Sorting Algorithms Survey

HAO WANG

ACM Reference Format:
Hao Wang. 2022. Sorting Algorithms Survey. 1, 1 (January 2022), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sorting algorithms are widely used while programmers need to manage data. Each algorithm has particular strengths and weaknesses and choose a proper algorithm could increase the efficiency of program.

In this survey, common used sorting algorithms will be introduced and different algorithms will be compared in terms of time complexity, operation complexity and usage of space.

2 SORTING ALGORITHMS SUMMARY

2.1 In-place Sorting Algorithms

Sorting methods that do not require additional space are called "in place". This property is particularly useful when dealing with large data sets, and some algorithms that seem to require additional space can be made in place with a bit of work.

2.1.1 Bubble Sort. Bubble sort is the simplest sorting algorithm. The bubble sort works by iterating down an array to be sorted from the first element to the last, comparing each pair of elements and switching their positions if the condition is satisfied. This process is repeated as many times as necessary, until the array is sorted. Since the worst case scenario is that the array is in reverse order, and that the first element in sorted array is the last element in the starting array, the most exchanges that will be necessary is equal to the length of the array. Notice that this will always loop n times from 0 to n, so the order of this algorithm is O(n*n). This is both the best and worst case scenario.

2.1.2 Modified Bubble Sort. modified bubble sort is a better version of bubble sort, includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made, then it should be clear that the array is already in order because no two elements need to be switched. In that case, the sort should end. The new best case order for this algorithm is O(n), as if the array is already sorted, then no exchanges are made. It only requires a few changes to the original bubble sort.

Author's address: Hao Wang, seraveea@ust.hk.

XXXX-XXXX/2022/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.1.3 Selection Sort. Selection sort is the most conceptually simple of all the sorting algorithms. It works by selecting the smallest (or largest, if you want to sort from big to small) element of the array and placing it at the head of the array. Then the process is repeated for the remainder of the array; the next largest element is selected and put into the next slot, and so on down the line. Suppose there are n numbers to sort, the first loop goes from 0 to n, and the second loop goes from x to n, so it goes from 0 to n, then from 1 to n, then from 2 to n and so on. The multiplication works out so that the efficiency is n*(n/2), though the order is still O(n*n).

2.1.4 Insertion Sort. Insertion sort inserts each element of the array into its proper position, leaving progressively larger stretches of the array sorted. What this means in practice is that the sort iterates down an array, and the part of the array already covered is in order; then, the current element of the array is inserted into the proper position at the head of the array, and the rest of the elements are moved down, using the space just vacated by the element inserted as the final space. In the worst case, the time complexity is O(n*n) and in the best case, there is no insert operation needed and the time complexity is o(n).

2.2 Out-place Sorting Algorithms

Common in-place sorting algorithms have been introduced. There are some faster sorting algorithms that can sort in time proportional to O(n*log(n)) in the average and best case time which is a significant speedup when the number of items to sort grows larger. Those faster sorts require additional storage are called "out-place".

2.2.1 Heap Sort. One of the simplest out-place algorithms is heap sort, which is based on the heap data structure. The first important thing to remember about heaps is that the top element of the heap is always "next" in order (either the next highest or next lowest, in the case of numbers). In a heap structure, adding a single element to and removing a single element from a heap both take O(log(n)) time. Consequently, the algorithmic efficiency of a heap sort is O(n*log(n)).

2.2.2 Merge Sort. Merge sort is the second guaranteed O(nlog(n)) sort. Like heap sort, merge sort requires additional memory proportional to the size of the input for scratch space, but, unlike heap sort, merge sort is stable, meaning that "equal" elements are ordered the same once sorting is complete. Merge sort works using the principle that if you have two sorted lists, you can merge them together to form another sorted list. This is a divide-and-conquer algorithm, sorting a list will be broken to sorting two smaller list until you can't break the list anymore.

Merge sort guarantees O(nlog(n)) complexity because it always splits the work in half. While computing the time complexity, two factors are involved: the number of recursive calls, and the time taken to merge each list together. The operation of break one list to two small list will result in a "binary tree" structure, and the depth of the tree is log(n)+1, which means we need to call the recursive function log(n)+1 times. Although every operation in each recursive function is different, it could be considered as O(n) complexity, and the time complexity of merge sort is O(nlog(n)).

2.2.3 Quick Sort. Quick sort, like merge sort, is also a divide-and-conquer recursive algorithm. The basic divide-and-conquer process for sorting a subarray S[p..r] is summarized in the following three easy steps:

Divide: Partition S[p..r] into two subarrays S[p..q-1] and S[q+1..r] such that each element of S[p..q-1] is less than or equal to S[q], which is, in turn, less than or equal to each element of S[q+1..r]. Compute the index q as part of this partitioning procedure

Conquer: Sort the two subarrays S[p...q-1] and S[q+1..r] by recursive calls to quicksort.

Combine: Since the subarrays are sorted in place, no work is needed to combing them: the entire array S is now sorted.

Algorithms	Time Complexity			Space Requirement
	Average	Best	Worst	
Bubble sort	O(n^2)	O(n^2)	O(n^2)	In place
Modified Bubble Sort	O(n^2)	O(n)	O(n^2)	**In place**
Selection Sort	O(n^2)	O(n^2)	O(n^2)	In place
Insertion Sort	O(n^2)	O(n)	O(n^2)	In place
Heap Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Out place
Merge Sort	O(n*log(n))	O(n*log(n))	O(n*log(n))	Out place
Quick Sort	O(n*log(n))	O(n*log(n))	O(n^2)	Out place

Table 1. Sorting Algorithms Summary

Same as merge sort, the average time complexity is also O(nlog(n)). But the performance of quick sort is influenced by the initial order of the list. At the worst case, the time complexity could be O(n*n)

3 SORTING ALGORITHMS COMPARED

Here we summarized sorting algorithms above in Table 1. When programmers need to pick up one sorting algorithm, we could consider from two perspective: time complexity and space requirement. If we need to sort a very large list and the program have a strict requirement of space management, in-place sorting algorithms like bubble sort, insertion sort are what we need. However, if we need high-speed algorithms and have free memory, maybe heap sort, merge sort and quick sort are better. And in most cases in reality, we need to find a balance between running time and the usage of memories.

REFERENCES