NQMSort: A Fast and Efficient Way of Sorting

Mr. Sanjay Chandrakant Khubchandani
School of Management
MIT-World Peace University
Pune, India

***Abstract*— We want to rearrange them in ascending or descending order, if we have said n items in a list. Different Sorting algorithms can be used to accomplish the task, such as Merge Sort, Quicksort, Bubble Sort, Insertion Sort, Selection Sort. But any Sorting algorithm either works in quadratic time or only works for shorter array sizes. A modern Sorting method called the NQMSort Algorithm is introduced here. NQMSort is a divide-and-conquer algorithm which aims to overcome the complexity of time and space that other Sorting techniques are facing. Keywords-component; Sort; ascending; descending; array;**

Keywords—sort, quicksort, merge sort, sorting

I. INTRODUCTION

Sorting algorithms are basically used by search engines. If you search online for something or any key words, the input information is provided and presented to you Sorted by the web page's significance and index. BubbleSort, SelectionSort and InsertionSort, all of them have an $O(N^2)$ time complexity, restricting their utility to a limited number of elements not more than a few thousand data points in a set.

The quadratic time complexity of current algorithms like BubbleSort, SelectionSort and InsertionSort limits their efficiency as array elements increase i.e. array length.

In this paper, we present NQMSort that is capable of rearranging list elements in ascending or descending order. The NQMSort functions as an algorithm for dividing and conquering. In 3 division ratio, it divides the given set into three parts.

Our main contribution is the implementation of NQMSort, an efficient algorithm can Sort a list of array elements with MergeSort efficiency and rapid Sorting time.

Several current Sorting algorithms are listed in the next section: BubbleSort, InsertionSort, and SelectionSort. Section 3 provides an explanation of our algorithm for the NQM Type. Section 4 deals with conceptual analysis and explanation. Section 5 summarizes and concludes our analysis.

II. EXISTING SORTING ALGORITHMS

A. Selection Sort

SelectionSort [1] works as follows: at each iteration of Sorting, we define two areas, sorted area (no item from the beginning) and un Sorted area. We "pick" from the unsorted region one of the smallest elements and place it in the Sorted area. In each iteration, the number of array elements in the Sorted region will increment by 1. Repeat on the rest of the un Sorted area until it is complete. This method is called SelectionSort because the smallest element in the remaining elements is constantly "selected" to work.

B. Insertion Sort

We sometimes use InsertionSort [2] to Sort bridge hands: We define two areas, sorted area (one data point from the beginning being the smallest or the lowest) and un Sorted area at each iteration. We take from the un Sorted field one data point and "drop" it into the Sorted area. In each iteration, the elements in the Sorted area will increment by 1. Repeat this without the 1st data point on the rest of the unsorted field. Astrachan's work [3] of Sorting strings in Java showed that the BubbleSort is about five times slower than the InsertionSort and forty percent slower than the SelectionSort, which shows that InsertionSort is the fastest of the 3 Sorting algorithms.

C. Bubble Sort

BubbleSort algorithm works as follows: keep going through the list, interchanging the adjacent element, if the list is out of order; when zero interchanges are required on some pass of list, the list is Sorted.

In BubbleSort, InsertionSort and SelectionSort, the O(N2) time complexity limits the performance when N gets very big.

D. Merge Sort

MergeSort is an efficient, widely used, Sorting algorithm that compares elements in software engineering. Some use produces a steady type, which means that the information and yield specification for comparable components is the equivalent. MergeSort is a calculation of Divide-and-Conquer created in 1945 by John von Neumann. In a study by Goldstine and von Neumann, a thorough description and analysis of MergeSort appeared as early as 1948. [4]

E. Quicksort

Quicksort is an efficient Sorting algorithm that fills in as a deliberate technique to bring together the components of a random-access file or an array. This tends to be around a few times faster at the stage when well actualized than other Sorting algorithms, Sorting Merge and heapsort. [5]

Quicksort is a Correlation Sort, which means that it can Sort things of any kind that are distinguished by a 'less than' connection. Quicksort Productive Implementations are not a Stable, meaning that there is no protection of the general requirement for equivalent Sorting things. Quicksort will operate on an Array, which needs little extra memory to perform the Sorting. It is exactly the same as Selection Sort, except that it typically does not select Worst-case Partition.

III. NQM SORT

NQMSort uses a divide-and-conquer approach:

1) Divide the array into three parts with subject to two pivots.

2) For each part,
   - Pick a pivot element, say P.
   - Re-arrange the elements into 3 sub-parts, Those less than or equal to <= P (the left-part); P (the only element in the middle-part);
     Those greater than or equal to >= P (the right-part)
   - Repeat the process for all sub-parts

3) Merge the three sub-arrays.

Pseudo-Code –

Array as the Input: A [1.N], indices p, q, r ($p \leqslant q < r$).

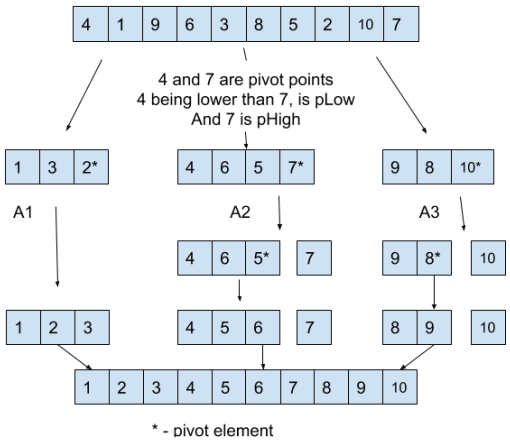
A [p…r] is the array to be Sorted.

Further implementation can be found in the box on side of the paper.

IV. EXPLANATION

The NQM Sorting Algorithm follows a divide and conquer approach where after division in 3 parts, every part is then Sorted into itself.

```
def nqmSort(arr, pLow, pHigh):
        a1 = []
        a2 = []
        a3 = []
        for element in arr:
                if (element < pLow):
                        a1.append(element)
                elif (element > pHigh):
                        a3.append(element)
                else:
                        a2.append(element)
        afterSort(a1,0,len(a1)-1)
        afterSort(a2,0,len(a2)-1)
        afterSort(a3,0,len(a3)-1)
        return(a1+a2+a3)
```



The NQM Sorting Algorithm follows a divide and conquer approach where after division in 3 parts, every part is then Sorted into itself. All 3 parts of the array are then merged into one final Sorted array using pivot and merging techniques.

CONCLUSIONS

The NQMSort Algorithm tries to encapsulate all the best features of Sorting algorithms into one. The theoretical implementation of the algorithm and pseudo code describes the flow and functionality of the algorithm. The time complexity of this algorithm may be better than Merge Sort algorithm and the Space Complexity of this algorithm may be better than Quicksort Algorithm for larger arrays.The Theoretical Analysis and Empirical Analysis of this algorithm is required to have further understanding and detailed comparison with other Sorting arrays.

REFERENCES

[1] Sedgewick, Algorithms in C++, pp.96 - 99, 102, ISBN 0 - 201-51059 - 6, Addison-Wesley , 1992

[2] Sedgewick, Algorithms in C++, pp.98 - 101, ISBN 0 - 201 - 51059-6 ,Addison - Wesley , 1992

[3] Owen Astrachan, BubbleSort: An Archaeological Algorithmic Analysis,SIGCSE 2003, http://www.cs.duke.edu/~ola/papers/bubble.pdf

[4] Katajainen, Jyrki, Träff, Jesper Larsson (1997). "A meticulous-analysis of MergeSort programs". pp. 217–228. CiteSeerX 10.1.1.86.3154.

[5] Skiena, Steven-S. (2008). The Algorithm Design Manual. Springer. p. 129. ISBN 978 1 84800 069 8.