Binary Search

~ Arbind

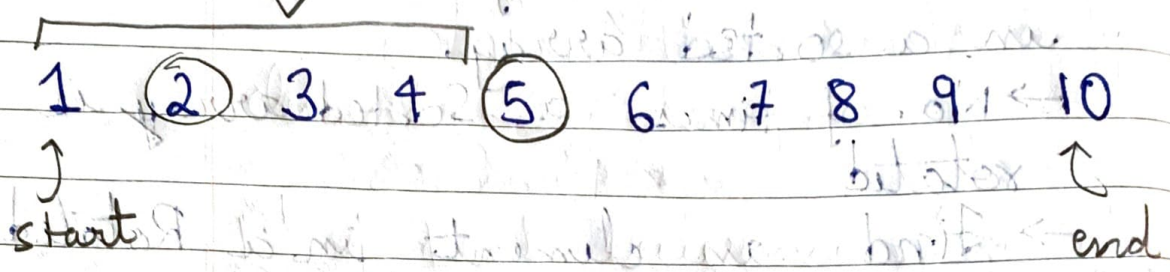
Binary Search

→ Binary Search
→ Binary Search on Reverse Sorted Array
→ Order Not known Search
→ First & last occurence of an Element
→ Count of an element in a sorted array.
→ No. of times a Sorted array is rotated
→ Find an element in a Rotated array
→ Searching in a Nearly Sorted Array
→ Floor / Ceil of an element
→ Next Alphabetic Element
* → Index of last 1
→ Find the position of an element
→ Index of first 1
→ Min diff element in an Sorted Array
→ Binary Search on Answer
→ Peak Element
→ Bitonic Array max element
→ Search in Bitonic Array
→ Search in row wise, column wise sorted array.
* → Find element in sorted array that appears only ones. (easy with bit manipulation, adjacent will not be same value)
→ Allocate min no. of pages.

Binary Search

	0	1	2	3	4	5	6	7
arr[] :	1	2	3	4	5	6	7	
start	8	9	10					
	7	8	9				end.	

ele = 2



⇒ mid = $\frac{start + end}{2} = \frac{0+9}{2}$

$= 4$

⇒ end = mid − 1

$= 1$

arr[mid] = ele

↳ so, return index

Code :

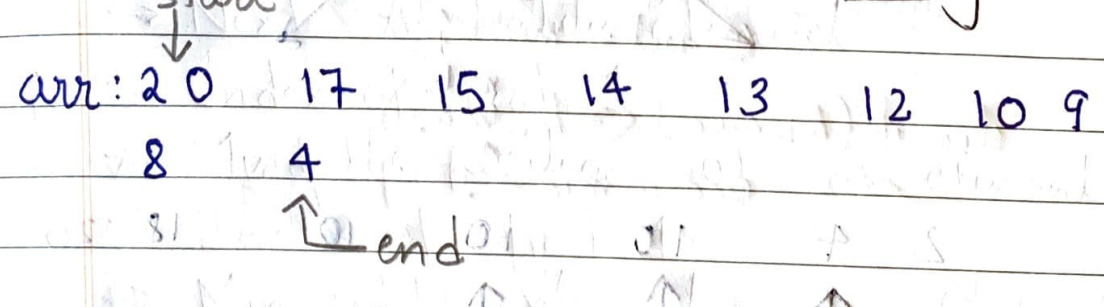
```
int start = 0;
int end = n-1;
while (start <= end)
{
```

$start + \frac{end - start}{2}$

↑ Optimised (prevents int overflow)

```
int mid = (start + end) / 2;
    if (arr[mid] == ele)
        return mid;
    else if (arr[mid] > ele)
        end = mid - 1;
    else
        start = mid + 1;
}
return -1;
```

Descending Sorted array



$mid = start + \frac{end - start}{2}$

```
else if (ele < arr[mid]) {
    start = mid + 1; }
else {
    end = mid - 1;
}
```

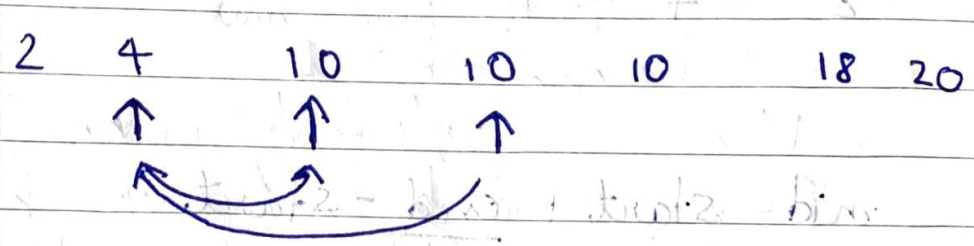
Order Not Known: (Order Agnostic Search)

arr[0] < arr[1] ⇒ asc
arr[0] > arr[1] ⇒ des

if arr[].size() == 1
then there exists only 1 element

1st & Last Occurence of an Element

0	1	2	3	4	5	6
2	4	10	10	10	18	20
		(2)	(3)	(4)		



$\frac{2+2}{2} = 2$

Code

```
int result = -1;
while ( start <= end) {
  if( ele == v[mid])
     result = mid
     end = mid -1;
}
```

→ First Occurance

```
int result = -1;
while( start <= end){
    if(ele == V[mid])
        result = mid
        start = mid + 1;
}
```

Count of an element in a Sorted Array

2 4 10 10 10 18 20

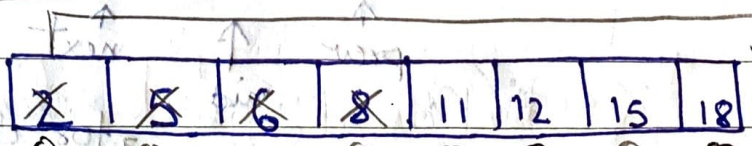
1st ↑ Last

length = last index - first index + 1

= 4 - 2 + 1

= 3

No. of times a binary array is rotated



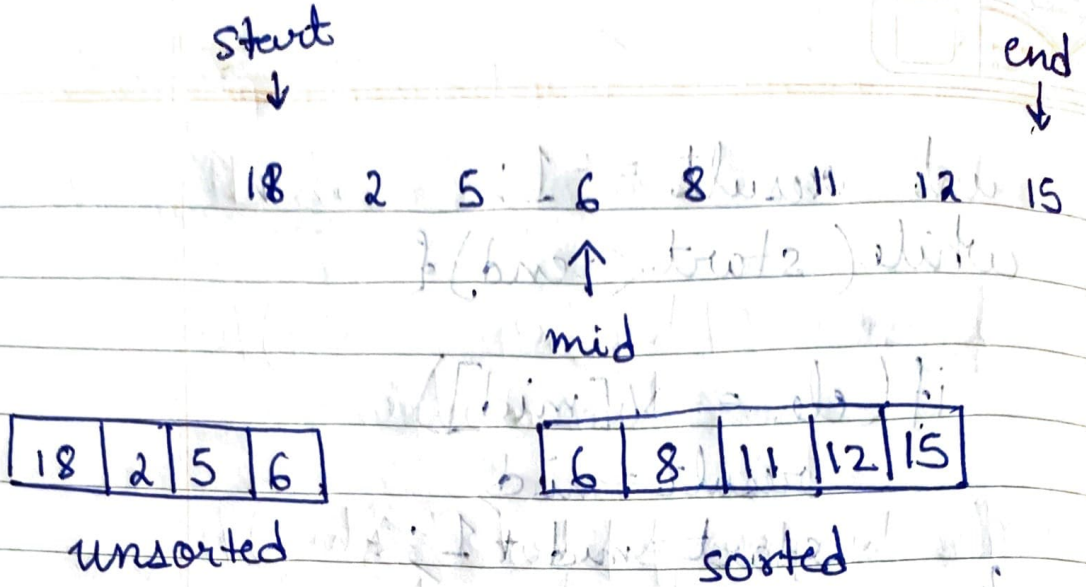
arr[]: 11 12 15 18 2 5 6 8

0 1 2 3 4 5 6 7

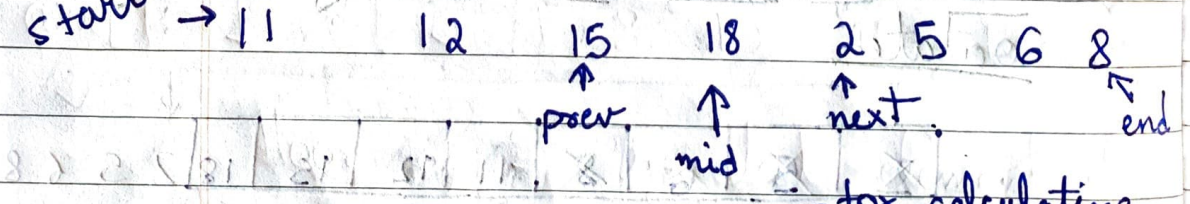
Pivot

index = 4

no. of times a arr is rotated ⇒ index of min number




```
while (start <= end)
{
    int mid = start + (end - start)/2;
if (A[mid] <= A[next] && A[mid] <= A[prev])
        return mid;
if (A[0] <= A[mid])
        start = mid + 1;
else if (A[mid] <= A[n-1])
        end = mid - 1
}
```



prev = (mid + N - 1) % N → for calculating prev at 0th index

next = (mid + 1) % N → suppose 7 (7+1) ⇒ 8 % 8 = 0

Find element in a rotated sorted Array

11 12 15 18 2 5 6 8

↑

no. of rotation → index of min element

```
binary_search(arr, 0, index - 1)
binary_search(arr, index, size - 1)
```

if both returns -1 then the element is not present in the array.

Searching in a Nearly Sorted Array

arr[]: | 5 | 10 | 30 | 20 | 40 |

nearly sorted = ith index

(i-1) (i) (i+1)

⇒ end = mid-2
⇒ start = mid+2

sorted array: 5 10 20 30 40

(0) (1) (2) (3) (4)

→ else == arr[mid] → mid - 1 >= start

Two new condition: elseif == arr[mid - 1]

elseif == arr[mid + 1] → mid + 1 <= end

Code

```
if(ele == arr[mid])
    return mid;
```

```
if(mid-1 >= start && arr[mid-1] == ele)
    return mid-1;
```

```
if(mid+1 <= end && arr[mid+1] == ele)
    return mid+1;
```

```
if(arr[mid] > ele)
    end = mid-2;
```

```
if(arr[mid] < ele)
    start = mid+2;
```

Find Floor of an element in a Sorted Array

	start			mid				end
arr[]:	1	2	3	4	8	10	10	12	19

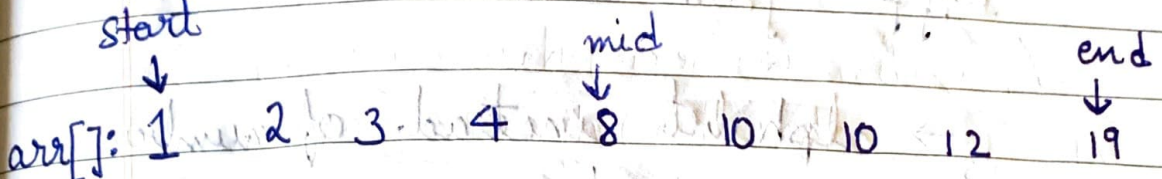
Floor of 5 = greatest element smaller than 5

Code:

```
if(arr[mid] <= ele){
    res = mid;        // index
    arr[mid];         // element
    start = mid+1
}
```

```
else if (arr[mid] > ele)
{
    end = mid - 1
}
```

Ceil of an element in a sorted Array



Ceil of 5 = Smallest element greater than 5

Code :

```
if (arr[mid] == ele)
    return arr[mid];

else if (arr[mid] > ele)
{
    ans = arr[mid]
    end = mid - 1;
}

else if (arr[mid] < ele)
{
    start = mid + 1;
}
```

Next letter

arr[]: [a c f h] key = f

→ Similar to ceil problem

Diff from ceil

→ alphabet instead of number
→ Key present x
next alphabet

Code:

```
arr[mid] > key
    res = arr[mid]
    high = mid - 1;
arr[mid] <= key
    low = mid + 1;
```

Position of an element in an infinite Sorted Array

1 2 3 4 5 6 7 8 9 ____

end → end * 2;

```
int low = 0
int high = 1

while (key > arr[high])
{
  low = high
  high = high * 2
}

binary search (arr, low, high)
```

Index of 1st in a Binary sorted Array

0 0 0 0 0 0 1 1 1 1 1

→ ∞
→ Binary
→ Sorted

```
while (key > arr [end])
{
  end
  start
}

if (arr[mid] == key)
    ress = mid;
    end = mid - 1;
```

Min diff element in a Sorted Array

arr[]: 4 6 10 Key = 7

7 7 7

3 1 3

1 3 8 | 10 15

↑ ↑ ↑

abs [arr[low] - key)] min

abs (arr[high - key)]

Binary Search on Answer

→ The array may not be sorted

Peak element:

arr[]: 5 10 20 15

↑

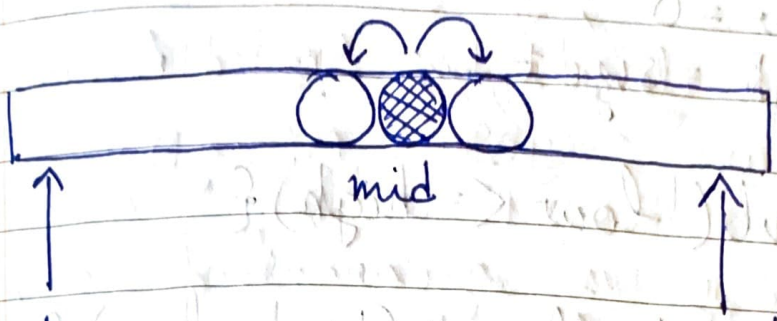
arr[]: 10 20 15 2 23 90 67

↑ ↑

arr[]: 10 20 30 40 50

↑

arr[]: 50 40 20 30 10



arr[mid] > arr[mid-1]

arr[mid] > arr[mid+1]

⇒ arr[]: 0: 5, 1: 10, 2: 20, 3: 15

start, end

$mid = \frac{0+3}{2} \Rightarrow mid = 1$

arr[m+1] >= arr[mid]
start = mid+1;

arr[mid] > arr[m+1]
end = mid-1;

(or)

if (arr[mid-1] > arr[mid]) ⇒ end = mid-1

else start = mid+1

Code :

```
low = 0 ;
high = size - 1 ;

while( low <= high){

int mid = low + (high - low)/2 ;

if( mid > 0 && mid < size-1){
  if (arr[mid] > arr[mid-1] &&
      arr[mid] > arr[mid+1])
        return mid;

elseif( arr[mid-1] > arr[mid])
      end = mid-1 ;
    (high)

else
      start = mid+1 ;
    (low)
}
```

Condition for 1st & last element :-

```
else if( mid == 0){
if(arr[0] > arr[1])
        return 0;
else return 1; }
```

```
else if(mid == size-1){
if(arr[size-1] > arr[size-2])
        return size-1;
else return size-2;
```

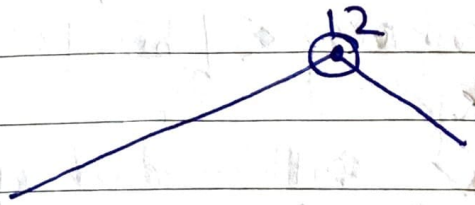
Max element in bitonic array

arr[]: 1 3 8 12 4 2

⇒ monotonical ↑ then monotonical ↓

5 10 15 19 21

arr[i+1] > arr[i]



max in Bitonic Array = Peak

only one peak in bitonic array.

Search in Bitonic array

	0	1	2	3	4	5
arr[]:	1	3	8	12	4	2

return index

key = 4

[1 3 8] [12 4 2]

Sorted Array ↑ Sorted Array ↓

Binary SearchA(arr, 0, index-1)
Binary SearchB(arr, index, size-1)

Better way exists

Find element in Sorted Array which occurs only ones O(log N) O(1)

→ Array is guaranteed to have unique element

	0	1	2	3	4	5	6
arr[] =	1,	1,	2,	2,	③,	4,	4]

Property

① Boundary Check
Pair

② index property

Before Unique Element
→ start at an even index
→ ends at an odd index

After Unique Element
for 4
→ start odd
→ end even

③ Unique Element Property

④ Partition Property

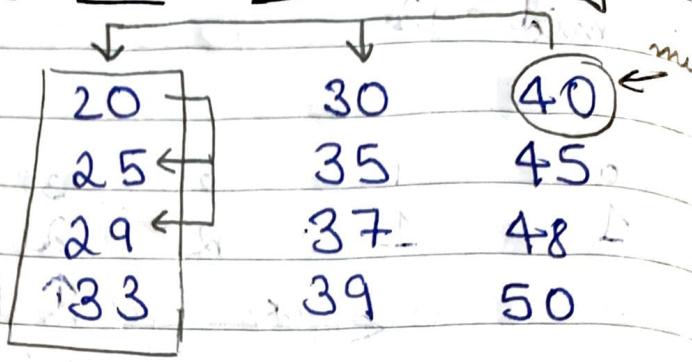
low = 0
high = 6
mid = 3

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

Boundary case

```
if (high == 0)
    return 0;
else if (nums[0] != nums[1])
    return nums[0];
else if (nums[high] != nums[high-1]
    return nums[high];

while(low <= high)
{
  mid = low + (high - low)/2;

  if(nums[mid] ! nums[mid-1] && nums[mid] !=
                                nums[mid+1])
    return nums[mid];

if(    mid % 2 == 0 && nums[mid] == nums[mid+1] ||
mid % 2 && nums[mid] == nums[mid-1])
          low = mid + 1;
else  high = mid - 1;
```

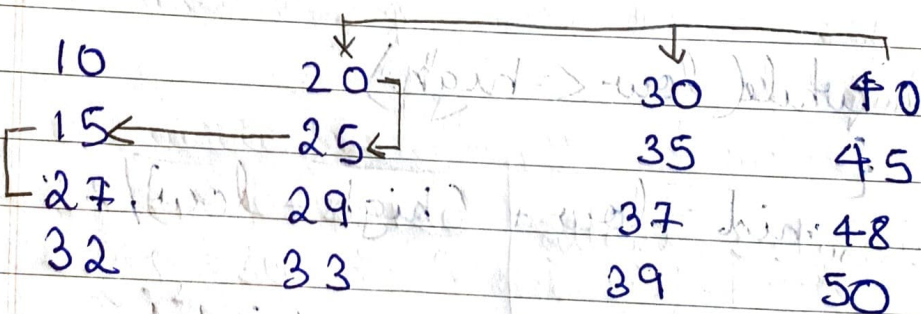
Search in Row wise & Column wise Sorted Array



Key = 29

Compare 40 < 29 ✗
So 30 < 29 ✗
20 < 29 ✓ ⇒ then check row

$O(n^2) \Rightarrow O(n+m)$



key = 23

Bound:

& $i >= 0$ && $i < n$

() $j >= 0$ && $j < m$

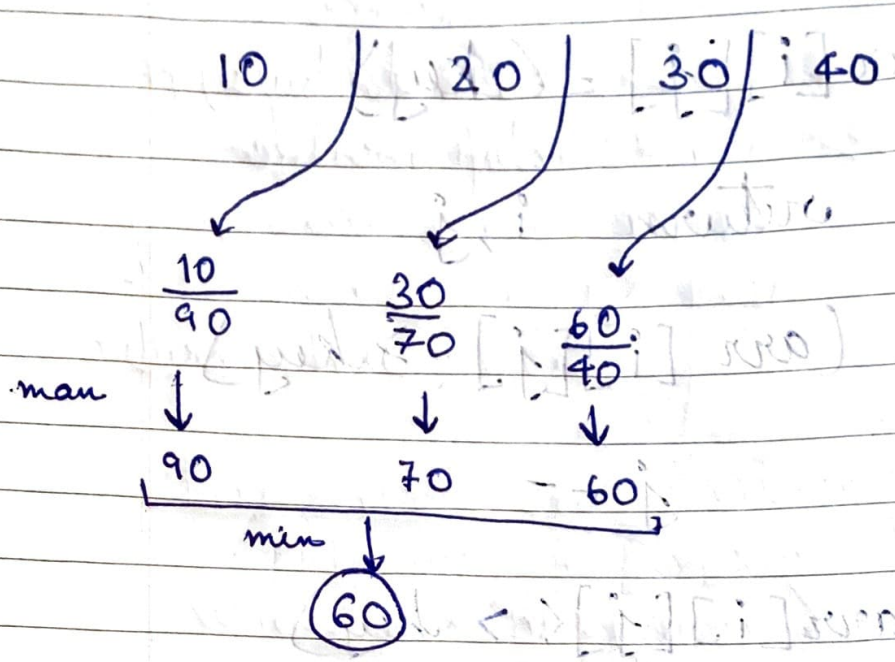
Code :

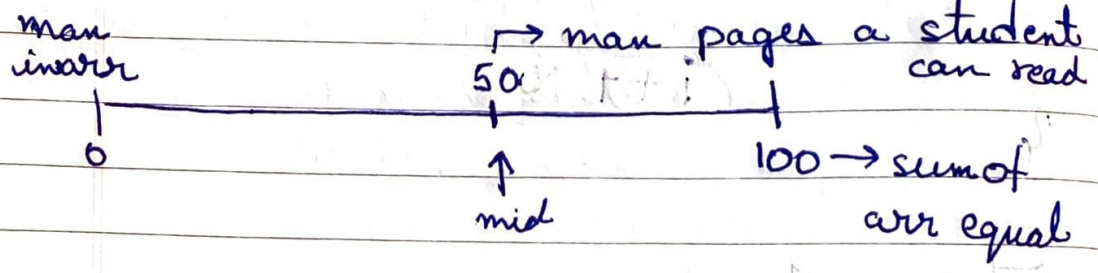
```
{ int i=0;
  int j=0;

while (i>=0 && i<n && j>=0
      && j<m){

if (arr[i][j] == key)

        return i,j

else if (arr [i][j] > key)

        j--

else (arr[i][j]< key)

        i++

}

return -1;

}
```

Allocate Pages of Books

	0	1	2	3
arr[]:	10	20	30	40

K = 2



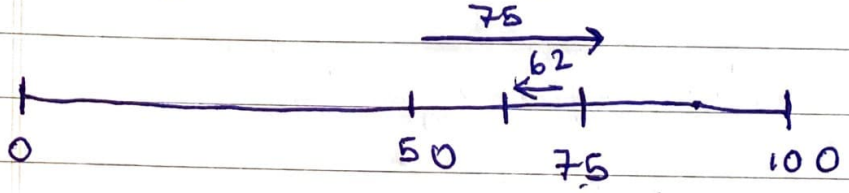


$S_1 = 10 + 20$

$S_2 = 30$

$S_3 = 40$

atleast 3 students



$S_1 = 10 + 20 + 30$

$S_2 = 40$

$62 \rightarrow 56 \rightarrow (60)$ ✓

Code

int start = max in array ⇒ 40

int end = sum of all array

int res = -1

```
while (start <= end) {
  mid = start + (end - start)/2

if (is valid (arr, n, k, mx) == true)
        res = mid
        end = mid - 1

  else
        start = mid + 1
```

Function

```
bool isvalid (arr, n, k, mx)
{
  int student = 1;
  int sum = 0;
  for (int i=0; i<n; i++){
    sum = sum + arr[i]
```

```
        if (sum > max) {
            student ++;
            sum = arr[i];
        }

    if (student > k)
        return false;
    }
return true;
```