Binary Search

- Array should be Sorted

(I) Iterative Approach : O(log n)

```
int f(int a[], int n, int x)
{
    int l=0, h=n-1;
    while(l<=h)
    {
        int mid=(l+h)/2;
        if(a[mid]==x)
            return mid;
        else if(x>a[mid])
            l=m+1;
        else
            h=m-1;
    }
    return -1;
}
```

(II) Recursive:

```
int f(vector<int> v, int t, int l, int h)
{
    if(l>h) return -1;
    int m=(l+h)/2;
    if(v[m]==t)
        return m;
    else if(t>v[m])
        return f(v, t, m+1, h);
    else
        return f(v, t, l, m-1);
}
```

Auxillary space O(log n)
Time complexity O(log n)

अगर array sorted है तो एक बार सोचो binary search के बारे में

(III) Index of first Occurence :

I/P → {1, 10, 10, 10, 20, 20} x = 20 → lower_bound() function can also be used

O/P → 4

Naive → O(n) , efficient → O(log n)

```
int f(int a[], int l, int h, int x)
{
    while(l<=h)
    {
        int m=(l+h)/2;
        if(x>a[m])
            l=m+1;
        else if(x<a[m])
            h=m-1;
        else
        {   if(m==0 || a[m]!=a[m-1])
                return m;
            else
                h=m-1;
        }
    }
    return -1;
}
```

(IV) Index of last Occurence :

I/P : {1, 10, 10, 20, 20} x = 20

O/P : 4

- Naive → O(n) , efficient → O(log n)
- upper_bound() function can also be used.

```
int f(int a[], int n, int x)
{
    int l=0, h=n-1;
    while(l<=h)
    {
        int m=(l+h)/2;
        if(a[m]
        if(x>a[m])
            l=m-1;
        else if(x<a[m])
            h=m+1;
        else
        {   if(mid m==n-1 || a[m]!=a[m+1])
                return m;
            else
                l=m+1;
        }
    }
    return -1;
}
```

⇒ lower_bound() & upper_bound(); O(log n)

a) lower_bound():
it returns an iterator the first element equal to or greater than a given value in a sorted array/vector.

b) upper_bound():
it returns an iterator to the first element greater than a given value in a sorted array/vector.

Iterator in C++ : it is used to point at a memory address in STL containers

↓

Array/vector etc.

Example

```
vector<int> v = {1,4,7,7,7,10,45,60};
auto a = lower_bound(v.begin(), v.end(), 90);
auto b = ________(________, 7);
auto c = ________(________, 8);
cout<< a-v.begin() << " " << b-v.begin() << " " << c-v.begin() << endl;

int d = upper_bound(v.begin(), v.end(), 90) - v.begin();
int e = ________(________, 7) ________;
int f = ________(________, 8) ________;

cout<<d<< " " <<e<< " " << f;
```

Output : 8 2 5
8 5 5

⇒ binary_search() : → Array should be sorted
return true(1) if element is present, else false(0);

```
vector<int> v = {2,8,9,10,12};
bool a = binary_search(v.begin(), v.end(), 9);
cout << a;
```

→ Binary search

O/P → 1

(V) Number of Occurrence : → Given an sorted array

I/P → {1,1,2,2,2,2,3} k=2

O/P → 4

```
int f(int a[], int n, int k){
    int b = lower_bound(a, a+n, k) - a;
    int c = upper_bound(a, a+n, k) - a;
    return c-b;
}
```

(VI) **Note**: while using upper_bound() & lower_bound() in array remember of the edge case, index should not point outside the array otherwise Runtime error will occur.

**

(VII) **N^th Root of a M**: $\sqrt[n]{m}$ (n^th root of m)

I/P → n=3, m=27 $(27)^{\frac{1}{3}}$
O/P → 3

I/P → n=3, m=29
O/P → 3.07232

```
int main(){
    int n,m;
    cin>>n>>m;
    getNthRoot(n, m);
    return 0;
}
void getNthRoot(int n, int m)
{
    double l=1, h=m, eps=1e-6;
    while((h-l)>eps)          // eps → calculating till 6 decimal places
    {
        double mid=(l+h)/2.0;
        if(multiply(mid,n)<m)
            l=mid;
        else
            h=mid;
    }
    cout<<l<<" "<<h<<endl;
    // just to check
    cout<<pow(m,(double)(1.0/(double)n));
}
```

```
double multibly(double num, int n)
{
    double ans = 1.0;
    for(int i=1; i<=n; i++)
        ans = ans * num;
    return ans;
}
```

I/P → 3 27	3 29
O/P → 3 3	3.207253 3.207253
3	3.207253

TC → $N \times \log_2(M \times 10^d)$ → d decimal places

Mx10 : [1, 1.1, 1.2, 1.3 ——— 2, 2.1, 2.2 ——— 3] for [1,3] & one decimal place

(1 to number, 2 to number)

to number when one decimal place

loo ——— duo ———

so on

∴ Mx10

eg: for n=3, m=27

Mx10 = 270

**

(VII) Matrix Median -

- find median of 2D matrix
- eg: I/P [[1,3,5], [2,6,9], [3,6,9]] O/P → 5

→ 1, 2, 3, 3, (5) 6, 6, 9, 9

median

```
int findMedian(vector<vector<int>> v)
{
    int l=1, h=1e9;
    while(l<=h)
    {
        int mid=(l+h)/2;
        int c=f(v,mid);
        if(c<=(v.size()*v[0].size())/2)
            l=mid+1;
        else
            h=mid-1;
    }
    return l;
}
int f(vector<vector<int>> &v, int mid)
{
    int c=0;
    for(int i=0;i<v.size();i++)
    {
        int l=0, h=v[i].size()-1;
        while(l<=h)
        {
            int md=(l+h)/2;
            if(v[i][md]<=mid)
                l=md+1;
            else
                h=md-1;
        }
        c=c+l;
    }
    return c;
}
```

→ upper_bound() function implementation.

matrix

$\begin{bmatrix} [1,3,5] \\ [2,6,9] \\ [3,6,9] \end{bmatrix}$ → 1, 2, 3, 3, 5, 6, 6, 9, 9

$2^{32} \approx 10^9$

TC → $\log_2(2^{32}) \times N \times \log_2 M$ → in every row (N) we are implementing binary search (upper_bound())

$\log_2(2^{32})$ → search space $N \times \log_2 M$ → # of rows

$TC = (32 \times N \times \log_2 M)$

$SC \rightarrow O(1)$

(VIII) Single Element in Sorted Array :

- every element appear twice, except one find it

a) Using XOR (O(n))

b) Using Binary Search (O(log n))

logic

1, 1, 2, 3, 3, 4, 4, 8, 8 n=9

0 1 2 3 4 5 6 7 8

num

repeating element start from even index on left side of num(2)

repeating element start from odd index on right side of num(2)

* num ≠ left & right element ≠ to num

```
int ff(vector<int> &v)
{
    int n=v.size();
    if(n==1)
        return v[0];
    else if(v[0]!=v[1])
        return v[0];
    else if(v[n-1]!=v[n-2])
        return v[n-1];
    int l=0, h=n-1;
    while(l<=h)
    {
        int m=(l+h)/2;
        if(v[m]!v[m-1] && v[m]!v[m+1])
            return v[m];
        else if((m%2==0 && v[m]==v[m+1]) ||
                (m%2==1 && v[m]==v[m-1]))
            l=m+1;
        else
            h=m-1;
    }
    return -1;
}
```

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

m even & (v[m]==v[m+1])
∴ l=m+1

1 1 2 2 3 4 5
0 1 2 3 4 5 6

m odd & v[m]==v[m-1]
∴ l=m+1

(IX) Search in Rotated Sorted Array

- distinct element — O(log n)
- I/P → [4,5,6,7,0,1,2] t=3
  O/P → -1

I/P → [3,5,7,9,1,2] t=7
O/P → 2
return index

```
int f(vector<int>& v, int t)
{
    int l=0, h=v.size()-1;
    while(l<=h)
    {
        int m=(l+h)/2;
        if(v[m]==t)
            return m;
        else if(v[l]<=v[m])          // left half sorted
        {
            if(t>=v[l] && t<=v[m])   // checking if element lies in b/w left half
                h=m-1;
            else
                l=m+1;
        }
        else                         // right half sorted
        {
            if(t>=v[m] && t<=v[h])
                l=m+1;
            else
                h=m-1;
        }
    }
    return -1;
}
```

→ Property:

[4, 5, 6, 7, 0, 1, 2]

(4, 5, 6, 7 → sorted; 0, 1, 2 → sorted)

[4, 5, 6, 7, 1, 2, 3]

m

→ m के left में sorted है या m के right में definitely unsorted होगा

(X) Median of Two Sorted Arrays of Different Size :

I/P a1[] → 1 3 4 7 10 12
a2[] → 2 3 6 15
O/P → 5

1 2 3 3 4 6 7 10 12 15

$\frac{(4+6)}{2} = 5$

(I) Merge Sort Merge method can be used (2 pointer approach)

O(n1+n2)

1 3 4 7 10 12 — i
2 3 6 15 — j

which ever is smaller take it

1 2 3 3 4 6 7 10 12 15 → store them in new array then return median

SC → O(n+m)

(or)

TC → O(n1+n2) just maintain count variable to remove extra way of space

(the merge 2-pointer approach here also)

1 2 3 3 4 6
count → 1 2 3 4 5 6

when count reaches 5 and 6 store the value at that point here $\frac{4+6}{2} = 5$

(II)

a1[] → 1 3 4 7 10 12
a2[] → 2 3 6 15

(5) (5)

⇒ 1 2 3 3 4 | 6 7 10 12 15

l1 r1

picking sorted element from a1[] → 1 3 4 7 | 10 12
l2 r2
→ 2 | 3 6 15

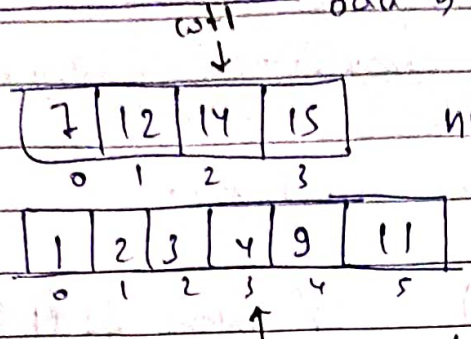
picking ——— a2[]

r2

l1 <= r2
l2 <= r1

1 3 4 | 7 10 12
2 3 | 6 15

- median when (n_1+n_2) is even is $\frac{\max(l_1, l_2) + \min(r_1, r_2)}{2}$
- median ——— odd is $\max(l_1, l_2)$



$l=0, h=4$

$cut1 = \frac{0+4}{2} = 2$

$cut2 = (5-2) = 3$

$\left(\frac{n1+n2+1}{2}\right)$

7, 12 | 14, 15

1, 2, 3 | 4, 9, 11

↓↓ 12 < 4 ✗ ↑↑

$h = cut1 - 1;$

$l=0, h=1$

similarly ———

```
double findmedian(vector<int> &v1, vector<int> &v2)
{
    int n1 = v1.size(), n2 = v2.size();
    if(n2<n1)
        return findmedian(v2, v1);
    int l=0, h=n1;
    while(l<=h)
    {
        int cut1 = (l+h)/2;
        int cut2 = (n1+n2+1)/2 - cut1;

        int l1 = cut1==0? INT_MIN : v1[cut1-1];
        int l2 = cut2==0? INT_MIN : v2[cut2-1];
        int r1 = cut1==n1? INT_MAX : v1[cut1];
        int r2 = cut2==n2? INT_MAX : v2[cut2];
```

we always take first array with min size

```
if(l1<=r2 && l2<=r1)
{
    if((n1+n2)%2==0)
        return (max(l1,l2)+min(r1,r2))/2;
    else
        return max(l1,l2);
}
else if(l1>r2)        // means l1 & r2 TT
    h=cut1-1;
else
    l=cut1+1;
}
return 0.0;
}
```

TC → O(min(n1, n2)) SC → O(1)

(XI) Kth element in two Sorted Array:

I/p a1 [2, 3, 6, 7, 9]
a2 [1, 4, 8, 10] K=7 ⇒ 1 2 3 4 6 7 (8) 9 10
o/p → 8

a) Merging also of Merge Sort (2-pointer approach maintain count variable for SC → O(1)

b) Using Binary Search: TC → O(min(n,m)) SC → O(1)
Same approach as of previous Question

```
int KthElement(int a[], int b[], int n, int m, int k)
{
    if(m<n)
        return KthElement(b, a, m, n, k);
```

```
int l = max(0, k-m), h = min(k, n);
while (l <= h)
{
    int cut1 = (l+h)/2;
    int cut2 = k - cut1;

    int l1 = cut1==0 ? INT_MIN : a[cut1-1];
    int l2 = cut2==0 ? INT_MIN : b[cut2-1];
    int r1 = cut1==n ? INT_MAX : a[cut1];
    int r2 = cut2==m ? INT_MAX : b[cut2];

    if (l1<=r2 && l2<=r1)
        return max(l1, l2);
    else if (l1 > r2)
        h = cut1 - 1;
    else
        l = cut1 + 1;
}
return 0;
}
```

* int l = max(0, k-m), h = min(k, n);

a

7	12	14	15
0	1	2	3

b

1	2	3	4	9	11
0	1	2	3	4	5

~~this array is only for this calculation~~

when k = 3

what can be l & h ——— 0

l → either don't take any element = 0

h → take ~~all~~ k element = 3

l = max(0, k-m)

when k = 7

l → we have to take some element from array a i.e (k-m) element

h → take all element (n)

Search space finding is very important in DS

classmate Date Page

**

(XII) Allocate Books :

a[] = [12, 34, 67, 90] student = 2

o/p → 113

- There are n books with certain no. of pages
- Find max. no. of pages allocated is minimum

Condition

(i) A Book will be allocated in whole (no pages splitting) to one student

(ii) Each student must get minimum of 1 Book

★ (iii) Allotment of Book should be continuous manner

[12, 34, 67, 90] S = 2

max

191 ←	S1: 12	S2: 34, 67, 90 → 191
157 ←	S1: 12, 34	S2: 67, 90 → 157
113 ←	S1: 12, 34, 67	S2: 90

113

min(191, 157, 113) = 113 Ans Maximum, Minimum का

a) Recursion Think of day partition 12 | 34, 67, 90 or 12, 34 | 67, 90 → day partition here

b) DP → O(N²)

c) Binary Search : Tell this to interviewer O(log N x N)

S(→ O(1)

Voimp

[12, 34, 67, 90] n=4, student = 2

[12 (low) 107 (mid, answer: maximum no. of pages) 203 (high)]

We go in array and allot pages to student such number any student can't have more than 107 pages.

S1 → 12+34+67 > 107 ∴ S1 = 12+34
S2 → 67+90 > 107 ∴ S2 = 67
S3 → 90 ∴ S3 = 90

students

we observe student_count > student
∴ this can't be our solution ∴ we move low = mid+1

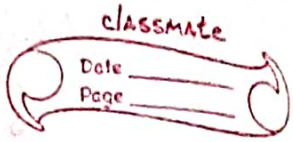
[108 (low) 155 (m.) 203 (h.)]

since student_count <= student
∴ it can be one of my answer
now we have to minimize it ∴ h = mid-1;

S1 → 12+34+67
S2 → 90

[108 131 154] h=m-1
[108 119 130] h=m-1
[108 113 118] h=m-1
[108 110 112] low=m+1

S1 = 12+34, S2 = 67, S3 = 90



→ [111 111 112]

l = m+1

S1 = 12+34
S2 = 67
S3 = 90

→ [112 112 112]

l = m+1

S1 = 12+34
S2 = 67
S3 = 90

l = 113, h = 112

l > h while loop stop

→ last is answer

main → int books(vector<int> &A, int S)

```
{
    if(S > A.size()) return -1;
    int l = max element(v.begin(), v.end());   // or min
    int h = accumulate(v.begin(), v.end(), 0), ans = -1;
    while(l <= h)
    {
        int m = (l+h)/2;
        if(AllocIsPossible(v, m, s))
        {
            ans = m;
            h = m-1;
        }
        else
            l = m+1;
    }
    return ans; // return l
}
```

```
bool AllocIsPossible(vector<int> v, int mid, int student)
{
    int stu=0, page_alloted=0;   // we allot page to student s1, s2, s3 ...
    for(int i=0; i<v.size(); i++)
    {
        if((page_alloted + v[i]) > mid)
        {
            stu++;   // when stu increase by 1 means we have alloted page/book to stu1 now it turns of stu2
            page_alloted = v[i];
        }
        else
            page_alloted += v[i];
    }
    if(stu > student)
        return 0;
    return 1;
}
```

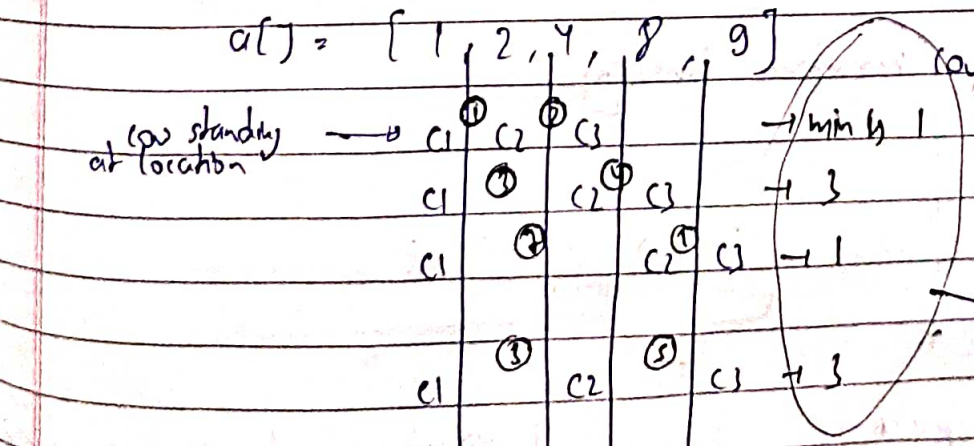
(XII) Aggressive Cows : of stalls

- Given an array of integers in unsorted order & no. of cows
- Find largest min distance

↓ [2, n]

a[] = [1, 2, 4, 8, 9]

cows = 3 (c1, c2, c3)



We have to place 3 cows such that minimum distance b/w cows is as large as possible.

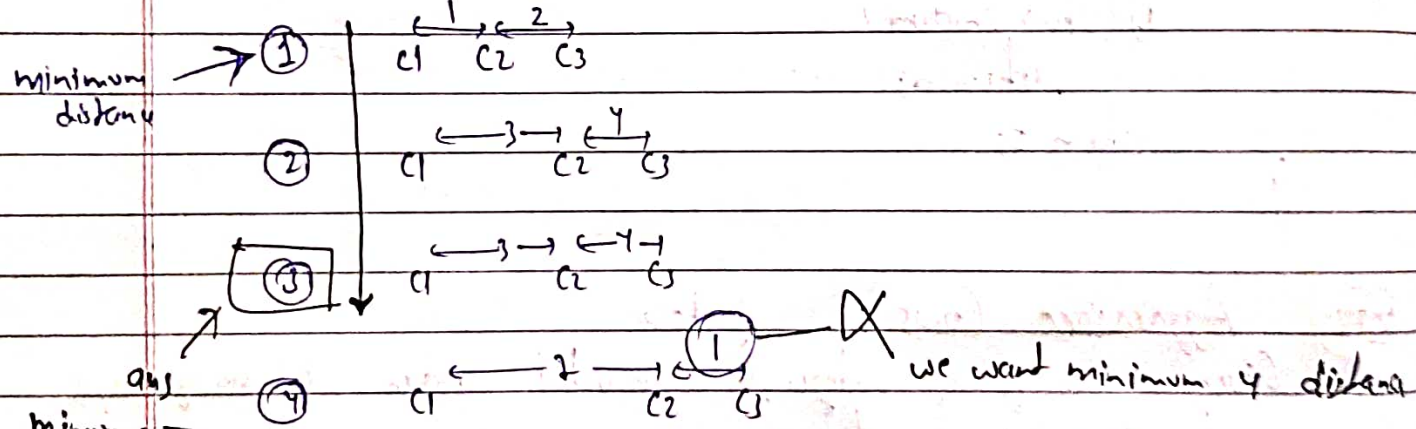
(a) By Reasion
check all the possible arrangement
Eg: If cows = 3, If size of array = 5
arrangement of 3 cows can be done in many ways.

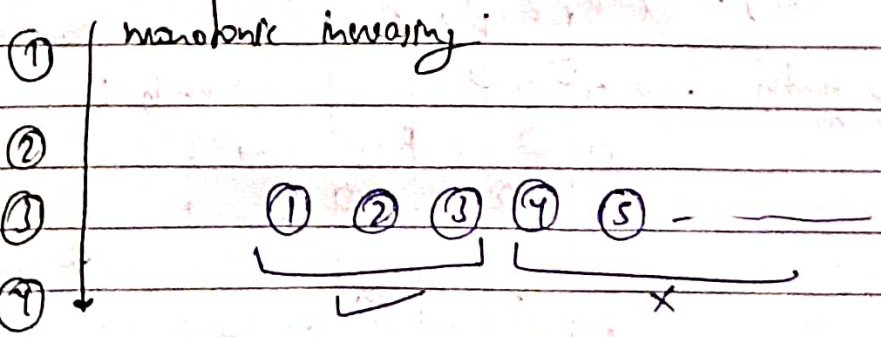
_ _ _ _ _ → 5 place
♀ ♀ ♀ → 3 cows

can be arranged in
5 × 4 × 3 = 60 ways

(b) Binary Search

a[] = [1, 2, 4, 8, 9] cows = 3





Search space
low = 1
high = v[n-1] - v[0]
here 9-1

If (4) is not possible arrangement of 3 cows then how can it be arranged in (5) minimum distance
∴ h = m - 1

Flow of BS

a = [1, 2, 4, 8, 9] cows = 3

[1 (low) 4 (mid) 8 (high)]

c1 ← 2 → c2
1, 2, 4, 8, 9 ✗

c3 not able to put
∴ mid = 4 not possible h = m-1;

now we will check can we place 3 cows by keeping the minimum distance of 4.

[1 (l) 2 (m) 3 (h)]

c1 ← 1 → c2 ← 1 → c3
1, 2, 4, 8, 9 ✓

l = m+1 b/z we have to maximize ans
∴ move right

[3 (l) 3 (m) 3 (h)]

c1 ← 3 → c2 ← 4 → c3
1, 2, 4, 8, 9 ✓ l = m+1

"

& l = 4 h = 3
while loop stop

Ans in h

```
int f(vector<int> &v, int c)
{
    sort(v.begin(), v.end());
    int l=1, h= v[v.size()-1]-v[0];
    while(l<=h)
    {
        int m= (l+h)/2;
        if(isPossible(v,c,m))
            l=m+1;
        else
            h=m-1;
    }
    return h;
}
bool isPossible(vector<int> &v, int no_of_cows, int min_distance)
{
    int cnt_cows=1;
    int lastPlaceCow = v[0];
    for(int i=0; i<v.size(); i++)
    {
        if((v[i] - lastPlaceCow) >= min_distance)
        {   cnt_cows++;
            lastPlaceCow= v[i];
        }
    }
    if(cntcows>= no_of_cows) return true;
    return false
}
```