BINARY SEARCH ALGORITHM

→ Algorithm:

1) Find middle element

2) Check:

if target > middle → search right else search in left.

3) if target == middle → we found.

→ $mid = \frac{start + end}{2} = \frac{0+9}{2} = 4.5 \approx (4)$

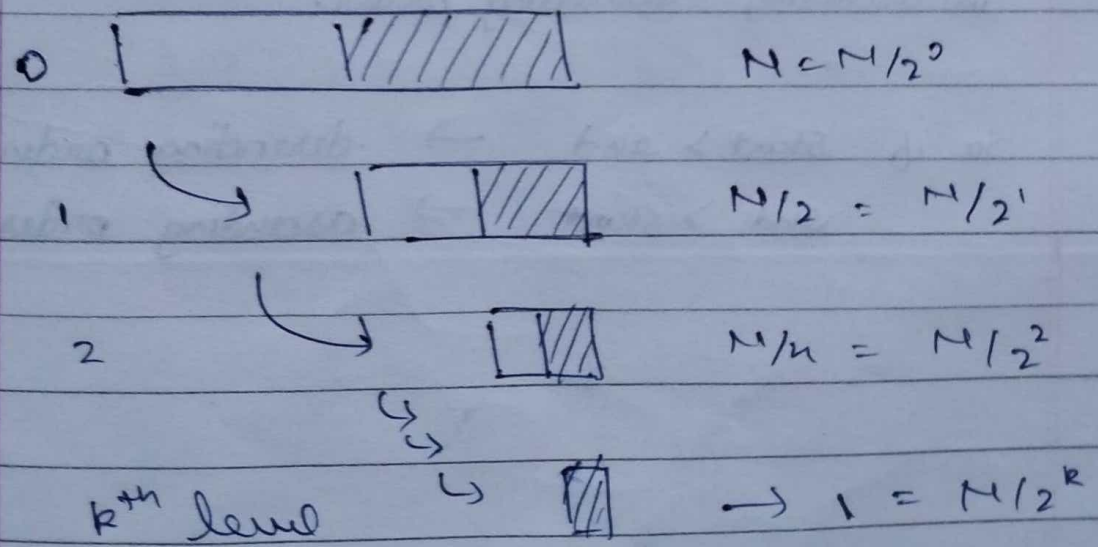
The space in which we are searching is getting divided into two spaces:

① Time complexity

Best case : O(1)

Worst case : O(log n)

explanation: find max number of comparisn



$$\frac{N}{2^k} = 1 \quad \therefore N = 2^k$$

$$\log(N) = \log(2^k) = k \log 2$$

$$k = \frac{\log N}{\log 2} \qquad \therefore \boxed{k = \log_2 N}$$

total no of comparision in worst case

size of array

Better way to find mid:

1) $mid = \left(\frac{start + end}{2}\right)$

2) $mid = s + \frac{e-s}{2} = \frac{s+e}{2}$

here it will not exceed int value.

* Order agnostic Binary Search.

→ if we dont know that the array is sorted in ascending / descending order.

So if start > end → descending order

end > start → ascending order.

5.2 2D BINARY SEARCH

* Matrix is sorted in row or column wise manner

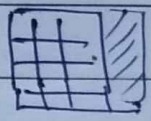
	0	1	2	3
0	10	20	30	40
1	15	25	35	45
2	28	29	37	40
3	33	34	38	50

target = 37

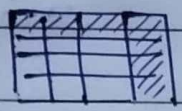
matrix can be m x n

Case 1: if element == target
→ ans found.

Case 2: if target < 40
→ eliminate column [3] : c--

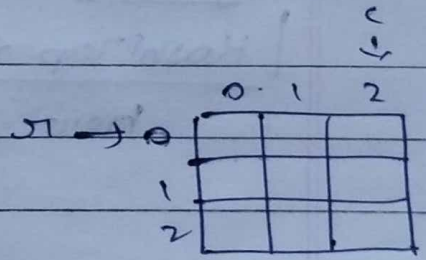


if target > 30
r++



* Complexity: n+n = O(2n) = O(n)
∵ n rows and n columns.

* Code




```
int r = 0
int c = matrix.length - 1
while (r < matrix.length && c >= 0) {
  if (mat[r][c] == target)
    { return new int[] {r, c}; }
  if (mat[r][c] < target) {
    r++ }
  else {
    c--;
  } return new int[] {-1, -1}
```