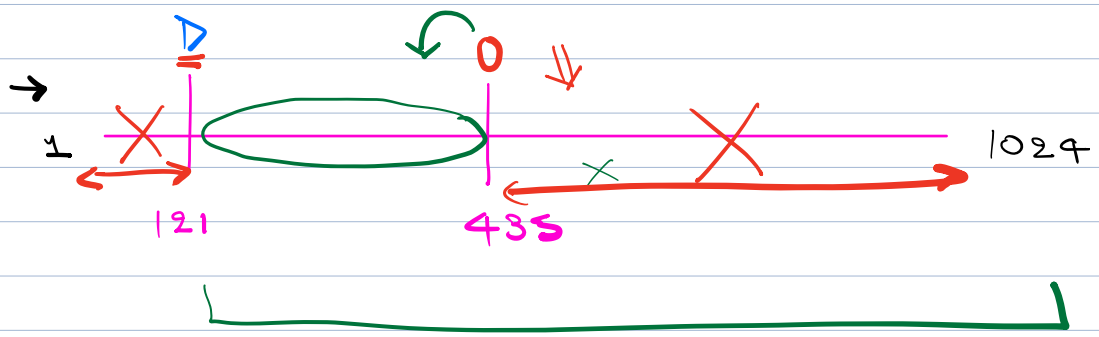
Search → Target (What to search)

→ Search Space (Where to search)

Eg : flabbergasted ⇒ Target

Dictionary ⇒ Search space



Searching can be optimised if data in target is sorted.

→ O(N) → linear

Ex : A: [1, 2, 8, 19, 31, 36, 42, 61, 72, 81, 99]

0 1 2 3 4 5 6 7 8 9 10 11

K = 19

Binary Search

⇒ Divide the search space into 2 halves.

⇒ Discard one half using some condition

Q Given a sorted array with distinct ele.

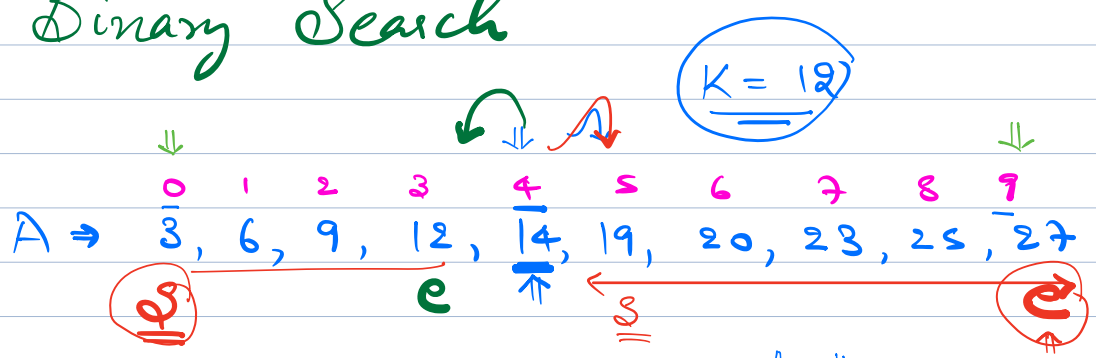
= Search & return the index of an element K (Given).
If K is not present in the array, return -1

Eg: A → 3, 6, 9, 12, 14, 19, 20, 23, 25, 27 (indices 0 1 2 3 4 5 6 7 8 9)

1) Linear Search

T.C. = O(N)
S.C. = O(1)

2) Binary Search



1) Calculate mid ⇒ (s+e)/2

$s + \frac{(e-s)}{2} \Rightarrow$

(i) If (A[mid] == K) {
→ return mid;
}

(ii) If (A[mid] > K) { Discard right

e = mid - 1;

// new search space ⇒ [s, mid-1]

(iii) If (A[mid] < K) { Discard left

s = mid + 1;

// new search space ⇒ [mid+1, end]

}

K = 12 / 11

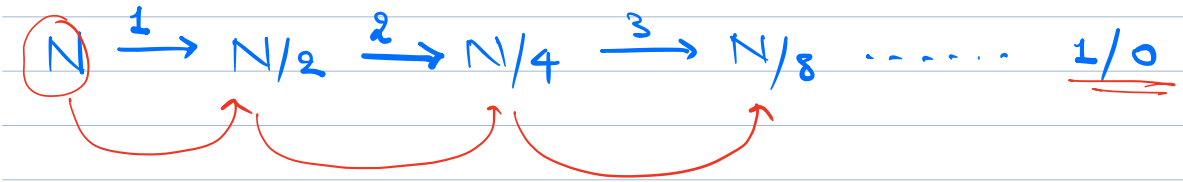
	0	1	2	3	4	5	6	7	8	9
A ⇒	3	6	9	12	14	19	20	23	25	27
	s			ans						e

s	e	mid = (s+e)/2	
0	9	(0+9)/2 = 4	A[4] > 12 Discard right
0	3	(0+3)/2 = 1	A[1] < 12 Discard left
2	3	(2+3)/2 = 2	A[2] < 12 Discard left
3	3	3	A[3] = 12 ↓ return
3	2		

Code

int searchIndex (A[], N, K) {

```
s = 0;
e = N-1;

while (s <= e) {

    mid = (s + e)/2

    if (A[mid] == K) {
        return mid;
    }
    else if (A[mid] > K) {
        e = mid - 1;
    }
    else {
        s = mid + 1;
    }
}

return -1;

}
```



T.C. = O(log N)

S.C. = O(1)

Q Given a sorted array of size N & K

Find the floor of a given no. K in the array

floor ⇓ greatest element <= K in the array.

	0	1	2	3	4	5	6	7	8
A:	-5,	2,	3,	6,	9,	10,	11,	14,	18

K = 5 ⇒ 3

K = 4 ⇒ 3

K = 10 ⇒ 10

K = -7 ⇒ floor does not exists. (INT_MIN)

K=

K = 5

	0	1	2	3	4	5	6	7	8
A:	-5,	2,	6,	7,	9,	10,	11,	14,	18
	s								e

s	e	mid	Comp	
0	8	4	A[4] > 5	Reject R
0	3	1	A[1] < 5	ans = 2
				Reject L
2	3	2	A[2] > 5	Reject R
2	1	X		

Code

```
int findfloor(A[], N, K) {
    if (A[0] > K) {
        return INT_MIN;
    }

    ans = INT_MIN;
    int s = 0, e = N-1, mid = 0;
    while (s <= e) {
        mid = (s+e)/2;
        if (A[mid] == K) {
            return A[mid];
        }
        else if (A[mid] > K) {
            e = mid-1;
        }
        else {
            ans = max(ans, A[mid]);
            s = mid+1;
        }
    }

    return ans;
}
```

T.C. = $O(\log N)$

S.C. = $O(1)$

Q Given a sorted array of size N.

Turn Given an element K (present in the array).

find the frequency of K

Eg: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
-5, -5, -3, 0, 0, 1, 1, 5, 5, 5, 5, 5, 5, 5, 8, 10, 10, 15

K = 5 (7)

① Iterate & count

T.C. = O(N)

S.C. = O(1)

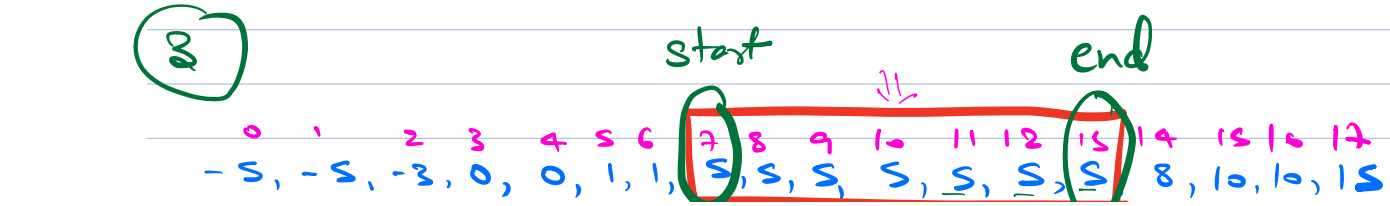
② BS + Iteration

→ 1) find K using BS O(log N)

→ 2) Go to left & right & count > O(N)

T.C. = O(N)

③

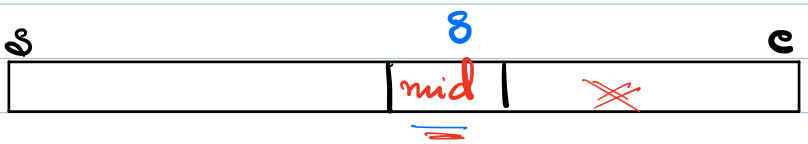


#count = end - start + 1

1) First occurance of K (BS) $\log N$

H.W. 2) Last occurance of K (BS) $\log N$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
-5	-5	-3	0	0	1	1	5	5	5	5	5	5	5	8	10	10	15

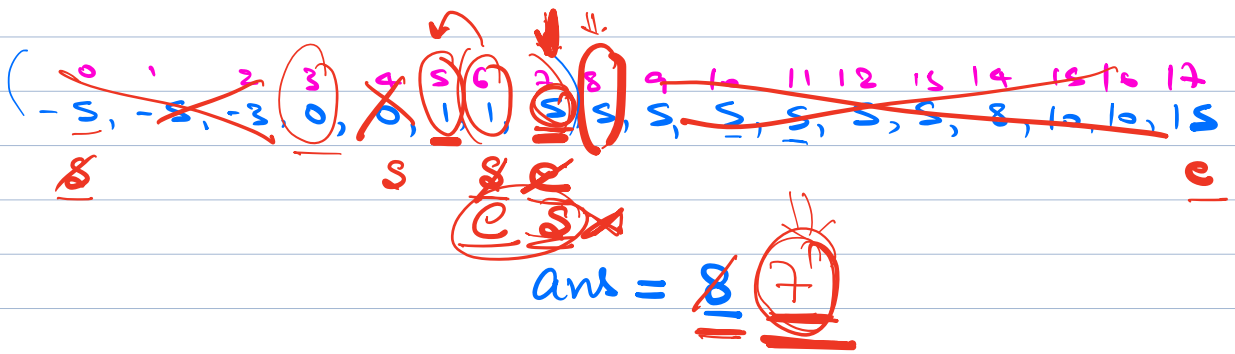


arr[mid] < K	arr[mid] = K	arr[mid] > K
Reject left	ans = mid	Reject Right
s = mid + 1	Reject Right	e = m - 1
	e = mid - 1	

H.W. 1) Last Occurance of K

2) Code

T.C. = $O(\log N)$



Q) Given an integer array of size N.
The array was initially sorted.
but was then rotated K times.
Search if an element X is present in the array or not.

A: -20, -14, -8, -4, 1, 2, 4, 7, 11, 14, 19, 23, 27

K = 5

A: 11, 14, 19, 23, 27, -20, -14, -8, -4, 1, 2, 4, 7,

0 K-1

BS in A[0, K-1]

BS in A[K, N-1]

H.W. if K is unknown ??

find K using BS