int pivot = arr[s]; count = 0;
for (int i = 1, i < e; i++)
if (arr[i] < pivot)
count++;
int pivot index = s + count;
= 0 + 2 = 2
Swap (arr[s], arr[pivot index]);

0 1 2 3 6 8 9 7 4
i, s pi e, j
i++ i++ j-- j-- j-- j--

pivot = a[s]
= 3
count = 0
= 3
| pi |

pivot index = s + c
= 0+3 = 3.

```
while (i < pi && j > pi)
    while (arr[i] <= pivot)
        i++;
⇒   while (arr[j] > pivot)
        j--;
```

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

pivot = 3
↓
partition index

recursion qs (s, pi−1)
qs (pi+1, e)

n log n
↓ ↳ breaking
combining

2 4 1 3 0 6 2
pi pi
↓
pivot <= pivot
count = 0

0 1 2 2

Kadane's Algo :→ ans = 6 O(n)

int arr[] = 5, −8, 1, 2, −1, 4

cmax = arr[0] = 5 ... 6
gmax = arr[0] = 5 6

```
for (int i = 1; i < size; i++) {
    cmax = max(arr[i], cmax + arr[i]);
    gmax = max(cmax, gmax);
}
return gmax;
```

Feedback → 11191
bigdatatraining.com