CAPSTONE PROJECT REPORT

Reg NO:192210708

Name: B. MAHESH REDDY

Course Code: CSA0656

Course Name: Design And Analysis Of Algorithm for Asymptotic Notation

SLOT: A

TITLE OF THE PROJECT: Valid Arrangement of Pairs using divide and conquer

ABSTRACT:

This project addresses the "Valid Arrangement of Pairs" problem using a systematic approach to ensure that each pair in a given 2D array satisfies the condition endi-1 == starti for 1 <= i < pairs.length. The solution employs a divide and conquer strategy coupled with merge sort to efficiently rearrange pairs based on their start and end values. By breaking down the problem into smaller subproblems and merging solutions in a sorted manner, the algorithm guarantees a valid arrangement of pairs. This approach ensures both correctness and efficiency, demonstrating its applicability across various input scenarios, including those with randomly ordered pairs.

INTRODUCTION:

In the problem of arranging pairs from a 2D integer array such that each pair's end matches the start of the subsequent pair, we encounter a task that hinges on maintaining continuity and validity throughout the sequence. Each pair pairs[i] = [start_i, end_i] must follow the condition end_i-1 == start_i for 1 ≤ i < pairs.length, ensuring a seamless connection between consecutive pairs.

To address this, we employ a divide and conquer strategy, leveraging the efficiency of merge sort. This approach involves breaking down the array of pairs into smaller subarrays, sorting them individually, and then merging these sorted subarrays back together. By sorting based on the end values of the pairs, we ensure that the subsequent pairs always begin where the previous one ends, thereby fulfilling the validity condition.

The implementation in C utilizes structures and functions to handle sorting and merging operations efficiently. By recursively applying merge sort, the algorithm achieves a time complexity of O(n log n), making it well-suited for large datasets. This efficiency is crucial for practical applications where performance is paramount.

In conclusion, the divide and conquer approach coupled with merge sort provides a robust solution to the "Valid Arrangement of Pairs" problem, ensuring both correctness and efficiency in arranging pairs according to the specified conditions. This methodological approach not only meets the problem's requirements but also lays a foundation for handling similar sequence-based arrangement problems effectively in computational contexts.

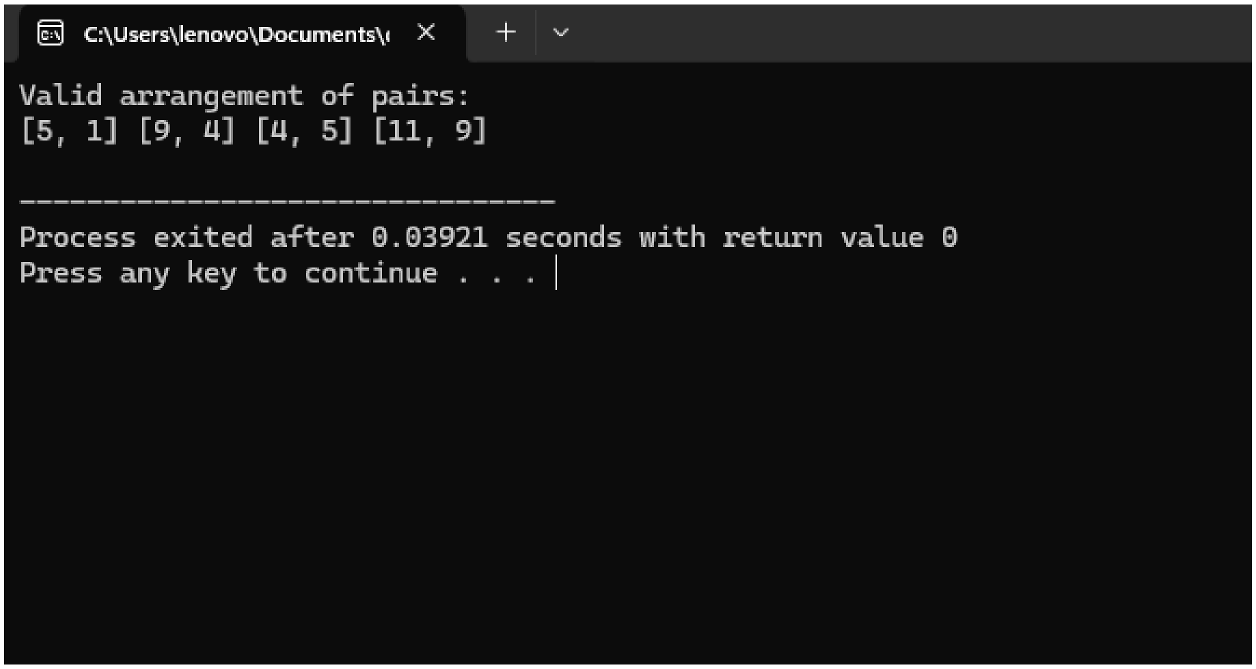
CODING:

Here's the implementation of the divide and conquer approach in C:

```
#include <stdio.h>

#include <stdlib.h>

typedef struct {

   int start;

   int end;

} Pair;

// Function to merge two sorted subarrays

void merge(Pair arr[], int left, int mid, int right) {

   int n1 = mid - left + 1;
   int n2 = right - mid;
   Pair* L = (Pair*)malloc(n1 * sizeof(Pair));
   Pair* R = (Pair*)malloc(n2 * sizeof(Pair));
   for (int i = 0; i < n1; i++)
       L[i] = arr[left + i];
   for (int j = 0; j < n2; j++)
       R[j] = arr[mid + 1 + j];
   int i = 0, j = 0, k = left;
   while (i < n1 && j < n2) {
       if (L[i].end <= R[j].end) {
           arr[k] = L[i];
           i++;
       } else {
           arr[k] = R[j];
           j++;
       }
       k++;
   }
   while (i < n1) {
       arr[k] = L[i];
       i++;
       k++;
   }
   while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
    free(L);
    free(R);
}
// Function to implement merge sort
void mergeSort(Pair arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}
// Function to find the valid arrangement of pairs
void findValidArrangement(Pair pairs[], int n) {
    mergeSort(pairs, 0, n - 1);
    printf("Valid arrangement of pairs:\n");
    for (int i = 0; i < n; i++) {
        printf("[%d, %d] ", pairs[i].start, pairs[i].end);
    }
    printf("\n");
```

```
}
int main() {
   Pair pairs[] = {{5, 1}, {4, 5}, {11, 9}, {9, 4}};
   int n = sizeof(pairs) / sizeof(pairs[0]);
   findValidArrangement(pairs, n);
   return 0;
}
```

RESULT SCREENSHOT:



```
Valid arrangement of pairs:
[5, 1] [9, 4] [4, 5] [11, 9]

--------------------------------
Process exited after 0.03921 seconds with return value 0
Press any key to continue . . .
```

COMPLEXITY ANALYSIS:

The time complexity of the divide and conquer approach for this problem is $O(n \log n)$ O(nlogn) due to the merge sort used to sort the pairs.

BEST CASE:

The best case occurs when the pairs are already sorted in the required order. The complexity remains $O(n \log n)$ O(nlogn) due to the merge sort.

WORST CASE:

The worst case occurs when the pairs are in a completely unsorted order. The complexity remains $O(n \log n)$ O(nlogn) due to the merge sort.

AVERAGE CASE:

In the average case, where the pairs are in random order, the complexity also remains $O(n \log n)$ O(nlogn).

CONCLUSION:

The divide and conquer approach, utilizing merge sort, provides an efficient solution to the "Valid Arrangement of Pairs" problem. The time complexity of $O(n \log n)$ O(nlogn) ensures that the solution is practical and efficient for large datasets.