



Dash

All

Articles

Videos

Problems

Quiz

<< Prev

Next >>

QuickSort analysis

Is QuickSort stable?

The default implementation is not stable. However any sorting algorithm can be made stable by considering indexes as comparison parameter.

Is QuickSort In-place?

As per the broad definition of in-place algorithm it qualifies as an in-place sorting algorithm as it uses extra space only for storing recursive function calls but not for manipulating the input.

What is 3-Way QuickSort?

In simple QuickSort algorithm, we select an element as pivot, partition the array around pivot and recur for subarrays on left and right of pivot.

Consider an array which has many redundant elements. For example, {1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 2, 2, 4, 1, 4, 4, 4}. If 4 is picked as pivot in Simple QuickSort, we fix only one 4 and recursively process remaining occurrences. In 3 Way QuickSort, an array $arr[l..r]$ is divided in 3 parts:

- $arr[l..i]$ elements less than pivot.
- $arr[i+1..j-1]$ elements equal to pivot.
- $arr[j..r]$ elements greater than pivot.

See [this](#) for implementation.

How to implement QuickSort for Linked Lists?

[QuickSort on Singly Linked List](#)

[QuickSort on Doubly Linked List](#)

Can we implement QuickSort Iteratively?

Yes, please refer [Iterative Quick Sort](#).

Why Quick Sort is preferred over MergeSort for sorting Arrays ?

Quick Sort in its general form is an in-place sort (i.e. it doesn't require any extra storage) whereas merge sort requires $O(N)$ extra storage, N denoting the array size which may be quite expensive. Allocating and de-allocating the extra space used for merge sort increases the running time of the algorithm. Comparing average complexity we find that both type of sorts have $O(N \log N)$ average complexity but the constants differ. For arrays, merge sort loses due to the use of extra $O(N)$ storage space.

Most practical implementations of Quick Sort use randomized version. The randomized version has expected time complexity of $O(n \log n)$. The worst case is possible in randomized version also, but worst case doesn't occur for a particular pattern (like sorted array) and randomized Quick Sort works well in practice.

Quick Sort is also a cache friendly sorting algorithm as it has good locality of reference when used for arrays.

Quick Sort is also tail recursive, therefore tail call optimizations is done.

Why MergeSort is preferred over QuickSort for Linked Lists ?

In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of $A[0]$ be x then to access $A[i]$, we can directly access the memory at $(x + i * 4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i 'th index, we have to travel each and every node from the head to i 'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

How to optimize QuickSort so that it takes $O(\log n)$ extra space in worst case?

Please see [QuickSort Tail Call Optimization \(Reducing worst case space to \$\log n\$ \)](#)



Dash



All



Articles



Videos



Problems



Quiz



- [Quiz on QuickSort](#)
- [Recent Articles on QuickSort](#)
- [Coding practice for sorting.](#)

Advantages of Quick Sort:

- It is a divide-and-conquer algorithm that makes it easier to solve problems.
- It is efficient on large data sets.
- It is a stable sort, meaning that if two elements have the same key, their relative order will be preserved in the sorted output.
- It has a low overhead, as it only requires a small amount of memory to function.

Disadvantages of Quick Sort:

- It has a worst-case time complexity of $O(n^2)$, which occurs when the pivot is chosen poorly.
- [It is not a good choice for small data sets.](#)
- [It can be sensitive to the choice of pivot.](#)
- [It is not cache-efficient.](#)

Time Complexity

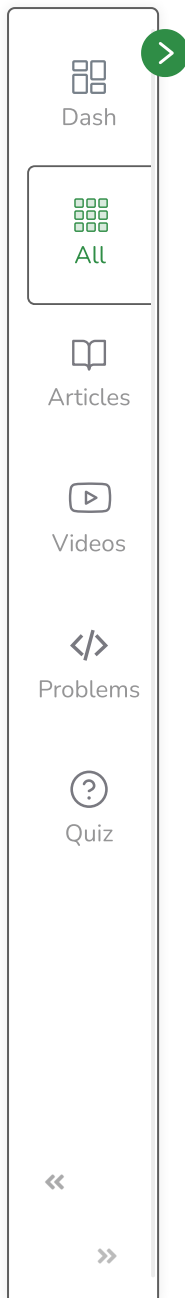
Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \theta(n)$$

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 QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If



we consider above 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.

$$T(n) = T(0) + T(n-1) + \theta(n)$$

which is equivalent to

$$T(n) = T(n-1) + \theta(n)$$

The solution of above recurrence is $\Theta(n^2)$.

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

$$T(n) = 2T(n/2) + \theta(n)$$

The solution of above recurrence is $\Theta(n \log n)$. It can be solved using case 2 of Master Theorem.

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.

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.

$$T(n) = T(n/9) + T(9n/10) + \theta(n)$$

Solution of above recurrence is also $O(n \log n)$

Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like



Dash



All



Articles



Videos



Problems



Quiz



Merge Sort and **Heap Sort**, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort 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.

Summary:

- Quick sort is a fast and efficient sorting algorithm with an average time complexity of $O(n \log n)$.
- It is a divide-and-conquer algorithm that breaks down the original problem into smaller subproblems that are easier to solve.
- It can be easily implemented in both iterative and recursive forms and it is efficient on large data sets, and can be used to sort data in-place.
- However, it also has some drawbacks such as worst case time complexity of $O(n^2)$ which occurs when the pivot is chosen poorly.
- It is not a good choice for small data sets, it is not cache-efficient, and is sensitive to the choice of pivot.



Marked as Read

 Report An Issue

If you are facing any issue on this page. Please let us know.

