Matrix

Matrix

```
arr[3][2] = [ [10, 20]
              [30, 40]
              [50, 60]
            ]
```

```
for (int i=0; i<3; i++)
    for (int j=0; j<2; j++)
        cout << arr[i][j] << " ";
    return 0;
```

→ o/p 10, 20, 30, 40, 50, 60

① element are stored in row major order.

10	20	30	40	50	60
2000	2004	2008	2012	2016	2020

② Internal curly brackets are optional

```
int arr[3][2] = [10, 20, 30, 40, 50, 60]
```

③ Only the Ist dimension can be omitted when we initialize.

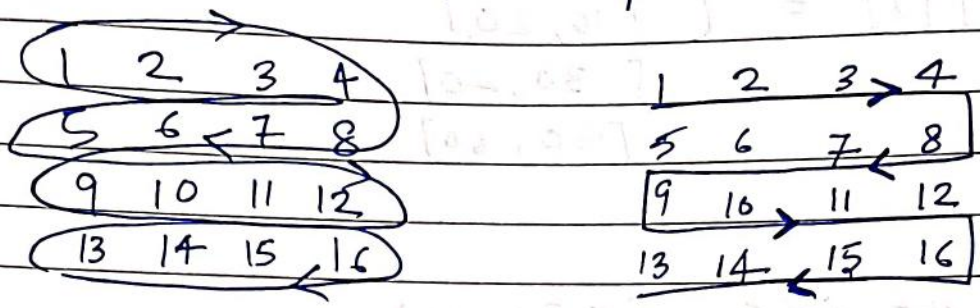
```
eg.: int arr[][2] = [[1,2], [3,4]]
     int arr[][2][2] = [ [[1,2] [3,4]
                         [5,6] [7,8]]
                       ]
```

Snake Pattern

Camlin Page
Date / /

Print a matrix in snake pattern.

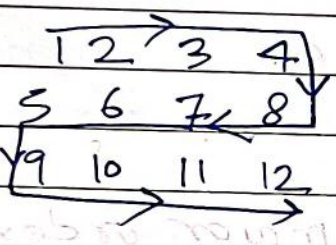
I/P



O/P To print the above matrix in below pattern.

1, 2, 3, 4, 8, 7, 6, 5, 9, 10, 11, 12,

I/P



O/P 1, 2, 3, 4, 8, 7, 6, 5, 9, 10, 11, 12

Time Complexity
O(r×c)

```
void printSnake (int mat[R][C])
{
for (int i=0; i<R; i++) {
    if (i%2 == 0)
    {
        for (int j =0; j<C; j++)
            print (mat[i][j] + " ");
    }
    else
    {
        for (int j = C-1; j >=0; j--)
            print (mat[i][j] + " ");
    }
}
}
```

Rows: 0, 2

Rows: 1, 3 ....

Print boundary element of matrix

I/P

1	2	3	4
5	6	7	8
9	10	11	12
13	4	15	16

O/P 1, 2, 3, 4, 8, 12, 16, 15, 4, 13, 9, 5

I/P

1 2 3 4
5 6 7 8

O/P 1 2 3 4 8 7 6 5

I/P 1 2 3 4

O/P 1 2 3 4

I/P

1 2
3 4
5 6

O/P 1 2 4 6 5 3 1

** Corner condition

① 1, 2, 3, 4

② 1
2
3
4

handle it by making separate condition.

Logic

	row (R)		
	1_{00}	2_{10}	3_{20}	4_{30}
column (C)	5_{01}	6_{11}	7_{21}	8_{31}
	9_{02}	10_{12}	11_{22}	12_{32}
	13_{03}	4_{13}	15_{23}	16_{33}

loop 0 to < C
mat[0][i]

print from row = 1
loop over row 1 to < R
mat[i][c]

loop over column C-1
mat[R-i][i]

loop over row R-1
mat[i][0]

```
void btravel(int mat[R][C])
{
    if (R == 1)
    {
        for(int i = 0; i < C; i++)
        { print mat[0][i]
        }
    }
    else if (C == 1)
    {
        for (int i = 0; i < R; i++)
            print (mat[i][0] + " ");
    }
    else {

    for (int i = 0; i < C; i++)
        print mat[0][i];

    for ( int i = 1; i < R; i++)
        print (mat[i][C-1]

    for (int i = C-2; i <= 2; i >= 0; i--)
        print mat[R-i][i]

    for (int i = R-2; i >= 1; i--)
        print mat[i][0]
```

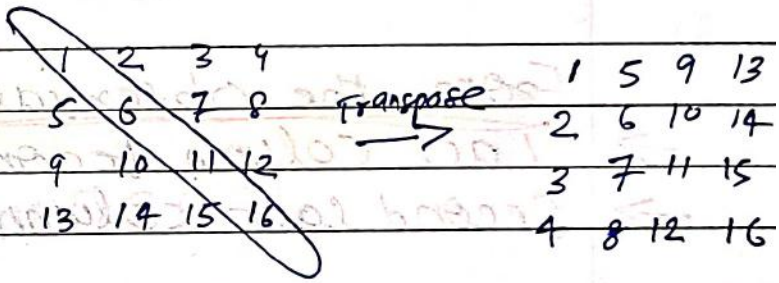
Transpose of matrix

I/p

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

O/p

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16



I/p

1	1
2	2

O/p

1	2
1	2

with temp variable

```
transpose ( int mat [n][n])
    {
      int temp [n][n]
      for (int i = 0 ; i<n; i++)
        for (int j =0 ; j<n ;j++)
            temp[i][j] = mat[j][i];
```

without temp variable

```
for (int i=0; i<n ; i++)
    for (int j = i+1; j<n; j++)
        Swap (mat [i][j] , mat [j][i])
```

Rotate Matrix Anti-clockwise by 90

I/P

1	2	3
4	5	6
7	8	9

O/P

3	6	9
2	5	8
1	4	7

I/P

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

O/P

4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

From the observation

→ Last column becomes first row.

→ Second last column become second rows.

$O(n^2)$ time

$O(n^2)$ space

Pseudo-code

$O(n^2)$

```
Void rotate90 (int mat[R][C])
{
    int temp [n][n];
    for (int i = 0; i<n; i++)
        for (int j = 0; j<n; j++)
            temp[n-j-1][i] = mat[i][j]

    for (int j=0; j<n; j++)
        for (int i=0; i<j; i++)
            mat[j][i] = temp[j][i]
}
```

Camlin Page
Date / /

I/P

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Reverse Columns →

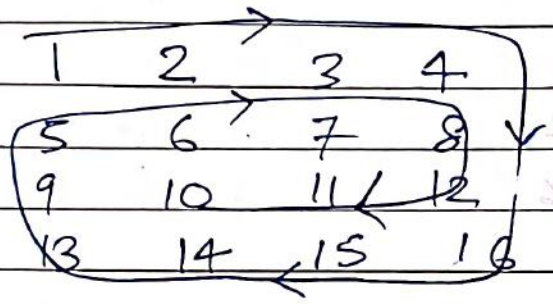
4	8	12	16
3	7	11	15
2	6	10	14
1	5	9	13

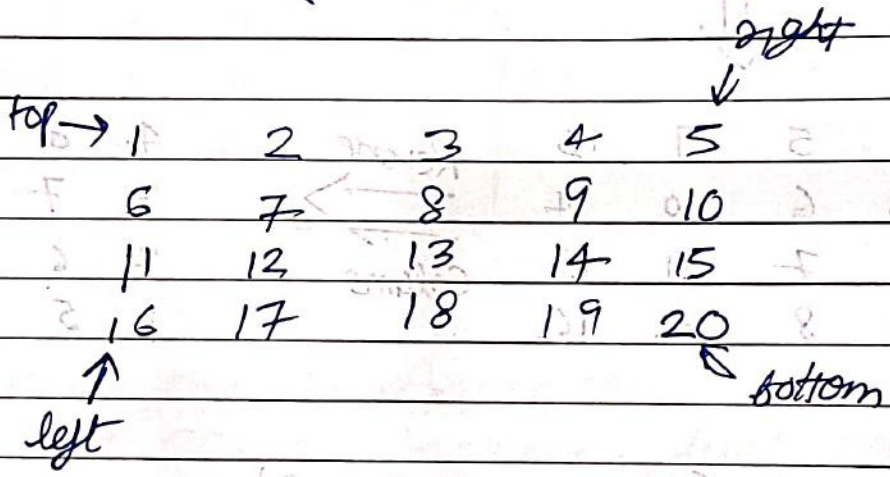
```
void rotate90 (int mat[n][n])
{
    // Transpose
    for (int i=0; i<n; i++)
        for (int j=j+1; j<n; j++)
            swap (mat[i][j], mat[j][i]);

    // Reverse columns
    for (int i=0; i<n; i++)
    {
        int low=0, high=n-1;
        while (low<high)
        {
            swap (mat[low][i], mat[high][i]);
            low++;
            high--;
        }
    }
}
```

Spiral Traversal of matrix

I/P





Steps

- Top row
- right column
- Bottom Raw (Reverse)
- left Column (Reverse)

```
void printSpiral (int mat[][], int R, int C)
{
    int top = 0, left = 0, bottom = R-1,
        right = C-1

    While (top <= bottom && left <= right) {

Top Raw.    for (int i = left ; i < right ; i++)
                print (mat [top][i] + " ");
            top++;

Right Column    for (int i = top ; i <= bottom ; i++
                    print (mat [i][right] + " ");
                right--;

bottom Raw (Reverse raw)    if (top <= bottom) {
                for (int i = right ; i >= left ; i--)
                    print (mat [bottom][i] + " ");
                bottom--;
            }

left Column (Reverse order)    if (left <= right {
                for (int i = bottom ; i >= top ; i--)
                    print (mat [i][left] + " ");
                left++;
            }
        }
    }
```

Search in a Row-wise and column-wise

I/P

$$mat[][] = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 15 & 25 & 35 & 45 \\ 27 & 29 & 37 & 48 \\ 32 & 33 & 39 & 50 \end{bmatrix}$$

$x = 29$

found at (2, 1)

(i, j)

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

```
void search (int mat[R][C], int x)
{
    for (int i = 0; i<R; i++)
    { for (int j = 0; j<C; j++)
        { if (mat[i][j] == x)
        { print ("found at " + i +
          " " + j + ").
          return;
        }
    }
    print("Not found")
}
```

Efficient Sol: O(R+C)

10	20	30	40
15	25	35	45
27	29	37	48
32	33	39	49

x = 29

→ Begin from the top right corner.
→ If x is same, print position & return.
→ If x is smaller, move left
→ If x is greater, move down

initially : i = 0, j = 3
after Iteration ① : j = 2
" " ② : j = 1
" " ③ : i = 1
" " ④ : i = 2
" " ⑤ : found at (2,3)

```
void search (int mat[R][C], int x)
{
    int i = 0, j = C-1;
    while (i<R && j>=0)
    {
        if (mat[i][j] == x)
        print ("Found at " +
            i + ", " + j + ");
        return;
    }
    else if (mat[i][j] > x)
        j--;
    else
        i++;
}
print ("Not found");
}
```

Ch - Searching

Binary Search

It is a searching algorithm for searching an element in a sorted list or array.

Binary

Binary Binary

① Binary Search (Iterative)

Find x and return the index from the array.

I/P arr[] = [10, 20, 30, 40, 50, 60]
(0, 1, 2, 3, 4, 5)

x = 20;

O/P 1

I/P arr[] = [10, 15]
(0, 1)

x = 20

O/P -1

I/P arr[] = [10, 10]
(0, 1)

O/P 0 or 1

10	20	30	40	50	60	70
0	1	2	3	4	5	6

low

high

Steps:

compute mid = (low + high)/2

case 1: (arr[mid] == x)

case 2: (arr[mid] > x)

case 3: (arr[mid] < x)

10	20	30	40	50	60	70
0	1	2	3	4	5	6

low ↑ high ↑

$x = 60$

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

Compare arr[3] & x

Steps :-

compute mid = ((low + high)/2)

return mid

Case 1: arr[mid] == x

$x = 30$

When you find the mid element is equal to x then return the same.

10	20	30	40	50
0	1	2	3	4

Repeat : high = mid - 1

Case 2: arr[mid] > x

$x = 10$

When mid is greater than x then - update high & repeat.

10	20	30	40	50

X exclude

Repeat : low = mid + 1

Case 3: arr[mid] < x

$x = 50$

When mid is smaller than x then update low & repeat.

10	20	30	40	50

X exclude

Example To search a number using Binary Search

10	20	30	40	50	60	70
0	1	2	3	4	5	6

low ↑ high ↑

$x = 60$

(I) $mid = \frac{0+6}{2} = 3$

Since $arr[mid] < x$

$40 < 60$, $low = mid + 1$

$1 + 3 = 4$, $x > [3] = 3+1 = 4$

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

since $arr[mid] == x$, return mid

```
(int bsearch (int arr[], int n, int x) {
    int low = 0;
    int high = n-1;
    while (low <= high)
    {
        int mid = (low + high)/2;
case 1  if (arr[mid] == x)
            return mid;
case 2  else if (arr[mid] > x)
            hig = mid - 1;
case 3  else low = mid + 1;
    }
return -1 }
```

Recursive Binary Search

10	20	30	40	50	60	70
0	1	2	3	4	5	6
↑	↑					↑
low	mid					high

$x = 20$

Steps:

$$mid = (low + high)/2$$

our base condition

case 1: $arr[mid] == x$, return mid

case 2: $arr[mid] > x$, $high = mid - 1$

case 3: $arr[mid] < x$, $low = mid + 1$

* What should be terminating condition

→ As in Iterative solution we are using while ($low <= high$)

→ As in recursive solution we are using if ($low > high$) return -1

```
int bSearch (int arr[], int low, int high,
             int x) {
    if (low > high) return -1;
    int mid = (low + high) / 2;
    if (arr[mid] == x) return mid;
    else if (arr[mid] > x)
        return bSearch (arr, low, mid-1, x);
    else
        return bSearch (arr, mid+1, high, x);
}
```

case 2

case 3

example 1)

10	20	30	40	50	60	70
0	1	2	3	4	5	6
low						high

x = 20

bSearch (arr, 0, 6, 20)
- mid = (0+6)/2 = 3
- arr[mid] > 3
- bSearch (arr, 0, 2, 20)
  - mid = (0+2)/2 = 1
  - arr[mid] == x, return mid

10	20	30	40	50	60	70
0	1	2	3	4	5	6

low ↑ high ↑

x = 25

bsearch(arr, 0, 6, 25)

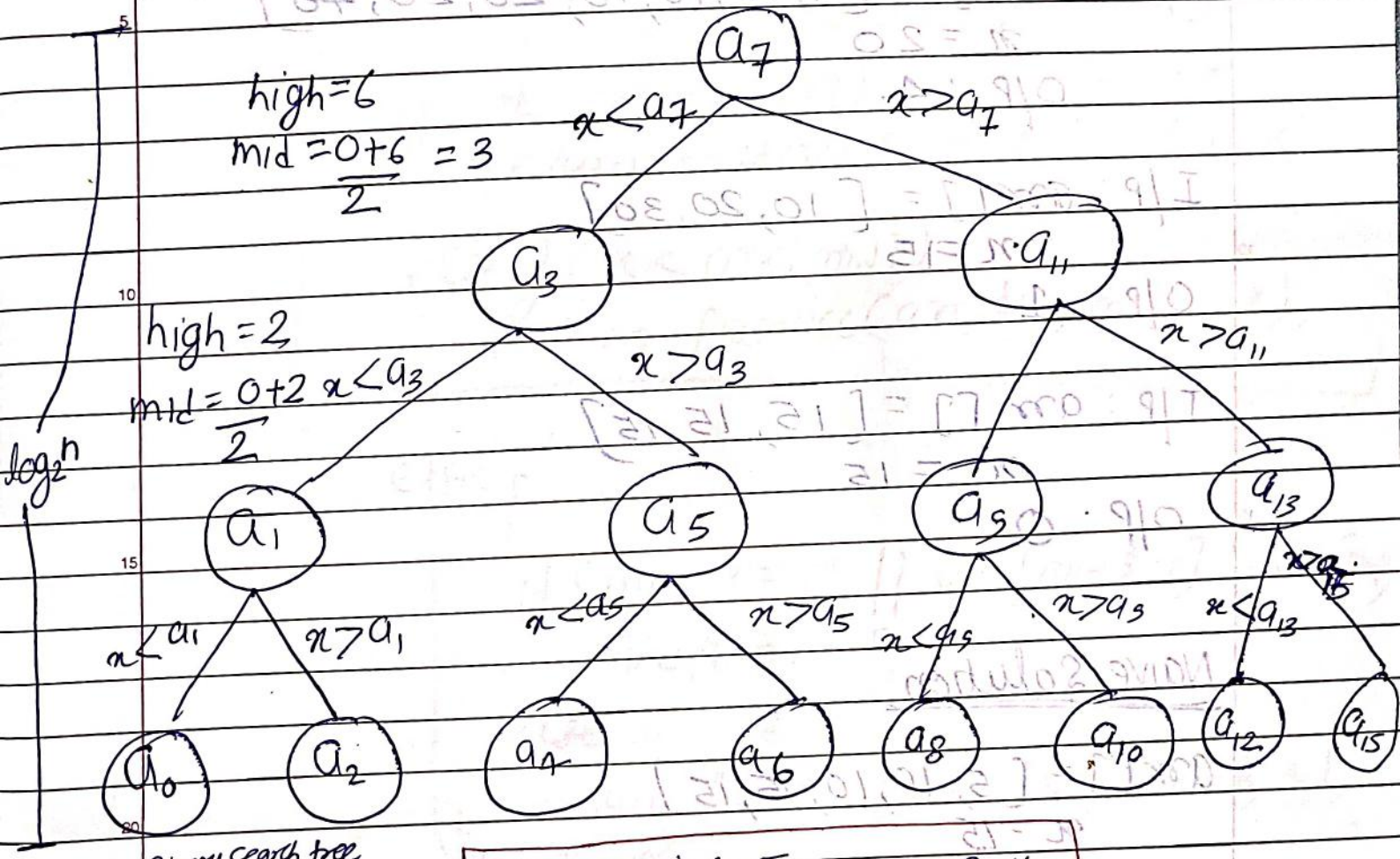
- mid = (0+6)/2 = 3
- arr[mid] > x
- bsearch(arr, 0, 2, 25)
  - mid = (0+2)/2 = 1
  - arr[mid] < x
  - bsearch(arr, 2, 2, 25)
    - mid = (2+2)/2 = 2
    - arr[mid] > x
    - bsearch(arr, 2, 1, 25)

-1 -1 -1

Analysis of Binary Search

a_0	a_1	.....	a_6	a_7	a_8	.....	a_{14}	a_{15}

$mid = (0+15)/2 = 7$



Binary search tree

height of $\lceil \log_2 n \rceil$

ceil of $\log_2 n$

All possible Execution Paths for Successful Search.

$x \rightarrow$ Element to be searched

Three cases

1. $x == a_{mid}$ return mid
2. $x < a_{mid}$ high = mid − 1
3. $x > a_{mid}$ low = mid + 1

Index of First Occurrence in Sorted

Example

I/P : arr[] = [1, 10, 10, 10, 20, 20, 40]
x = 20
O/P : 4

I/P : arr[] = [10, 20, 30]
x = 15
O/P : -1

I/P : arr[] = [15, 15, 15]
x = 15
O/P : 0

Naive Solution

arr[] = [5, 10, 10, 15, 15]
x = 15

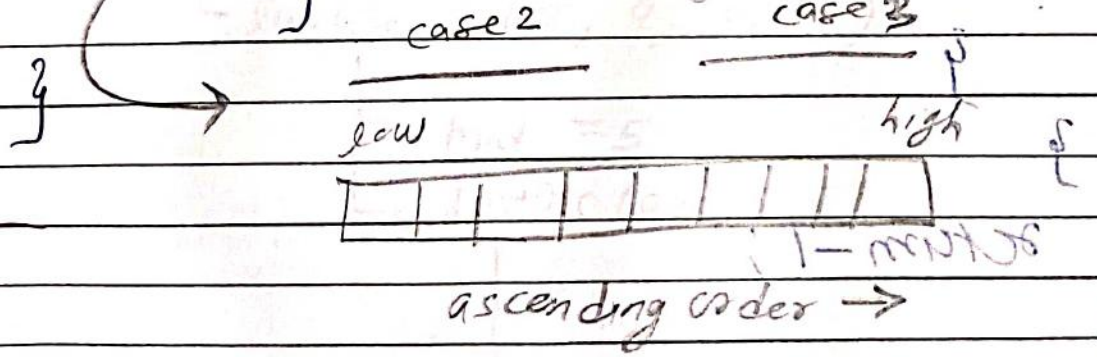
```
int firstOccurrence (int arr[], int n, int x)
{
    for (int i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

O(n) time
O(1) Aux Space

Recursive Solution

```
int firstOcc(int arr[], int low, int high, int x)
{
    if (low > high) return -1;
    int mid = (low + high) / 2;
    if (x > arr[mid])
        return firstOcc(arr, mid+1, high, x);
    else if (x <= arr[mid])
        return firstOcc(arr, low, mid-1, x);
    else {
        if (mid == 0 || arr[mid-1] != arr[mid])
            return mid;
        else
            return firstOcc(arr, low, mid-1, x);
    }
}
```

Normal Binary Search



```
int firstOcc(int arr[], int n, int x) {
    int low=0, high=n-1;
    while (low<=high)
    {
        int mid = (low+high)/2;
        if (arr[mid] > x)
            high = mid-1;
        else if (arr[mid] < x)
            low = mid+1;
        else {
            if (mid==0 || arr[mid-1] != arr[mid])
                return mid;
            else
                high = mid-1;
        }
    }
    return -1;
}
```

Index of Last Occurrence

Camlin Page
Date / /

I/P arr[] = [10, 15, 20, 20, 40, 40]
x = 20

O/P 3

I/P arr[] = [5, 8, 8, 10, 10]
x = 10

O/P 4

I/P arr[] = [8, 10, 10, 12]
x = 7

O/P -1

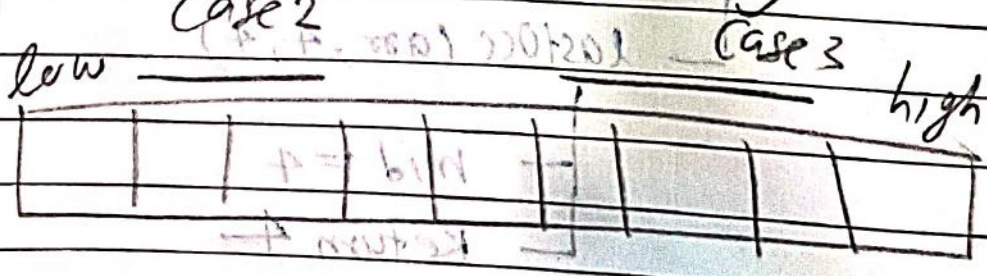
arr[] = [5, 10, 10, 10, 20, 20]
x = 10

lastOcc(arr, 0, 6, 10)
- mid = 3
- lastOcc(arr, 4, 6, 10)
  - mid = 5
  - lastOcc(arr, 4, 4)
    - mid = 4
    - Return 4

Recursive sol

```
int lastOcc (int arr[], int low, int high, int x)
                                    ↳ length of arr
{
    if (low > high) return -1;
    int mid = (low + high)/2;
    if (arr[mid] > x)
        return lastOcc(arr, low, mid-1, x);
    else if (arr[mid] < x)
        return lastOcc(arr, mid+1, high, x);
    else {
        if (mid == n-1 || arr[mid] != arr[mid+1])
            return mid;
        else
            return lastOcc(arr, mid+1, high, x);
    }
}
```

Normal Binary Search



ascending order →

Iterative Sol.

```
int lastOcc(int arr[], int n, int x)
{
    int low = 0, high = n-1;
    while (low <= high)
    {
        int mid = (low + high)/2;
        if (arr[mid] < x)
            low = mid + 1;
        else if (arr[mid] > x)
            high = mid - 1;
        else {
            if (mid != n-1 || arr[mid] != arr[mid+1])
                return mid;
            else
                low = mid + 1;
        }
    }
    return -1;
}
```

Normal Binary Search

Count Occurences in a Sorted Array

I/P arr[] = [10, 20, 20, 20, 30, 30]
x = 20
o/p : 3

I/P arr[] = [10, 10, 10, 10, 10 10]
x = 10
o/p 6

I/P arr[] = [5, 8, 10]
x = 7
o/p 0

```
int countOcc (int arr[], int n, int x)
{
    int first = firstOcc(arr, n, x);
    if (first == -1)
        return 0
    else
        return (lastOcc(arr, n, x) - fir
}
```

eg

10	20	20	20	44
	↑		↑	
	1		3	

Count 1's in a Sorted Binary Array

Sampling Page
Date / /

I/P arr[] = [0, 0, 0, 1, 1, 1, 1]
O/P 4

I/P arr[] = [1, 1, 1, 1, 1]
O/P 5

I/P arr[] = [0, 0, 0, 0]
O/P 0

```
int countOnes(int arr[], int n)
{
    int low = 0, high = n-1;
    while (low <= high)
    {
        int mid = (low + high)/2;
        if (arr[mid] == 0)
            low = mid + 1;
        else
            if (mid == 0 || arr[mid-1] == 0)
                return (n - mid);
            else
                high = mid - 1;
    }
}
```

Case 3

Case 3

0011111
low = 0, high = 6

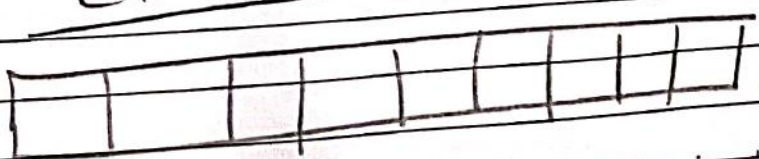
I st Iteration
mid = 3
high = 2

II nd Iteration
mid = 1
low = 2

III rd Iteration
mid = 2
return (7 - 2);

Case 2

Case 3



Square Root

I/P x = 4
O/P 2

I/P x = 14
O/P 3

I/P x = 25
O/P 5

x = 10
law = 1, high = 10

I^st Iteration
mid = 5
msq = 25
high = 4

II^nd Iteration
mid = 2
msq = 4
law = 3, ans = 2

III^rd Iteration
mid = 3
msq = 9
law = 4, ans = 3

IV^th Iteration
mid = 4
msq = 16
high = 3

```
int sqRootFloor(int x)
{
    int law = 1, high = x, ans;
    while (law <= high)
    {
        int mid = (law + high)/2;
        int msq = mid * mid;
        if (msq == x)
            return mid;
        else if (msq > x)
        {
            high = mid - 1;
        }
        else {
            law = mid + 1;
            ans = mid;
        }
    }
    return ans;
}
```

Search in Sorted Rotated Array

I/P arr[] = [10, 20, 30, 40, 50, 8, 9]
x = 30

O/P 2 ← Index

I/P arr[] = [100, 200, 300, 10, 20]
x = 40

O/P -1 ← Index

```
int search(int arr[], int n, int x)
{
    int low = 0, high = n-1;          // Like Normal Binary search
    while (low <= high)
    {
        int mid = (low + high)/2;
        if (arr[mid] == x) return mid;

        if (arr[low] < arr[mid])      // Left half Sorted
        {
            if (x >= arr[low] && x < arr[mid])
                high = mid - 1;
            else
                low = mid + 1;
        }
        else                          // Right half Sorted
        {
            if (x > arr[mid] && x <= arr[high])
                low = mid + 1;
            else
                high = mid - 1;
        }
    }
    return -1;
}
```

Find a Peak Element

(Not smaller than Neighbours)

I/P arr[] = [5, 10, 20, 15, 7]
O/P 20

I/P arr[] = [10, 20, 15, 5, 23, 90, 67]
O/P 20 or 90

I/P arr[] = [80, 70, 60]
O/P 80

I/P arr[] = [80, 70, 90]
O/P 80 or 90

Naive Solution

```
int getPeak(int arr[], int n)
{
    if (n == 1) return arr[0];
    if (arr[0] >= arr[1]) return arr[0];
    if (arr[n-1] >= arr[n-2]) return arr[n-1];

    for (int i = 1; i <= n-1; i++)
        if (arr[i] >= arr[i-1] &&
            arr[i] >= arr[i+1])
            return arr[i];
}
```

```
int getAPeak(int arr[], int n){
    int low = 0, high = n-1;
    while (low <= high)
    {
        int mid = (low + high)/2;
        if ((mid == 0 || arr[mid-1] <= arr[mid])
        && (mid == n-1 || arr[mid+1] <= arr[mid]))
            return mid;
        if (mid > 0 && arr[mid-1] >= arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}
```

Two Pointer Approach

① Given an unsorted array and a number x, we need to find if there is a pair in the array with sum equals to x.

I/p : arr[] = [3, 5, 9, 2, 8, 10, 11]

O/p x = 17

Yes

Pair is 5, 12

left pointer = 0

right pointer = n-1

I/p arr[] = [3, 8, 13, 18]

x = 14

O/p No

[3, 5, 9, 2, 8, 10, 11]

l = 0

r = n-1

let desired sum = DS

DS < l + r

then x--

DS > l + r

then l++

I/p arr[] = [3, 8, 13, 18]

x = 14

O/p = No

(2) Given a sorted array and a sum find if there is a pair with given sum.

```
bool isPair (int arr[], int n, int x)
{
    int left = 0, right = n-1;
    while (left < right)
    {
        if (arr[left] + arr[right] == x)
            return true;
        else if (arr[left] + arr[right] > x)
            right--;
        else left++;
    }
    return false;
}
```

(3) Given a sorted array and a sum find if there is a triplet with given sum.

I/P arr[] = [2, 3, 4, 8, 9, 20, 40]

x = 32

O/P Yes

```
for (int i=0; i<n; i++)
    if (isPair (a, i+1, n-1, x - a[i]))
        return true;
return false;
```

Median of Two Sorted Arrays

I/P
$a_1[] = [10, 20, 30, 40, 50]$
$a_2[] = [5, 15, 25, 35, 45]$

O/P
27.5 // [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

avg. of these 2 element

I/P
$a_1[] = [1, 2, 3, 4, 5, 6]$
$a_2[] = [10, 20, 30, 40, 50]$

O/P
6.0 = // [1, 2, 3, 4, 5, 6, 10, 20, 30, 40, 50]

I/P
$a_1[] = [10, 20, 30, 40, 50, 60]$
$a_2[] = [1, 2, 3, 4, 5]$

O/P
10.0 = // [1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 60]

Median of Two sorted Arrays

Naive : $O((n_1 + n_2) * \log(n_1 + n_2))$

$n_1 \rightarrow$ size of all []
$n_2 \rightarrow$ size of a_2 []

1. Create an array temp [] of size $(n_1 + n_2)$
2. Copy element of a_1 [] & a_2 [] to temp
3. Sort temp []
4. If $(n_1 + n_2)$ is odd, then return middle of temp
5. Else return average of middle two elements.

$a_1[] = [10, 20, 30]$

$a_2[] = [5, 15, 25]$

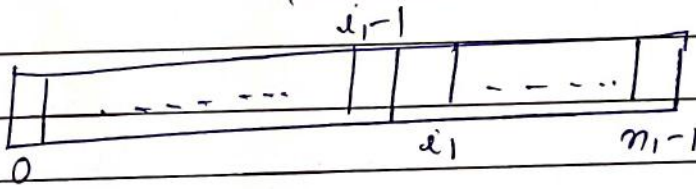
temp [] = [10, 20, 30, 5, 15, 25]

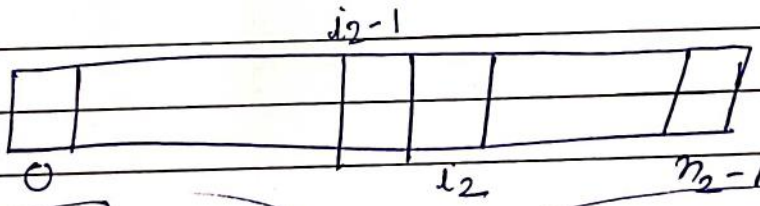
~~After~~ sorting.

temp [] = [5, 10, 15, 20, 25, 30]

$n_1 = 5$ $a_1[] = [10, 20, 30, 40, 50]$

$n_2 = 9$ $a_2[] = [5, 15, 25, 35, 45, 55, 65, 75, 85$





$a_1[0 \ldots\ldots i_1-1]$
$a_2[0 \ldots\ldots i_2-1]$

left half

$a_1[i \ldots\ldots n_1-1]$
$a_2[i_2 \ldots\ldots n_2-1]$

right half

$$i_2 = \left| \frac{n_1 + n_2 + 1}{2} \right| - i_1$$