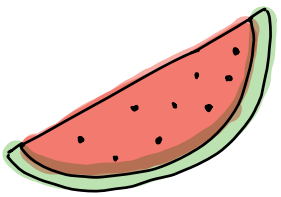


Searching

↳ products on Amazon.

→ Facebook | Instagram.

→ Google.

Target : Something that has to be searched.

Search Space :- Where to search for the target.

Ex :-

1) Search a word in a newspaper. ⇒ O(N)

word → target; newspaper → Search Space.

2) Search a word in a dictionary.

word → target; dictionary → Search Space.

3) Search a ph. no in a hand written list of nos.

ph. no → target; hand written list of nos → Search Space.

4) Search a phone no. in a phone directory.

phone no. → target; phone directory → Search Space

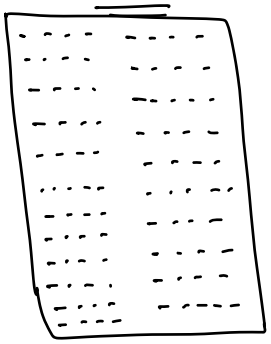
5) Search an int in an Array.

int → target; Array → Search Space.

6) Search an int in a sorted Array.

target — search space

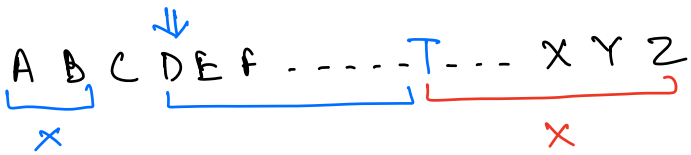
Newspaper



$\Rightarrow$ Linear Search

TC: O(No. of words in newspaper)

Dog



Ex

Sorted Array is given

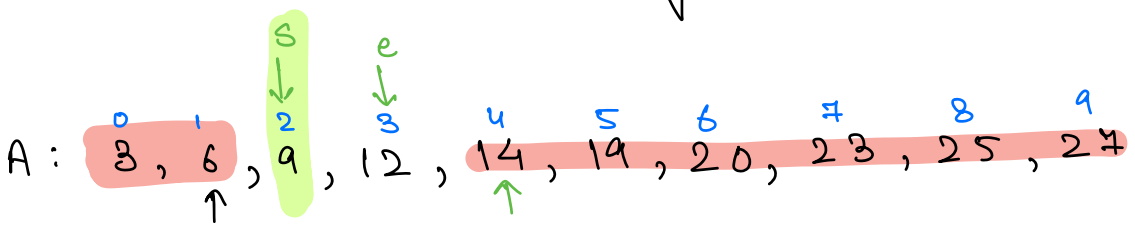
A: 0: 3, 1: 6, 2: 9, 3: 12, 4: 14, 5: 19, 6: 20, 7: 23, 8: 25, 9: 27

K = 12

s	e	randomIndex
0	9	6 : A[6] > K
0	5	1 : A[1] < K
2	5	2 : A[2] < K
3	5	5 : A[5] > K
3	4	3 : A[3] == K ans.

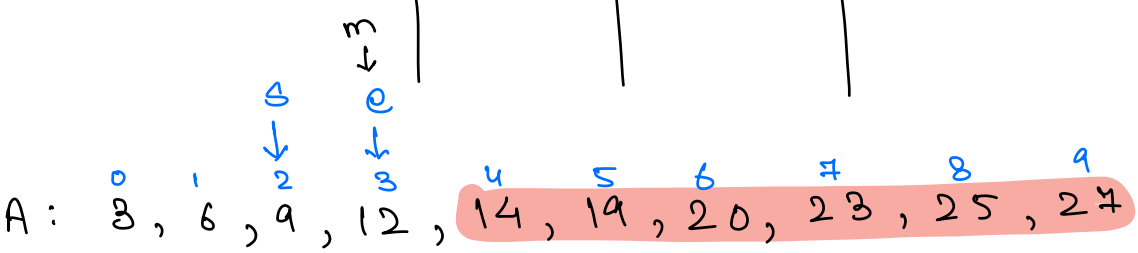
Best choice for random index :-

⇒ Middle index of the search space.



K = 9

s	e	mid	A[mid]
0	9	4	14 > K
0	3	1	6 < K
2	3	2	9 == K return 2



K = 11

s	e	mid	A[mid]
0	9	4	14 > 11
0	3	1	6 < 11
2	3	2	9 < 11
3	3	3	12 > 11
3	2	2	

s > e ⇒ break

return -1

Binary Search

* binarySearch fun^ returns the index at which target is present (OR) it will return -1 if target is NOT present.

```
int binarySearch (int A[], int K) {
    l = 0
    r = N-1
    while ( l<=r ) {
        mid = (l+r)/2
        if ( A[mid] == K)
            return mid;
        else if ( A[mid] < K )
            l = mid+1;
        else {
            r = mid-1;
        }
    }
    return -1;
}
```

Time Complexity :-

$N \xrightarrow{1^{st}\ iteration} \frac{N}{2} \xrightarrow{2^{nd}\ iteration} \frac{N}{4} \longrightarrow \frac{N}{8} \longrightarrow \cdots\cdots (1)$

$\log_2 N$

TC : $O(\log_2 N)$ **SC : $O(1)$**

Recursive Code $\Rightarrow$ TC : $O(\log_2 N)$, SC : $O(\log_2 N)$

Q.1 Given a sorted Array (ascending order) find the floor of a given value (K)

floor (K) $\Rightarrow$ The largest no. less than or equal to K (in the Array)

	0	1	2	3	4	5	6	7	8
A :	-5	2	3	6	9	10	11	15	18

floor (20) = 18

floor (2) = 2

floor (5) = 3

floor (14) = 11

floor (-6) = ✗

floor (8) = 6

	0	1	2	3	4	5	6	7	8
A :	-5	2	3	6	9	10	11	15	18

K = 4

floor (4)

s	e	mid	ans
0	8	4	x
0	3	1	2
2	3	2	3
3	3	3	3
3	2		

s > e ⇒ Break

return 3

```
int floor ( int A[], int K) {
    l = 0
    r = N-1
    ans = -∞
    while ( l <= r) {
        mid = (l+r)/2;
        if( A[mid] == K) {
            return A[mid];
        }
        else if ( A[mid] < K ) {
            ans = A[mid];
            l = mid + 1;
        }
        else {
            r = mid - 1;
        }
    }
    return ans;
}
```

TC : O(log N)

SC : O(1)

10:32 pm

Q.2 Amazon Given an Array of size N, sorted in ascending order. Find the frequency of K.

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

freq(1) = 3

freq(5) = 6

freq(0) = 2

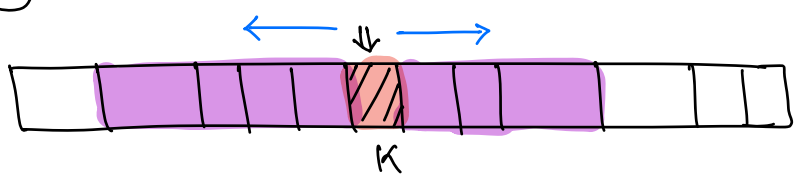
Brute Force

TC : O(N)

SC : O(1)

Approach # 1

(1) Apply Binary search & find the occurrence of (K) → O(log N)



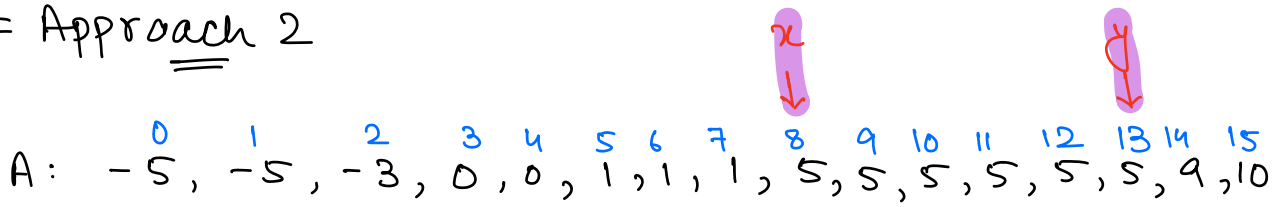
(2) Iterate over the Array towards left & right to count the freq.

⇒ O(N)

TC: O(N)

SC: O(1)

Approach 2



$$freq(5) = y - x + 1$$

Steps

1) find the first occurrence of K.
(left most index at which (K) is present)

2) find the last occurrence of K.
(right most index at which (K) is present)

first occurrence

K = 5

l	r	mid	A[mid]	s (1st occurrence)
0	15	7	1	-1
8	15	11	5	11
8	10	9	5	9
8	8	8	5	8
8	7			

l > r Break

```
int firstOcc ( int A[], int K) {
    l = 0
    r = N-1
    first_occ_index = -1
    while( l <= r) {
        mid = (l+r)/2
        if( A[mid] == K) {
            first_occ_index = mid
            r = mid-1;
        }
        else if ( A[mid] < K ) {
            l = mid + 1;
        }
        else {
            r = mid - 1;
        }
    }
    return first_occ_index;
}
```

last occurrence

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

$k = 5$

l	r	mid	A[mid]	e (last occurrence)
0	15	7	1	-1
8	15	11	5	11
12	15	13	5	13
14	15	14	9	13
14	13			

$l > r \Rightarrow$ Break

$s = 8$
$e = 13$
$\text{freq} = 13 - 8 + 1$
$= 6$

```
int firstOcc( int A[], int K) {
    l = 0
    r = N-1
    last_occ_inden = -1
    while( l <= r) {
        mid = (l+r)/2
        if( A[mid] == K) {
            last_occ_inden = mid
            l = mid+1;
        }
        else if ( A[mid] < K ) {
            l = mid + 1;
        }
        else {
            r = mid - 1;
        }
    }
    return last_occ_inden;
}
```

return last_occ_inden - first_occ_inden +1

TC : O(log N)

SC : O(1)

* Where can we apply the Binary Search ?

⇒ Binary Search can be applied when we can come up with a logic of discarding half of the search space in every iteration.
