Research Review For Sorting Algorithm Limitations

Shish ram , Vijay Kumar Sharma , Dr. Saroj Hiranwal

Abstract— In order to perform any operation computer systems uses various algorithms with help of sorting algorithms. The computational complexity, ordering and other properties are performs under sorting algorithm. This paper is discus two sorting algorithm: Quick sort and Bubble sort which are one of the best sorting algorithms exists and also analyzes their performances, limitations an advantages and gives a deep insight about them. This paper also discuss about the limitations of sorting algorithms based on spatial complexity and further suggest a solution to overcome the problem of spatial complexity in brief.

Index Terms—Quick sorts, sequential order, merge space and Bubble sort.

I. INTRODUCTION

In general, simple sorting algorithms perform two operations such as compare two elements and assign one element. These operations proceed over and over until the data is sorted [1]. Moreover, selecting a good sorting algorithm depending upon several factors such as the size of the input data, available main memory, disk size, the extent to which the list is already sorted and the distribution of values [2]. To measure the performance of different sorting algorithm we need to consider the following facts such as the number of operations performed, the execution time and the space required for the algorithm [2]. Since sorting algorithms are common in computer science, some of its context contributes to a variety of core algorithm concepts such as divide-and-conquer algorithms, data structures, randomized algorithms, etc. The majority of an algorithm in use have an algorithmic efficiency of either O(n2) or O(n log n) [7].

II. BUBBLE SORT

Bubble sort is a simple and the slowest sorting algorithm which works by comparing each element in the list with its neighboring elements and swapping them if they are in undesirable order. The algorithm continues this operation until it makes a pass right through the list without swapping any elements, which shows that the list is sorted. This process makes the algorithm works slower when the size of the input n increased. Because of this reason it considered to be the most inefficient sorting algorithm with large amount of data. The algorithm [4] for bubble sort is as follows.

A. Algorithm: Bubble Sort

1: procedure Bubble Sort. Records R1... RN are rearranged in place; after sorting is not known to be in its final position; thus we are indicating that nothing is known at this point.

2: [Initialize BOUND.] Set BOUND ← N. (BOUND is the highest index for which the record is not known to be in its final position; thus we are indicating that nothing is known at this point.)

3: [Loop on j.] Set t ← 0. Perform step 4 for j = 1, 2..., BOUND-1, and then go to step 5. (If BOUND =1, this means go directly to step 5.)
4: [Compare/exchange Rj: Rj + 1.] If Kj > Kj + 1, interchange Rj ↔ Rj + 1 and set t ← j.

5: [Any exchanges?] If t = 0, terminate the algorithm. Otherwise set BOUND ← t and return to step 3.

6: end procedure

B. Performance Analysis

Bubble sort is considered to be the most inefficient algorithm for the reason that it has a worst case and average case complexity of O (n^2), where n is the number of elements to be sorted. Likewise some other simple sorting methods such as insertion sort and selection sort has the same worst case complexity of O(n^2) however the efficiency of bubble sort is comparatively lesser than these algorithms. Hence these computational complexity shows that bubble sort should not be considered over a large amount of data items.

Moreover, there is a better way of implementing the bubble sort described in the modified bubble sort. It suggests a few changes to the standard bubble sort which 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 certainly show that the array is already in order. Which gives the best case complexity of O(n) if the array is already sorted.

Besides another variant of bubble sort is Bidirectional Bubble Sort or Cocktail Sort [4] which sorts the list in both directions each pass through the list. This process slightly reduced the number of comparisons. Moreover Butcher [4] proposed a method whose running time is better than both straight bubble sort and Bidirectional Bubble sort.

<table>
<thead>
<tr>
<th>No. of Elements</th>
<th>Execution Time (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>50</td>
<td>72</td>
</tr>
<tr>
<td>100</td>
<td>167</td>
</tr>
<tr>
<td>200</td>
<td>368</td>
</tr>
</tbody>
</table>

Table 1: Execution time of Bubble Sort Algorithm

C. Advantages and Disadvantages

Even though Bubble sort considered to be the most inefficient algorithm, it has some advantage over other algorithms such as simplicity and ease of implementation and the ability to identify the list is already sorted if it is efficiently implemented. Moreover bubble sort uses O (1) auxiliary space for sorting. On the other hand the drawbacks of bubble sort include code inefficient, inappropriate for large volumes of data elements and repetitive problems as well.

III. QUICK SORT

Quick sort [5] is the fastest general purpose internal sorting algorithm on the average among other sophisticated algorithms. Unlike merge sort it does not require any additional memory space for sorting an array. For the reason that it is widely used in most real time application with large data sets. Quick sort uses divide and conquer approach for solving problems. Quick sort is quite similar to merge sort. It works by selecting elements from unsorted array named as a pivot and split the array into two parts called sub arrays and

Figure 2: Steps of Quick Sort

reconstruct the former part with the elements smaller than the pivot and the latter with elements larger than the pivot. This operation is called as partitioning [6]. The algorithm repeats this operation recursively for both the sub arrays. In general, the leftmost or the rightmost element is selected as a pivot.

Selecting the left most and right most elements as pivot was practiced in the early version of quick sort and this causes the worst case behavior, if the array is already sorted. Later it was solved by various practices such as selecting a random pivot and taking the median of first, middle and last elements. Quick sort is an in-place algorithm and it works very well, even in a virtual memory environment. The algorithm for quick sort is as follows.

A. Algorithm: Quick Sort

1: procedure QUICK SORT (list, start, end)
2: if start<end then
3: index = PARTITION (list, start, end)
4: QUICKSORT (list, start, index − 1)
5: QUICKSORT (list, index − 1, end)
6: end if
7: end procedure

B. Performance Analysis

Quick sort is the fastest sort on the average running time complexity of O (n log n) when compared to other sophisticated algorithms. Usually, selecting the leftmost or rightmost element as a pivot causes the worst case running time of O (n^2) when the array is already sorted. Likewise, it is not efficient if all the input elements are equal, the algorithm will take quadratic time O (n^2) to sort an array of equal elements. However, these worst case scenarios are infrequent. There are more advanced version of quick sort are evolved with a solution to selecting pivot. Qsort [6] is one of
the variants of quick sort, which is faster and more robust than the standard method. Quick sort is the better option if speed is greatly significant and also for large data sets. It is quite complicated to sort an array of smaller size, so we can implement a quick sort often with insertion sort to sort smaller arrays.

**Table 2:** Execution time of Quick Sort Algorithm

<table>
<thead>
<tr>
<th>Quick Sort</th>
<th>Bubble Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Elements</td>
<td>Execution Time (in ms)</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>50</td>
<td>42</td>
</tr>
<tr>
<td>100</td>
<td>78</td>
</tr>
<tr>
<td>200</td>
<td>142</td>
</tr>
</tbody>
</table>

**C. Advantages and Disadvantages**

One of the great advantages of quick sort is that it is fast and efficient for large data sets. However it is not efficient if the array elements are already sorted and also each element in the array are equal. This gives the worst case time complexity of $O(n^2)$. Moreover it’s not efficient to sort real objects. Quick sort might be space expensive for large data sets due to the fact that it uses $O(\log n)$ auxiliary space for recursive function calls. Moreover quick sort carry out sequential traverse through array elements which results in good locality of reference and cache behavior for arrays [5].

### IV. PERFORMANCE COMPARISON

In this section the comparison of the performance is done for Quick and bubble sort sorting algorithms.

<table>
<thead>
<tr>
<th>Time</th>
<th>Quick sort</th>
<th>Bubble sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>$O(n\log n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Best</td>
<td>$O(n\log n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Worst</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Space</td>
<td>Constant</td>
<td>Constant</td>
</tr>
<tr>
<td>Stability</td>
<td>Stable</td>
<td>Stable</td>
</tr>
</tbody>
</table>

**Remarks**

- Randomly picking a pivot value (or shuffling the array prior to sorting) can help avoid worst case scenarios such as a perfectly sorted array.
- Always use a modified bubble sort

**Table 3:** Time complexity analysis for Quick sort and Bubble sort Algorithms

From the results calculated from the comparison of execution time of quick sort and bubble sort the figure is plotted which shows the comparison in execution time for Quick & Bubble Sort Algorithm.

**Figure 3:** Comparison in Execution time for Quick & Bubble Sort Algorithm

From figure 3 it is clear that execution time of quick sort is always better than the bubble sort algorithm. As the number of elements increases the time taken by the quick sort is also improved in comparison with bubble sort algorithm.

### V. CONCLUSION

Results prove that Quick sort is more efficient than Bubble sort in all scenarios. As the number of elements increases the performance of quick sort is much better than Bubble sort.

But the time taken by both the algorithm increases significantly. In calculation if we increase the number of elements from few thousands to few millions then execution time taken by these algorithms cannot be acceptable in practical scenarios. Result shows that these algorithms cannot work with large datasets efficiently.

Another Problem is that these algorithm uses memory for computation, so the available memory needs to be larger than the datasets need to be sorted. In practical scenario sometimes datasets can be larger as of multiple MBs then these algorithms will be inefficient for Sorting.
VI. FUTURE SCOPE

The algorithms can perform with large datasets as some text files of multiple MBs of data consisting unsorted elements, to identify the significance of sorting algorithms in practical scenario. Any algorithm can also be improved in order to overcome the memory limitations of sorting algorithm that can sort large data when limited memory is available.

REFERENCES