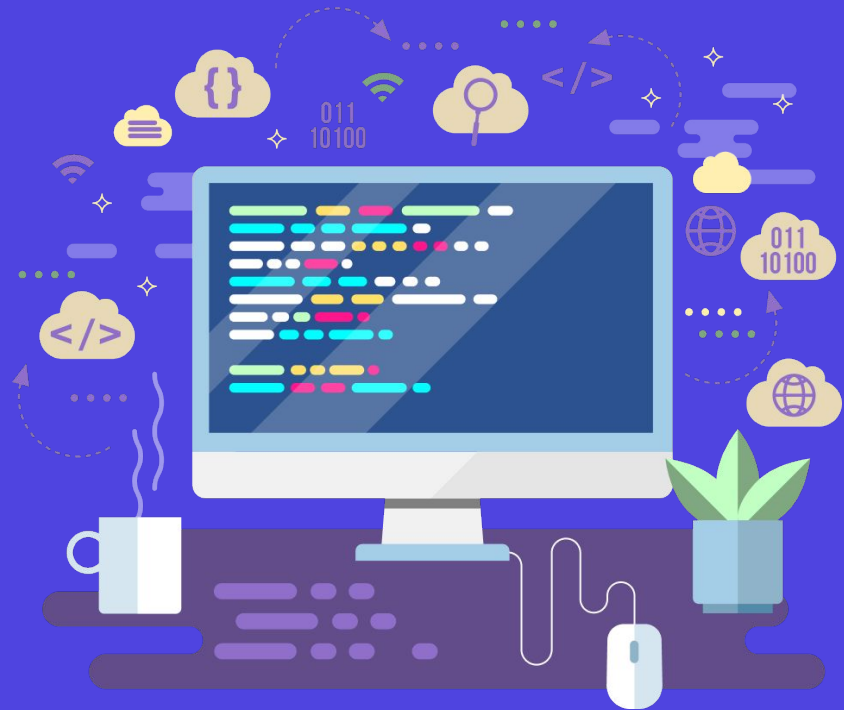


# Advanced Problem Solving: Binary Search

**Relevel**  
by Unacademy



## What does it take? ( 5 minutes)

In this lesson, we will see some advanced concepts on binary search that help you master it.

Please recap the following concept covered in the previous session:

- Problem solving using binary search.

## List of problems:- ( 5 minutes)

- Problem 1 - Aggressive Cows
- Problem 2 - Book allocation
- Problem 3 - Painter partition
- Problem 4 - Median of sorted arrays
- Problem 5 - Find the Peak Element

# Recap to Binary Search: ( 5 minutes)

- Binary Search is an efficient algorithm, and it is very fast to search an element in a sorted list of elements.
- The algorithm works on the divide and conquer method, whereas it refers to breaking down a problem into a few smaller problems of the same size until there are a few smaller problems.
- Binary Search is popular for being  $O(\log n)$ , which means the time complexity of the operation is proportional to the log of its input size.

# Divide and Conquer Algorithm: ( 5 minutes)

Divide and Conquer is an algorithm paradigm similar to greedy and dynamic programming. Follow these steps in this algorithm to solve the problems.

- Divide: we need to break the problem into smaller parts of the same type. Each sub-problem should represent a part of the main problem. This step is achieved by a recursive approach to divide the problem until no sub-problem is further divisible.
- Conquer: this step gets a lot of sub-problems to be solved recursively. Generally, in this stage, problems are solved on their own.
- Combine: when the smaller sub problems are resolved again by using recursion we need to combine the sub problem until the solution for the original problem is obtained.

# PROBLEM 1: Aggressive Cows (30 minutes)

Farmer Vijay has a larger barn, with 'n' stalls. The stalls are located at the position from 'b1.....bn' along a straight line. His cows 'c' are not interested in his barn layout and become aggressive towards each other once put into a stall, to avoid the cows getting hurt each other. The Farmer wishes to assign the cows to stalls, such that the minimum distance between any two of them is as large. Find the largest minimum distance.

## Constraints:

- $n(2 \leq n \leq 100000)$
- $b1.....bn(0 \leq ai \leq 1000000)$
- $c(2 \leq c \leq N)$

## Input:

- No of stalls (n)
- Array of stalls (arr[n])
- No of cows(c)

## Sample Input:

5

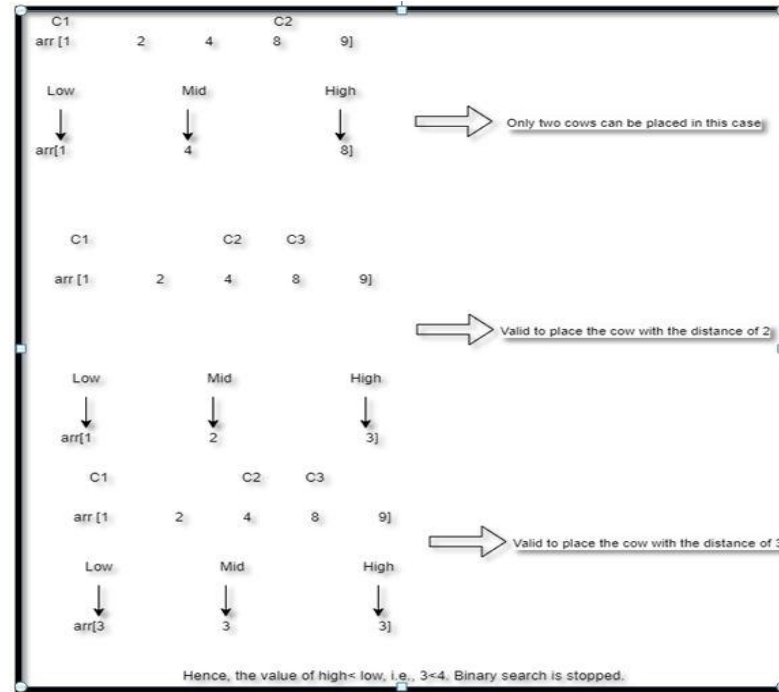
[1 2 4 8 9]

3

Sample Output: 3

# Solution:

Given, the number of stalls are 5 and an array of stalls, which contains the location of the stalls and cows are 3.



We can place the cows among the 5 stalls, such that we get the largest distance between them.

- **Case 1:** In this case, 1 is low and 9-1 is 8 will be high. Here, the search space is 1 to 8 and the mid value is 4. i.e., ( $\text{mid} = \text{low} + \text{high} / 2$ ). We can analyse that if we can place the cows with the distance 4 in between them, C1 is placed at stall 1 and C2 is placed at stall 8. C3 cannot be placed at stall 9 as it needs a minimum distance of 4. Hence, only two cows can be placed in this approach.

In our case, we need to place three cows. Let's reduce the search space to 1 to 3 and the mid value is 2.

- **Case 2:** The C1 is placed at stall 1, C2 at stall 4 and C3 at stall 8. With the distance of 2. We were able to place all the three cows in the stalls.

We need the largest minimum distance, so increase the search space and move to the right, low is 3, high is 3 and mid is 3.

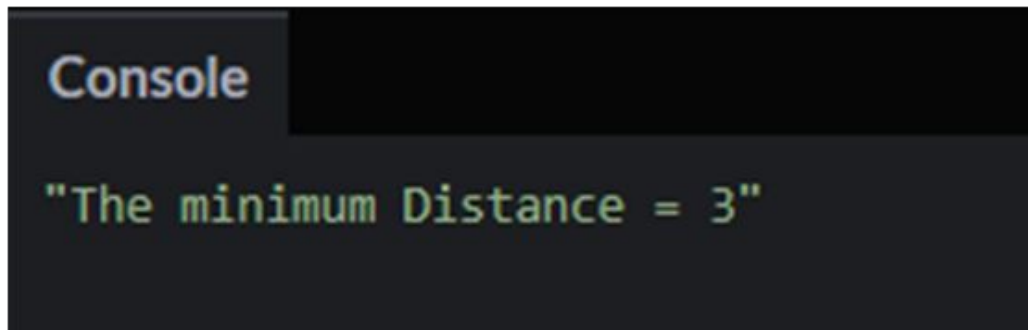
- **Case 3:** The C1 is placed at stall 1, C2 at stall 4 and C3 at stall 8. With the distance of 3. We were able to place all the three cows in the stalls.

After increasing the search space, low to 4 and high to 3. In this case, the value of low is greater than the value of high. Hence, we can stop the binary search at this point.

Therefore, 3 is the largest minimal distance between the cows among all.

**Code link :** [link:https://jsfiddle.net/gokul5a738/hqpu27sy/](https://jsfiddle.net/gokul5a738/hqpu27sy/)

**Output:**



```
Console  
"The minimum Distance = 3"
```

**Complexity Analysis:**

Time complexity:  $O(N \log N)$

Space complexity:  $O(N)$



## PROBLEM 2: Book Allocation (30 minutes)

- We are given the 'n' number of books arranged in sorted order. Where each book has a different number of pages. You have to allocate books to 's' count of students so that the maximum number of pages allotted to a student is minimum.
- Each student will be allotted at least one book and in contiguous order. Find the minimum possible number and if the assignment is not valid, return -1.

### Constraints:

- $n(1 \leq n \leq 10^5)$
- $arr(1 \leq arr[i] \leq 10^5)$

### Input:

- Array of books(arr[])
- No of students(s)

### Sample Input:

[12, 34, 67, 90]

2

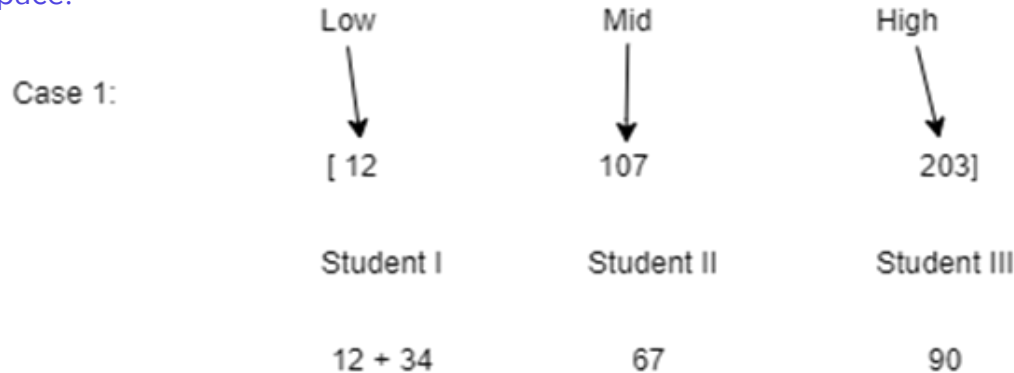
### Sample Output:

113

# Solution:

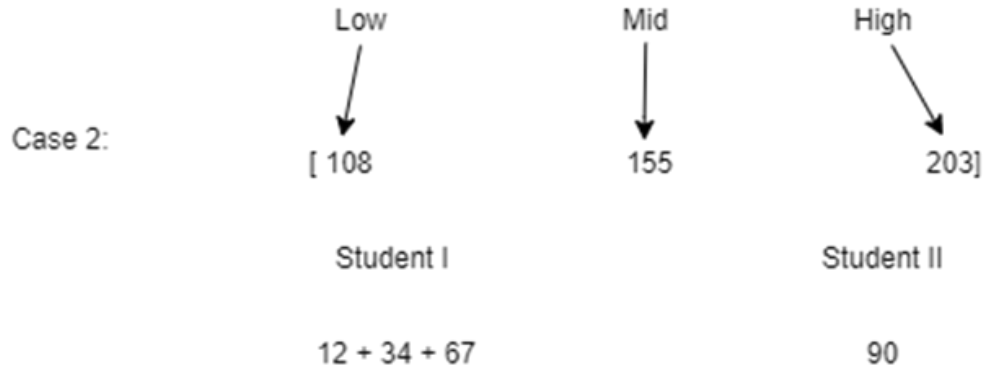
- The students should be assigned with the books such that the combined sum of the pages will be minimum comparing all the other possible combinations.
- We would be using the same binary search approach to find the minimum pages combination from the array of books. Check, if we are able to allocate books, such that the number of pages allocated to each student is less than or equal to the value of mid.
- **Case 1:** Initially, the search space has the value low is 12 and the value of high is the sum of array values i.e.,  $12 + 34 + 67 + 90 = 203$ . Mid value is 107. We start allocating pages to two students such that the number of pages allocated do not exceed the mid value.

Here, we are able to allocate all the pages to 3 of the students but, in our case we need to allocate it to two students. So, we can increase the search space.



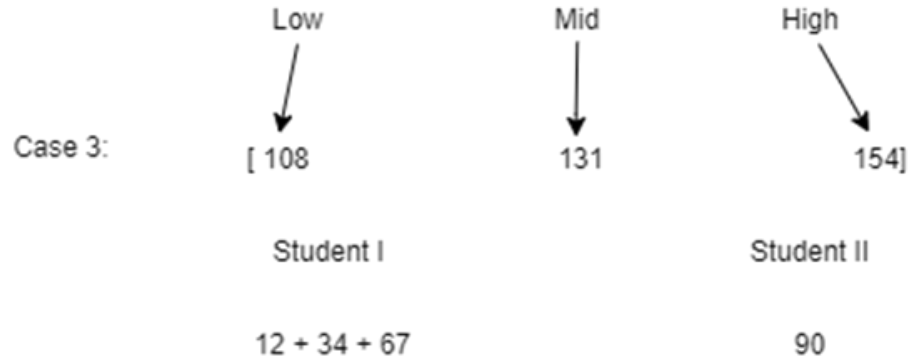
- **Case 2:** By assigning the mid value to 155, we are able to allocate all the books to two students. The possible result will be 155.

Whereas, we need the minimal value so, we repeat the above steps until we reach the minimal value.

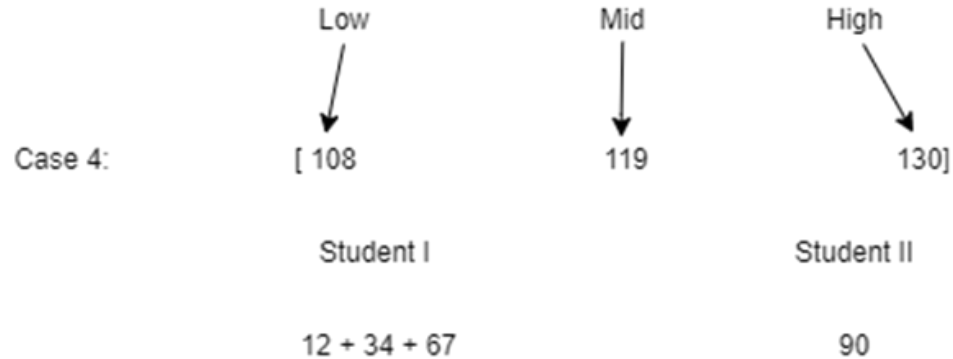


- **Case 3:** By assigning the mid value to 131, we are able to allocate all the books to not more than two students. The possible result will be 131.

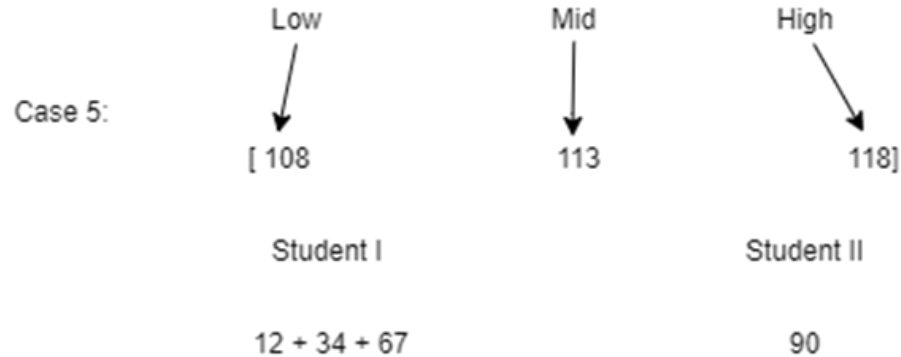
So, we can reduce the search space.



- **Case 4 :** Repeat the step until we get the minimal value.

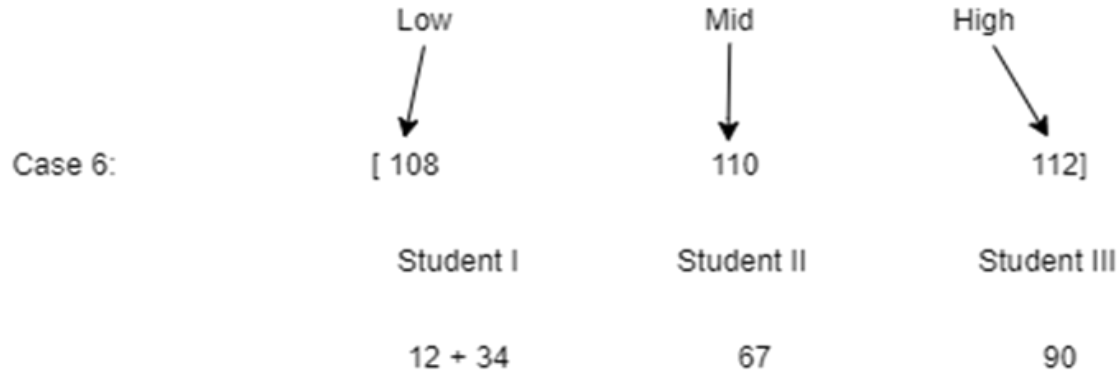


- **Case 5:** By assigning the mid value to 113, we are able to allocate all the books to not more than two students. The possible result will be reduced to 113.



- **Case 6:** In this case, as it exceeds the number of students it is not possible to allocate the books to two students.

The search space will be reduced from low- 113 to high - 112. Hence, the search space is beyond the limit where the value of low exceeds the value of high. (i.e.,  $high < low$ ).



Therefore, the minimum value of the maximum number of pages in book allocation is 113.

**code link:** <https://jsfiddle.net/gokul5a738/au2y68vs/3/>

### Output:

Console

```
"Minimum number of pages = 113"
```

### Complexity Analysis:

Time complexity:  $\log N(N)$

Space complexity:  $O(1)$

## PROBLEM 3: Painter partition (30 minutes)

You are provided with 'n' number of boards of length [b1, b2,...bn]. There are 'a' painters available in the market and each painter takes one(1) unit time to paint one(1) unit of board. We need to find the minimum time to get the job done under the constraints that any painter will only paint contiguous sections of the board.

**Note:** Two painters cannot share boards to paint.

### Constraints:

- $a(1 \leq n \leq 1000)$
- $arr(1 \leq arr[i] \leq 10^5)$

### Input:

- Array of boards(arr[])
- No of painters(a)

### Sample Input:

[10, 20, 30, 40]

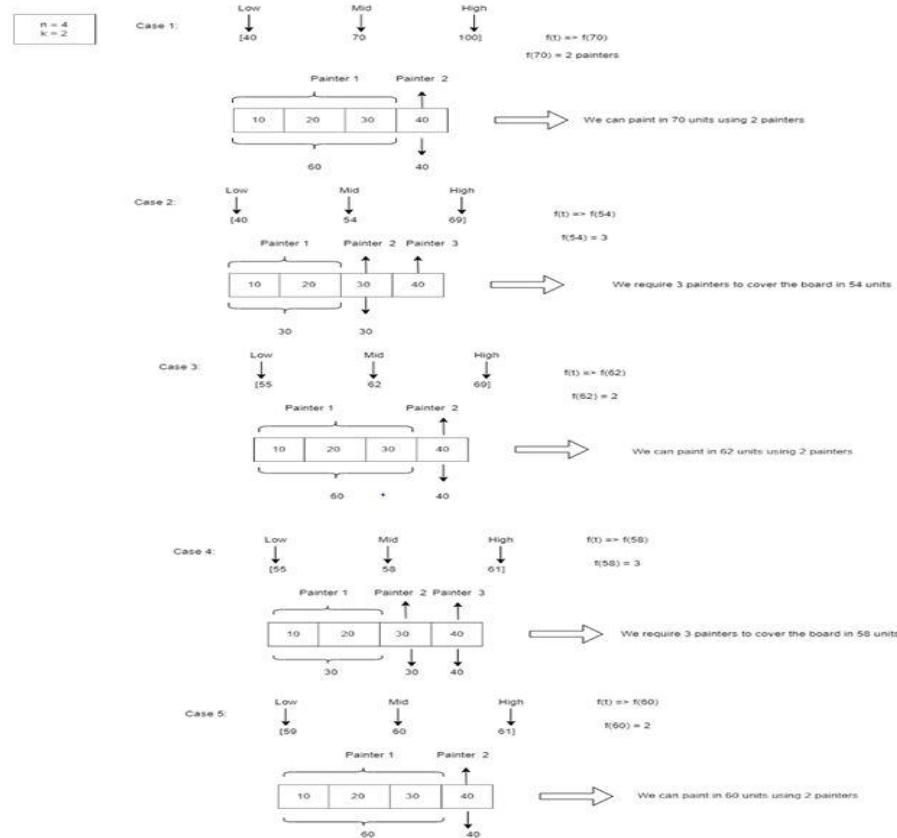
2

### Sample Output:

60



# Solution:



The highest value will be the sum of the board is 100 and lowest possible value is 40 for the minimum time.

The search space range is from 40 to 100.

- **Case 1:** In this case, the mid value will be 70. The minimum number of painters required to paint the board in the 70 units of time is 2 painters.

In the function, we keep assigning boards to the painters till the time taken is less than or equal to  $t$  in  $f(t)$ . Whenever we run out of time, we assign the board to other painters and repeat the steps.

Since it takes 70 units of time, our goal is to minimize the time. So, reduce the search space.

- **Case 2, 3 & 4:** Repeat the step until we get the minimal time.
- **Case 5:** In the 60 units of time, the two painters can paint the board by painting B1, B2 and B3 by painter 1. And B4 by painter 2,  $f(60) = 2$ .

After reducing the search space range value, low to 59, high to 59 and mid to 59. In this case, we need three painters. So, we can terminate at this point.

$t = 5$ , time unit = 60

$\Rightarrow 60 * 5$

$\Rightarrow 300$

In this case,  $t = 5$  so we need 300 units of time to paint the board with 2 painters.

**Code link:** <https://jsfiddle.net/gokul5a738/fntzxqww/1/>

**Output:**

Console

```
"Minimum time taken is 17 units"
```

**Complexity Analysis:**

Time Complexity:  $O(n \log n)$

Space Complexity:  $O(1)$

## PROBLEM 4: Median of two sorted arrays (30 minutes)

We are given with two sorted arrays  $a[]$  and  $b[]$ , the task is to find the median of these two sorted arrays, where the number of elements in the first array is  $n$  and the number of elements in the second array is  $m$ .

### Constraints:

- $a(1 \leq n \leq 100)$
- $b(1 \leq m \leq 100)$
- **Input:**
- Array of numbers
- Array of numbers

### Sample Input:

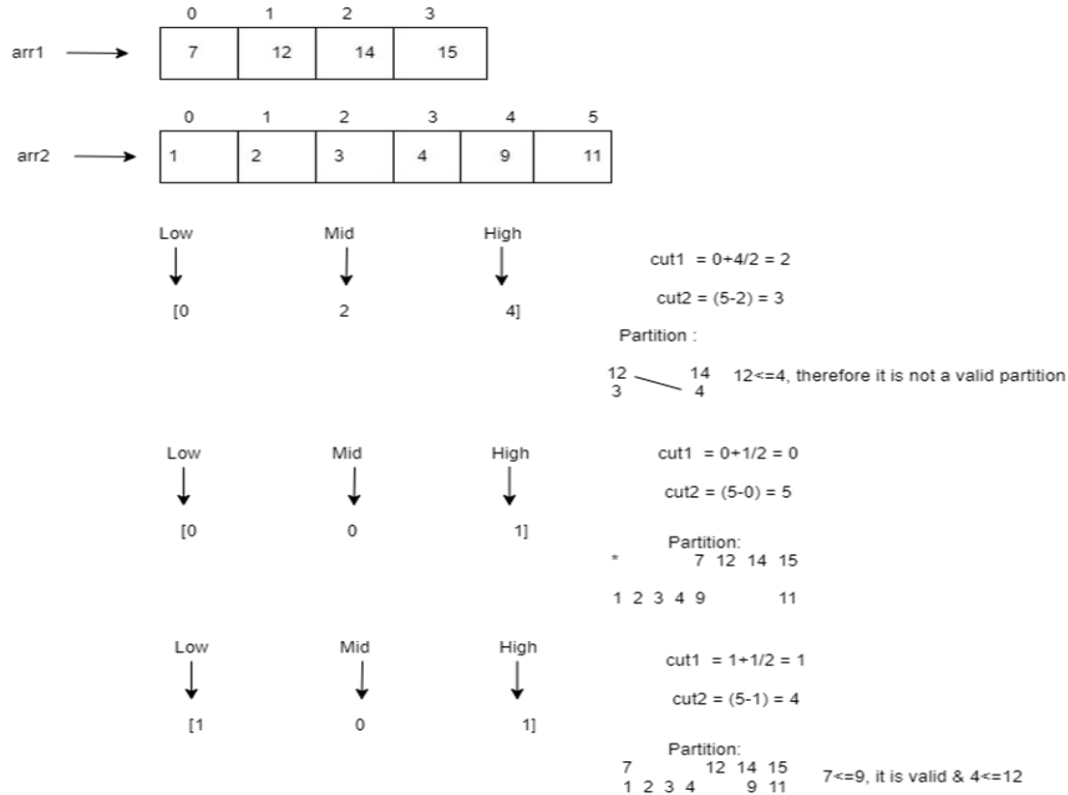
[7, 12, 14, 15]

[1, 2, 3, 4, 9, 11]

### Sample Output:

The median is 8.

# Solution:



To check the partition until it is valid, we can take a minimum of 0 element and maximum of 4 from arr1. Therefore, low is 0 and high is 4.

- **Case 1:** In this case, according to the cut we take 2 elements from arr1 and 3 elements from arr2.

#### To check the partition:

- l1, l2, r1, r2 - compare and check if they are valid or not.
- $l1 = arr1[cut1-1]$
- $l2 = arr2[cut2-1]$
- $r1 = arr1[cut1]$
- $r2 = arr2[cut2]$
- $median = \max(l1, l2) + \min(r1, r2) / 2$
- $cut1 = (low + high) / 2$
- $cut2 = (n1 + n2) / 2 - cut1$

Comparing l1 and r1,  $l2 \leq 4$ ; it is not a valid partition. So reduce the value of l1. (i.e.,  $high = mid - 1$ ).

- **Case 2:** Picking up 0 elements from arr1 and 5 elements from arr2. Here,  $l1 < r2$  and to check  $l2 \leq r1$ ,  $9 \leq 7$  is not valid. So, decrease the value of  $l2$  i.e., 9 and increase the value of  $r1$  i.e., 7.

**Note:** If there is no element in  $l1$ , assign the value of  $l1$  to `int_min`(minimal value).

- **Case 3:** In this case, the value of low is moved to  $mid+1$  i.e.,  $low = mid + 1$ ,  $low = 0 + 1 \Rightarrow low = 1$ . Picking up one element from arr1 and 4 elements from arr2.

Here, to check the partition,

- $l1 \leq r2 \Rightarrow 7 \leq 9$  - valid
- $l2 \leq r1 \Rightarrow 4 \leq 12$  – valid

**To find the median:-**

- Take the maximum number from the left and minimum number from the right.
- $7 + 9 / 2 = 8$ .

Therefore, the median of two sorted arrays is 8.

**Code link:** <https://jsfiddle.net/gokul5a738/2xkb6ga3/1/>

### Output:

Console

```
"Median of the two arrays are 3"
```

### Complexity Analysis:

Time Complexity:  $O(\min(\log m, \log n))$

Space Complexity:  $O(1)$



## PROBLEM 5: Find Peak Element (30 minutes)

We are given an array of 'n' numbers. Our task is to find the peak element from the given array of numbers and return its index. The array element is a peak element if it is not smaller than its neighbor element. Sometimes an array can have multiple peak elements, so return the index of the first peak element of the array.

### Constraints:

- $arr[1] \leq n \leq 100$

### Input:

- Array of numbers

### Sample Input:

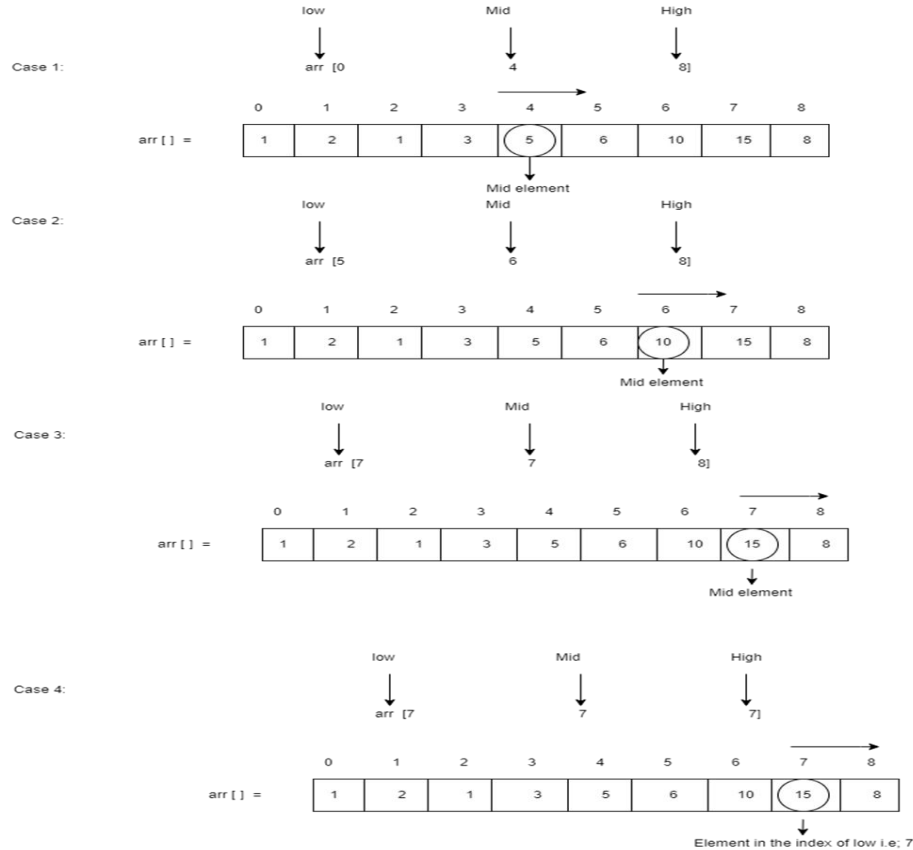
[1, 2, 1, 3, 5, 6, 10, 15, 8]

### Sample Output:

15

## Solution:

- Let's take the input array, and iterate in the linear method to check whether the current element is higher than the both of its neighbours. (i.e,  $i-1$  &  $i+1$ ). We repeat the steps until we reach this condition.
- If the condition is met, we can abort the process and return that particular index of  $i$ .
- Here, the corner elements will have only one neighbor. First element will have only one neighbor at the right and the last element will have only one neighbor at the left.
- We need to return only one peak element.
- We compare the middle element with its neighbors. If the middle element is not smaller than both of its neighbors, then we return it.

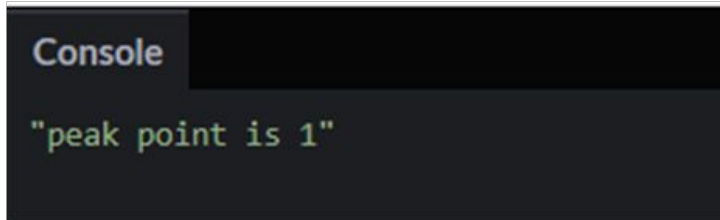


- **Case 1:** The index of the middle element is 4, the element is 5. We can compare the middle element 5 to the neighboring element 6 on the right. (i.e.,  $low = mid + 1 = 5 + 1 \Rightarrow 6$ ). The middle element is lesser than the adjacent neighbor so, move to the right direction.
- **Case 2:** The middle element is 10 and the index is 6. Comparing it with the adjacent element 15, it is greater. So, increasing the search space, low value is increased to 7.
- **Case 3:** In this case, the middle element would be 15 and the index is 7. By comparing with the neighbor element, (i.e.,  $low = mid + 1 = 7 + 1 \Rightarrow 8$ ). The element is lesser so, we assign the value of high to low i.e.,  $low = 7$ .
- Hence, we find that both the values are equal; we shall abort the process here.

The element which is held at the index low is the peak element. Therefore, the peak element of the array is 15.

**Code link** [h:https://jsfiddle.net/gokul5a738/2xfeo0qu/](https://jsfiddle.net/gokul5a738/2xfeo0qu/)

**Output:**

A screenshot of a web browser's console. The console has a dark background with a tab labeled 'Console'. Below the tab, the text '"peak point is 1"' is displayed in a green monospace font.

**Complexity Analysis:**

Time Complexity:  $O(\log 2(n))$

Space Complexity:  $O(1)$

# Practice Homework:(10 minutes)

- Count of Range Sum
- Search in Rotated Sorted Array

**Reference:** <https://leetcode.com/tag/binary-search/>

**Thank you**