Max-Min problem

The Max-Min problem involves finding the maximum and minimum elements in an array. Using the **Divide and Conquer** method, the problem is solved by dividing the array into smaller subarrays, solving the problem for these subarrays, and then combining the results to find the final maximum and minimum.

Algorithm:

1.Divide:

1. Split the array into two halves until each half contains only one or two elements.

2.Conquer:

1. If the subarray contains a single element, both the maximum and minimum are the same (that element).
2. If the subarray contains two elements, compare them to find the maximum and minimum.

3.Combine:

1. Recursively merge results from the two halves by comparing their maximums and minimums to determine the overall maximum and minimum.

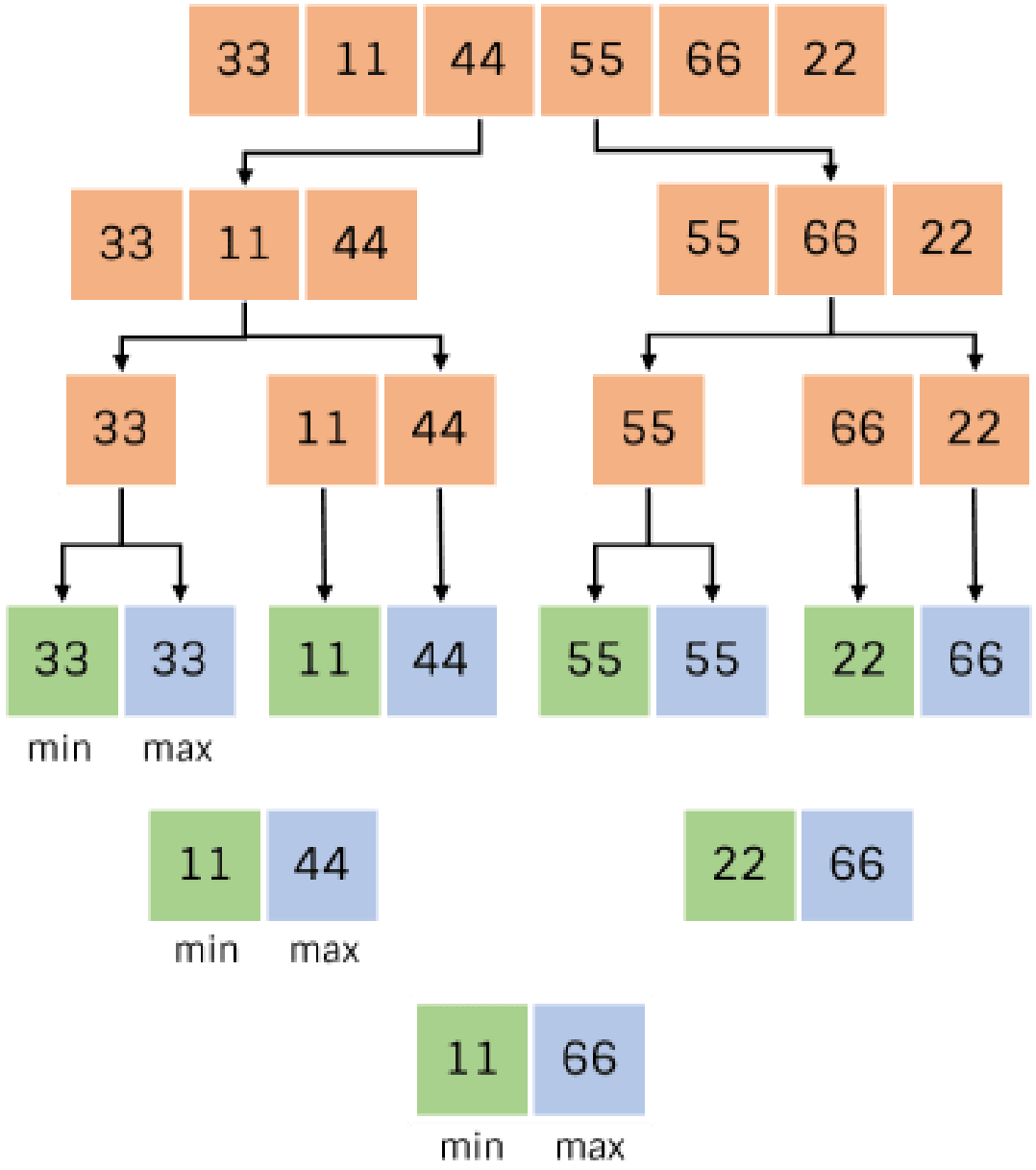
```
MaxMin(arr, low, high):
    If low == high:
        Return (arr[low], arr[low]) // Single element: max and min are the same

    Else If high == low + 1:
        If arr[low] > arr[high]:
            Return (arr[low], arr[high]) // Two elements
        Else:
            Return (arr[high], arr[low])

    Else:
        mid = (low + high) // 2
        (max1, min1) = MaxMin(arr, low, mid)
        (max2, min2) = MaxMin(arr, mid + 1, high)

        overall_max = max(max1, max2)
        overall_min = min(min1, min2)

        Return (overall_max, overall_min)
```



Complexity:

•**Time Complexity**: O(n) as each element is compared at most once during merging.

•**Space Complexity**: O(logn), due to the recursion stack

Median of Two Sorted Arrays using Divide and Conquer Method

The **Divide and Conquer** approach is an efficient way to find the median of two sorted arrays without merging them completely. It is based on the idea of binary search and runs in **O(log(min(m, n)))** time.

Approach:

1. **Partition the smaller array** using **binary search**.
2. **Ensure correct partitioning** so that all elements on the left side are smaller than those on the right.
3. **Find the median** based on the partition

Algorithm (Binary Search on the Smaller Array)

1. Identify the smaller of the two arrays and apply binary search on it.
2. Set two pointers: low = 0 and high = size of smaller array.
3. Find the partition index partitionX in the smaller array.
4. Determine the corresponding partition index partitionY in the larger array.
5. Find the left and right max/min elements:
    1. maxLeftX = Largest element in the left part of nums1
    2. minRightX = Smallest element in the right part of nums1
    3. maxLeftY = Largest element in the left part of nums2
    4. minRightY = Smallest element in the right part of nums2
6. Check if partitioning is correct:
    1. If maxLeftX ≤ minRightY and maxLeftY ≤ minRightX, then we found the correct partition.
    2. If the total number of elements (m + n) is odd, return max(maxLeftX, maxLeftY).
    3. If (m + n) is even, return (max(maxLeftX, maxLeftY) + min(minRightX, minRightY)) / 2.
7. If partitioning is incorrect:
    1. If maxLeftX > minRightY, move the high pointer left.
    2. If maxLeftY > minRightX, move the low pointer right.

Example

Input:

nums1 = [1, 3, 8]
nums2 = [7, 9, 10, 11]

Step-by-Step Execution

- Nums1 has 3 elements, nums2 has 4 elements.
- We apply binary search on the smaller array nums1.
- Initial partition at partition X = (0+3)//2 = 1, so

Left part of nums1 = [1] Right part of nums1 = [3, 8]

Since nums2 has 4 elements, partitionY = (3+4+1)//2 - partitionX = 4 - 1 = 3, so:

Left part of nums2 = [7, 9, 10] Right part of nums2 = [11]

Find max/min values :

maxLeftX = 1, minRightX = 3
maxLeftY = 10, minRightY = 11

Adjust Partitioning:

Since maxLeftX (1) ≤ minRightY (11),

but maxLeftY (10) > minRightX (3), we move right in nums1.

Set low = 2, new partition at partitionX = 2.

Now

maxLeftX = 3, minRightX = 8
maxLeftY = 7, minRightY = 9

Condition satisfied: maxLeftX (3) ≤ minRightY (9) and maxLeftY (7) ≤ minRightX (8).

Compute the median: Since (3 + 4) = 7 (odd), median = max(3, 7) = 7.

Time Complexity:

- **O(log(min(m, n)))** because we apply **binary search** on the smaller array.