What is Merge Sort?

There are many sorting algorithms possible to sort the data like bubble sort, quick sort, counting sort, etc. All these algorithms have their own techniques to sort the data, along with some advantages and disadvantages. Merge sort is one of these sorting algorithms based on the divide and conquer technique.

The principle working of merge sort is that it divides the list of data into two halves and again calls these subparts to further divide it into two halves. It keeps repeating the process until each subpart of the list contains only one element.

Later, it will join these sub-parts of one element into two elements by sorting them. Again, the subpart containing two elements will be joined with the other two elements after sorting. This process keeps repeating by calling the function recursively until we get the final sorted list of elements.

Merge sort is very popular for its efficiency to sort the data in a small amount of time. It is one of the best examples of applications for divide and conquer approach in Python. If you don't know, Divide and conquer is a famous algorithmic strategy in computer science that involves dividing a problem down into smaller subproblems until the subproblems become simple enough to be addressed directly.

Algorithm for Merge Sort

The divide and conquer approach to sorting n numbers using merge sort consists of separating the array T into two parts where the sizes are almost the same. These two parts are sorted by recursive calls and then merged with the solution of each part while preserving the order.

The algorithm considers two temporary arrays U and V, into which the original array T is divided. When the number of elements to be sorted is small, a relatively simple algorithm is used. ‘mergeSort’ separates the instance into two halves sized sub instances, solves them recursively, and then combines the two sorted half arrays to obtain the solution to the original instance.

Time and Space Complexity

The running time complexity for best-case, worst-case, and average-case scenarios is O(n*log n) where n is the number of elements to be sorted. The space complexity of the algorithm is O(n) where we need an array of size n to place the sorted element.

Advantages & Disadvantages of Merge Sort

The biggest advantage of Merge sort is that it can sort large data sets easily because it has faster time complexity. It is also a stable sorting algorithm which means it maintains the relative order of equal elements. Also, merge sort works when sorting linked lists because of its divide-and-conquer approach.

But there are some limitations as well. First, merge sort requires an array of the same size as the original list to store the final sorted array which consumes the large memory space. It is also slower when it comes to sorting smaller data sets.

Overall, it is an amazing sorting algorithm that is used in different domains of the industry. It is a part of many programming languages, libraries, and frameworks for sorting because of its efficiency. A sector which it is a big part of is Database Systems for query processing, result merging, and indexing. It is also utilized in Google PageRank results, E-commerce algorithm, and External sorting.

```
def mergeSort(arr):
    if len(arr) > 1:
        a = len(arr)//2
        l = arr[:a]
        r = arr[a:]
        # Sort the two halves
        mergeSort(l)
        mergeSort(r)
        b = c = d = 0
        while b < len(l) and c < len(r):
            if l[b] < r[c]:
                arr[d] = l[b]
                b += 1
            else:
                arr[d] = r[c]
                c += 1
            d += 1
        while b < len(l):
            arr[d] = l[b]
            b += 1
            d += 1
        while c < len(r):
            arr[d] = r[c]
            c += 1
```

```
            d += 1

def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

# Driver program
if __name__ == '__main__':
    arr = [0,1,3,5,7,9,2,4,6,8]
    mergeSort(arr)
    print("Sorted array is: ")
    printList(arr)
```