Binary search to find a target

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int low=0;
        int high=nums.size()-1;
        while(low<=high){
            int mid = low+(high-low)/2;
            if(nums[mid]==target){
                return mid;
            }
            else if(nums[mid]<target){
                low = mid+1;
            }
            else{
                high = mid-1;
            }
        }
        return -1;
    }
};
```

Floor&Ceil:

```
int findFloor(int arr[], int n, int x) {
            int low = 0, high = n - 1;
            int ans = -1;

            while (low <= high) {
                        int mid = (low + high) / 2;
                        // maybe an answer
                        if (arr[mid] <= x) {
                                    ans = arr[mid];
                                    //look for smaller index on the left
                                    low = mid + 1;
                        }
                        else {
                                    high = mid - 1; // look on the right
                        }
            }
            return ans;
}

int findCeil(int arr[], int n, int x) {
            int low = 0, high = n - 1;
            int ans = -1;

            while (low <= high) {
                        int mid = (low + high) / 2;
                        // maybe an answer
                        if (arr[mid] >= x) {
                                    ans = arr[mid];
                                    //look for smaller index on the left
                                    high = mid - 1;
                        }
                        else {
                                    low = mid + 1; // look on the right
                        }
            }
            return ans;
}
```

Count Occurences of a Number in sorted array:
Formula: Last Occurence-First Occurence+1

```
class Solution{
public:
            /* if x is present in arr[] then returns the count
                        of occurrences of x, otherwise returns 0. */
                        int firstOccurence(int nums[], int n,int target){
        int fo=-1;
        int low=0;
        int high=n-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target){
                fo=mid;
                high=mid-1;
            }
            else{
                if(nums[mid]<target){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
        return fo;
    }
    int lastOccurence(int nums[], int n,int target){
        int lo=-1;
        int low=0;
        int high=n-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target){
                lo=mid;
                low=mid+1;
            }
            else{
                if(nums[mid]<target){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
        return lo;
    }
            int count(int arr[], int n, int x) {
                int fo=firstOccurence(arr,n,x);
        int lo=lastOccurence(arr,n,x);
        if(fo==-1||lo==-1){
            return 0;
        }
        return lo-fo+1;
            }
};
```

Aggressive Cows:

```
class Solution {
public:

    bool canCowsBePlaced(int gap,int n,vector<int>&stalls,int k){
        int cows=1;
        int pos=0;
        for(int i=1;i<n;i++){
            if(stalls[i]-stalls[pos]>=gap){
                cows++;
                pos=i;
            }
        }
        if(cows>=k){
            return true;
        }
        else{
            return false;
        }
    }
    int solve(int n, int k, vector<int> &stalls) {
    sort(stalls.begin(),stalls.end());
    int low=1;
    int high=stalls[n-1]-stalls[0];
    int maxGap=0;
    while(low<=high){
        int mid=(low+high)/2;
        if(canCowsBePlaced(mid,n,stalls,k)==true){
            maxGap=mid;
            low=mid+1;
        }
        else{
            high=mid-1;
        }
    }
    return maxGap;
    }
};
```

Smallest Divisor Given a Threshold:

```
class Solution {
public:
    bool canBeSmallestDivisor(vector<int>nums,int n,int mid,int threshold){
        int sum=0;
        for(int i=0;i<n;i++){
            if(nums[i]%mid!=0){
                sum+=1;
            }
            sum+=nums[i]/mid;
        }
        return sum<=threshold;
    }
    int smallestDivisor(vector<int>& nums, int threshold) {
        int n=nums.size();
        int lo=1;
        int hi=*max_element(nums.begin(),nums.end());
        int ans=-1;
        while(lo<=hi){
            int mid=(lo+hi)>>1;
            if(canBeSmallestDivisor(nums,n,mid,threshold)==true){
                ans=mid;
                hi=mid-1;
            }
            else{
                lo=mid+1;
            }
        }
        return ans;
    }
};
```

Search in a 2D Matrix:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
    int N=matrix.size();
    int i=0;
            int M=matrix[i].size();
    int j=(N*M)-1;
            while(i<=j){
                        int mid=i+(j-i)/2;
                        if(matrix[mid/M][mid%M]==target){
                                    return true;
                        }
                        else{
                                    if(matrix[mid/M][mid%M]<target){
                                                i=mid+1;
                                    }
                                    else{
                                                j=mid-1;
                                    }
                        }
            }
            return false;
    }
};
```

Search in a Row-wise and column-wise sorted matrix:

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int n=matrix.size();
        if(n==0){
            return false;
        }
        int m=matrix[0].size();

        int lo=0;
        int hi=m-1;
        while(lo<n&&hi>=0){
            if(matrix[lo][hi]==target){
                return true;
            }
            else{
                if(matrix[lo][hi]>target){
                    hi--;
                }
                else{
                    lo++;
                }
            }
        }
        return false;
    }
};
```

Koko Eating Bananas

```
class Solution {
public:
    bool isPossible(vector<int>& piles, int h,int speed){
        long long int hours=0;
        for(int x:piles){
            hours+=x/speed;
            if(x%speed!=0){
                hours++;
            }
        }
        return hours<=h;
    }
    int minEatingSpeed(vector<int>& piles, int h) {
        int lo=1;
        int hi=*max_element(piles.begin(),piles.end());
        int ans=0;
        while(lo<=hi){
            int mid=lo+(hi-lo)/2;
            if(isPossible(piles,h,mid)){
                ans=mid;
                hi=mid-1;
            }
            else{
                lo=mid+1;
            }
        }
        return ans;
    }
};
```

Capacity to ship packages withing D Days:

```
class Solution {
public:
    bool isPossible(int capacity,vector<int>&weights,int days){
        int d=1;
        int sum=0;
        for(int i=0;i<weights.size();i++){
            if(sum+weights[i]>capacity){
                sum=weights[i];
                d++;
            }
            else{
                sum+=weights[i];
            }
        }
        return d<=days;
    }
    int shipWithinDays(vector<int>& weights, int days) {
        int lo=*max_element(weights.begin(),weights.end());
        int hi=accumulate(weights.begin(),weights.end(),0);
        int ans=1e9;
        while(lo<=hi){
            int mid=(lo+hi)/2;
            if(isPossible(mid,weights,days)==true){
                ans=mid;
                hi=mid-1;
            }
            else{
                lo=mid+1;
            }
        }
        return ans;
    }
};
```

Search in a rotated sorted array:

```
class Solution {
public:
    int searchInARotatedSortedArray(vector<int>nums,int target){
        int i=0;
        int j=nums.size()-1;
        int pos=-1;
        while(i<=j){
            int mid=i+(j-i)/2;
            if(nums[mid]==target){
                return mid;
            }
            else{
                if(nums[i]<=nums[mid]){
                    if(nums[i]<=target&&target<=nums[mid]){
                        j=mid-1;
                    }
                    else{
                        i=mid+1;
                    }
                }
                else if(nums[mid]<=nums[j]){
                    if(nums[mid]<=target&&target<=nums[j]){
                        i=mid+1;
                    }
                    else{
                        j=mid-1;
                    }
                }
            }
        }
        return pos;
    }
    int search(vector<int>& nums, int target) {
        return searchInARotatedSortedArray(nums,target);
    }
};
```

Row with max number of ones:

```
class Solution {
public:
    int firstOccurence(vector<int>&nums,int target){
        int fo=-1;
        int low=0;
        int high=nums.size()-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target){
                fo=mid;
                high=mid-1;
            }
            else{
                if(nums[mid]<target){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
        return fo;
    }
    int lastOccurence(vector<int>&nums, int target){
        int lo=-1;
        int low=0;
        int high=nums.size()-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(nums[mid]==target){
                lo=mid;
                low=mid+1;
            }
            else{
                if(nums[mid]<target){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
        return lo;
    }
    vector<int> rowAndMaximumOnes(vector<vector<int>>& mat) {
        vector<int>ans;
        int n=mat.size();
        int m=mat[0].size();
        int ansrow=0;
        int maxOnesCount=0;
        for(int i=0;i<n;i++){
            int count=0;
            //bruteforce
            // for(int j=0;j<m;j++){
            //     if(mat[i][j]==1){
            //         count++;
            //     }
            // }
            //optimised

            vector<int>row=mat[i];
            sort(row.begin(),row.end());
            int fo=firstOccurence(row,1);
        int lo=lastOccurence(row,1);
        cout<<lo<<","<<fo<<endl;
          if (fo != -1 && lo != -1) {
                count = (lo - fo) + 1;
         }
            if(count>maxOnesCount){
                ansrow=i;
                maxOnesCount=count;
            }
        }
        ans.push_back(ansrow);
        ans.push_back(maxOnesCount);
        return ans;
    }
};
```