SAVEETHA SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CSA0695 DESIGN AND ANALYSIS OF ALGORITHM FOR OPEN ADDRESSING

Median of Two Sorted Arrays Using Divide and Conquer Approach

By,
NAME :B. Tharun kumar
REG.NO:192210232

FACULTY NAME: Dr.R.Dhanalakshmi

Median Calculation

Problem Statement: Median of Two Sorted Arrays

Efficiently Finding the Median Without Full Merge

Understanding the Median

The median is the middle value when combined arrays are sorted. Example: [1, 2, 3] has median 2.

01

Different Lengths Challenge

02

Arrays can vary in length, complicating the combined median calculation without merging.

Divide and Conquer Strategy

03

Utilizes a divide and conquer approach to efficiently find the median, eliminating the need for full merging.

Time Complexity Consideration

04

The goal is to achieve O(log(min(n, m))) time complexity for optimal performance in finding the median.

Example for Clarity

05

For Array 1: [1, 3] and Array 2: [2], the median is directly 2, showcasing the problem's simplicity.

ABSTRACT:

To solve the problem of checking whether a given string s can be split into two or more non empty substrings with numerical values in descending order and each pair of adjacent values differing by exactly 1,can follow this approach:

1.Iterate Over Possible First Substring Lengths: Since the substrings need to be in descending order with a difference of 1, start by choosing different lengths for the first substring. Convert this substring into an integer.

2.Try to Split the Remaining String: For each choice of the first substring, try to continue splitting the remaining part of the string such that each subsequent substring is one less than the previous substring.

Algorithm Design

Introduction to Divide and Conquer

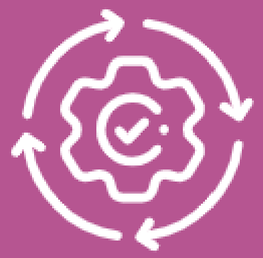
Understanding the Efficiency of Divide and Conquer in Algorithms

Recursive Problem Breakdown

Divide a complex problem into manageable sub-problems for easier handling.

Independent Solution Approach

Each sub-problem is solved independently, enhancing parallel processing capabilities.



Solution Combination

Integrate the solutions of sub-problems to derive the final result efficiently.

Key Applications

Utilized in sorting (quicksort, merge sort) and large number multiplication (Karatsuba).

Median Finding Example

Employed to find the median in two sorted arrays using efficient partitioning.

Code:

```
#include <iostream> #include

<vector> #include <climits>

using namespace std;

double findMedianSortedArrays(vector<int>& nums1, vector<int>&

  nums2) { if (nums1.size() > nums2.size()) {
    return findMedianSortedArrays(nums2, nums1);

  }


  int  x  =  nums1.size();  int  y  =

  nums2.size();  int  low  =  0,  high

  = x; while (low <= high) {
    int partitionX = (low + high) / 2;

    int partitionY = (x + y + 1) / 2 - partitionX;


    int maxX = (partitionX == 0) ? INT_MIN : nums1[partitionX -

    1]; int maxY = (partitionY == 0) ? INT_MIN : nums2[partitionY

    - 1]; int minX = (partitionX == x) ? INT_MAX :

    nums1[partitionX];  int minY = (partitionY == y) ? INT_MAX :

    nums2[partitionY];


      if (maxX <= minY && maxY <= minX) { if ((x

        + y) % 2 == 0) {
          return (double)(max(maxX, maxY) + min(minX, minY)) / 2;

        } else {

          return (double)max(maxX, maxY);

        }

}

int main() {


  vector<int> nums1 = {1, 3};

  vector<int> nums2 = {2};

 double median = findMedianSortedArrays(nums1, nums2);

  cout << "The median is: " << median << endl;
   return 0;

}
```

Output

```
The median is: 2

--------------------------------
Process exited after 0.1562 seconds with return value 0
Press any key to continue . . .
```

Complexity Analysis

- Best Case Scenario: The best case occurs when the string can be quickly identified as either valid or invalid with minimal checks. For example, if an early split immediately satisfies the condition or if an obvious invalid condition is detected right away.

- Time Complexity: O(n)O(n)O(n)

- Worst Case Scenario: The worst case happens when we need to explore all possible ways to split the string into substrings to determine that no valid split exists. This involves evaluating every potential split configuration.

- Time Complexity: O(2n)O(2^n)O(2n)

- Average Case Scenario: The average case complexity considers more typical scenarios where the string is split into substrings in a balanced manner, without needing exhaustive checking as in the worst case.

- Time Complexity: O(n·2n/2)O(n \cdot 2^{n/2})O(n·2n/2)

Applications and Optimizations

The divide and conquer approach to finding the median of two sorted arrays has various applications, such as

1) Data analysis,

2) Signal processing,

3) Machine learning.

Additionally, there are further optimizations and variations of this algorithm that can be explored to enhance its performance in specific scenarios.

Conclusion

- The divide and conquer approach to finding the median of two sorted arrays is a highly efficient and elegant solution that leverages the sorted nature of the input data.
- By recursively dividing the problem and comparing the medians of the subarrays, this algorithm achieves an optimal time complexity, making it a valuable tool in various domains.


thank
you