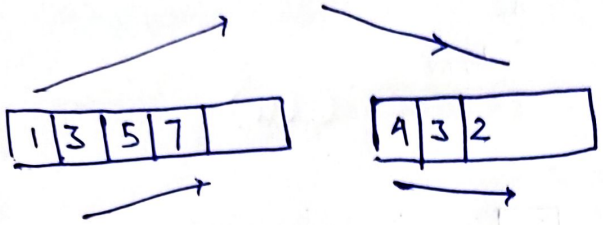
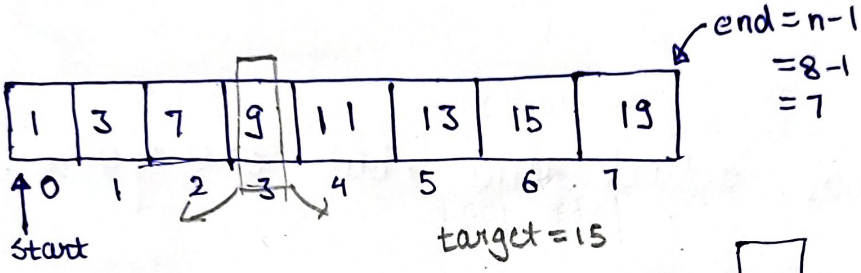
WEEK 4

Lec1:

Binary Search :-

Cond^n: element should be in monotonic order.

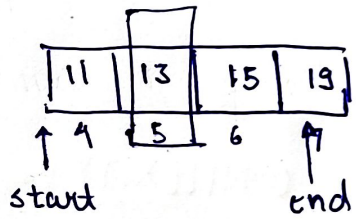




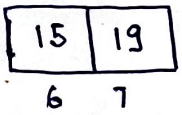
① start → 0 index

② end → last index
$n-1 = 7$

③ mid = $\left(\frac{start + end}{2}\right)$
$= \frac{0+7}{2} = 3$



mid = $\frac{4+7}{2} = \frac{11}{2} = 5$

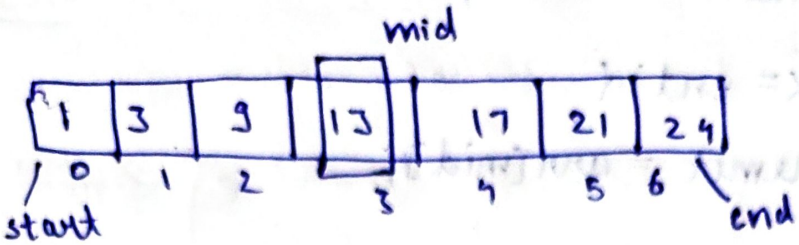


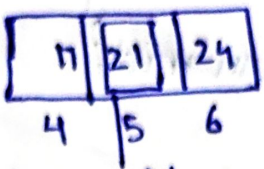
mid = $\frac{6+7}{2} = \frac{13}{2}$ 6

15
6 if 15 on 6 is target?
Yes

```
#include <iostream>
using namespace std;

int binarySearch(int arr[], int size, int target){
    int start=0;
    int end = size-1;
```

```
    int mid = (start + end) / 2;

    while(start <= end){
        int element = arr[mid];

        if (element == target){
            return mid;
        }
        else if (target < element){
            // search in left
            end = mid - 1;
        }
        else{
            start = mid+1;   // search in right
        }
        mid = (start + end)/2;
    }
    // element not found
    return -1;
}

int main(){
    int arr[] = {2,4,6,8,10,12,16};
    int size = 7;
    int target= 2;

    int indexOftarget = binarySearch (arr, size, target);
    if (indexOftarget ==-1){
        cout << "target not found" << endl;
    }else{
        cout << "target found at " << indexOftarget << endl;
    }
    return 0;
}
```





22 > 21

start > e

↓

Loop stop

Here element not found

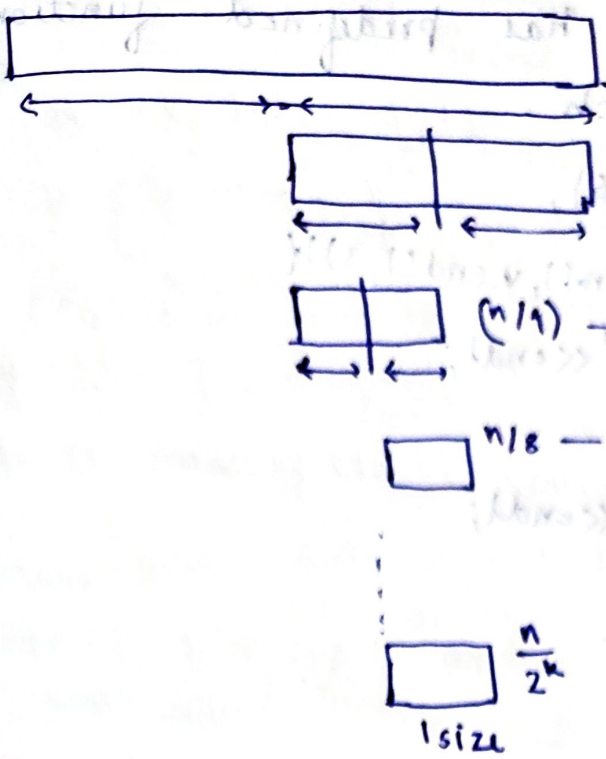
Note:- $mid = \frac{s+e}{2}$ has a issue.

Integer overflow हो सकता है!

---

so, we use:

$$mid = s + \frac{(e-s)}{2}$$



Time complexity (Derivation)

n size array.

I iteration → $\frac{n}{2^1}$

2nd " → $\frac{n}{2^2}$

.

.

kth " → $\frac{n}{2^k}$

$\frac{n}{2^k} = 1$ as if only one element is now there

$n = 2^k$

$$\boxed{k = \log_2 n}$$

So, Time. C of Binary search is $O(\log n)$.

- Binary Search is faster than linear search.

STL: This already has predefined function of binary search.

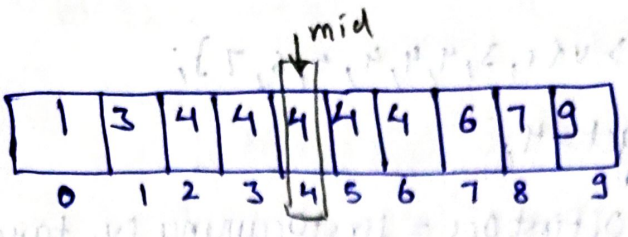
```
vector<int> v{1,2,3,4,5,6};
if(binary_search(v.begin(), v.end(), 3)){
        cout << "Found" << endl;
}
else{
    cout<< "No Found." <<endl;
}
return 0;
}
```

To use binary_search, we need to #include <algorithm> in our program.

Also, For array:

```
int arr[] = {1, 2, 3, 4};
int size = 4;
if (binary_search (arr, arr + size, 3)){
                cout << " Found " <<endl;
}
 else{
      cout << "Not Found" <<endl;
}
```

① find the first occurence of an element, using B.S.



Ans-

It is monotonic.

start = 0

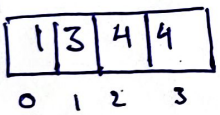
end = 9

$mid = \frac{s+e}{2} = 4$

- if(arr[mid] == target)

  4 == 4 → yes/true → element store → index 4

  ↳ search in left



mid = 3

4 > 3

mid = 2

arr[mid] == target → array store → index → 2

↘ L-S search → e = mid-1

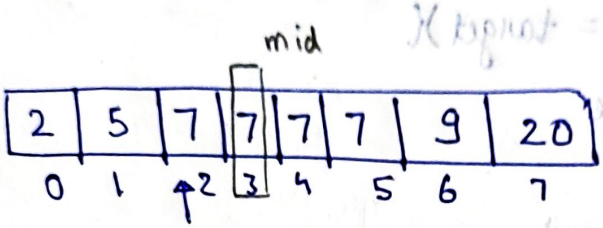
```
int main(){
    vector<int> v{1,3,4,4,4,4,6,7};
    int target=4;
    int indexOfFirstOcc = firstOccurence(v, target);
    cout<<"ans is"<<ans<<endl;
    return 0;
}

int firstOccurence(vector<int>arr,int target){
    int s=0;
    int e=v.size()-1;
    int mid=(s+e)/2
    int ans=-1
    while(s<=e){
        if(arr[mid]==target){
            //ans store and then left
            ans = mid;
            e= mid-1;
        }
        else if(target>arr[mid]){
            //right search
            s= mid+1;
        }
        else if(target<arr[mid]){
            //left search
            e=mid-1;
        }
    mid=(s+e)/2
    }
    return ans;
}
```

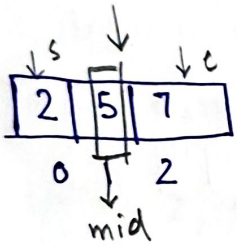
Dry Run

① if (arr[mid] == target)
↳ ans = mid;
↳ e = mid - 1;

② if (target < arr[mid])
e = mid - 1;

③ if (target > arr[mid])
s = mid + 1;

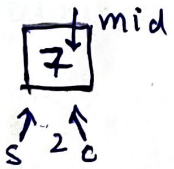




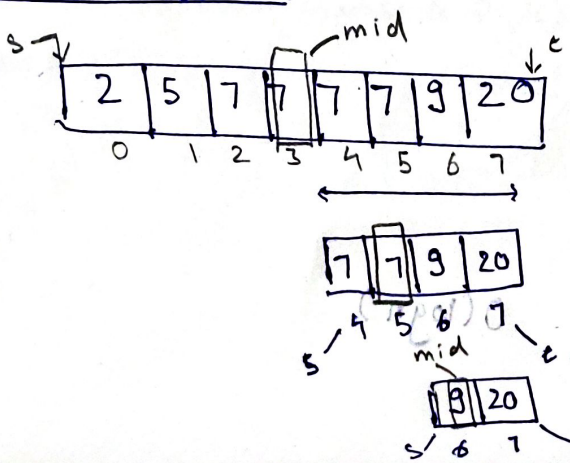
~~target < arr[mid]~~ ✗

~~mid~~ target > arr[mid]

s = mid + 1



Last occurence :-



if (arr[mid] == target)
↳ ans = mid
↳ s = mid + 1

if (target < arr[mid]
L.S ↳ e = mid - 1

if (target > arr[mid]
R.S ↳ s = mid + 1

```
#include <iostream>
#include <vector>
using namespace std;
int lastOcc(vector<int> arr, int target){
    int s = 0;
    int e = arr.size()-1;

    int mid= s+(e-s)/2
    Int ans = -1;

    while(s<=e){

        if(arr[mid]== target){
            //ans store
            ans=mid;
            //right search
            s=mid+1;
        }
        else if(target< arr[mid]){
            // left search
            e=mid-1;
        }
        else if(target >arr[mid]){
            // right search
            s=mid+1;

        }
    mid=s+(e-s)/2
    }
    return ans;
}
int main(){
```

- Time complexity O(logn)

STL functions for first and last occurence :-

lower_bound (v.begin(), v.end(), 20);
                                  ↳ target

Lower bound is used to find the first occurence and it returns an iterator.

Similarly, upper_bound is used to find last occurence.

upper_bound (v.begin(), v.end(), 20)
                                  ↳ Target element.

2) Total number of occurences of element.

i/p — 2, 4, 4, 4, 4, 5, 6

o/p → 4

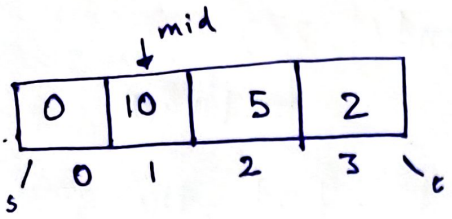
Ans:

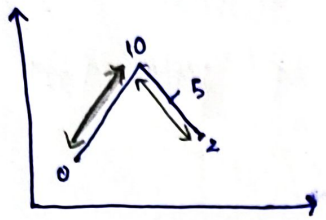
Total occurence = lastOccurence - firstOccurence + 1.

Code:

```
int first = firstOccurence (v, 4);
int last  = lastOccurence (v, 4)
cout << (last - first + 1) << endl;
```

Leetcode: # Peak Index in a Mountain Array:





Brute force →

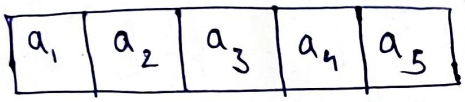
LS → max element

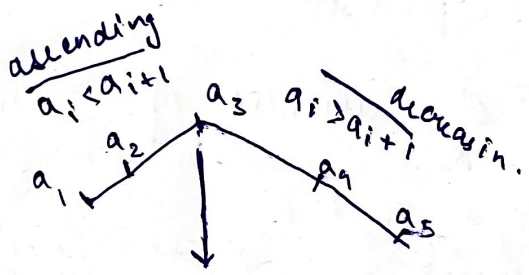
↓

O(n)

BS → O(log n)

optimal



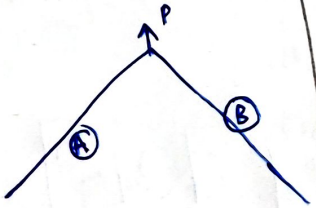


arr[i-1] < arr[i] > arr[i+1]

if (arr[mid] > arr[mid+1]) → mid is line A

if (arr[mid] < arr[mid+1]) → mid element may be a peak element

→ mid element in Line B



Line (A) →

arr[i] < arr[i+1]

(P) → peak element

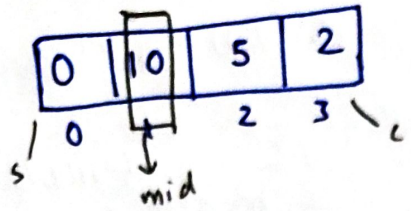
arr[i-1] < arr[i] > arr[i+1]

Line (B) → arr[i] > arr[i+1]

```
if(arr[mid] < arr[mid+1]
        s = mid + 1
else
  {
      e = mid;
  }
```

Code:-

```
class Solution{
public:
    int findPeakIndex(vector<int> arr){
        int s=0;
        int e = arr.size()-1;
        int mid = s+(e-s)/2;
        while(s< e){
            if(arr[mid]< arr[mid+1]){
                //right search
                s= mid+1;
            }else{
                e=mid;
            }
            mid = s + (e-s)/2;
        }
        return s;
    }
    int peakIndexInMountainArray(vector<int>& arr){
        return findPeakIndex(arr);
    }
};
```



s < e

↓

go in loop

if (arr[mid] < arr[mid+1]) — FALSE

↳ R-S

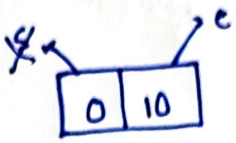
↓

else {

e = mid;

}

return s;
OR
return e;



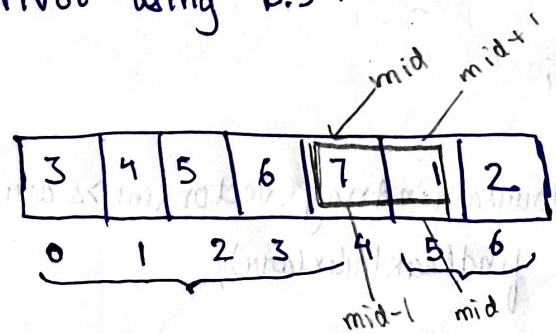
arr[mid] < arr[mid+1] ✓

① Find pivot using Binary search

②

Lec2:-

① Pivot using B.S.

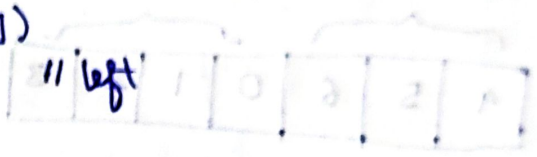


```
if (arr[mid] > arr[mid+1])
        return mid;
```

→ Jiske baad monotonic fun" break krrha hai usse 'pivot'.

```
if (arr[mid-1] > arr[mid])
        return mid-1
```

```
if (arr[s] > arr[mid])
        e = mid-1          // 'left'
elseif (arr[s] < arr[mid])
        s = mid+1          // right
```

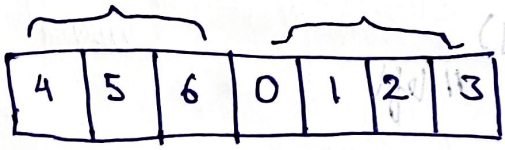


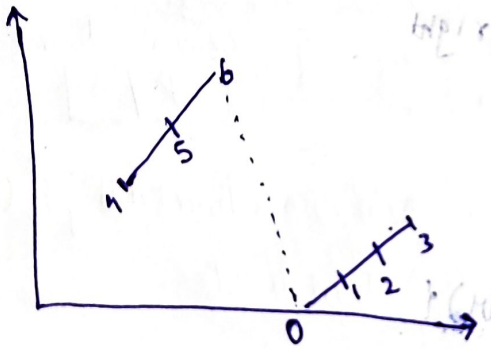
CODE:

```
int findPivot(vector<int> arr){
  int s=0;
  int e= arr.size()-1;
  int e= arr.size()-1;
  int mid = s+(e-s)/2;
  while(s<=e){
      if(arr[mid]>arr[mid+1])
          return mid;
      if(arr[mid-1] > arr[mid])
          return mid-1;
      else
          s= mid+1;
```

```
        mid = s + (e-s)/2;
    }
    return -1;
}
```

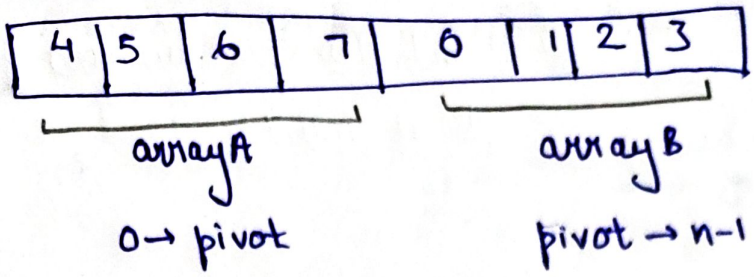
② Search in rotated sorted array.





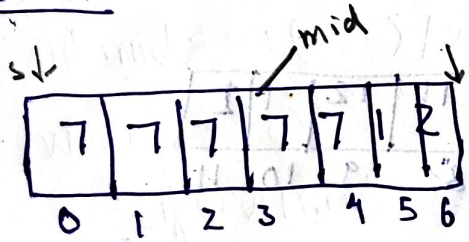
sorted → 0 → pivot

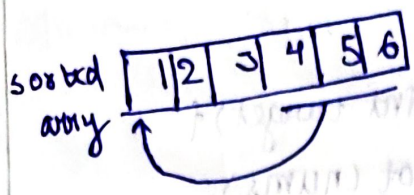
sorted 2 → pivot+1 → n-1

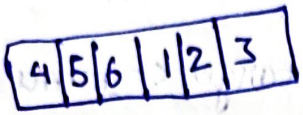



```
int search(vector<int> & nums, int target){
    int pivotIndex = findPivot(nums);
    if(target >= nums[0] && target <= nums[pivot]){
        //search A
        int ans = binarySearch(nums, target, 0, pivot);
        return ans;
    }
    if(target >= nums[pivot+1] && target <=
        //search B
        int ans = binarySearch(nums, target, pivot+1,
                                nums.size()-1);
        return ans;
    }
    return -1;
}
```

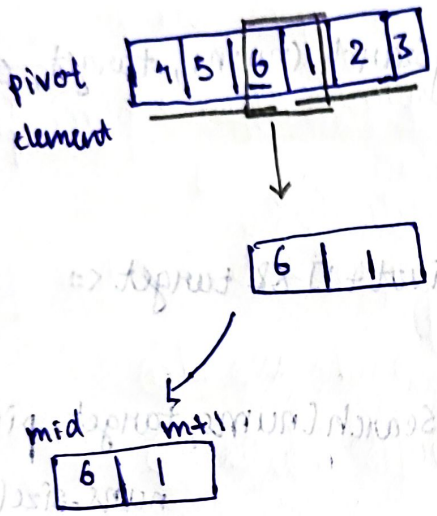
DRun







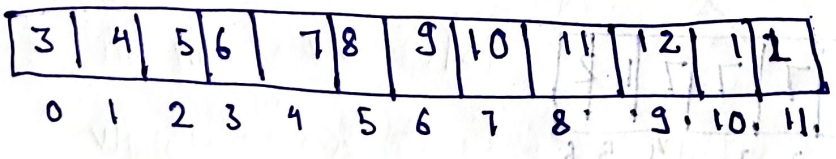
sorted & Rotated

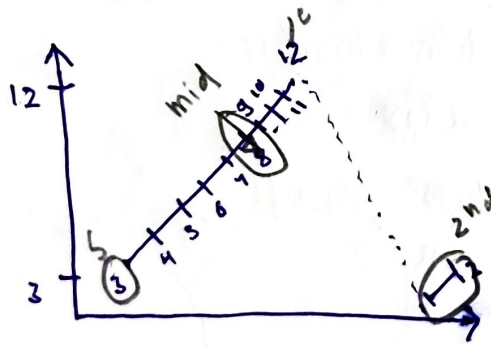


① if arr[mid > arr mid+1)
return mid

② if (arr[mid-1] > arr[mid])
return (mid-1);

Now, check left search or right search:

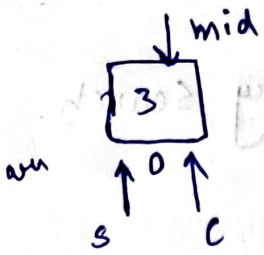




③ if(arr[mid] < arr[s])
s = mid + 1;

else

④ e = mid - 1;



s <= e ✗

↓

need nhi nai

s < e ✓

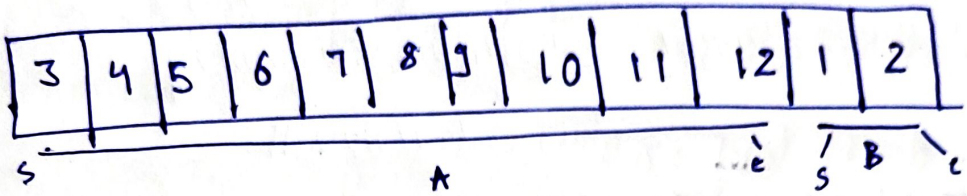
{

returns;

s = mid

or

e = mid



rotated & sorted Arr mei search.

A → (0 to pivot)

B → (pivot+1 → n-1)

3 — 12

↓

2 exist ✗

1 — 2

↓

2 exist ✓

⑤ Square Root of no. using Binary Search.

$\sqrt{12}$ → 0 ⟶ 12

$\sqrt{8}$ — 0 ⟶ 8

$\sqrt{100}$ — 0 ⟶ 100 (search space)

Soln

Algo:

if (mid * mid > target)

↳ L.S

if (mid * mid < target)

① ↳ ans store

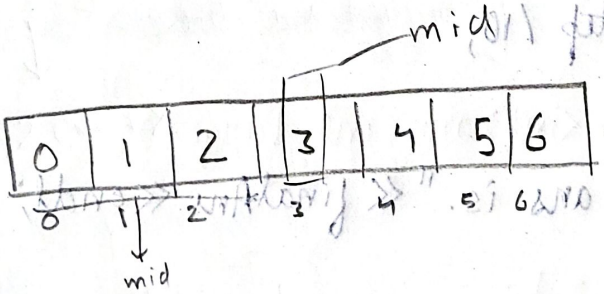
② Right search

if (mid * mid == target)

return mid

```
int findSqrt(int n){
    int s = 0
    int e = n;
    int mid = s + (e-s)/2;
    while(s <= e){
        if(mid * mid == target)
            return mid;
        if(mid * mid > target)
            // LS
            e = mid-1;
    }
```

```
    else{
        //ans store
        ans = mid;
        //R.s
        s = mid+1;
    }
    mid = s+(e-s)/2;
}
return ans;
}
int main(){
    int n;
    cin>>n;
    int ans = findSqrt(n);
    cout<<"Ans is"<<ans<<endl;
    return 0;
}
```



```
int main(){
    int ans = findSqrt(n);
    cout << "Ans is " << ans << endl;
    int precision;
    cout << "Enter no. of floating digit in precision" << endl;
    cin >> precision;
    double step = 0.1
    double finalAns = ans;
    for (int i = 0; i < precision; i++){
        for(double j = finalAns; j * j <= n; j += step){
            finalAns = j;
        }
        step = step / 10;
    }
    cout << "Final ans is." << finalAns << endl;
    return 0;
}
```

Dry Run :-

```
for (int i = 0; i < precision; i++){ //adding new digit
    for (double j = finalAns; j * j <= n; j = j + step){
        finalAns = j; // ans store
    }
    step = step / 10;
}
```

J = 3
3 * 3 = 9 <= 10 → T
J = 3 + 0.1 = 3.1
3.1 * 3.1 = 9.61 <= 10 ✓
i : 3.1 + 0.1 = 3.2
3.2 * 3.2 = 10.24 <= 10 ✗

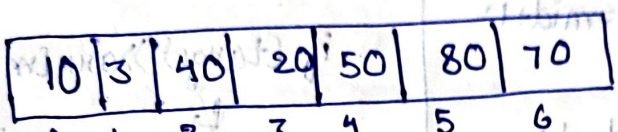
① Search in a nearly sorted array in log time.

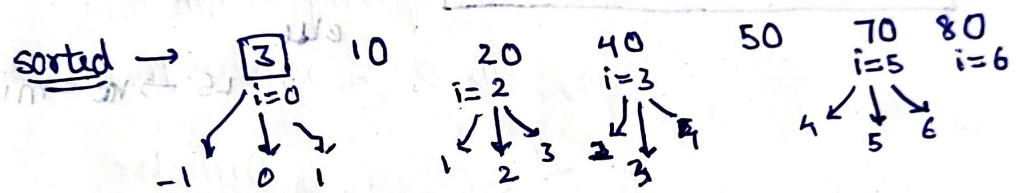
② Division of 2 no. using B.S.

③ Find ODD occuring element using Binary Search.

Lec3:

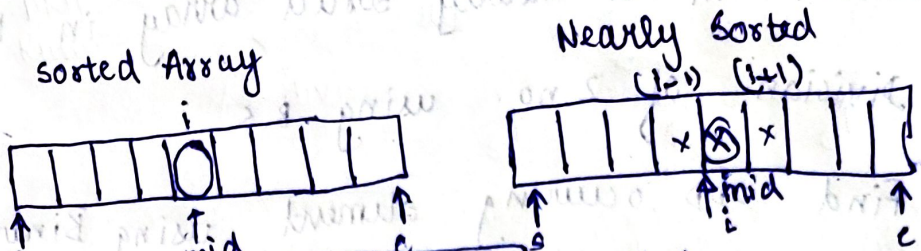
① Binary Search in nearly Sorted Array:-





Nearly sorted array: -

Element
↓
sorted array में
↓
i^{th} index पे hai.
↓
then, in Nearly sorted array में
↓
$(i-1)^{th}$, i^{th}, $(i+1)^{th}$ th indexes




```
if (arr[mid] == target)
    return mid;
if (target > arr[mid])
    ↳ R.S → s = mid+1;
else
    ↳ LS → e = mid-1;
```

```
if (arr[mid] == target)
    return mid;
if (arr[mid-1] == target)
    return mid-1;
if (arr[mid+1] == target)
    return mid+1;
if (target > arr[mid])
    ↳ R.S → s = mid+2;
else
    LS → e = mid-2;
```

CODE:

```
int binarySearch(vector<int> arr, int target){
    int s=0;
    int e = arr.size() - 1;
    int mid = (s + (e-s)) / 2;
    while (s<=e){
        if (arr[mid]==target)
            return mid;
        if (mid-1 >=0 && arr[mid-1]==target)
            return mid-1;
        if (mid+1 < arr.size() && arr[mid+1] == target)
            return mid+1;

            if(target > arr[mid]){ //R.S
                s= mid+2;
            }
```

```
        else {    //L.S
            e = mid - 2;
        }
        mid = s + (e - s) / 2;
    }
    return -1;
}

int main(){
    vector<int> arr{10,3,40,20,50,80,70},
    int target = 40;
    int ans = binarySearch (arr, target);
    cout << " index of " << target << "is" << ans << endl;
    return 0;
}
```

T.C → O(log n)

(2) Divide two no.s using Binary Search.

dividend = 10
divisor = 2
quotient = ?

2 | 10 | 5 Quotient

dividend

10

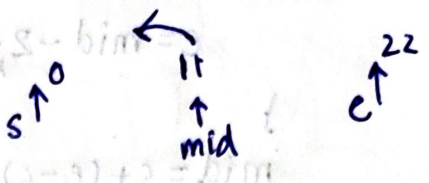
0 remainder

quotient * divisor + remainder = dividend

quotient * divisor <= dividend (CONDITION)

Dry Run:-

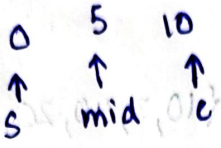
22 ← dividend
7 ← divisor



mid * divisor > dividend

11 * 7 = 77 > 22

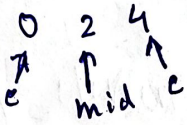
L.S.



→ mid * divis.

5 × 7 = 35 > 22

L.S

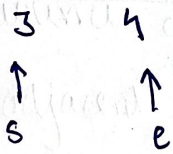


→ mid * divisor

2 × 7 = 14 ≤ 22

↳ ans → store

↳ right search



= 21 ≤ 22

ans → store & RS

4 × 7 = 28 > 22

L.S

s = 4
e = 3

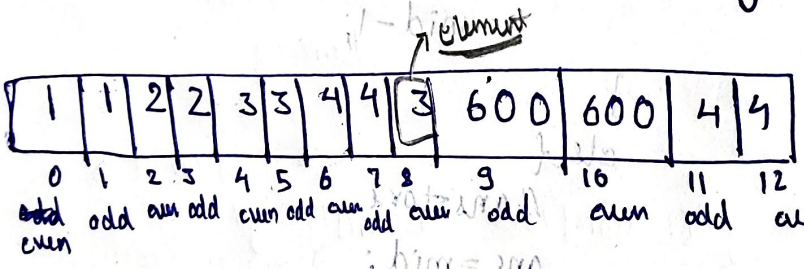
code:

```
int solve(int dividend, int divisor){
    int s=0;
    int e = dividend; // abs(dividend)
    int ans = 0;
    int mid = s+ (e-s)/2;

    while(s<=e){
        // perfect solution
        if(mid* divisor == dividend){
        //abs(                 abs
            return mid;
        }
        // not perfect answer
        if(mid * divisor > dividend){
        //abs                 abs
            // left.s
            e = mid-1;
        }
        else{
            // ans store
            ans = mid;
            // right search
            s = mid+1;
        }
        mid = s+(e-s)/2;
    }
    return ans;
}
int main(){
```

// if(divisor<0 && dividend<0)
|| (divisor>0 && dividend>0))
    return ans;
else
    return -ans;

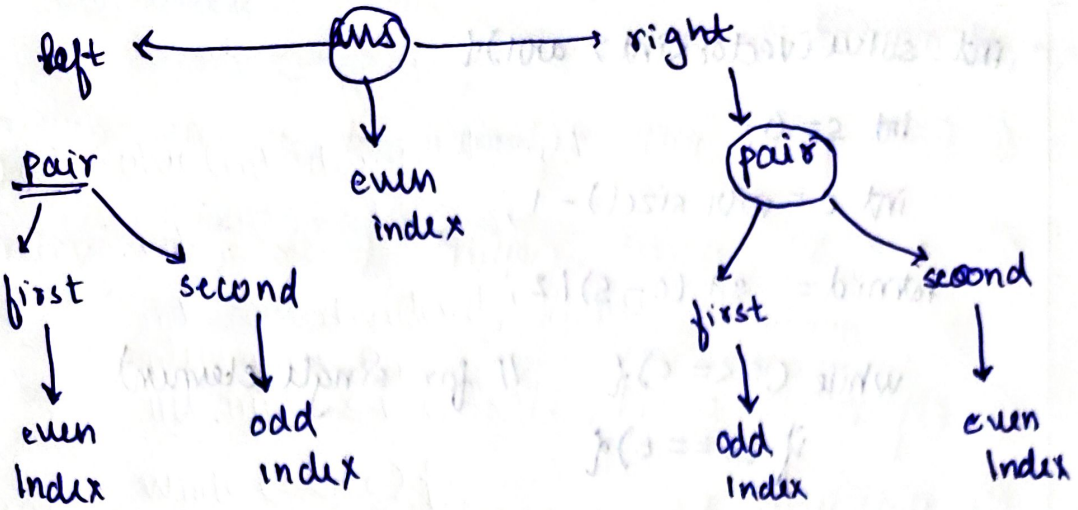
```
int ans= solve();
cout <<"Ans is" <<ans<<endl;
int precision;
cout<<Enter no. of floating digits: "<<endl;
cin>> precision;
double step =0.1;
```

③ Find ODD occurring element in array in B.S..



→ all repeating occurance of element appear in pairs, are not adjacent. (there cannot be more than two consecutive occurence of any element).

→ find element that appears odd no. of times.



Conditions:-

```
s=0;
e=n-1;
mid= s+ (e-s)/2
while(s<=e){
    if(s==e)
        return s;

if(mid % 2 ==0) // even
    // even

else //odd
{

}
```

①

Go to Left | Iam on right side

```
if(arr[mid] == arr[mid+1])
    ↳ R.S → s = mid+2
else
    e= mid
```

Goto R.S | Iam on LS

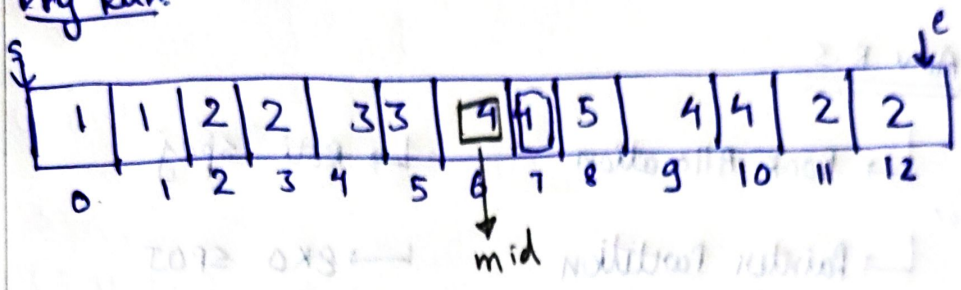
```
if(arr[mid-1] == a[mid])
    ↳ s = mid+1;
```

□ != □ → else
mid-1 (even) | mid (odd)

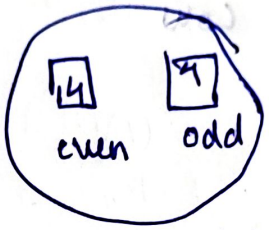
```
int solve(vector<int> arr){
    int s=0;
    int e = arr.size()-1;
    int mid= s+(e-s)/2;
    while(s<=e){    // for single element
        if(s==e){
            return s;
        }
        // 2 cases → mid even or odd
        if(mid%2==0){
            if(arr[mid]==arr[mid+1]){
                s=mid+2;
            }
            else{
                e=mid;
            }
        }else{
            if(arr[mid]==arr[mid-1]){
                s=mid+1;
            }else{
                e=mid-1;
            }
        }
        mid=s+(e-s)/2;
    }
    return -1;
}
```

```
int main(){
    vector<int> arr{1,1,2,2,3,3,4,4,3,600,600,4,4};
    int ans = solve(arr);
    cout << arr[ans] << endl;
    return 0;
}
```

Dry Run



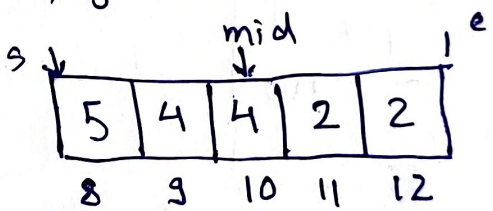
(i)



LS mei kdu ho

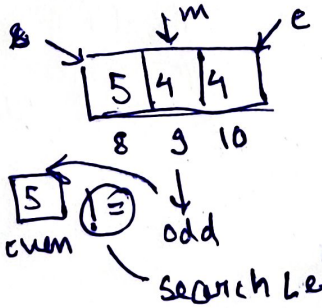
So, go R.S

(ii)



(iii)

mid=10



search Left

e = mid - 1

5

Homework:

1. Find pairs with difference 'k' in an array.
2. Find 'k' closest element to a given value in an array.
3. Exponential search
4. Unbounded BS
5. Adv B.S
   - ↳ Book Allocation
   - ↳ Painter Partition
   - ↳ Aggressive Cow
   - ↳ Roti .spoj
   - ↳ EKO SPOJ