Volume 6, Issue 1, January 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review and Analysis of Sorting Techniques in Various Cases

Ravendra Kumar
Aligarh College of Engineering & Technology,
Aligarh, Uttar Pradesh, India

Abstract: ***Sorting is a key data structure operation, which makes easy arranging, searching, and finding the information. I have discussed various sorting algorithms with their comparison to each other. Since every sorting algorithm have some its advantage and some disadvantages. Sorting problem gain more popularity, as efficient sorting is more important to optimize other algorithm. A number of sorting algorithms has been proposed with different constraints e.g. number of iterations (inner loop, outer loop), time complexity, and space complexity. This paper presents a comparison of different sorting algorithms along with Best Case, Average Case, and worst case. There are lot of design strategies of algorithms such as including Divide and Conquer, Dynamic Programming, The Greedy-Method, Backtracking and Branch-and-Bound. The aim of this comparison between these algorithms is to know the running time of each one according to the number of input elements.***

Keywords: Sorting, Quick Sort, Merge Sort, Selection Sort, Insertion Sort, Bubble Sort, Complexity, Divide-and-Conquer, Dynamic Programming, Greedy-Method and Backtracking.

I. INTRODUCTION

An algorithm is a well-defined procedure or set of instructions, that takes some input in the form of some values, processes them and gives some values as output. There are five characteristics as input, output, definiteness, finiteness and effectiveness of algorithm.
Sorting can formally defined as:
Input: A sequence of data set having n numbers of random order data= (1, 2, 3 … ,− 1)
Output: A permutation of the input sequence′= (′1, ′2, ′3… , ′ −1 ,′), such that ′1 ≤ ′2 ≤ ′ ≤, … , ≤ − 1 ≤)

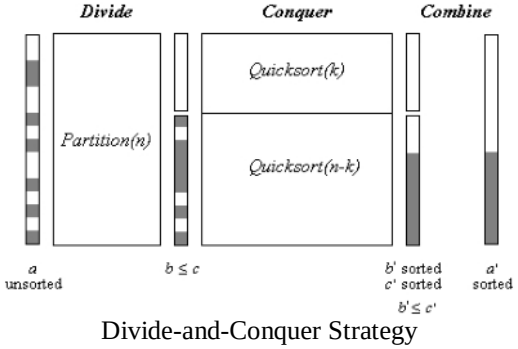
The use of algorithms did not begin with the introduction of computers, but people using them while they are solving problems. We can describe algorithms as a finite sequence of rules which describes and analyses the algorithms. Each sorting algorithm uses its own technique in execution. It is also possible that a single problem can be solved by using more than one algorithms. Here I will compare between the sorting algorithms based on best-case B (n), average-case A(n), and worst-case efficiency W(n) that refer to the performance of the number n of elements. The Big O, Theta (Θ) and big omega (Ω) notations are used to give approximate result not exact due to their inherent property.

II. MAJOR DESIGN STRATEGIES FOR THE ALGORITHMS

The main aspect of design algorithms is to produce a solution which gives sufficient run time. Below, there is a brief overview of each strategy.

2.1 Divide-and-Conquer

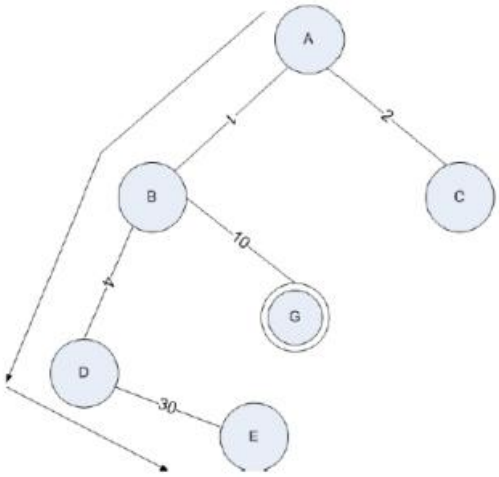
First, the sequence to be sorted A is partitioned into two parts, such that all elements of the first part B are less than or equal to all elements of the second part C (divide). Then the two parts are sorted separately by recursive application of the same procedure (conquer). Recombination of the two parts yields the sorted sequence (combine). [2] Figure 1illustrates this approach.



Divide-and-Conquer Strategy

2.2 The Greedy Method

This algorithm solves the problem by choosing an optimal choice for the solution that leads to the goal but that will cost much time. [2] It generally focus on the solution which is local, not global in nature. Figure 2 shows a case of greedy method to reach the goal G



Greedy Method

2.3 Dynamic Programming

Like divide-and-conquer, dynamic programming technique also builds a solution to a problem from solution to sub- problems, and all sub-problems must be solved before proceeding to next sub-problems [2].

2.4 Backtracking and Branch-and-Bound

Backtracking is using depth-first search in the space tree while Branch-and-Bound is using the breadth-first search. Both depend on how to take a decision to reach the solution. For example traveling salesman problem(TSP).

III. RESPONSIBLE FACTORS FOR CLASSIFICATION OF SORTING

1. Computational complexity (worst, average and best behavior) of element comparisons in terms of the size of the list. For typical sorting algorithms good behavior is O(n logn) and bad behavior is $O(n^2)$..
2. Memory usage (and use of other computer resources). In particular, some sorting algorithms are "in place". This means that they need only O(1) memory beyond the items being sorted and they don't need to create auxiliary locations for data to be temporarily stored, as in other sorting algorithms.
3. Some algorithms are either recursive or non-recursive while others may be both (e.g., merge sort).
   Stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).

Sorting algorithms are an important part of managing data. Most sorting algorithms work by comparing the data being sorted. In some cases, it may be desirable to sort a large volume of data based on only a portion of that data. The piece of data actually used to determine the sorted order is called the key. Sorting algorithms are usually judged by their efficiency[1]. Sorting is the process of arranging data in specific order which benefits searching and locating the information in an easy and efficiency way. Sorting algorithms are developed to arrange data in various ways; for instance, an array of integers may be sorted from lower to highest or from highest to lower or array of string elements may sorted in alphabetical order.

Bubble Sort

Bubble sort is a simple sorting algorithm. The algorithm starts at the beginning of the data set. It compares the first two elements, and if the first is greater than the second, then it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass.[4]

Bubble sort starts comparing the last item M in list with the (M-1) and swaps them if needed. The algorithm repeats this technique until the required order as shown in the Pseudo code:

```
For i = 1:n,
        swapped = false for j = n:i+1,
                if a[j] < a[j-1],
                        swap a[j,j-1] swapped = true
        break if not swapped end
```

In case of sorted data, bubble sort takes O(n) (best-case) time, since it passes over the items one time. But it takes O(n2) time in the average and worst cases because it requires at least 2 passes through the data.1) Advantage: Simplicity and ease-of-implementation.2) Disadvantage: Code inefficient.[3]

$$O(n^2) = n + (n + 1) + (n + 2) + :: + 1$$

Insertion sort

Insertion sort is a simple sorting algorithm that is relatively efficient for small lists and mostly sorted lists, and often is used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list. In arrays, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by one.[6..12]

Quick Sort

Quick Sort is a divide and conquer algorithm which relies on a partition operation: to partition an array an element called a pivot is selected. All elements smaller than the pivots are moved before it and all greater elements are moved after it. This can be done efficiently in linear time and in-place. The lesser and greater sub-lists are then recursively sorted. Efficient implementations of quick sort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice[6..12].

Merge Sort

Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. It starts by comparing every two elements (i.e., 1 with 2, then 3 with 4...) and swapping them if the first should come after the second. It then merges each of the resulting lists of two into lists of four, then merges those lists of four, and so on; until at last two lists are merged into the final sorted list.[6..12]

Counting sort

Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers; that is, it is an integer sorting algorithm.[6..12].It comes into the stable category.

Selection Sort

Scan all items and find the smallest. Swap it into position as the first item. Repeat the selection sort[6..12] on the remaining N-1 items. *Selection sort* is an in place comparison sort. It has $O(n^2)$ complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

Gnome sorting

Gnome sorting algorithm is similar to bubble sort [4]. The movement of the elements depends on series of swapping and no need for nested loops to reach the final order of the list. The algorithm always finds the first place where to the adjacent elements are in the wrong order and swaps them It is a sorting algorithm which is similar to insertion sort, except that moving an element to its proper place is accomplished by a series of swaps, as in bubble sort. It is conceptually simple, requiring no nested loops. The average, or expected, running time is $O(n^2)$, but tends towards $O(n)$ if the list is initially almost sorted[6..12]

IV. IMPLEMENTED SORTING ALGORITHMS

```
procedure gnomeSort(a[])
   pos := 1
   while pos < length(a)
      if (a[pos] >= a[pos-1])
         pos := pos + 1
      else
         swap a[pos] and a[pos-1]
         if (pos > 1)
            pos := pos - 1
         end if
      end if
   end while
end procedure
```

Current array	**Action to take**
[6, **4**, 3, 5]	a[pos] < a[pos-1], swap:
[4, **6**, 3, 5]	a[pos] >= a[pos-1], increment pos:
[4, 6, **3**, 5]	a[pos] < a[pos-1], swap and pos > 1, decrement pos:
[4, **3**, 6, 5]	a[pos] < a[pos-1], swap and pos <= 1, increment pos:
[3, 4, **6**, 5]	a[pos] >= a[pos-1], increment pos:
[3, 4, 6, **5**]	a[pos] < a[pos-1], swap and pos > 1, decrement pos:

[3, 4, **5**, 6] a[pos] >= a[pos-1], increment pos:

[3, 4, 5, **6**] a[pos] >= a[pos-1], increment pos:

[3, 4, 5, 6] pos == length(a), finished.

I concluded that Big O notation $O(n^2)$ for the three cases (Best, Average and worst) is the same since selection sort algorithm continue sorting even if the remaining elements in the array is already sorted, in figure 3 after loop3 no need to do more sorting [3]. Selection algorithm is easy for implementation and useful for sorting small data as it does not occupy a large space of memory, but it is inefficient for large list. [5]

	Time		
Sort	Avg.	Best	Worst
Bubble sort	O(n^2)	O(n^2)	O(n^2)
Selection Sort	O(n^2)	O(n^2)	O(n^2)
Insertion Sort	O(n^2)	O(n)	O(n^2)
Merge Sort	O(n*logn)	O(n*logn)	O(n*logn)
Quick Sort	O(n*logn)	O(n*logn)	O(n^2)
Counting sort	O(n)	O(n)	O(n)
Gnome Sort	n2	n	n2

Summary of the best-case, average-case and worst-case [2]

V. CONCLUSION

From above discussion, it can be said that, every sorting algorithm has some advantages and disadvantages. Here I have write a little about those who have already their elaboration and tried to elaborate others, still there are some chances further enhance their efficiency by developing more strategies. To determine the good sorting algorithm, speed is the top consideration but other factors include handling various data type, consistency of performance, complexity of code etc. Quick Sort is the most efficient algorithm. It has been shown that gnome sort algorithm is the quickest one for already sorted data but selection sort is more quick than gnome and bubble , in unsorted data. Bubble sort and gnome sort swap the elements if required. But In selection sort it continues sorting even if the elements are already sorted.

REFERENCES

[1] Pankaj Sareen. Comparison of Sorting Algorithms (On the Basis of Average Case). Department of Computer Applications. International Journal of Advanced Research in Computer Science and Software Engineering, 2013. isbn: 2277 128X

[2] Kenneth A. Berman and Jerome L. Paul. Algorithms: Sequential, Parallel, and Distributed. Computer Science and Engineering. Thomason course technology, 2002. Isbn :0-534-42057-5..pdf.

[3] Khalid Suleiman Al-Kharabsheh et al. Review on Sorting Algorithms A Comparative Study. Computer Science. International Journal of Computer Science and Security (IJCSS), 2013

[4] https://en.wikipedia.org/wiki/**Seymour_Lipschutz**

[5] Ramesh Chand Pandey, Comparison Of various sorting algorithms,Computer Science and Engineering 2008,isbn 80632019

[6] https://en.wikipedia.org/wiki/Gnome_sort

[7] https://en.wikipedia.org/wiki/selection_sort

[8] https://en.wikipedia.org/wiki/bubble_sort

[9] https://en.wikipedia.org/wiki/insertion_sort

[10] https://en.wikipedia.org/wiki/quick_sort

[11] https://en.wikipedia.org/wiki/merge_sort

[12] https://en.wikipedia.org/wiki/counting_sort

