Study of Sorting Algorithm to Optimize Search Results

Savina¹, Surmeet Kaur²

1,2 Department of Computer Science and Engineering Lovely professional university, Jalandhar, India

ABSTRACT: Sorting involves rearranging information into either ascending or descending order. There are many sorting algorithms; Sorting has been an area of interest for all those who have been involved into the mining field of data. Sorting data is important to search the data appropriately. For different types of works different types of sorting algorithm has been proposed by different scientist. There are several issues with sorting to taken into account but In this paper, we will compare the bubble sort with binary sort then merge them to take the better performance of be focusing on the time factor of the sorting. This research is about comparing and merging such an algorithm which fits into each and every account of sorting and searches the data appropriately. For the same we will have to look into the previous done algorithms also and then we would have to compare the previous work with our latest done work.

Keywords: Sorting, algorithms, comparison, bubble sort, binary sort, merge sort.

1. INTRODUCTION

In this paper we have seen that there are different types of sorting and their internal area. Now let’s get to know what exactly are the problems in sorting. The first problem is selecting the best suited algorithm for sorting. The Selection Sort is used for selecting the algorithm. There are many sorting algorithms. All sorting algorithm are for different kind of problem. To choose an algorithm depends upon what kind of structure we do have. Accordingly the architecture and complexity is different. Hence we need to design a kind of binary algorithm for sorting which can fit in most of the cases. If we would like to compare the bubble algorithms, it does have different scenario. We can merge them according to the complexity which they possess. It is also necessary that we choose the best category, i.e., whether internal or external sorting algorithm. Now whatever we choose as an algorithm, our aim is to emphasis on the timing factor. By timing factor I mean that how much time it takes to execute the structure of the searching. Hence our aim is to compare the best suited generated algorithm for searching in terms of time compatibility.

1.1 Insertion Sort: Insertion sort is a simple a comparison sorting algorithm. Every iteration of insertion sort removes an element from the input data, inserting it into the correct position in the already-sorted list, until no input elements remain. The choice of which element to remove from the input is arbitrary, and can be made using almost any choice algorithm. Illustration of insertion sort for input list [5, 2, 4, 6, 1, 3]:

![Example of straight selection sort list](image)

1.2 Selection Sort: Selection sort is also a simple a comparison sorting algorithm. The algorithm works as follows:
1. Find the minimum value in the list
2. Swap it with the value in the first position
3. Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

Figure 1.2: Example of straight selection sort list [1,3,4,5,7]:

![Example of straight selection sort list](image)

1.3 Merge Sort

2. RELATED WORK

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms such as search and merge algorithms which require input data to be in sorted lists; it is also often useful for data and for producing human-readable Output. More formally, the output must satisfy two conditions. The output is in non decreasing order each element is no smaller than the previous element according to the desired total order. The output is a permutation reordering of the input.

2.1 Bubble Sort

Although the shell sort algorithm is significantly better than insertion sort, there is still room for improvement. One of the most popular sorting algorithms is quick sort. Quick sort executes in \(O(n \log n)\) on average and \(O(n^2)\) in the worst-case. However, with proper precautions, worst-case behavior is very unlikely. Quick sort is a non-stable sort. It is not an in-place sort as stack space is required. The quick sort algorithm works by partitioning the array to be sorted, then recursively sorting each partition. In Partition, one
of the array elements is selected as a pivot value. Bubble Sort is not known to be a good algorithm because it is a quadratic-time sorting algorithm. However, efforts have been made to improve the performance of the algorithm. With Bidirectional Bubble Sort, the average number of comparisons is slightly reduced and Batch’s Sort similar to Shell sort also performs significantly better than Bidirectional Bubble Sort by carrying out comparisons in a novel way so that no propagation of exchange is necessary. Bubble sort is a simple sorting algorithm. It works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. First pass bubble out the largest element and places in the last position and second pass place the second largest in the second last position and so on. Thus, in the last pass smallest items is placed in the first position. Because, it only uses comparisons to operate on elements, it is a comparison sort.

For I = 1: n,
    Swapped = false
    For j = n: i + 1,
        If a[j] < a[j-1],
            Swap a[j], a[j-1]
            Swapped = true
    → Invariant: a[1…i−1] in final position
    Break if not swapped
End

Example: - Bubble sort (sorted elements are shaded)

```
   Original List
     |     |     |     |     |
    23  78  45  8  32  56
```
```
   after Pass 1
     |     |     |     |     |
    8   23  78  45  32  56
```
```
   after Pass 2
     |     |     |     |     |
    8   23  32  78  45  56
```
```
   after Pass 3
     |     |     |     |     |
    8   23  32  45  78  56
```
```
   after Pass 4
     |     |     |     |     |
    8   23  32  45  56  78
```

2.2 BINARY SORT: - In computer science, a binary sort algorithm finds the position of a specified value within a sorted array. In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is less than the middle element’s key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special “Not found” indication is returned. However, if we place our items in an array and sort them in either ascending or descending order on the key first, then we can obtain much better performance with an algorithm called Binary Sort. These are following functions in this paper which can be implemented in Binary Sort Order:

1. Bin sort is recursive: it determines whether the search key lies in the lower or upper half of the array, then calls itself on the appropriate half.
2. There is a termination condition two of them in fact!
   - If low > high then the partition to be searched has no elements in it and
   - If there is a match with the element in the middle of the current partition, then we can return immediately.
3. AddToCollection will need to be modified to ensure that each item added is placed in its correct place in the array. The procedure is simple:
   - Search the array until the correct spot to insert the new item is found,
   - Move all the following items up one position and
   - Insert the new item into the empty position thus created.
4. Bin search is declared static. It is a local function and is not used outside this class and if it were not declared static, it would be exported and be available to all parts of the program. The static declaration also allows other classes to use the same name internally.
2.3 MERGE SORT: - Merge sort takes advantage of the ease of merging already sorted lists into a new sorted list. Merge sort is good for data that's too big to have in memory at once because its pattern of storage access is very regular. It also uses even fewer comparisons with binary sort and heap sort. It is especially suited for data stored as linked lists. Merge Sort is a $O(n \log n)$ sorting algorithm. 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. Of the algorithms described here, this is the first that scales well to very large lists, because its worst-case running time is $O(n \log n)$. Merge sort has seen a relatively recent surge in popularity for practical implementations, being used for the standard sort routine in the programming languages Perl, Java and among others. Merge sort is an $O(n \log n)$ comparison-based divide and conquer sorting algorithm. Conceptually, a merge sort works as follows:

1. If the list is of length 0 or 1, then it is already sorted. Otherwise:
2. Divide the unsorted list into two sub lists of about half the size.
3. Sort each sub list recursively by re-applying merge sort algorithm.
4. Merge the two sub lists back into one sorted list.

3. RESULT & DISCUSSION

Sorting algorithms used in computer science are often classified by: 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 $O(n \log n)$ and bad behavior is $O(n^2)$. 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. Recursion some algorithms are either recursive or non-recursive, while others may be both e.g., merge sort. Stability stable sorting algorithms maintain the relative order of records with equal keys i.e., values. Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator. Adaptability Whether or not the pre sorted of the input affects the running time. Let get to know about two sorting techniques and analyze their performance. The two techniques are: 1. Internal Sorting, 2. External Sorting.

<table>
<thead>
<tr>
<th>Name</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
<th>Memory</th>
<th>Stable</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>$n$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>l</td>
<td>Yes</td>
<td>Exchanging</td>
</tr>
<tr>
<td>Binary Sort</td>
<td>$n$</td>
<td>$n\log n$</td>
<td>$n\log n$</td>
<td>n</td>
<td>Yes</td>
<td>Insertion</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>$n\log n$</td>
<td>$n\log n$</td>
<td>$n\log n$</td>
<td>Depends: Worst case is $n$</td>
<td>Yes</td>
<td>Merging</td>
</tr>
</tbody>
</table>

At the result of this paper is in the sorting algorithm is that the binary sort is merge with bubble sort to obtain better performance of the time complexity. In this comparison of bubble sort and binary sort there should be check the complexity of best case, average case and worst case. Then merge the bubble sort with binary sort and insertion method is used for binary sort and selection method is used for bubble sort. These algorithms are used to depend on need.

4. CONCLUSION

In this paper, we can use sorting algorithms. We present different sorting algorithms models. This research is about finding or creating such an algorithm which fits into each and every account of sorting and searches the data appropriately, such as Bubble Sort, B Sort and Merge Sort. However, we study sorts because general sorts do not work in all situations and because sorting is a simple illustration of algorithmic techniques. This paper started by giving an overview over selected algorithms for time factor of the sorting, which uncovered both weaknesses and strengths of existing approaches. Algorithms should give a time complexity for the selected set of quality characteristics and explain the relationships between concepts. This paper has introduced two sorting algorithms than the merge sorting algorithms. The first is based upon sorting by the insertion technique, whereas the second is based upon sorting by the exchange technique. The analysis of Merge algorithms shows that they score the best performance when binary sort and heap sort can be merged. This paper is best compare to privies this is compare based on the time, complexity if the current is time is less then thus is the best algorithm.

REFERENCES


Savina received the MCA degree from Punjab Technical University and pursuing M.tech in Lovely professional university, Jalandhar, India