Merge Sort

Like Quick Sort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge () function is used for merging two halves. The merge (arr, l, m, r) is key process that assumes that arr[l..m] and arr [m+1...r] are sorted and merges the two sorted sub-arrays into one.

**Time Complexity:** Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.  
T(n) = 2T(n/2) + O(n)

The above recurrence can be solved either using Recurrence Tree method or Master method. The solution of the recurrence is:O(nlogn)

Time complexity of Merge Sort is O(nlogn) in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and take linear time to merge two halves.

**Auxiliary Space:** O(n)

**Algorithmic Paradigm:** Divide and Conquer

Quick Sort

Like Merge Sort, Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quick Sort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quick Sort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

**Analysis of Quick Sort**  
Time taken by Quick Sort in general can be written as following:

T(n)=T(k)+T(n-k-1) + O(n)

The first two terms are for two recursive calls; the last term is for the partition process. k is the number of elements which are smaller than pivot.  
The time taken by Quick Sort depends upon the input array and partition strategy. Following are three cases.

***Best Case:*** The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

T(n) = 2T(n/2) + O(n)

The solution of above recurrence is O(nlogn).

***Worst Case:*** The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider about partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

T(n) = T (0) + T(n-1) + O(n)

Which is equivalent to:

T(n) = T(n-1) + O(n)

The solution of above recurrence is O(n2).

***Average Case:*** To do average case analysis, we need to [consider all possible permutation of array and calculate time taken by every permutation which doesn’t look easy](https://www.geeksforgeeks.org/analysis-of-algorithms-set-2-asymptotic-analysis/). We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

T(n) = T(n/9)+ T(9n/10) + O(n)

Solution of above recurrence is also O(nLogn)

Although the worst case time complexity of Quick Sort is O(n2) which is more than many other sorting algorithms like Merge Sort and Heap Sort, Quick Sort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. Quick Sort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

Machine configuration: Processor: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80 GHz

64 bit operating system

Code Run by: Code Blocks

Compiler: GNU GCC Compiler

Table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | N= | 10 | 100 | 1000 | 10000 | 100000 | 1000000 |
| Case | Sort |  |  |  |  |  |  |
| Best | Merge | 0 | 0 | 0 | 0.0013 | 0.0116 |  |
|  | Quick | 0 | 0 | 0.001 | 0.154 | 5.5756 |  |
| Worst | Merge | 0 | 0 | 0.0003 | 0.0066 | 0.0103 |  |
|  | Quick | 0 | 0 | 0.002 | 0.1513 | 5.6363 |  |
| Average | Merge | 0 | 0 | 0 | 0.0016 | 0.0183 |  |
|  | Quick | 0 | 0 | 0.0023 | 0.1536 | 5.6407 |  |