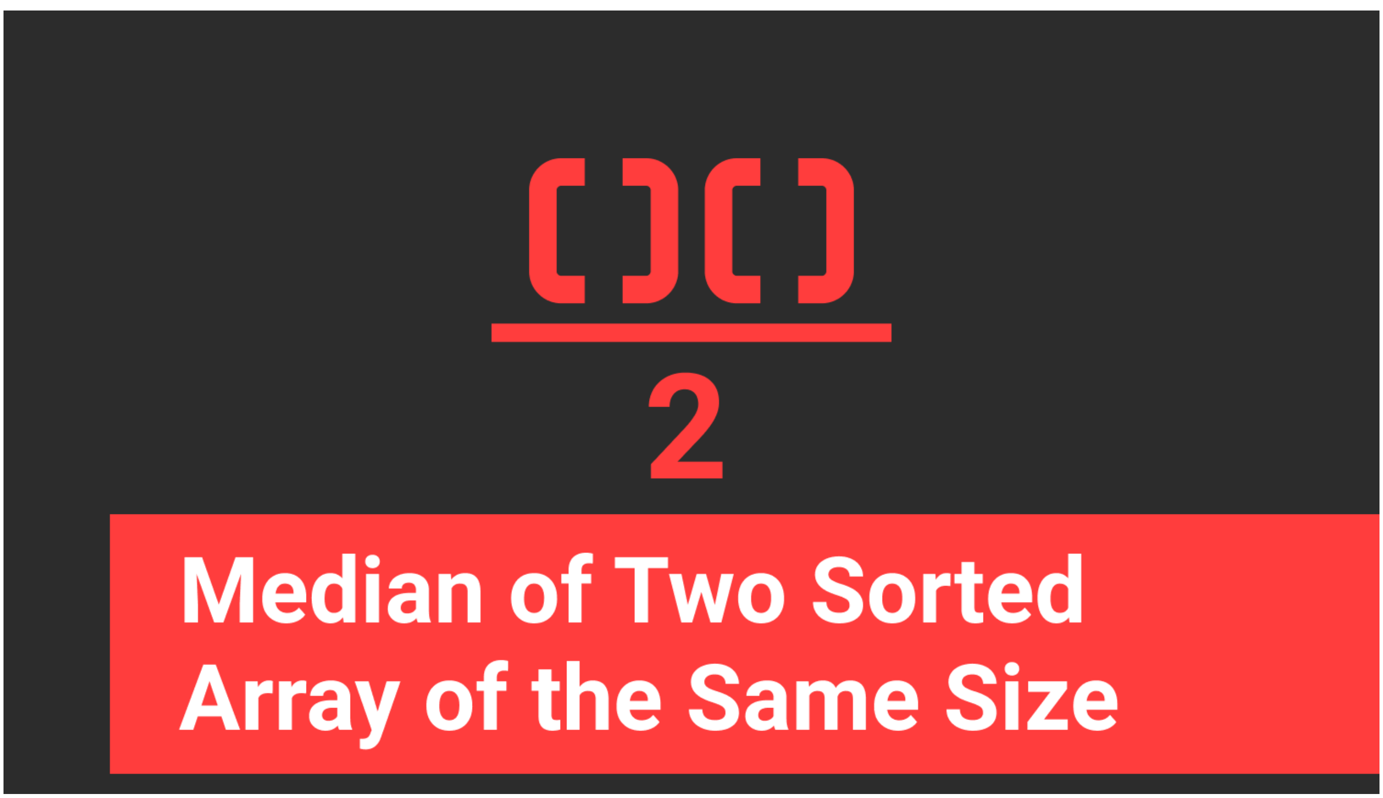
AfterAcademy

Admin AfterAcademy
29 Feb 2020

Median of Two Sorted Array of the Same Size



Understanding The Problem

Problem Description

There are two sorted arrays **nums1** and **nums2** of size n. Find the median of the two sorted arrays.

You may assume **nums1** and **nums2** cannot be both empty.

Example 1:

```
nums1 = [1, 3]
nums2 = [2, 4]

The median is 2.5 as (2 + 3) / 2 = 2.5
```

Example 2:

```
nums1 = [1, 2, 9]
nums2 = [3, 4, 7]

The median is (3 + 4)/2 = 3.5
```

What is Median?

The median is the value separating the higher half of a data sample, a population, or a probability distribution, from the lower half. For a data set, it may be thought of as the “middle” value.

For odd n, Median (*M*) = `value of ((n + 1)/2)th item term.`

For even n, Median (*M*) = `value of {(n/2)th item term + (n/2 + 1)th item term}/2`

Note → Since the size of the two input arrays for which we are looking for the median is even(2n), we need to take the average of the middle two numbers and return it.

Solutions

We will be discussing linear and logarithmic solutions for the problem

1. **Counting while comparing**—Find the middle elements after merging the sorted array using the merge procedure of merge sort.
2. **Comparing medians**—Get the medians of two sorted arrays and compare the medians and move accordingly until the two medians become equal following divide and conquer approach.

1. Counting While Comparing

For this approach, we actually have to find those two elements that could be in the middle when the two arrays are merged in sorted order.

If we follow the merge procedure like the one in merge sort then we could just compare the values at the two pointers pointing two the indexes of the two arrays and then increment the pointers accordingly while keeping track of the count. If count becomes n(For 2n elements), we have reached the median. Take the average of the elements at indexes n-1 and n in the merged array.

Solution Steps

1. Create two pointers `i` and `j` pointing two `nums1` and `nums2` respectively and initialize them with `0`
2. If the value at `nums1[i] < nums2[j]` then increment `i` otherwise, increment `j`
3. Repeat step 2 until `i` or `j` becomes `i+j` equals to n
4. Return median of the two elements at ith and jth indexes.

Pseudo Code

```
float getMedian(int num1[], int num2[], int size) {
   int i = 0
   int j = 0
   int m1 = -1, m2 = -1
   for (count = 0 to size) {
      if (i == size) {
         m1 = m2
         m2 = num2[0]
         break
      }
      else if (j == size) {
         m1 = m2
         m2 = num1[0]
         break
      }
      if (num1[i] < num2[j]) {
         m1 = m2
         m2 = num1[i]
         i = i + 1
      } else {
         m1 = m2
```

```
            m2 = num2[j]
            j = j + 1
        }
    }
    return (m1 + m2)/2
}
```

Complexity Analysis

Time Complexity: O(n), where is the size of both the input arrays

Space Complexity: O(1)

Critical Ideas to Think

- Did you recognize that the merge approach is similar to the one used in merge sort?
- Why we are returning (m1 + m2)/ 2 in the end?
- If we have created an auxiliary array and save both the arrays in a sorted manner and then return its median, will it give the correct result? If yes then what would be its time and space complexity?

2. Comparing medians

This approach works by comparing the two medians of both the array following the Divide & Conquer paradigm. If the first median is greater than the second median then we will look for the median of the two sorted arrays in `num1[0 to median1]` and `num2[median2 to n]`. (**Why?)**

We know that `median1` is larger than or equal to the first half of `num1`. And the same for `median2` and `num2`. At the same time, we also know that `median1` is smaller than or equal to the second half of `num1`.

Let's consider the case `median1 > median2`

```
num1:      [.....median1.....]
num2:      [.....median2.....]
num1+num2: [.....median2...median1........]
```

Let us call the median of the merged array `m*`. Since the first halves of `num1` and `num2` come before `median1`. So, `median1 >= m*`, this means that no values larger than `median1` need to be considered in `num1`. So we only need to look in the first half or ar1.

Similarly, since the second halves of `num1` and `num2` come after `median1`, we have that `median2 <= m*`, this means that no values smaller than `median2` need to be considered and we only need to look in the second half of `num2`.

Example →

```
num1 = {1, 12, 15, 26, 38}
num2 = {2, 13, 17, 30, 45}

med1 = 15 and med2 = 17 for num1[0 to 4] and num2[0 to 4]

med1 < med2. So median is present in {15, 26, 38} and {2, 13, 17}
now, med1 = 26 and med2 = 13

med1 > med2. So median is present in {15, 26} and {13, 17}

The size of both the subarrays are now 2. So,
median = (max(15,13) + min(26, 17))/2 = 16
```

Solution Steps

1. Calculate the median of both the arrays, say `m1` and `m2` for `num1[]`

and `num2[]`.

2. Repeat till the size of `num1` and `num2` becomes two.

- If `m1 == m2` then return m1.
- If `m1 > m2` then median will be present in either of the sub-arrays `num1[0..m1]` and `num2[m2 to n]`.
- If `m2 > m1` then median will be present in either of the sub-arrays `num1[m1 to n]` and `num2[0 to m2]`.

3. Return `Median = (max(array1[0],array2[0]) + min(array1[1],array2[1]))/2`

Pseudo Code

```
float getMedian(int num1[], int num2[], int size) {
   // base cases
   if (size <= 0)
      return -1
   if (size == 1)
      return (num1[0] + num2[0]) / 2
   if (size == 2)
      return (max(num1[0], num2[0]) + min(num1[1], num2[1])) / 2
   int med1 = median(num1, size)
   int med2 = median(num2, size)
   // compare the medians
   if (med1 == med2)
      return med1
   if (med1 < med2) {
      // recurse for the subarray num1[m1 to size] and num2[0 to m2]
      if (size % 2 == 0)
         return getMedian(num1 + size/2 - 1, num2, size - size/2 + 1)
      else
         return getMedian(num1 + size / 2, num2, size - size / 2)
   } else {
```

```
        // recurse for the subarray num1[0 to m1] and num2[m2 to n]
        if (size % 2 == 0)
            return getMedian(num2 + size / 2 - 1, num1, size - size / 2 + 1)
        else
            return getMedian(num2 + size / 2, num1, size - size / 2)
}

// Utility function to find median of array
int median(int arr[], int array_size) {
    if (array_size % 2 == 0)
        return (arr[array_size / 2] + arr[array_size / 2 - 1]) / 2
    else
        return arr[array_size / 2]
}
```

Complexity Analysis

Time Complexity: O(log n), where n is the size of both the arrays

Space Complexity: O(log n) **(Why?)**

Critical Ideas to Think

- Can you code this approach in an iterative manner?
- Why does the comparison of medians and moving accordingly will reach the correct result?
- What would be the median when the two sorted arrays are of length two?

Comparison of Different Solutions

Approach	Time Complexity	Space Complexity
Counting while	O(**n**)	O(1)

comparing		
Comparing medians (Divide & Conquer)	O(log n)	O(log n)

Suggested Problems to Solve

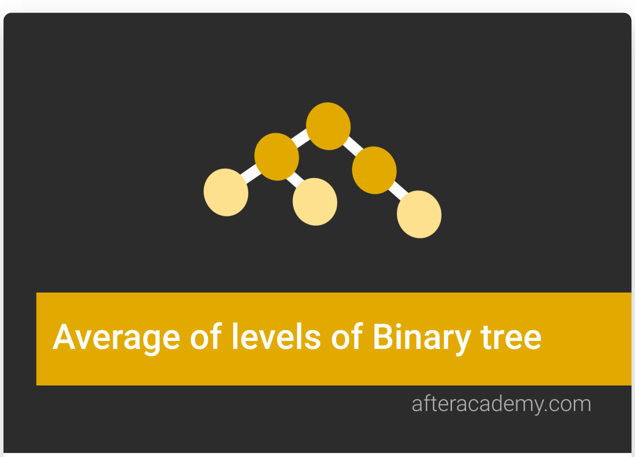
- Median of two sorted arrays of different sizes
- Median in a row-wise sorted matrix

...d Interview

Google Android Engineer Interview Process and Preparation.

CHECK NOW

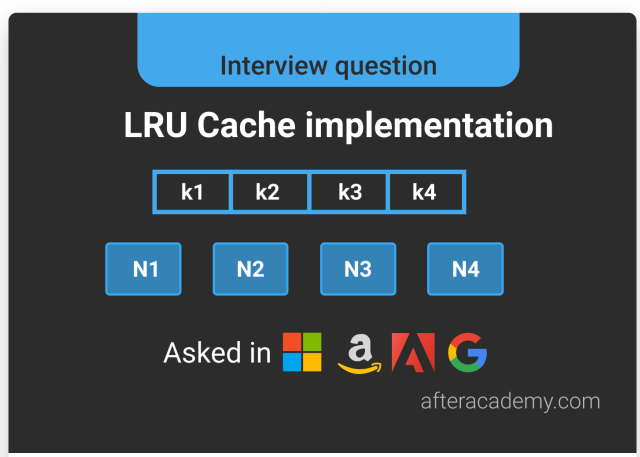
Recommended for You



Average of Levels in Binary Tree

Admin AfterAcademy
4 Nov 2020

Given a binary tree, write a program to return

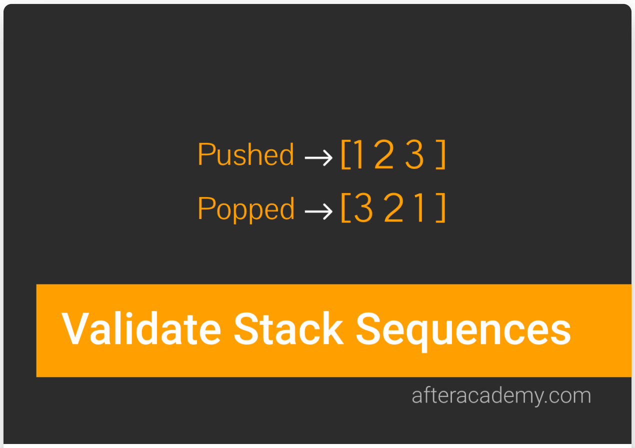


LRU Cache Implementation

Admin AfterAcademy
12 Oct 2020

...n and implement a data structure for Least Recently Used(LRU) cache. Your data the average value of the nodes on each level in the form of an array. The range of the node's value is in the range of 32-bit signed integer.

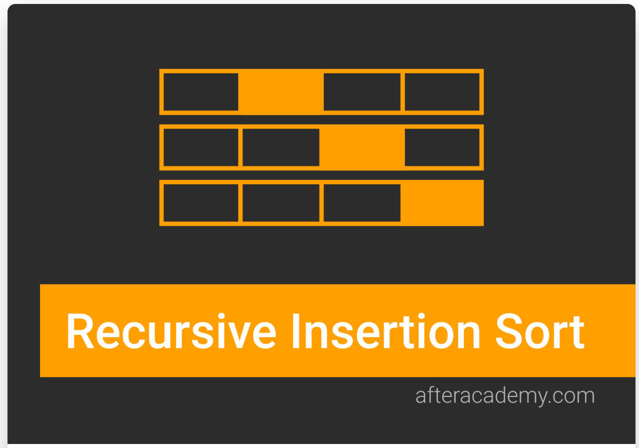
structure must support two operations: get(key) and put(). The problem expects a constant time solution



Validate Stack Sequences

Given two sequences pushed and popped with distinct values, write a program to return true if and only if this could have been the result of a sequence of push and pop operations on an initially empty stack.

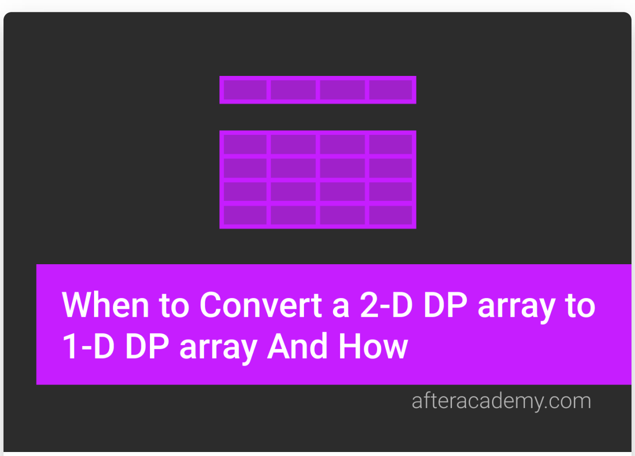
Admin AfterAcademy
4 Oct 2020



Recursive Insertion Sort

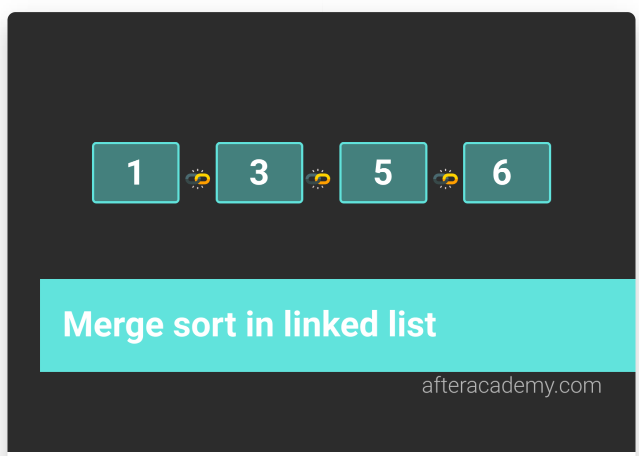
Write a program for the recursive implementation of Insertion Sort. Insertion Sort is used to sort a given array. This problem will sharpen your recursion skills.

Admin AfterAcademy
22 Sep 2020



When to Convert a 2-D DP array to 1-D DP array And How?

Admin AfterAcademy
15 Sep 2020



Sort List - Merge Sort

linked list using Merge Sort. This is a very famous interview problem that

Admin AfterAcademy
12 Sep 2020

In which situation 2 dimensional DP can be dropped to 1 dimension? Is there any principle or regular pattern? This is a very important question when it comes to optimization of DP arrays. Let's find out.

demonstrates the concept of recursion. This problem is quite similar to Merge Sort in Arrays.