Agenda

i) Linear search to find K in A[]

ii) Binary search to find K in A[]

iii) First occurrence of K in array

iv) Floor of K in array

v) Local minima

Q.1 Given A[], find if K is present or not.

A =	2	9	8	17	42	1
	0	1	2	3	4	5

K = 19 ans = -1

K = 17 ans = 3

Linear search

```
int   search ( int [ ]A, int K) {
        int n= A.length;
        for (int i=0; i<n; i++) {
                if (A[i] == K) {
                        return i;
                }
        }
        return -1;
}
```

TC: O(n)

Organised vs unorganised data :

Searching in organised data takes less efforts.
(organised clothes section, dictionary, ...)

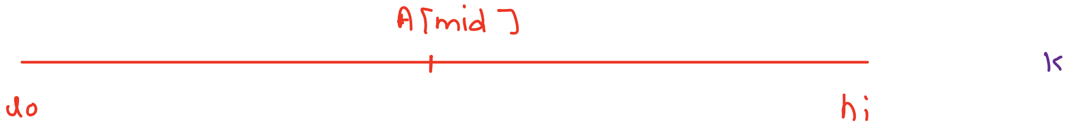
Q.2 Given a sorted A[], find if k is present or not.

A =	2	9	13	15	19	24	31	48	52
	0	1	2	3	4	5	6	7	8

k = 13 ans = 2

k = 48 ans = 7

k = 10 ans = -1



Binary Search

A[mid] == k : got the answer , return mid

A[mid] < k : discard left side
⇒ lo = mid + 1

A[mid] > k : discard right side
⇒ hi = mid - 1;

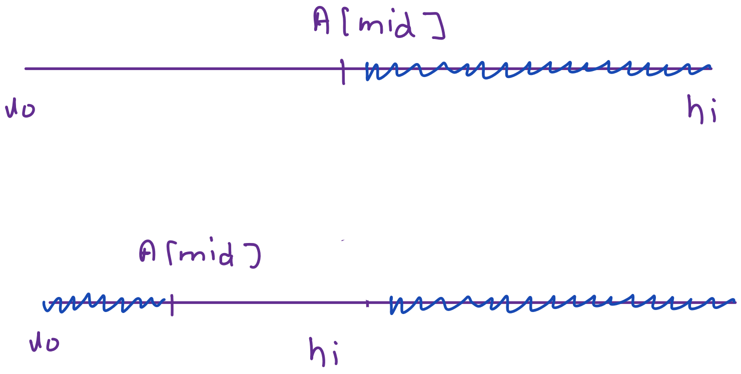
```
int  search ( int [] A , int  K) {
  int  n= A.length;
   int  lo = 0 , hi = n-1;

   while ( lo <= hi)  {
        int  mid=  (lo+hi)/2;
        if( A[mid] == K)  {
              return  mid;
        }
        else if (A[mid] < K) {
              //discard left side
               lo = mid+1;
        }
        else if ( A[mid] > K)  {
               // discard right side
               hi= mid-1;
        }
   }
   return -1;
}
```

```
                                   K= 8
A =  2   4   8   13   19   29   38   42   49
     0   1   2   3    4    5    6    7    8
             lo  hi
             m
                              ans= 2
```

```
                                   K= 42
A =  2   4   8   13   19   29   38   42   49
     0   1   2   3    4    5    6    7    8
                                     lo   hi
                                     m
                              ans= 7
```

```
                                   K= 13
A =  2   4   8   13   19   29   38   42   49
     0   1   2   3    4    5    6    7    8
                 hi
                 lo
                 m
                                   ans = 3
```

```
                                   K= 10
A =  2   4   8   13   19   29   38   42   49
     0   1   2   3    4    5    6    7    8
             hi
                 lo
                 m
                             get out of loop
                              ans = -1
```



$\Rightarrow n \downarrow \frac{n}{2} \downarrow \frac{n}{4} \cdots 1$

TC: $O(\log_2 n)$

Array is sorted

example: $n = 10^5$ $(10^5 = \sim 2^{16})$

Linear search : $n \Rightarrow 10^5$ steps

Binary search : $\log_2 n \Rightarrow \log_2(10^5)$

$= \log_2(2^{16}) = 16$ steps

Q.3 Given a sorted A[] , find first occurrence of K.

A =	2	2	3	4	5	5	5	7	7	8	9	12	19	20
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Expected TC : $O(\log_2 n)$

K = 3	ans = 2
K = 5	ans = 4
K = 10	ans = -1

Slight modification of Binary search

↳ A[mid] == K

⇒ keep searching on left

```
int search ( int [] A, int K) {
  int n = A.length;
  int lo = 0 , hi = n-1 , ans = -1;

  while ( lo <= hi)  {
      int  mid =  (lo+hi) /2 ;
      if( A[mid] == K) {
             ans = mid ;
             hi =  mid-1;
      }
      else if (A [mid] < K) {
             //discard left side
             lo = mid+1;
      }
      else if ( A[mid] > K) {
             // discard right side
             hi = mid-1;
      }
  }
  return ans;
}
```

K = 5

A =	2	2	$\boxed{5}$	5	5	5	7	8	9	9
	0	1	$\boxed{2}$	3	4	5	6	7	8	9
		hi	lo							
			m							

ans = ~~-1~~ ~~4~~ 2

K = 9

A =	2	2	5	5	5	5	7	$\boxed{9}$	9	12
	0	1	2	3	4	5	6	$\boxed{7}$	8	9
							hi	lo		
							m			

ans = ~~-1~~ 7

Q-4 Given a sorted array, find floor of K in the array.

floor (K) => $\boxed{\text{max of all the values which are <= K}}$

A =	12	19	21	25	28	32	35	38	42	51
	0	1	2	3	4	5	6	7	8	9

K	floor
24	21
28	28
40	38
55	51

$$\text{floor}(K) = \begin{cases} K & \text{when K is present in A[]} \\ \text{just small than K} & \text{when K is absent in A[]} \end{cases}$$

A =	12	19	21	25	28	32	35	38	42	51
	0	1	2	3	4	5	6	7	8	9

```
if (A[mid] <= K) {
        ans = A[mid];
        lo = mid + 1;
}
```

```
else {
        hi = mid-1;
}
```

```
int floor (int [] A, int K ) {
    int n = A.length;
     int lo = 0, hi = n-1, ans = -1;
    while (lo <= hi) {
        int mid = (lo+hi)/2;
        if (A[mid] <= k) {
            ans = A[mid];
            lo = mid + 1;
        }
        else {
            hi = mid - 1;
        }
    }
    return ans;
}
```

K = 10

A = 2 8 13 14 24 31 48 51

0 1 2 3 4 5 6 7

hi

lo

m

ans = ~~-1~~ 8

K = 10

A = 2 7 9 13 14 24 31 48 51

0 1 2 3 4 5 6 7 8

hi

lo

m

ans = ~~-1~~ ~~7~~ 9

Q. 5 Local minima

Given an array, find any local minima. Local minima is the element smaller than both of its neighbours. Corner elements will have only one neighbour. { Array contains distinct values }

A = 12 $>$ $\boxed{10}$ $<$ 15 20 ans : 10

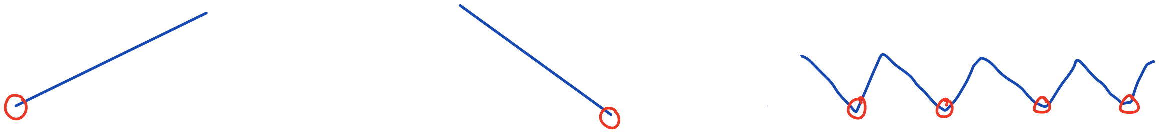
A = 12 10 9 $>$ $\boxed{7}$ $<$ 15 ans : 7

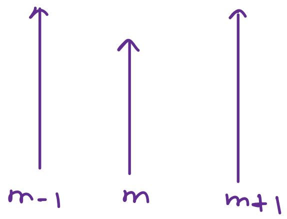
A = 12 15 17 19 8 20 ans : 8 or 12

→ Simple idea : O(n)

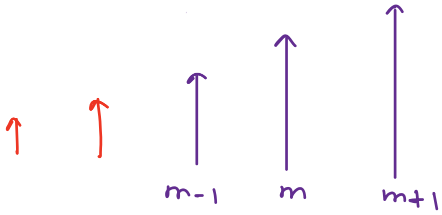
go on every element and check if it smaller than its neighbours.

→ local minima will always be there





m-1
m
m+1



m-1
m
m+1

```
int   local_minima(int [] A) {
   int  n= A.length;
    // corner cases
    if (A[0] < A[1]) {
          return A[0];
     }
     else if (A[n-1] < A[n-2]) {
            return A[n-1];
     }

     int lo = 1, hi = n-2;
     while (lo <= hi) {
           int  m = (lo+hi)/2;
           if (A[m] < A[m-1]  && A[m] < A[m+1]) {
                return  A[m];
            }
            else if (A[m-1] < A[m]) {
                    hi = m-1;
            }
            else if (A[m+1] < A[m] {
                   lo = m+1;
             }
     }
     return -1; (to satisfy compiler)
}
```

Java

Dry run

```
while (lo <= hi) {
    int m = (lo+hi)/2;
    if (A[m] < A[m-1] && A[m] < A[m+1]) {
        return A[m];
    }
    else if (A[m-1] < A[m]) {
        hi = m-1;
    }
    else if (A[m+1] < A[m]) {
        lo = m+1;
    }
}
```

20	$\boxed{18}$	22	25	28	16	50
0	1	2	3	4	5	6
	lo m	hi				

20	19	30	25	23	$\boxed{16}$	50
0	1	2	3	4	5	6
				lo	m	hi

TC: $O(\log_2 n)$