Searching 1

Today's Content:

- Binary Search
- Search for element k.
- Search first and last occurrence
- Single element in a sorted array
- Peak element
- Local minima

Mock Interview

- DSA Mock interview for 60 mins
- At the end of DSA module
- 30 days after your DSA module ends, the interview expires
- Only if you pass, you can sit for placements.

Base topics for 80% DSA problems:

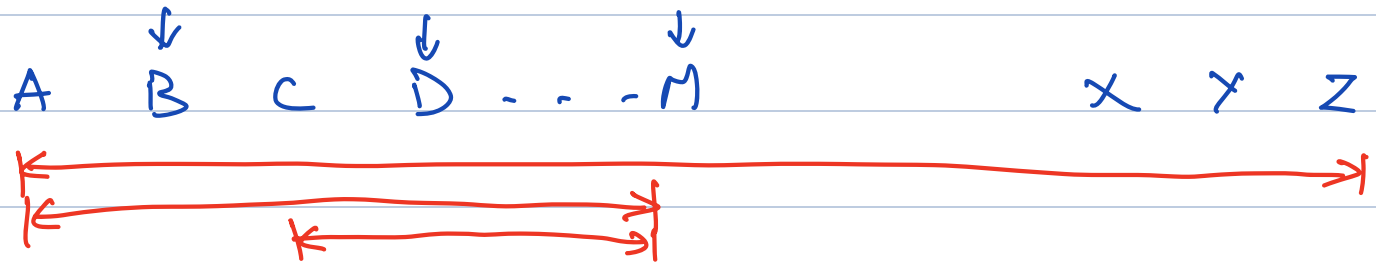
1) Hashing 2) Binary Search 3) Two pointers
4) Recursion

Scenario

Police → Target (How they look)
↳ Search space (Where you found them last)

Example: word → {Dictionary, Book, Newspaper}
phone no → {Phone Directory, Diary}

* Sorting makes searching faster



$\boxed{\text{Dog}}$ → Target

Criteria required for Binary Search:

1) Target
2) Search space
3) Condition to neglect half of the search space

What is Binary Search?

- Divide search space into 2 parts
- Repeatedly keep on neglecting one half of the search space.

Question: Given a sorted array with distinct elements, return the index of an element k. If k is not present, return -1.

ex: arr = [3 6 9 12 14 19 20 23 25 27]

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

k = 14 → O/P = 4 k = 99 → O/P = -1

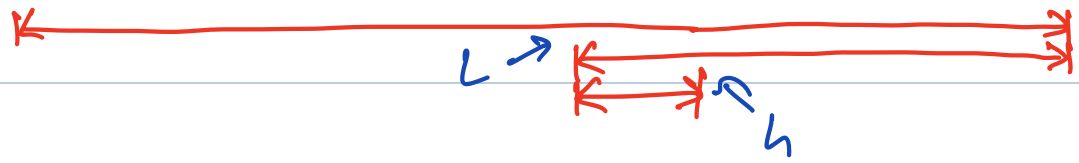
Idea 1: Linear search (Iterate over all elements)

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

Idea 2: Binary search

- Target (k)
- Search space (array)
- Condition to neglect half of search space
  - arr[mid] > k → Go left.
  - arr[mid] < k → Go right.
  - arr[mid] == k → Return mid.

	0	1	2	3	4	5	6	7	8	9
arr =	3	6	9	12	14	19	20	23	25	27



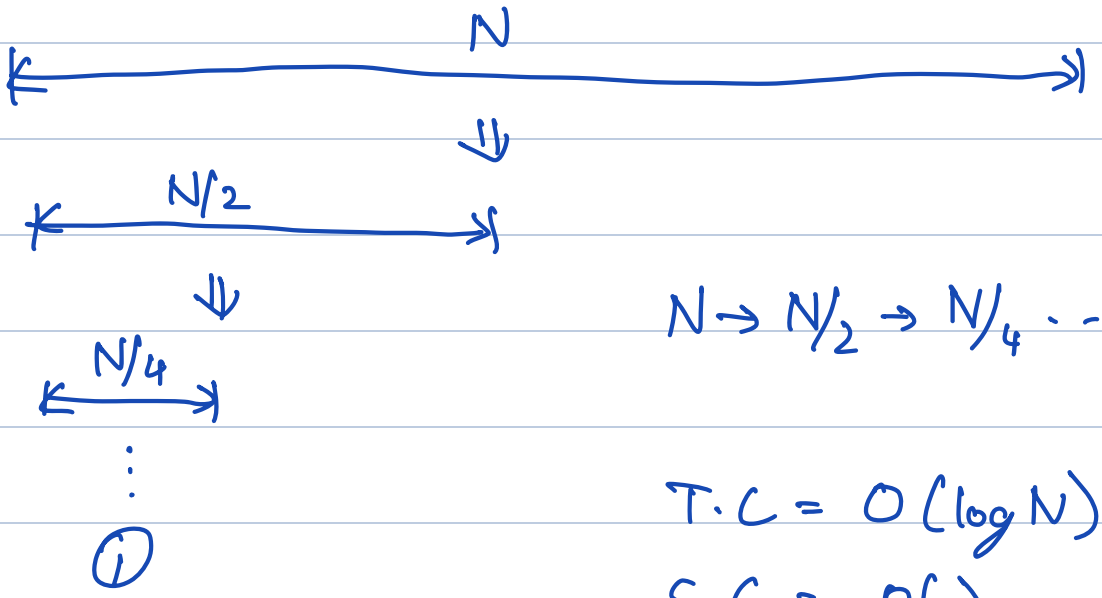
k = 15

Low	High	Mid	Result
0	9	4	Go right
5	9	7	Go left.
5	6	5	Go left
5	4		

Steps:

1) Low = 0 ; high = N-1
2) mid = (low + high)/2
3) Apply the conditions.
4) Go left ⇒ h = m-1
5) Go right ⇒ l = m+1
6) Redo the same

```
func searchK(int[] arr, int k) {
    l = 0 ; h = arr.length - 1 ;
    while (l <= h) {
        m = (l + h) / 2 ;
        if (arr[m] == k)  return m;
        else if (arr[m] < k)  l = m+1 ;
        else  h = m-1 ;
    }
    return -1 ;
}
```



$N \to N/2 \to N/4 \cdots 1$

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

$S.C = O(1)$

Best Practice

$m = (l+h)/2 \to$ int.

$l = 10^9$

$h = 10^9$

$l+h = 2*10^9 \to$ int ✗ Can't store in int

Solution 1:

$m = ((\text{long})\, l + h)/2$

Solution 2:

$m = l + (h-l)/2 \Rightarrow \dfrac{2l + h - l}{2} = \dfrac{l+h}{2}$

Question: Given a sorted array of N elements, find the first occurrence index of a given element k.

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

$k = 5$ O/P = 7

Idea: Find element using binary search

Then keep going left 1 by 1 till 1st occurrence

L m h

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

Iterations = 1 + 1 + 1 + 1 + 1 = 5 = N/2 = O(N)

Optimised:

L m h

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

low	high	mid	ans (-1)	Result
0	9	4	4	Go left
0	3	1	1	Go left
0	0	0	0	Go left
0	-1	→ Break.		

T.C = O(log N)

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

L m h

k = 5

Low	High	Mid	Ans	Result
0	14	7	7	Go left.
0	6	3		Go right
4	6	5		Go right
6	6	6	6	Go left
6	5	→ Break		

```
func firstOccurrence (int[] arr, int k){
    ans = -1;
    l = 0;  h = arr.length -1;
    while (l<=h){
        m = l + (h-l)/2;
        if(arr[m] == k)  ans = m;   h = m-1;
        else if(arr[m] > k)  h = m-1;
        else  l = m+1;
    }
    return ans;
```

Question: Given an array where every element occurs twice except for one unique element, find that unique element.

Note: Duplicates are adjacent to each other.

Idea 1: XOR of whole array

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

Optimised:

	0	1	2	3	4	5	6	7	8	9	10	11	12
arr =	3	3	1	1	8	8	10	10	19	6	6	2	2

l m h

index of first occurrence is even

index of 1st occurrence is odd.

1st occurrence index is even ⇒ Go right.

1st occurrence index is odd ⇒ Go left.

Low	High	Mid	isUnique (Check left & right)	arr[m] == arr[m-1] (m = m-1)	m % 2	Result
0	12	6	No	–	True	Go right. (l = m+2)
8	12	10	No	9	False	Go left (h = m-1)
8	8	8	Yes → return arr[8]			

```
func findUnique (int[] arr) {
    l = 0 ; h = arr.length - 1 ;
    while (l <= h) {
        m = l + (h - l)/2 ;
        // is Unique
        if ((m > 0 && arr[m] != arr[m-1]) &&
            (m < N-1 && arr[m] != arr[m+1]))
            return arr[m] ;
        // Go to first occurrence
        if (m > 0 && arr[m] == arr[m-1])
            m = m - 1 ;
        // Check if index is even or odd
        if (m % 2 == 0)  // even -> Go right
            l = m + 2 ;  // l = m+1 is also duplicate
                            element.
        else  // odd -> Go left.
            h = m - 1 ;
    }
    if (N > 1 && arr[0] != arr[1]) return arr[0];
    if (N > 1 && arr[N-1] != arr[N-2]) return arr[N-1];
    if (N == 1) return arr[0];
```

}

Question: Given an increasing decreasing array with distinct elements, find the maximum element.

ex: arr = [1 3 5 2]

O/P = 5

arr = [1 3 5 10 15 12 6]

O/P = 15

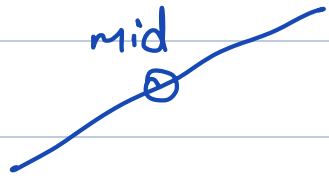
Case 1:

mid

⇒ Return mid.

Case 2

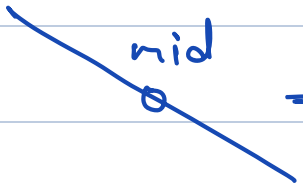
mid



⇒ Go right

Case 3

mid



⇒ Go left

Code ⇒ TODO.

Question: Given an array of distinct elements, find any local minima in the array.

Note: Local minima is a number which is smaller than its adjacent neighbours.

ex: arr = [3 6 1 0 9 15 8]

arr = [21 20 19 17 15 9 7]

arr = [5 9 15 16 20 21]

Binary Search

- Target (Local minima)
- Search (Array).
- Condition

Case 1:

⇒ Return mid

Case 2

⇒ Go right

Case 3

⇒ Go left.

Case 4

⇒ Go anywhere

Case 5

⇒ Go anywhere

```
func localMinima(int[] arr) {
    l = 0;  h = N-1;
    while (l <= h) {
        m = l + (h-l)/2;
        // Case 1
        if ((m>0 && arr[m] < arr[m-1]) &&
            (m<N-1 && arr[m] < arr[m+1]))
            return m;
        // Case 2
        else if((m>0 && arr[m] < arr[m-1]) &&
                (m<N-1 && arr[m] > arr[m+1]))
            l = m+1;
        // Case 3
        else if((m>0 && arr[m] > arr[m-1]) &&
                (m<N-1 && arr[m] < arr[m+1]))
            h = m-1;
        // Case 4 - Not possible as no duplicates
        // Case 5
        else h = m-1;
}
```

```
    if ( N>1 && arr[0] < arr[1]) return 0;
    if ( N>1 && arr[N-1] < arr[N-2]) return N-1;
    if (N==1) return 0;
}
```

L h

	0	1	2	3	4	5	6
arr =	3	6	0	1	9	15	8

low	high	mid	Case	Result
0	6	3	⇒ mid	Go left
0	2	1	mid	Go anywhere (Left)
0	0	0	→ No execution, so go left.	
0	-1	Break.		

Base case 1 ⇒ True ⇒ Return 0

L m h

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

Low	High	mid	Case	Result
0	11	5	None	Go left.