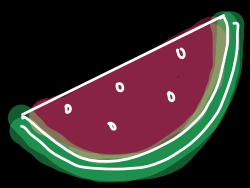


Searching

Target : Something that has to be searched

SearchSpace : Where to search for the target

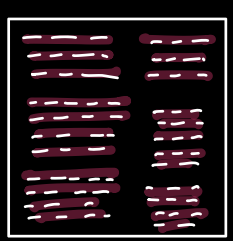
- Search a **word** in **newspaper**. → Linear Search
- Search a **word** in **dictionary**. ✔ → Binary Search

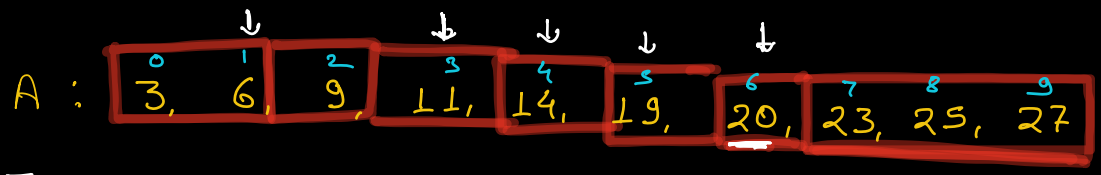
Search a **ph no of contact** in a **hand written large list**.

Search a **ph no of contact** in a **directory** ✔

Search an **int** in an **array** (unsorted)

Search an **int** in a **sorted array** (ASC/DESC) ✔



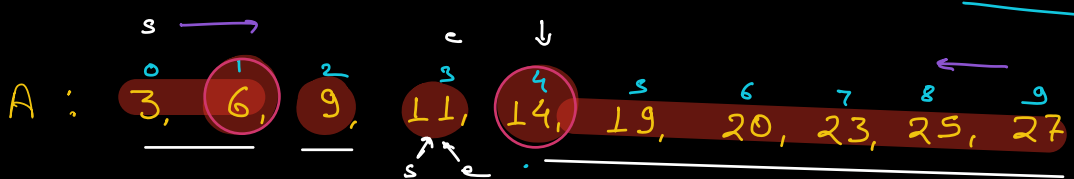


Target = 11 ✓

12 ✗

s	e	Rand & Iterat
0	9	A[6] ⇒ 20
0	5	A[1] ⇒ 6
2	5	A[2] ⇒ 9
3	5	A[5] ⇒ 19
3	4	A[4] ⇒ 14

Best choice for this random index ⇒ mid of search space.



Target : 9

12

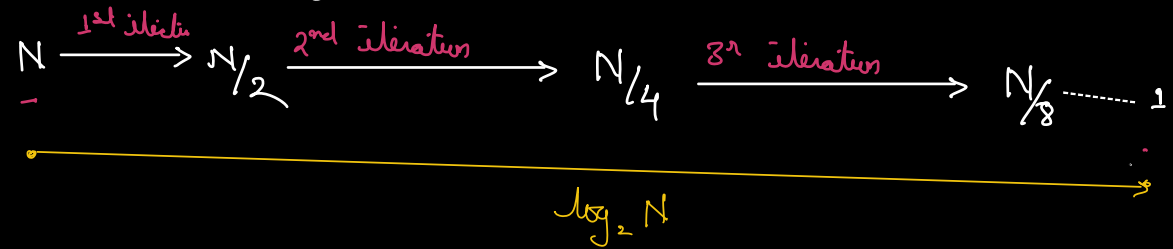
	s	e	mid (s+e)/2	A[mid]
⇒	0	9	4	14
⇒	0	3	1	6
→	2	3	2	9 ✓
→	3	3	3	11
→	4 ⟷ 2			

→ Break

Return the index of target in a sorted array. ret -1 if it is not present.

```
int BinarySearch(A[], K){
    // define the search space
    l = 0 ; r = N-1 ;
    while( l <= r ){
        mid = (l + r)/2 ;
        if( A[mid] == K){
            ret mid ;
        }
        else if( A[mid] < K){
            l = mid + 1;
        }
        else{
            r = mid - 1;
        }
    }
    ret -1;
}
```

Search space size



TC : O(log N)

SC : O(1)

Q Given an array sorted in ASC order. Find the floor of a given no. K.

floor(K) ⟶ The greatest no. less than or equal to K (in the array)

A : $\underset{0}{-5}, \underset{1}{2}, \underset{2}{3}, \underset{3}{6}, \underset{4}{9}, \underset{5}{10}, \underset{6}{11}, \underset{7}{15}, \underset{8}{18}$

K = 1

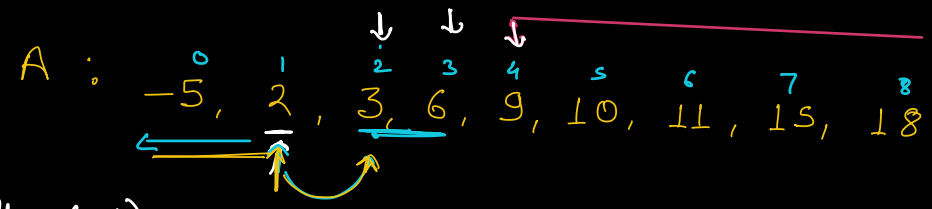
floor(20) = 18

floor(2) = 2

floor(5) = 3

floor(14) = 11

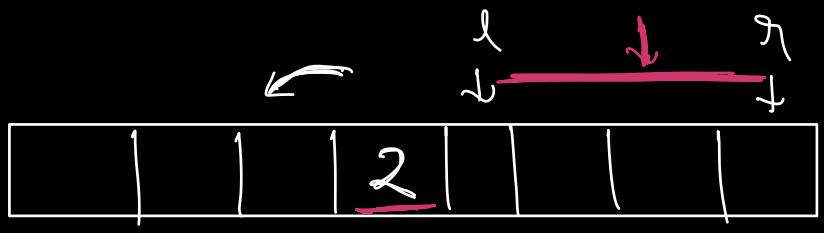
floor(8) = 6



floor (4)

l	r	mid	ans
0	8	4	$-\infty$
0	3	1	2
2	3	2	3
3	3	3	3
3	2	→ Break	

```
int FindFloor(A[], K){
    // define the search space
    l = 0; r = N-1;
    while( l <= r ){
        mid = (l + r)/2;
        if( A[mid] == K){
            ret A[mid];
        }
        else if( A[mid] < K){
            ans = A[mid]
            l = mid+1;
        }
        else{
            r = mid-1;
        }
    }
    ret -1;
}
```



K = 4

A[mid] > K

go to left.

A[mid] < K

ans = A[mid];

l = mid + 1;

Amazon

Q Given an array of N elements sorted in ASC order. Find the freq of a given target K.

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

s = 8, e = 13

freq(1) → 3

freq(5) → 6

freq(0) → 2

[s, e] = e − s + 1

13 − 8 + 1 = 6

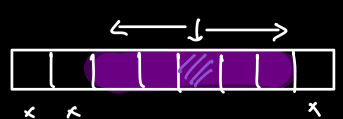
Brute Force : Linear Search

TC: O(N)

SC : O(1)

Approach 1 :

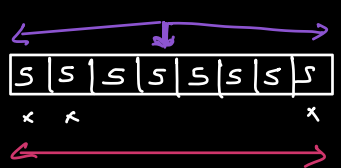
① Do a BS & find an occurance of the target ⇒ O(log N)



② Iterate towards left & right to count occ.

TC: O(N)

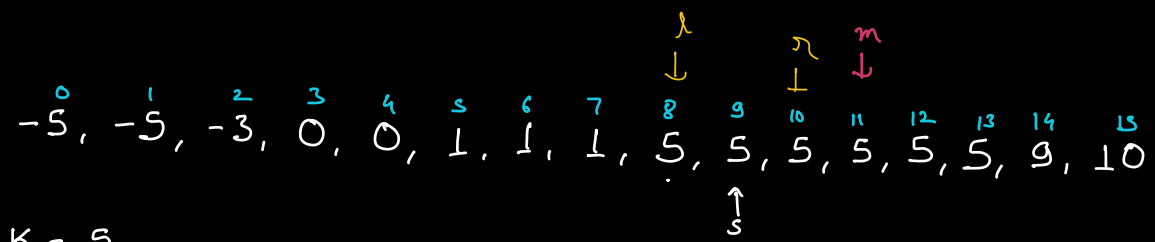
TC : O(log N + N) ⇒ O(N)



Step I : Find first occurance of K
Prob 1 (left most index where K is present) ⇒ S

Step I : Find last occurance of K
Prob 2 (right most index where K is present) ⇒ E

ret freq → E - S + 1

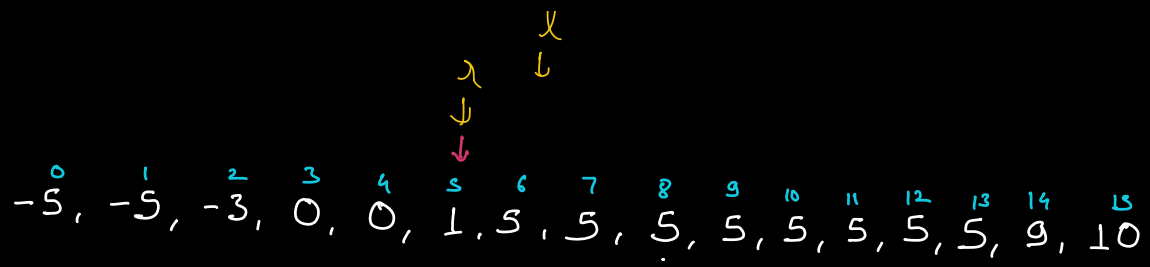


K = 5

l	r	mid	A[mid]	S → 1st occ
0	15	7	1	-1
8	15	11	5	11
8	10	9	5	9
8	8	8	5	8
8	7			

→ Break

TC : O(logN + logN) ⇒ O(logN)



K=5

l	r	mid	A[mid]	ans.
0	15	7	5	7
0	6	3	0	7
4	6	5	1	7
6	6	6	5	6
6	5	→ Break.		

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

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

```
freq (K) = e - s + 1;
```

Binary search can be applied when you can come up with a logic of discarding half of the search space.

Google

Q Given an array of N distinct elements.. Find any local minima in the array.

local minima → a no. smaller than all available neighbour (adjacent element)

if A[i] is local minima

$A[i-1] > A[i] < A[i+1]$

$3 < 6 > 1$ ×

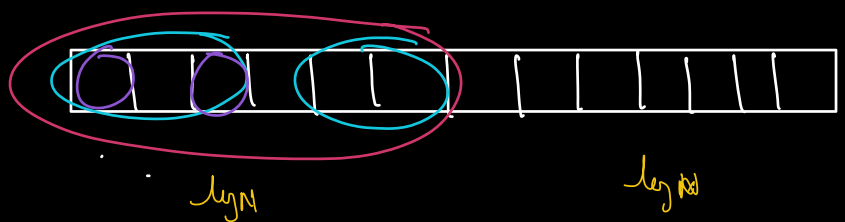
$6 > 1 > 0$ ×

$1 > 0 < 9$

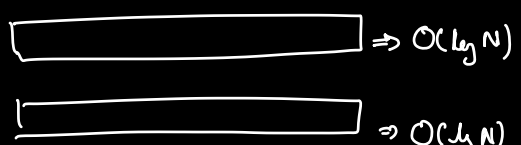
$0 < 9 < 15$

$9 < 15 > 8$

$15 > 8$



$$\begin{cases} T(N) = \underline{2T(N/2)} + 1 \\ \qquad\qquad \hookrightarrow O(N) \\ T(N) = T(N/2) + 1 \\ \qquad\qquad \hookrightarrow O(\lg N) \end{cases}$$



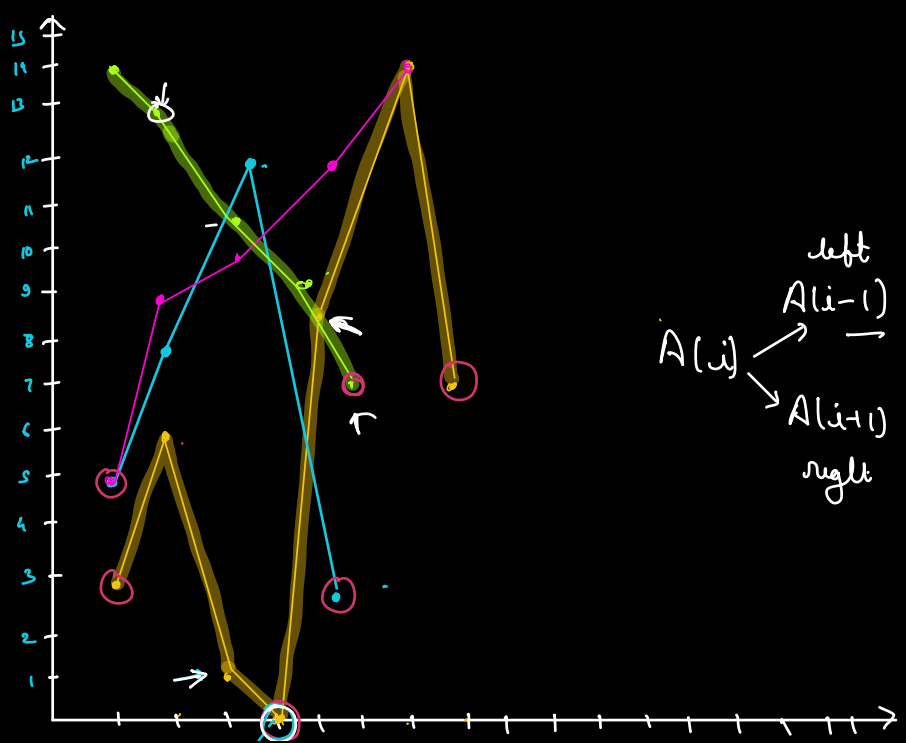
$O(\lg N)$

A: 3, 6, 1, 0, 9, 15, 8 (indices 0 1 2 3 4 5 6)

B: 5, 8, 12, 3

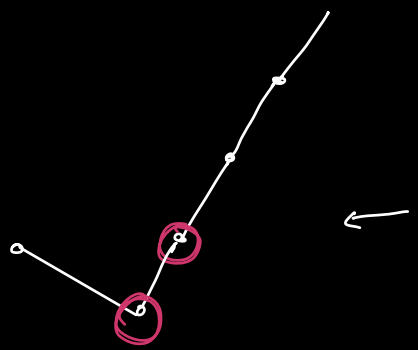
C: 5, 9, 10, 12, 15

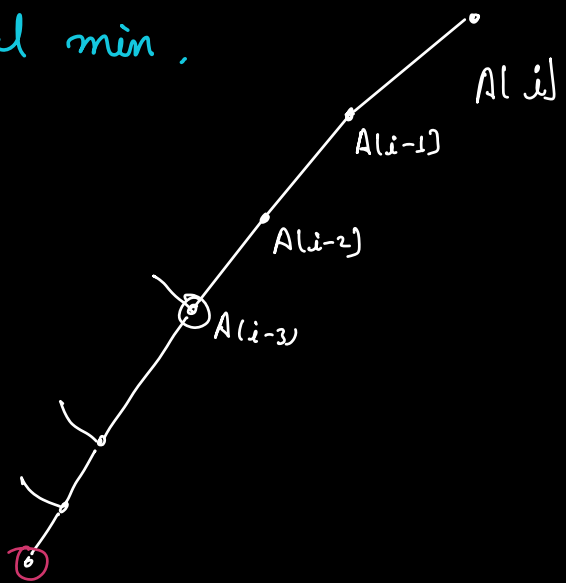
D: 14, 13, 11, 9, 7

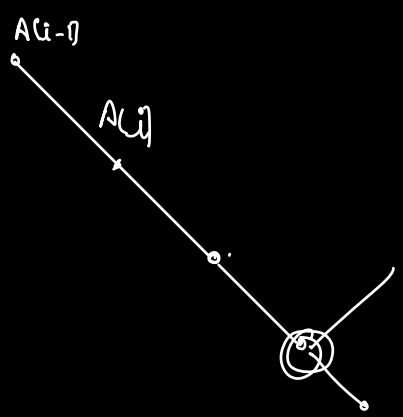


$A[i-2] < A[i-1] < A[i]$

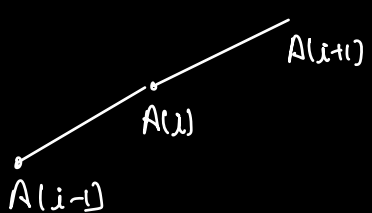
Moving towards the smaller neighbour guarantees the use of local min.



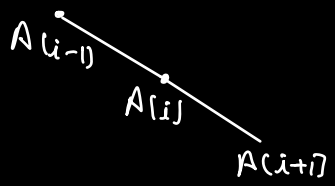




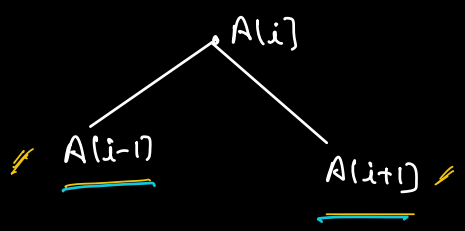
$A[i-1] < A[i] < A[i+1] \Rightarrow$ move to left



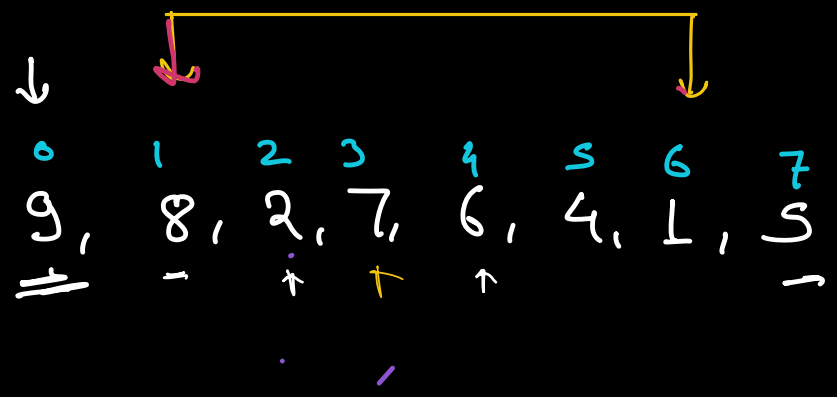
$A[i-1] > A[i] > A[i+1] \Rightarrow$ move to right



$A[i-1] < A[i] > A[i+1]$




```
int localMinima(A[]) {
    if(A[0] < A[1]) ret A[0];
    if(A[N-1] < A[N-2]) ret A[N-1];
    l = 1; r = N-2;
    while(l <= r) {
        mid = (l+r)/2;
        if(A[mid] < A[mid-1] && A[mid] < A[mid+1])
            ret A[mid];
        else if(A[mid-1] < A[mid]) {
            r = mid-1;
        }
        else
            l = mid+1;
    }
```



l	r	mid	A[mid]	A[mid+1]	A[mid-1]
1	6	3	7	6	2
1	2	1	8	2	9
2	2	2	2	7	8