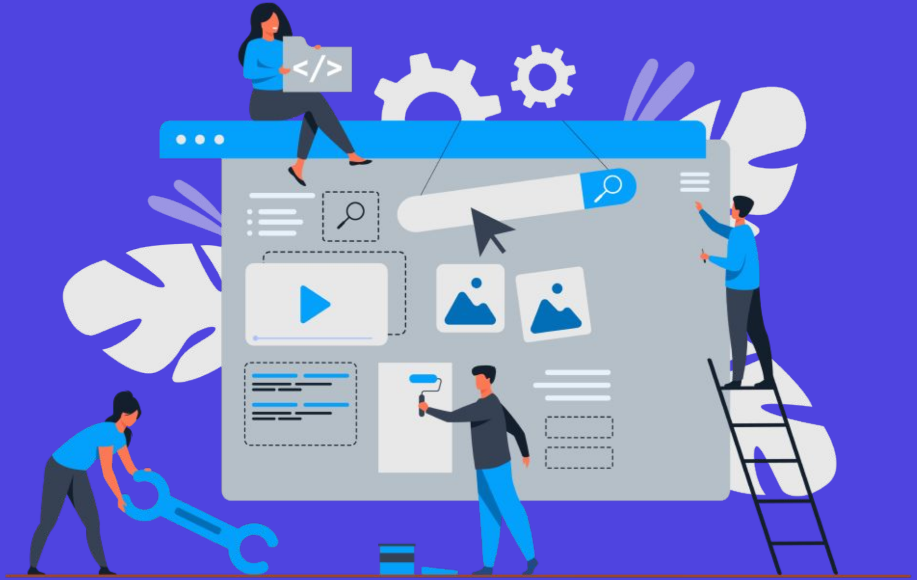
# Binary Search (2D Arrays configuration) - Part 1

Relevel
by Unacademy



## List of Problems Involved

- Exam Hall Problem
- Common Books Problem
- Search Egg Problem
- Rat in a Grid Problem

## Exam Hall Problem

**Problem Statement –** Consider an exam hall, where students are sitting on chairs. Each chair is assigned a roll number which is a unique number and students should sit on their assigned roll number chair only. We need to arrange chairs on the basis of increasing roll numbers. Your task is to find the roll number of a student who should be seated on the middle chair.

**Input Format -**
A 2D array of M*N size will be given where each cell will represent a roll number.

- M*N will always be odd.
- Every row given is sorted

**Output Format -**
Roll number of middle chair

**Input -**
11 13 15
12 16 19
13 16 19

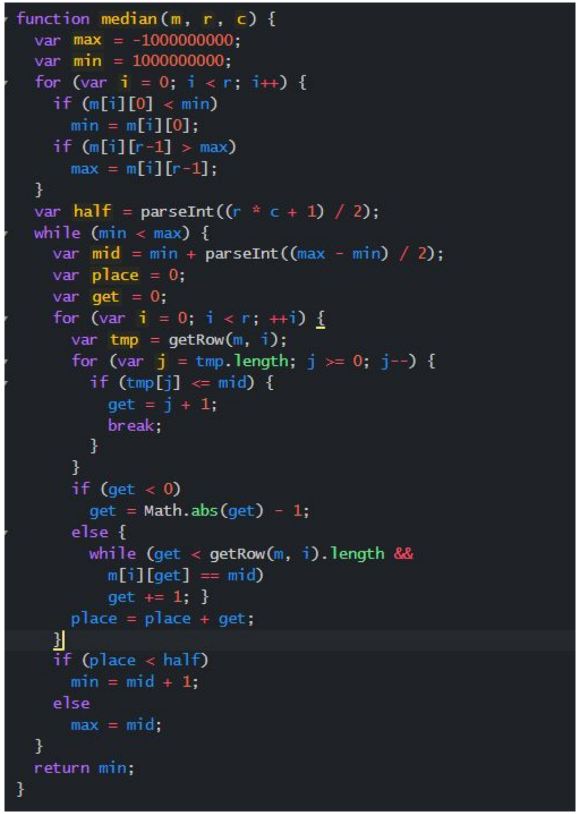
**Output -**
15

**Approach –** If we observe the problem, we need to find the median of the given matrix. A median is the middle number of sorted arrays.
It is already given that M*N is odd, hence we can use this to find the required output. We will use Binary Search here to find the middle number efficiently.
A number will be median if there are m*n/2 numbers are less than it.

Let's see each step –

1. Calculate the minimum and maximum element in the given matrix
2. Calculate mid using min and max
3. Get count of numbers which are less than or equal to mid
4. Repeat step 3 for every number and compare count with the number
5. If count < number, median > number
6. Else median < number

Code Link - https://jsfiddle.net/hf7kmnrd/ , https://ideone.com/AuichO



```
function median(m, r, c) {
  var max = -1000000000;
  var min = 1000000000;
  for (var i = 0; i < r; i++) {
    if (m[i][0] < min)
      min = m[i][0];
    if (m[i][r-1] > max)
      max = m[i][r-1];
  }
  var half = parseInt((r * c + 1) / 2);
  while (min < max) {
    var mid = min + parseInt((max - min) / 2);
    var place = 0;
    var get = 0;
    for (var i = 0; i < r; ++i) {
      var tmp = getRow(m, i);
      for (var j = tmp.length; j >= 0; j--) {
        if (tmp[j] <= mid) {
          get = j + 1;
          break;
        }
      }
      if (get < 0)
        get = Math.abs(get) - 1;
      else {
        while (get < getRow(m, i).length &&
          m[i][get] == mid)
          get += 1; }
      place = place + get;
    }
    if (place < half)
      min = mid + 1;
    else
      max = mid;
  }
  return min;
}
```

## Common Books Problem

**Problem Statement –** Consider a library where multiple books are arranged in different racks. Consider a rack where there are multiple layers in which books are placed. Every book has a unique ID. There are some books which are common in the layers. Your task is to find those common books.

**Input Format -**
A 2D array of M*N size will be given where each cell will represent a unique book id.

**Output Format -**
Common id of books

**Input -**
```
16 13 19
12 16 19
13 16 19
```

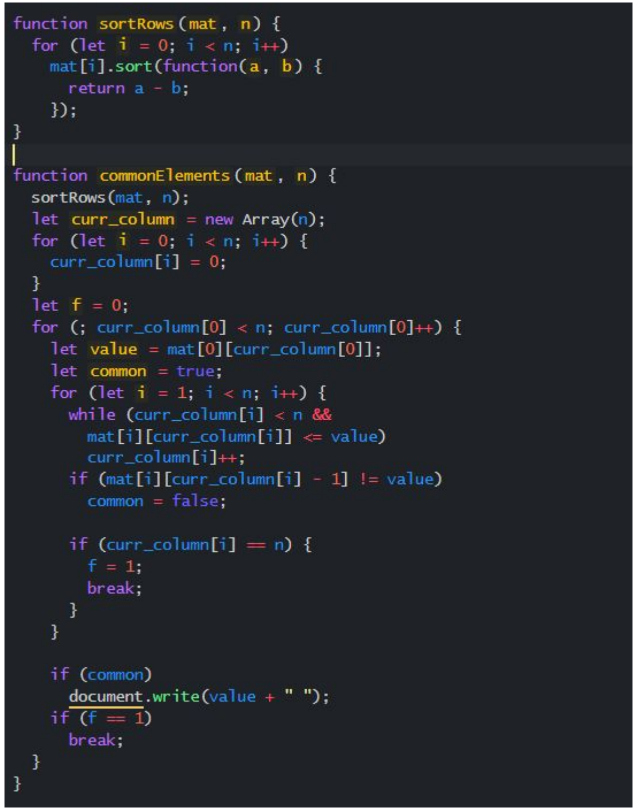
**Output -**
```
16, 19
```

**Approach –** If we observe the problem, we need to find the common elements in all the rows of a given matrix. We can sort each row of a given matrix and the problem will become like common elements in multiple sorted arrays.

Let's see each step –

1. Sort each row
2. Create a current_column array to store current column index for each row
3. Fetch value at current column for first row
4. Search value in remaining rows
5. If value present in all rows, print the value
6. Once all rows have been traversed, end the loop

Code Link - https://jsfiddle.net/jfmwk3ge/  Code Link - https://ideone.com/8BMFyH



```
function sortRows(mat, n) {
  for (let i = 0; i < n; i++)
    mat[i].sort(function(a, b) {
      return a - b;
    });
}

function commonElements(mat, n) {
  sortRows(mat, n);
  let curr_column = new Array(n);
  for (let i = 0; i < n; i++) {
    curr_column[i] = 0;
  }
  let f = 0;
  for (; curr_column[0] < n; curr_column[0]++) {
    let value = mat[0][curr_column[0]];
    let common = true;
    for (let i = 1; i < n; i++) {
      while (curr_column[i] < n &&
        mat[i][curr_column[i]] <= value)
        curr_column[i]++;
      if (mat[i][curr_column[i] - 1] != value)
        common = false;

      if (curr_column[i] == n) {
        f = 1;
        break;
      }
    }

    if (common)
      document.write(value + " ");
    if (f == 1)
      break;
  }
}
```

## Search Egg Problem

**Problem Statement –** Consider an egg tray. Each egg is assigned a unique ID. In each row and column of the given tray, eggs are already sorted based on there unique IDs. Your task is to find an egg with given ID.

**Input Format -**
A 2D array of M*N size will be given where each cell will represent a unique id of an egg.
An integer = ID of egg

**Output Format -**
Position of egg in the given tray

**Input -**

```
16 13 19
12 36 29
23 26 39
13
```

**Output -**

```
0,1
```

**Approach –** If we observe the problem, we need to find search an element in the given 2D array.

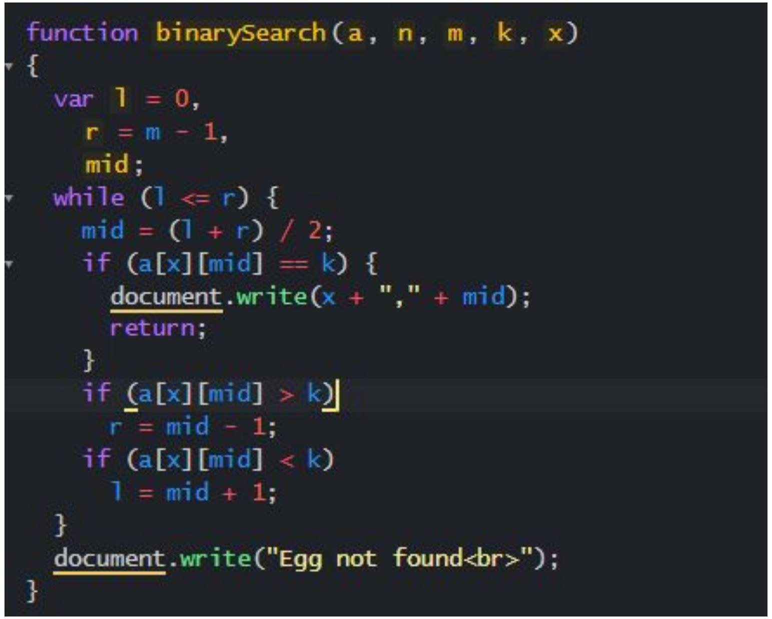
As per the conditions, eggs are already sorted row-wise and column-wise.
Based on this, we can implement Binary Search algorithm to find the required output.

We will apply binary search first to find the row where our required egg is present and then we will again apply binary search to find the position of given egg.

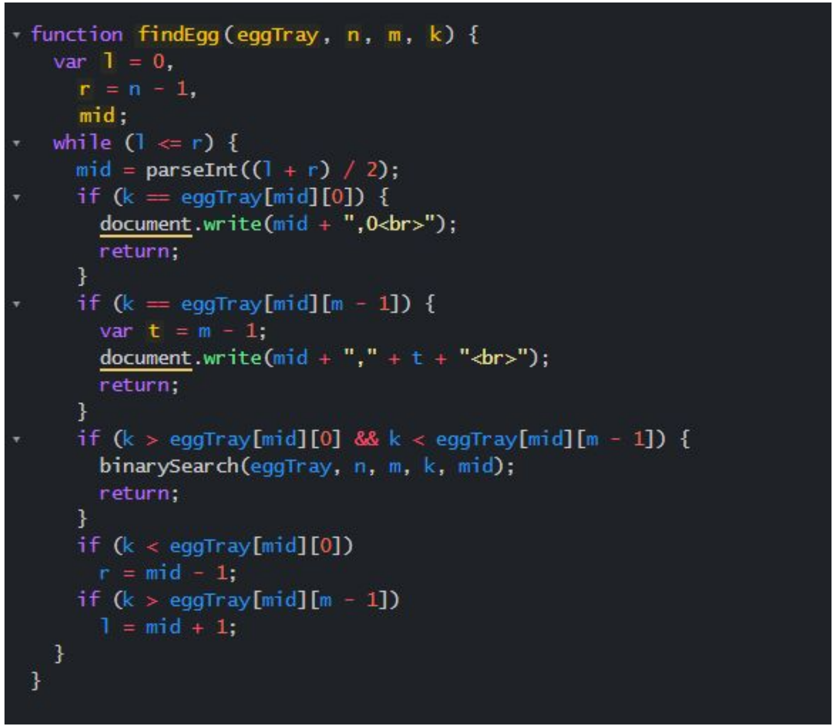
Let's see each step –

1. Initialize low = 0 and high = n-1 where n = number of columns
2. Initialize mid = middle row of given egg tray
3. Let say K = egg Id which we needs to find
4. If k < eggTray[mid][0] -> high = mid-1
5. If k > eggTray[mid][m-1] -> low = mid + 1
6. If k > eggTray[mid][0] and k < eggTray[mid][m-1] – implement binary search in current row taking it as 1D array
7. If k == eggTray[mid][0] and k == eggTray[mid][m-1] -> print the position

Code Link - https://jsfiddle.net/rwexqmf7/  Code Link -  https://ideone.com/Y4qzSA



```
function binarySearch(a, n, m, k, x)
{
  var l = 0,
    r = m - 1,
    mid;
  while (l <= r) {
    mid = (l + r) / 2;
    if (a[x][mid] == k) {
      document.write(x + "," + mid);
      return;
    }
    if (a[x][mid] > k)
      r = mid - 1;
    if (a[x][mid] < k)
      l = mid + 1;
  }
  document.write("Egg not found<br>");
}
```



```
function findEgg(eggTray, n, m, k) {
  var l = 0,
    r = n - 1,
    mid;
  while (l <= r) {
    mid = parseInt((l + r) / 2);
    if (k == eggTray[mid][0]) {
      document.write(mid + ",0<br>");
      return;
    }
    if (k == eggTray[mid][m - 1]) {
      var t = m - 1;
      document.write(mid + "," + t + "<br>");
      return;
    }
    if (k > eggTray[mid][0] && k < eggTray[mid][m - 1]) {
      binarySearch(eggTray, n, m, k, mid);
      return;
    }
    if (k < eggTray[mid][0])
      r = mid - 1;
    if (k > eggTray[mid][m - 1])
      l = mid + 1;
  }
}
```

## Rat in a Grid Problem

**Problem Statement -** Consider a grid in which a rat is present at some specific position and we need to search that rat. There can be multiple path to reach to that rat. Your task is to count the number of unique path which are present to reach to that rat.
Source and Destination position will be given in the input. Destination is where rat is present in the grid.

**NOTE –** we can move only in right and downward direction.

**Input Format -**
A 2D array of M*N size will be given which is representing a grid.
Source – position where you need to start to search rat.
Destination – position of rat

**Output Format -**
Count of unique path present to reach rat

**Input -**

11 12 13

14 15 16

Source = (0,0), Destination – (1,2)

**Output -**

3

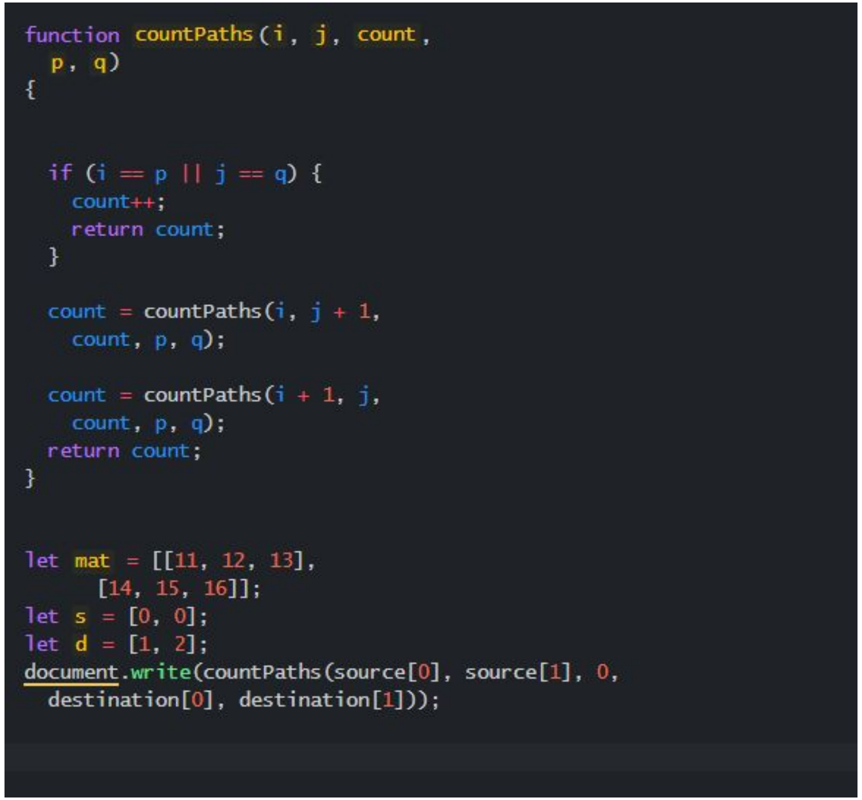
**Approach –** If we observe the problem, we need to find the count of unique path between source and destination.

We can use recursion technique here to find the required output.

Let's see each step –

1. Create a recursive function – countPaths()
2. Initialize (i,j) as source position and (p,q) as destination position
3. Initialize count=0
4. Base condition in countPaths() – if i=p and j=q -> increment count and return
5. To move right – call countPaths with j = j+1
6. To move down – call countPaths with i = i+1
7. return count

Code Link - https://jsfiddle.net/41np6z9t/  Code Link - https://ideone.com/9U32Ux



```
function countPaths(i, j, count,
  p, q)
{


  if (i == p || j == q) {
    count++;
    return count;
  }

  count = countPaths(i, j + 1,
    count, p, q);

  count = countPaths(i + 1, j,
    count, p, q);
  return count;
}


let mat = [[11, 12, 13],
      [14, 15, 16]];
let s = [0, 0];
let d = [1, 2];
document.write(countPaths(source[0], source[1], 0,
  destination[0], destination[1]));
```

Thank You!