Binary Search

Algorithm.

2.

Binary Search

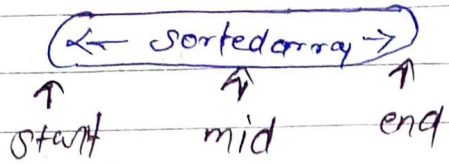
Introduction, arry Format

(1.) Binary Search.
(2.) Order agnostic BS.
(3.) 1st and last occurrence of an element.
(4.) Count of element in a sorted array.
(5.) Number of times → array is rotated.
(6.) Find an Element in rotated sorted array.
(7.) Searching in nearly sorted array.
(8.) Floor/Ceil of an array.
(9.) Next letter.
(10.) Index of the last one in a sorted array.
(11.) Find the position of an element in ∞ sorted array.
(12.) min. d/f element in a sorted array.
(13.) Bitonic Array max. element.
(14.) Search in a Bitonic Array.
(15.) Search in Row wise + col wise sorted matrix.
(16.) Find element in sorted array that appears once.
(17.) Allocate min # of pages.

2.

Binary Search

Given



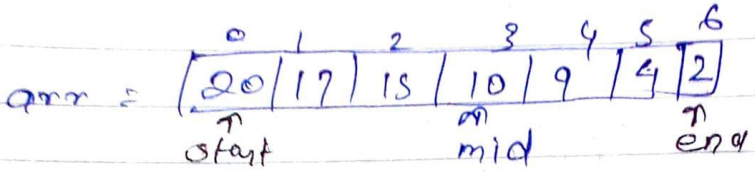
T.C $O(\log_2 n)$

CODE

```
int start = 0;
int end = n-1;

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

    if( element == arr[mid])
        return mid;

    else if (element < arr[mid])
        end = mid-1;

    else
        start = mid+1;
}
return -1;
```

8. Search in Descending sorted Array

Given an array in desending order;



ele = 2

$$mid = 0 + \frac{6-0}{2} = 3$$

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

else if( arr(mid) > ele)
    start = mid+1;

else
    end = mid-1;
```

4. Agnostic Search → Order Not known

If there is single element in the array.

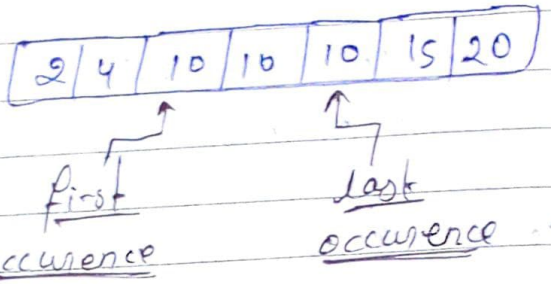
```
if (arr[0] == ele)
        return 0;

else if (arr[0] < arr[1])
        return AccendingOrder(all());

else if (arr[0] > arr[1])
        return DecendingOrder(all());
```

5.

First and Last Occurence of an Element.

Given an, ele = 10

arr =

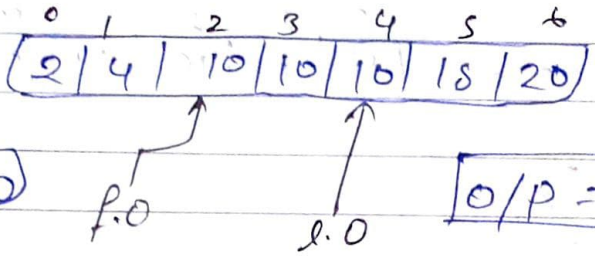


CODE

```
int start = 0;
int end = n-1;
int res = -1;
while(start < end)
{
    int mid = start + (end - start)/2;

if(ele == arr[mid])
{
    res = mid;
    end = mid-1;   <->   start = mid+1;
}                        (f.o)        (l.o)
else if(ele < arr[mid])
        end = mid-1;
else
        start = mid+1;
}
return res;
```

Q. Count of an Element in a sorted Array.

Given an

arr[] = 


```
int firstOccurence = findFirstOccurence();
int lastOccurence = findLastOccurence();

int ans = lastOccurence - firstOccurence + 1;

return ans;
```

(V. Imp)

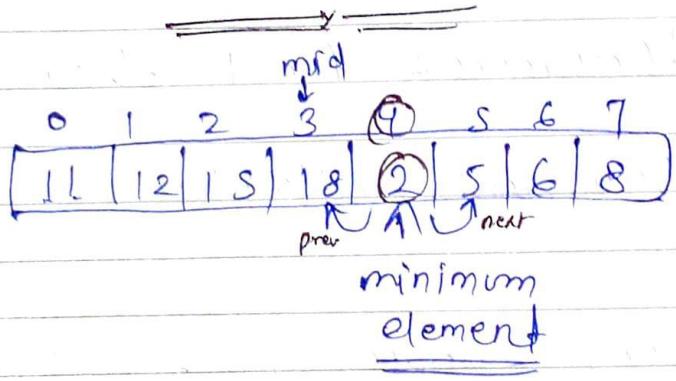
7. Number of times a Sorted Array is Rotated.

Given,

arr[] =

0	1	2	3	4	5	6	7
11	12	15	18	2	5	6	8

O/P: 4



Count of rotation = Index of minimum element.

So basically we have to find out minimum element in the given array.

By Using Binary Search

(1.) Calculate mid.
(2.) Check if it is minimum element.
(3.) Moving factor.

(1.) Calculate mid =>

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

(2.) Check if it is minimum element =>

Minimum element will always be smaller than its adjacent elements.

~~prev next = (mid+1) % N~~

~~next = (mid+N-1) % N~~

(1.)
```
next = (mid+1) % N
prev = (mid+N-1) % N
```

B.C if array is already sorted

```
if(arr[start] < arr[end]) return arr[start];
```

(2.)
```
if ( arr[mid] < next && arr[mid] < prev)
    return mid
```

(3.)
```
if ( arr[start] <= arr[mid] )
    start = mid+1;
else if ( arr[mid] <= arr[end] )
    end = mid-1;
```

always move towards unsorted side.

return arr[start]; -1;

8. Find an Element in a Rotated Sorted Array.

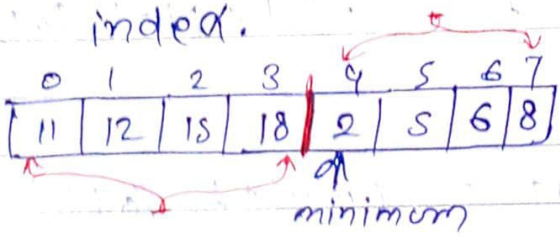
Given,

arr[] ⇒

0	1	2	3	4	5	6	7
11	12	15	18	2	5	6	8

ele = 15 o/p ⇒ 2

Step 1 ⇒ Find out the minimum element index.



index = 4

Now, we have two sorted array, so we can implement binary search one by one in both of the arrays.

```
if(start, ive ← (arr, start, index-1)
-ve ← (arr, index, size()-1)
return arr(+ve)
```

Searching in a Nearly Sorted Array.

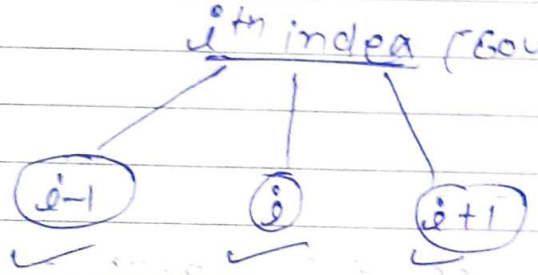
Q:

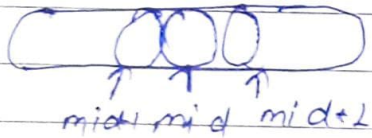
Given,

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

ele = x

Nearly Sorted = i^{th} index (could be at)





pseudo 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;
```

Boundry Check

remaining code is same.

10 Find Floor of an Element in a Sorted Array

Given,

arr[] = [1, 2, 3, 4, 8, 10, 10, 12, 19]

x = 5 floor → [O/P ⇒ 4]
ceil → 3

Floor ⇒ Greatest element smaller than x.

candidate
[1, 2, 3, (4)]
↑ res

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

if (arr[mid] < x)
{
    res = arr[mid];
    start = mid+1;
}
else if( arr[mid] > x)
{
    end = mid-1;
}

return res;
```

11.) **Find Ceil of an Element in a Sorted Array.**

Ceil => Smallest element greater than x

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

if( arr[mid] < x )
{
    start = mid+1;
}

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

return res;
```

12. Next Alphabetical Element

Given,

arr[] : [a, c, f, h]

I/P : f

O/P : h

CODE

char res = #

if key is greater or equals to

```
if( arr[mid] <= key)
    start = mid+1;
```

if key is smaller then it could be an ans.

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

return res;

LVIP)

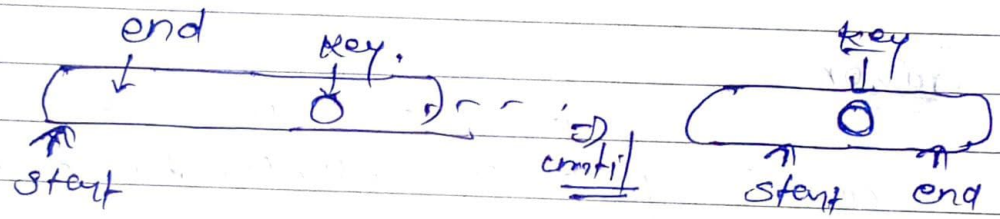
13. **Find position of an element in an Infinite Sorted Array**

Given an infinite array,

0	1	2	3	4	5	6	7	8	9	10	
1,	2,	3,	4,	5,	6,	7,	8,	9,	10,	10	....∞

arr[] = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10 ....∞]

key = 7 | O/P ⇒ 6



CODE

Initialization

```
int low = 0;
int high = 1;
```

taking key under control

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

Simple BS

```
BinarySearch(arr, low, high);
```

14. Index of first 1 in a Binary Sorted Array.

This Question is a combination of these two questions:-

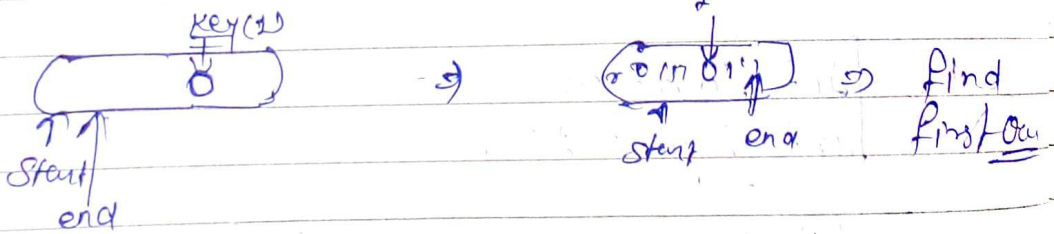
(1.) Find element in an ∞ sorted array.

(2.) First first occurence of given element.

Given

```
      0  1  2  3  4  5  6  7  8
arr[] = [0, 0, 0, 0, 0, 1, 1, 1, 1 ... ∞]
```

o/p => 5




```
while( key > arr[end] )
  { start = end;
    end = end * 2;
  }

while ( arr[mid] <= key )
{ res = mid;
  start end = mid - 1;
}
```

(Proceeding)

15. Minimum Difference Element in a Sorted Array.

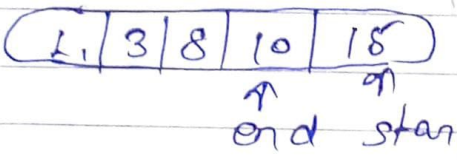
arr[] = [1, 3, 8, 10, 15]

key = 12. [O/P:- 10]

Special feature of simple Binary Search.

When a Binary Search gets over its start index ~~points~~ and end index points to the nearest value of a key.

Eg:



We just need to figure out which one gives the minimum result and return that.

return min (abs(arr[end] - key), abs(arr[start] - key));

Concept

10. Binary Search On Answer.

Note :- Binary Search can also be applied on unsorted array.

Requirment or modifications :-

criteria

(1.) mid to ans.
(2.) left / right ↔

(VSP)

(17.) # Peak Element.

Given an array,

arr[] = [10, 20, 30, 5] (indices 0, 1, 2, 3)

O/P :) 2

definition of peak element (mid)

mid-1 ✓

mid+1 ✓

CODE

Initialization

```
int start = 0;
int end = arr.size() - 1;
```

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

if this is the peak element

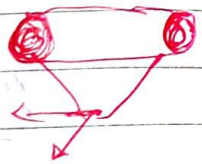
```
    if (mid > 0 && mid < arr.size() - 1)
    {
        if (arr[mid] > arr[mid-1] &&
            arr[mid] > arr[mid+1])
            return mid;
```

moving criteria

```
else if (arr[mid-1] > arr[mid])
        end = mid - 1;

else
        start = mid + 1;
}
```

first element

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

   else
        return 1;
}
```



Boundary Check

```
else if (mid == size(0-1)
{
   if( arr[size-1] > arr[size-2)
        return size-1;

   else
        return size-2;
}
```

last element

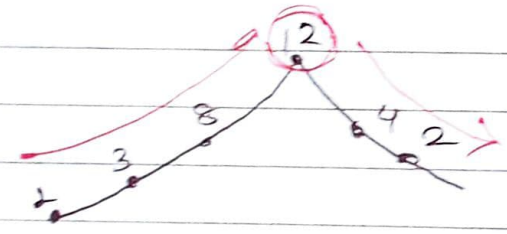
18. # find maximum Element in a Bitonic Array

Bitonic Array :-

The array which has no repitation till its peak and then starts decreasing without repeating same element in one go.

Given arr,

arr[] = [5, 3, 8, 12, 4, 2]



monotonic Increase/Decrease

That means,

This question is same as finding peak element in an array.

Code is ditto same 😊

19. Search An Element in Bitonic Array.

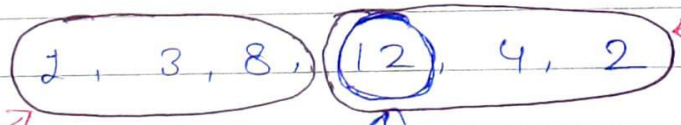
Given a Bitonic array

	0	1	2	3	4	5
arr[] =	1,	3,	8,	12,	4,	2

ele = 4

O/P = 4

Step 1 :

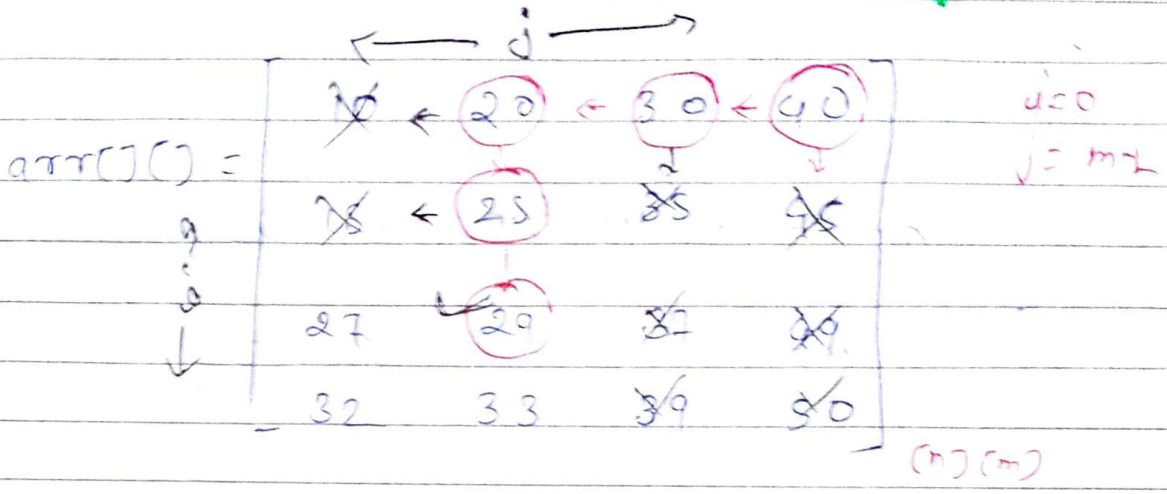


index = peak element

```
BS(arr, 0, index-1);
BS(arr, index, size()-1);
```

Search In Row wise And Column wise Sorted Array.

20.



key = 29

O/P = [2, 1]

CODE

Initialization from top-right.

```
int i = 0;
int j = m-1;
```

```
while( i>=0 && i<n && j>=0 && j<m)
{
    if( arr[i][j] == key)          // if key is present
        return pair{i, j};
    else if( arr[i][j] > key)      // if curr. element is greater
        j--;
    else if( arr[i][j] < key)      // increment
        i++;
}
return -1;                         // If key isn't present.
}
```

(VVIP)

Q1.

Allocate Minimum Number of Pages.

Given,

arr[] = [10, 20, 30, 40]

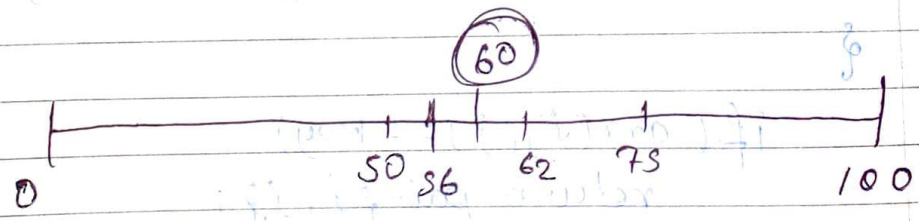
K = 2

O/P ⇒ 60

minimum ~~Maximum~~ ~~minimum~~ Number of pages for one student.

10 | 20 | 30 | 40

10 / 90 — 30 / 70 — 60 / 40



S_1 = 10 + 20 + 30

S_2 = 40

CODE (Psudo code)

B.C ← `if(n<k) return -1;`

```
int start = max. in arr

int end = sum of all arr. element.

int res = -1;

while(start <= end)
{

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

    if( isValid(arr, n, k, mid) == true )
    {
        res = mid;
        end = mid-1;
    }

    else
        start = mid+1;

}

return res;

}
```

```
bool isValid(arr, n, k, mid)
{
    int student = 1;
    int sum = 0;

for(int i=0; i<n; i++)
 {
    sum += arr[i];

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

 if(student > k)
    return false;

 }

 return true;

}
```