Name: Suraj Balaso Desai

Merge Sort Algorithm

Introduction

Merge sort is a popular sorting algorithm that uses the divide-and-conquer strategy to sort a list of items. It is an efficient algorithm that has a time complexity of O(n log n), which makes it a suitable choice for sorting large datasets.Merge Sort is a sorting technique that follows a divide and conquer approach. It involves breaking down a problem into smaller, related sub-problems recursively until they become simple enough to be solved directly. The sub-problems are solved individually, and their solutions are merged to form the solution to the original problem. In the case of Merge Sort, the algorithm first divides the array into two equal halves and then sorts and combines them in a manner that results in a fully sorted array.

Example of Merge sort being used in real-world scenarios can be observed in the process of consolidating sorted lists. For instance, let's take the example of a music streaming service that manages multiple playlists, each of which is sorted by artist name. If a user wishes to access all their favorite songs from a particular artist, the service needs to consolidate the relevant playlists into a single sorted list.

Merge sort comes in handy here, as it can efficiently merge these sorted lists into a single sorted list with a time complexity of O(n log n). The algorithm works by comparing the first element of each sorted list and adding the smallest element to a new sorted list. This process is repeated until all the elements have been merged into the new list. Finally, the resulting list is sorted and can be presented to the user as a single, sorted playlist of their favorite songs by the selected artist.

The Merge sort algorithm has a significant advantage is that it ensures a worst-case time complexity of O(n log n). This feature makes it an ideal sorting algorithm for large datasets, particularly when compared to others like bubble sort or insertion sort, which have higher time complexities. Moreover, Merge sort is considered a stable sorting algorithm because it preserves the relative position of equal elements in the sorted list.

Algorithm

The Merge sort algorithm consists of two primary steps:

Step 1: the unsorted list is divided into two sub-lists of equal size, recursively, until each sub-list contains only one element. This is accomplished by a technique called "recursive splitting," in which the input list is repeatedly divided in half until only individual elements remain.

Step 2: The sub-lists are combined together in a sorted manner to create a new sorted list. This is achieved by comparing the smallest element from each sub-list and transferring them to the new sorted list. This procedure is repeated until all the elements are merged together, resulting in a single sorted list.

Example

Let's assume that we have an array that includes the following elements: 1, 6, 3, 2, 7, 5, 8, 4. Our task is to sort this array using the merge sort algorithm.

Since the array contains a total of 8 elements, we can determine that the midpoint of the array is 4. Consequently, we will split the array into two arrays, each having a size of 4, as illustrated below:

Following that, we will continue the process of dividing each of the two previously created arrays recursively into halves. Since the midpoint of an array of size 4 is 2, each of the previous arrays will be split into two new arrays, with each having a size of 2. As a result, we will end up with a total of 4 arrays, each having a size of 2.

We will keep dividing the arrays into halves until we arrive at an array size of 1. This means that the recursive splitting process will continue until each sub-array contains only a single element, as demonstrated below:

Once we have arrived at sub-arrays containing only a single element, we will start the combining step. During this step, we will compare each element with its adjacent element and arrange them in a sorted order. In other words, we will merge the smaller sub-arrays back into larger sub-arrays, sorting them as we go.

During the next iteration, we will compare two sub-arrays and merge them into a single sorted sub-array, as demonstrated below:

Lastly, we will compare the elements of the two sub-arrays, each having a size of 4, and merge them into a single, sorted array, resulting in the final sorted array as demonstrated below:

Code

```
def merge(sub1, sub2):
    a = 0
    b = 0
    result = []
    while a < len(sub1) and b < len(sub2):
        if sub1[a] <= sub2[b]:
            result.append(sub1[a])
            a+=1
        else:
            result.append(sub2[b])
            b+=1

    if a < len(sub1):
        for i in range(a, len(sub1)):
            result.append(sub1[i])
    else:
        for i in range(b, len(sub2)):
            result.append(sub2[i])

    return result


def merge_sort(arr):
    if len(arr) < 2:
        return arr
    else:
        mid = len(arr)// 2
        sub1 = merge_sort(arr[0:mid])
        sub2 = merge_sort(arr[mid:])

        return merge(sub1, sub2)


if __name__ == "__main__":
    print(merge_sort([6, 5, 12, 10, 9, 1]))
```

Output

[1, 5, 6, 9, 10, 12]

Time Complexity

Merge sort has a time complexity of O(n log n), which implies that as the number of elements (n) increases, the time taken to sort the array will increase at a rate proportional to n multiplied by the logarithm of n. Merge sort is highly efficient and is often preferred over other sorting algorithms, such as Bubble sort or Insertion sort, which have higher time complexities.

However, it is worth noting that Merge sort utilizes additional memory space for the temporary arrays used during the merging process, which can have an impact on its space complexity.

Advantages

1. Efficiency: To sort large datasets efficiently, Merge sort utilizes the divide-and-conquer approach to divide the dataset into smaller sub-lists that can be sorted separately, resulting in a worst-case time complexity of O(n log n). This makes it an efficient algorithm for sorting large datasets.
2. Stability: In Merge sort, equal elements in the input list retain their relative position in the sorted list, making it a stable sorting algorithm. This feature is crucial in situations where the initial order of the data must be preserved.
3. Predictability: To put it simply, Merge sort's performance is highly predictable, even for different types of input data. This is due to its worst-case time complexity of O(n log n), which ensures that the algorithm will not have sudden spikes in execution time.
4. Ease of implementation: Merge sort is considered to be a straightforward algorithm to implement, which means that programmers of any skill level can easily understand and use it. Moreover, there are many libraries and existing implementations of Merge sort available in different programming languages, making it even more accessible to developers.
5. Parallelization: Merge sort can be easily divided into multiple independent sub-tasks that can be executed concurrently on different processors or threads, making it a highly parallelizable algorithm. This property allows Merge sort to take advantage of multi-core or distributed systems and can lead to significant improvements in performance.

Disadvantages

1. Space complexity: The merge sort algorithm needs extra memory space to use temporary arrays during the merging process. This could become problematic when working with large datasets because it can result in a significant increase in memory consumption.
2. Not in-place: Merge sort cannot sort the data in place, which means that it needs additional memory space to store the temporary arrays used during the merging step. This can be a disadvantage when working with limited memory or when the memory is a bottleneck in the system, as it can cause the algorithm to consume a lot of memory.

Conclusion

To sum up, Merge sort is a widely used and effective sorting algorithm that employs the divide-and-conquer approach to sort a collection of items. It achieves a worst-case time complexity of O(n log n) by splitting the input list into smaller sub-lists, sorting them recursively, and merging them back together. Additionally, Merge sort preserves the relative order of equal elements in the sorted list.

References

- TechVidvan
- Medium