Searching

→ Finding an element in array/ linked list called searching.

08→ **Linear Search** = A simple approach is to do a linear search. The time complexity of the linear search is O(n).

09

A simple approach is to do a linear search, i.e.,

10 (1) Start from the leftmost element of arr[] and one by one compare x with each element of arr[].

11 (2) If x matches with an element, return the index.
(3) If x doesn't match with any of element, return -1.

12

eg find [20]

0	1	2	3	4	5	6	7	8
10	50	30	70	80	60	20	90	40

01

02
```
def search (arr, n, n):
    for i in range (0, n):
        if (arr[i] == n):
            return i
03
    return -1
```

→ best case — 1st position element found at

→ worst case — at last(n) not available

04 TC — O(n) [ε C]

05

→ **Binary Search**

06

(1) Begin with the mid element of the whole array as search key.
07 (2) If the value of the search key is equal to the item then return index of the search key.
(3) Or if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
(4) Otherwise, narrow it to the upper half.
(5) Repeatedly check for the second point until the value is found or the interval is empty.

MARCH
26
TUESDAY
'24
13th Week
086-280

BS → It only works for sorted arrays. l.sort()

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAR 2024

eg

0	1	2	3	4	5	6
10	20	30	40	50	60	70

low, high

mid = (low + high)//2 mid = (0+6)//2 = 3

Case 1: (l[mid] == x), if x = 40 → return mid

Case 2: (l[mid] > x), if x = 10 → high = mid - 1 [change high]

Case 3: (l[mid] < x), if x = 60 → low = mid + 1

Iterative

```
def bs(l,x):
    low = 0
    high = len(l) - 1
    while low <= high:
        mid = (low + high)//2
        if l[mid] == x:
            return mid
        elif l[mid] < x:
            low = mid + 1
        else:
            high = mid - 1
    return -1
```

— TC — O(log n)
— SC — O(1)

Left H available नहीं है then high = mid - 1

Right H available नहीं है then low = mid + 1.

Recursive

```
def bs(l,x, low, high):
    if low > high:
        return -1
    mid = (low + high)//2
    if l[mid] == x:
        return mid
    elif l[mid] > x:
        return bs(l,x,low,mid-1)
    else:
        return bs(l,x,mid+1,high)

def bsmain(l,x):
    return bs(l,x,0,len(l)-1)
```

— TC — O(log n)
— SC — O(log n)

Iterative Solution should be preferred when we are having a choice

Index of first occurrence in sorted array :-

(a)
```
def fo(arr,n,x):
    for i in range(0,n):
        if(arr[i]==x):
            return i
    return -1
```

(b)
```
def fo(arr,n,x):
    low=0
    high=n-1
    while(low<=high):
        mid=(low+high)//2
        if x>arr[mid]:
            low=mid+1
        elif x<arr[mid]:
        else:
            if mid==0 or
            arr[mid-1]!=arr[
                return mid
            else:
                high=mid-1
    return -1.
```

Index of last occurrence in sorted array :-

(a)
```
def la(l,n):
    low=0
    high=len(l)-1
    while low<=high:
        mid=(low+high)//2
        if l[mid]<x:
            low=mid+1
        elif l[mid]>x:
            high=mid-1
        else:
            if mid==len(l)-1 or l[mid]!=l[mid+1]:
                return mid
            else:
                low=mid+1
    return -1
```

(b)
```
def la(l,n):
    for i in reversed(range(len(l))):
        if l[i]==x:
            return i
    return -1.
```

```
for i in range(len(l)-1,
    -1,-1):
```

MARCH
28
'24
13th Week
088-278
THURSDAY

S	M	T	W	T	F	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

MAR 2024

Count occurrence in sorted array

(a) def count OCC (lin): → first occurrence function of Previous function.
    first = fo (lin)
    if first == -1:
        return 0
    else:
        return last OCC (lin) - first + 1
    lo → function = last occurrence of Previous Page.

TC = O(n)

```
res = 0
for i in range(n): if x == arr[i]:
    res += 1 , return res
```

(b) l.count( ) — TC = O(n)
    ↳ any thing which you want.

Repeating Element (a) arr(size) >= 2 (b) only one element is repeated

(c) All the element from 0 to max(arr) are present => $0 \le \max(arr) \le n-2$

i/p = l = [0,2,1,3,2,2] o/p = 2.
i/p = l = [1,2,3,0,3,4,5] o/p = 3.

we need to solve
O(n) , O(1)
No modification to original array

M-1
```
for i in range(0,n-1):
    for j in range(i+1,n):
        if (arr[i] == arr[j]):
            return arr[i]
```
TC = $O(n^2)$
SC = $O(1)$

M-2 Sort the array.
[0,1,2,2,2,3]
TC = O(nlogn)
```
for i in range(0,n):
    if (arr[i] == arr[i+1])
        return arr[i]
```

M-4
```
def repeat(arr,n):
    slow = arr[0]
    fast = arr[0]
    slow = arr[slow]
    fast = arr[arr[fast]]
    while slow != fast: slow = arr[slow]
    fast = arr[arr[fast]]
    slow = arr[0]
    while slow != fast: slow = arr[slow]
        fast = arr[fast]
return slow.
```

M-3
```
def repeat(arr,n):
    visit = [False]*n
    for i in range(n):
        if visit[arr[i]]:
            return arr[i]
        visit[arr[i]] = True
    return -1
```

Square root

(a)
```
def sqrt(n):
    i = 1
    while i*i <= n:
        i += 1
    return i-1
```

(b)
```
def sqrt(n):
    low = 1
    high = n
    ans = -1
    while low <= high:
        mid = (low+high)//2
        msq = mid*mid
        if msq == n:
            return mid
        elif msq > n:
            high = mid-1
        else:
            low = mid+1
            ans = mid
    return ans.
```

Linear search for first & record occurrence.

```
def ls(l,key):
    tl = []
    for i in range(len(l)):
        if key == l[i]:
            tl.append(i)
    return tl[0:2]
```

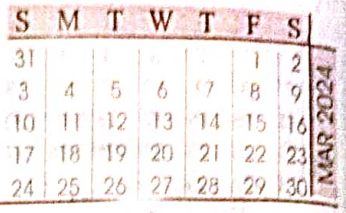
Search in an Infinite sized array.

i/p = arr = [1,10,15,20,40,80,90,100,120,500..]
n = 100, o/p = 7 ✓

i/p = arr = [20,40,100,300,...]
n = 50, o/p = -1.

M-1
```
def search(arr,n):
    i = 0
    while True:
        if arr[i] == n:
            return i
        if arr[i] > n:
            return -1
        i += 1.
```
TC = O(n)

M-2
```
def search(arr,n):
    if arr[0] == n:
        return 0
    i = 1
    while arr[i] < n:
        i = i*2
    if arr[i] == n
        return i
    return (arr,i//2+1,i-1,n)
```
TC = O(log n)



Peaks Element

Given an array of integers. Find a peak element.
i.e an element that is not smaller than its neighbours.

i/p : l = [5, 10, 20, 15] , o/p = 20
i/p : l = [10, 20, 15, 2, 23, 90, 67] , o/p = 20 (or) 90

M1

```
def peak(arr,n):
    if n==1:
        return arr[0]
    if arr[0] >= arr[1]:
        return arr[0]
    if arr[n-1] >= arr[n-2]:
        return arr[n-1]
    for i in range(1,n-1):
        if arr[i] >= arr[i-1] and arr[i] >= arr[i+1]:
            return arr[i]
```

TC = O(n)
SC = O(1)

M2

```
def peak(arr,n):
    l=0
    r=n-1
    while(l<=r):
        mid = (l+r)>>1
        if((mid==0 or arr[mid-1]<=arr[mid]) and
           (mid==n-1 or arr[mid+1]<=arr[mid])):
            break
        if(mid>0 and arr[mid-1]>arr[mid]):
            r=mid-1
        else:
            l=mid+1
    return mid
```

TC = O(log n)
SC = O(1)

Two Pointer Approach

Two pointers is really an easy & effective technique that is typically used for searching pairs in a sorted array.

eg Given a sorted array A (sorted) having N integers, find if there exist any pair of elements (A[i], A[j]) such that their sum is equal to x.

i/p: A = [10, 20, 35, 50, 75, 80]
x = 70, o/p: True

```
def ispair(arr,n):
    n= len(arr)
    for i in range(n-1):
        for j in range(i+1,n):
            if arr[i] + arr[j]==n:
                return True
    return False
```

TC = O(n²)
SC = O(1)

M-2

We move either of them towards each other.

eg [2, 4, 8, 9, 11, 12, 20, 30], x = 23

→ It is giving smaller than expected so it will not create impact

→ It is giving larger than expected value pair so it will get impact

i=0, J=7
(2+30) > 23 : i=0, J=6
(2+20) < 23 : i=1, J=6
(4+20) > 23 : i=1, J=5
(4+12) < 23 : i=2, J=5
(8+12) < 23 : i=3, J=5
(9+12) < 23 : i=4, J=5
(11+12) == 23 : return True

APRIL
01
MONDAY
'24
14th Week
092-274

This technique will work well when you have sorted array, for unsorted array we will have to first sort the array.

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

APR 2024

We take two pointers, one representing the first element and other representing the last element of the array, and then we add the values kept at both the pointers. If their sum is smaller than x then we shift the left pointer to right or if their sum is greater than x then we shift the right pointer to left, in order to get closer to the sum. We keep moving the pointers until we get the sum as x.

```
def Tp(arr, x):
    i = 0
    J = len(arr) - 1
    while i < J:
        if arr[i] + arr[J] == x:
            return True
        elif (arr[i] + arr[J] < x):
            i = i + 1
        else:
            J = J - 1
    return False.
```

TC = O(nlogn)
SC = O(1)

Triplet in a sorted Array:—

i/p = l = [2, 3, 14, 8, 9, 20, 40], x = 32
o/p = True. Triplet is (4, 8, 20)

```
def Triplet(arr, x):
    n = len(arr)
    for i in range(n):
        for J in range(i+1, n):
            for K in range(J+1, n):
                if arr[i] + arr[J] + arr[K] == x:
                    return True
    Return False
```

TC = O(n3)
SC = O(1)

① Traverse the array from left to right

② For every element arr[i], check if there is a pair on right side with sum (n - arr[i])

starting index.

```
def isPair(arr, n, si):
    i = si
    J = len(arr) - 1
    while i < J:
        if arr[i] + arr[J] == n:
            return True
        elif arr[i] + arr[J] < n:
            i = i + 1
        else:
            J = J - 1
    return False
```

```
def Triplet(arr, n):
    for i in range(len(arr) - 2):
        if (isPair(arr, n - arr[i], i+1)):
            return True
    return False
```

TC = $O(n^2)$