BINARY SEARCH

Basic binary search:

```
class Solution:
    def binarysearch(self, arr, n, k):
        start=0
        end=n-1
        while(start<=end):
            mid= int((start+end)/2)
            if(arr[mid]==k):
                return mid
            elif arr[mid]<k:
                start=mid+1
            else:
                end=mid-1
        return -1
```

Q: Find first and last occurrence of any element in sorted array/ Left most and right most index:

```python
class Solution:
    def indexes(self, v, x):
        start=0
        end=len(v)-1
        left_index=-1
        right_index=-1
        #find left most occurance
        while(start<=end):
            mid=(start+end)//2
            if v[mid]==x:
                left_index=mid
                end=mid-1
            elif v[mid]>x:
                end=mid-1
            else:
                start=mid+1
        #find right most occurance
        start=0
        end=len(v)-1
        while(start<=end):
            mid= (start+end)//2
            if v[mid]==x:
                right_index=mid
                start=mid+1
            elif v[mid]>x:
                end=mid-1
            else:
                start=mid+1
        return left_index,right_index
```

Q: Number of occurrence of any element in sorted array:
Intuition: same as above logic, find left and right most index of that element and then : right_index-left_index +1

```python
class Solution:

    def count(self,arr, n, x):
        start=0
        end=n-1
        left_index=-1
        right_index=-1
        #find left most occurance
        while(start<=end):
            mid=(start+end)//2
            if arr[mid]==x:
                left_index=mid
                end=mid-1
            elif arr[mid]>x:
                end=mid-1
            else:
                start=mid+1
        if left_index==-1:
            return 0   #as no need to check for right occurance
        #find right most occurance
        start=0
        end=n-1
        while(start<=end):
            mid= (start+end)//2
            if arr[mid]==x:
                right_index=mid
                start=mid+1
            elif arr[mid]>x:
                end=mid-1
            else:
                start=mid+1

        return right_index-left_index+1

        |
```

Q: Find no of times the array is rotated:

Intuition: we observe that by finding minimum element in the array, we easily find out how many times array is rotated.

```python
class Solution:
    def findKRotation(self,arr,  n):
        start = 0
        end = n - 1
        while start <= end:
            mid = (start + end) // 2
            if arr[mid] < arr[mid - 1] and arr[mid] < arr[(mid + 1)%n]:
                return mid
            elif arr[mid] > arr[end]:
                start = mid + 1
            else:
                end = mid - 1
        return 0
```

Q: Find out the element in rotated array:

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        start=0
        end=len(nums)-1
        rotation=0
        #find out the min element to know how many times array is rotated
        while(start<=end):
            mid=(start+end)//2
            if(nums[mid]<nums[mid-1] and nums[mid]<nums[(mid+1)%len(nums)]):
                rotation=mid
                break
            elif nums[mid]>nums[end]:
                start=mid+1
            else:
                end=mid-1
        s=rotation
        e=len(nums)-1
        if(target>=nums[s] and target<=nums[e]):
            while(s<=e):
                mid=(s+e)//2
                if nums[mid]==target:
                    return mid
                elif nums[mid]<target:
                    s=mid+1
                else:
                    e=mid-1
        else:
            s=0
            e=rotation-1
            while(s<=e):
                mid=(s+e)//2
                if nums[mid]==target:
                    return mid
                elif nums[mid]<target:
                    s=mid+1
                else:
                    e=mid-1
        return -1
```

Q: Find floor of a given number:

```
#Complete this function
def findFloor(self,A,N,X):
    ans=-1
    s=0
    e=N-1
    while(s<=e):
        mid=(s+e)//2
        if A[mid]==X:
            ans=mid
            break
        elif A[mid]>X: #not possible candidate coz greater than X
            e=mid-1
        else:
            ans=mid        #possible candidates
            s=mid+1

    return ans
```

Q: Single element in sorted array:

```
class Solution:
    def singleNonDuplicate(self, nums: List[int]) -> int:
        n=len(nums)
        s=0
        e=n-1
        if n==1:
            return nums[0]
        #first occurance always on odd index and secong index on even index
        while(s<=e):
            mid=(s+e)//2
            #if element's prev and next are different, then that element occurs single time
            if nums[mid]!=nums[mid-1] and nums[mid]!=nums[(mid+1)%n]:
                return nums[mid]
            #if mid is odd and second occurance then till now everything good , move right
            elif mid%2!=0 and nums[mid]==nums[mid-1]:
                s=mid+1
            #if mid is even and first occurance then also everything good, move right
            elif mid%2==0 and nums[mid]==nums[(mid+1)%n]:
                s=mid+1
            #else then move left
            else:
                e=mid-1
```

Q: Sort the almost sorted array: (IMP)(Flipkart interview)

Q: Search in almost sorted array: (Dell interview)

Q: Median of Two Sorted Arrays

```
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
        n1=len(nums1)
        n2=len(nums2)
        if n1 > n2:
            return self.findMedianSortedArrays(nums2, nums1)
        low=0
        high=n1
        med=(n1+n2+1)//2

        while(low<=high):
            cut1=(low+high)//2
            cut2=med-cut1

            if cut1==0:
                left1=float('-inf')
            else:
                left1=nums1[cut1-1]

            if cut2==0:
                left2=float('-inf')
            else:
                left2=nums2[cut2-1]

            if cut1==n1:
                right1=float('inf')
            else:
                right1=nums1[cut1]

            if cut2==n2:
                right2=float('inf')
            else:
                right2=nums2[cut2]

            if(left1<=right2 and left2<=right1):
                if (n1+n2)%2==0:
                    return (max(left1,left2)+min(right1,right2))/2.0
                else:
                    return max(left1,left2)
            elif (left1>right2):
                high=cut1-1
            else:
                low=cut1+1
        return 0.0
```

Q: Metric median:

Intuition: find the min and max from the matrix and as we know we can find the median between this range only.min can be found in first col and max in last col

We know that the this matrix is odd , ie r*c =odd always , so no of element before median is equal to (r*c)/2. And then we can use BS row wise to calculate the no of element less than equal to median.

If (r*c)/2<=count , that means we have to reduce the max , so max=mid-1

Otherwise min=mid+1, atlast min contain the result.

```python
def countLessThanMid(arr, mid):
    count = 0
    low = 0
    high = len(arr) - 1
    while low <= high:
        m = (low + high) // 2
        if arr[m] <= mid:
            count = m + 1
            low = m + 1
        else:
            high = m - 1
    return count

class Solution:
    def findMedian(self, A):
        row = len(A)
        col = len(A[0])
        minn = float('inf')
        maxx = float('-inf')

        for i in range(row):
            minn = min(minn, A[i][0])
            maxx = max(maxx, A[i][col - 1])

        desired = (row * col) // 2
        while minn <= maxx:
            mid = (maxx + minn) // 2
            count = 0
            for i in range(row):
                count += countLessThanMid(A[i], mid)
            if count <= desired:
                minn = mid + 1
            else:
                maxx = mid-1
        return minn
```

Q: Allocate books:

```python
class Solution:
    # @param A : list of integers
    # @param B : integer
    # @return an integer
    def books(self, A, B):
        n = len(A)
        if n < B:
            return -1

        start = max(A)
        end = sum(A)
        ans = float('inf')

        while start <= end:
            mid = (start + end) // 2
            count = 1
            curr_sum = 0
            for i in range(n):
                if curr_sum + A[i] > mid:
                    count += 1
                    curr_sum = A[i]
                else:
                    curr_sum += A[i]

            if count > B:
                start=mid+1   #agar count kamm h to bcche ko jyde pages allocate krne ki jrut h
            else:
                ans = mid   #trying to minimize all possible value
                end = mid - 1
        return ans
```

Q: Aggressive cows:

TC: O(nlogn)

```python
def aggressiveCows(stalls, k):
    stalls.sort()
    n=len(stalls)
    low=1
    high=stalls[n-1]-stalls[0]

    def isPossible(mid):
        count=1
        lastplacedcow=stalls[0]
        for i in range(n):
            if stalls[i]-lastplacedcow>=mid:
                count+=1
                if count==k:
                    return True
                lastplacedcow=stalls[i]
        return False

    while(low<=high):
        mid=(low+high)//2
        if isPossible(mid):
            res=mid
            low=mid+1
        else:
            high=mid-1
    return res
```