A Study of Comparison for Sorting Algorithms Based on Data Sequences

Cho Cho Lwin  
University of Computer Studies (Bannmaw)  
chocholwin216@gmail.com

Nwey Zin Moe  
University of Computer Studies (Bannmaw)  
nweyzinmoe@gmail.com

Pan Nu Wai  
University of Computer Studies (Bannmaw)  
pannuwaicu@gmail.com

Abstract

There are many problems in different fields of computer sciences, computer architecture, artificial intelligence, database applications and computer network. In computer science, the sorting algorithm is fundamental and important. Sorting is usually refers to bringing the set of data items into some well-defined order. When dealing with research data, sorting is a common method used to visualize data in a form that makes it easier to understand the stories the data tells. To do this, we must specify the concept of line item order consider. I also want to work on existing sorting algorithms and presenting my finding after studying some well-known which are Bubble sort, merge sort, Insertion sort, Quick sort and Selection sort. In this paper we will study these algorithms with the type of data structure obtained by using data sequences, time complexities and efficiency extra overhead.

Keywords: Bubble sort, Merge sort, Insertion sort, Quick sort and Selection sort.

1. Introduction

Sorting is important having things organized makes it easier to find things. There are many sorting algorithms but there differ complexity. Sorting algorithms have attracted the attention of many researchers, with the need for sorting algorithms taking into consideration performance and memory space along with the rapid growth of information [9]. We can reduce the average (and worst) complexity of finding something O (log2 n) steps, this will be an O (n) step without sorting. Therefore, if we have frequent searches, it is worth the effort to sort the entire collection [8]. This white paper describes five sort algorithms already available as insertion sort, bubble sort, selection sort, and merge sort. Compare their best-case, worst-case, average-case, space complexity and speed up. Next, let's look at the performance and comparison of all five sorting algorithms based on their growth of the number of n elements. All five sorting algorithms take time to execute. We check which algorithm makes it faster to process elements and which sorting algorithm the best performance and a least bad case to show.

2. Sorting Algorithm

Sorting algorithms are important tools for programmers. Different algorithms Different situations have different "best" sorting methods [7]. This algorithm is designed to aggregate information in different ways, for example. The numbers are arranged by ascending and descending order, and the string elements can be arranged alphabetically.

2.1. Bubble Sort

Bubble sort follows an exchange pattern. This is a sorting algorithm that works by exchanging adjacent elements. It is very easy to implement, but there will be especially slow to run. Let's say we have size n that we want to sort. Bubble sort start by comparing a [n-1] with a [n-2] and swaps them if they are in the wrong order [1].

Algorithm BubbleSort(arr)

Pre: list ≠ ∅
Post: value of the ordered size of the list items
for a ← 0 to arr.n - 1
for m ←0 to arr.n -1
if arr[a] < arr[m]
swap(arr[a], arr[m])
end if
end for
end for
return arr
end BubbleSort

Using bubble sort algorithm to sorting elements in the given data sequence following table.

Table 1. Set of elements for bubble sort

<table>
<thead>
<tr>
<th>Data</th>
<th>8</th>
<th>6</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass1</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Pass2</td>
<td>6</td>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Pass3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

So the number of comparisons is O (n) and the bubble sort performance O (n) operation. Hence the time complexity of bubble sort is O (n²) [9]. The spatial width for Bubble Sort is O (1) because a single additional memory space is a temp variable. Therefore worst-case time complexity of bubble sort is O (n²).

2.2. Merge Sort

The merge sort an also sorting algorithm which is based on divide and conquers strategy which is more popular solving technique. The algorithm divides each list into two lists of the same size (left and right). Compiling a list of each and then based on merging the
split lists together [5]. Recursive divide and conquer approach a list of \( n \) length.

algorithm MgSort(list)

Pre: list \( \neq {} \)

Post: sorted values of ascending order

if list.Count = 1

return list

end if

\( m \leftarrow \text{list.count} = 2 \)

\( l \leftarrow \text{list(m)} \)

\( r \leftarrow \text{list(list.count - m)} \)

for \( k \leftarrow 0 \) to \( l\).count - 1

\( l[k] \leftarrow \text{list}[k] \)

end for

for \( q \leftarrow 0 \) to \( r\).count - 1

\( r[q] \leftarrow \text{list}[q] \)

end for

\( l \leftarrow \text{MgSort}(l) \)

\( r \leftarrow \text{MgSort}(r) \)

return MergeOrdered(l, r)

end MgSort

Following are the procedure to sort a given set of elements (8, 6, 10 and 5).

<table>
<thead>
<tr>
<th>Table 2. Set of elements for merge sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Pass1</td>
</tr>
<tr>
<td>Pass2</td>
</tr>
<tr>
<td>Pass3</td>
</tr>
</tbody>
</table>

In addition, \( O(n) \) time will be required to compile sub-arrays to integrate sub-arrays. Divide and conquer will be \( O(\log n) \). However \( n \) sub-lists need to be sorted. Therefore, the total time for merge sort operation becomes \( (\log n + 1) \), which gives us the time complexity of \( O(n \cdot \log n) \). The constant time is \( O(n) \). Therefore worst-case time complexity of merge sort is \( O(n \log n) \).

2.3. Insertion Sort

Insertion sort is a simple sorting algorithm. It's easier to think of insertion when we sorted the middle process, half the sequence. At this point, the current position is somewhere in the middle of the list [3]. To performs current position at the left-most element of the array and invoke insert to insert each element encountered into its correct position. Compare the current element will all elements in the left side.

algorithm InsertSort(l)

Pre: \( l \neq {} \)

Post: Array into values of ascending order

\( \text{unst} \leftarrow 1 \)

while \( \text{unst} \leq l\).Count

\( \text{hold} \leftarrow \text{unst} \)

\( i \leftarrow \text{unst} - 1 \)

while \( i \geq 0 \) and \( \text{hold} \leq l[i] \)

\( l[i + 1] \leftarrow \text{hold} \)

\( \text{unst} \leftarrow \text{unst} + 1 \)

end while

return \( l \)

end InsertSort

Using insertion sort algorithm to sorting elements in the given data sequence following table.

<table>
<thead>
<tr>
<th>Table 3. Set of elements for insertion sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Pass1</td>
</tr>
<tr>
<td>Pass2</td>
</tr>
<tr>
<td>Pass3</td>
</tr>
</tbody>
</table>

The time complexity of insertion sort depends on the number of inversions in the input array [12]. In this case, the algorithm runs for \( O(n) \) time. The worst case time is still \( O(n^2) \). The worst case is when the array is rearranged [8]. If require only a constant amount \( O(1) \) of additional memory space.

2.4. Selection Sort

This is the simplest of sorting method. Finding the smallest element bring smallest element in first position [11]. Compare the lowest to the second element. If the second element is less than the minimum, specify the second element. This process occurs until the last element. After each element is searched repeatedly, the lowest element is positioned in front of the unsorted list and then repeated until all the elements are placed at their correct positions [2].

Algorithm Selectionsort(list)

Pre: list \( \neq {} \)

Post: list has been sorted values

for \( i \leftarrow 0 \) to list.Count - 1

\( k = i \)

for \( j \leftarrow i + 1 \) to list.Count

if \( a[j] < a[k] \) \( k = j \) swap \( a[i] \) and \( a[k] \)

endfor

endfor

return list

end Selectionsort

Following are the procedure to sort a given set of elements (8, 6, 10 and 5).

<table>
<thead>
<tr>
<th>Table 4. Set of elements for selection sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Pass1</td>
</tr>
<tr>
<td>Pass1</td>
</tr>
<tr>
<td>Pass1</td>
</tr>
</tbody>
</table>

The time complexity of selection sort is \( O(n^2) \) in all the same case. Which \( O(n^2) \) time complexity is in terms of number of comparisons. Selection sort is an in-place algorithm. Hence, the space complexity works out to be \( O(1) \) [3].

27
2.5. Quick Sort

Quick sort has approach sequence to be sorted is partitioned into two parts, such that all elements of the first part are less than or equal to all elements of the second part. Then the two parts are sorted separately by recursive application of the same procedure conquer [6]. The combination of the two parts is constantly flowing. If you look for an element called pivot, the element on the left is smaller than the pivot, and the second half is bigger than the pivot.

```
algorithm QuickSort(list)
    Pre: list ≠ ∅
    Post: list has sorted order
    if list.Count = 1 // already sorted
        return list
    end if
    pivot ← MedianValue(list)
    for i ← 0 to list.Count - 1
        if list[i] = pivot
            equal.Insert(list[i])
        end if
        if list[i] < pivot
            less.Insert(list[i])
        end if
        if list[i] > pivot
            greater.Insert(list[i])
        end if
    end for
    return Concatenate(QuickSort(less), equal, QuickSort(greater))
end Quicksort
```

Using Quick sort algorithm to sorting elements in the given data sequence following table.

Table 5. Set of elements for quick sort

<table>
<thead>
<tr>
<th>Data</th>
<th>8</th>
<th>6</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass1</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Pass2</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>Pass3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

It is found when the element of the element is the greatest or smallest element [10]. A sub-plot is always empty, and a subdivision contains n-1 elements. Best-case complexity is \( O(n \log n) \). The space complexity for quick sort is \( O(\log n) \) [6].

3. Comparison Table

In comparison-based sorting algorithms, the elements of the array are compared with each other and determine which element should be the first in the last sorted list. The table 6 shows the time complexity and time of speed operation of these sorting algorithms [4]. There are described big O notations of five algorithms in figure 1.

![Figure 1. Sorting algorithm of time complexity](image)

Table 6. Comparison of sorting algorithms

<table>
<thead>
<tr>
<th>Name of algorithm</th>
<th>Time complexity</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Merge sort</td>
<td>( O(n \log n) )</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Quick sort</td>
<td>( O(n \log n) )</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>( O(n) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Selection sort</td>
<td>( O(n^2) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>

In table 7, five algorithms are implemented using the same java programming language. The sizes of array elements are varied from 100 to 5000. This calculates the timing of input sequences measuring the time of milliseconds in java. All tests are performed on our windows system. Depending on the operating system, the running time may change.

In figure 2 is show the running time of sorting algorithms to perform the data sequences.
would like to thank prof. Dr.Nwe Nwe Hlaing(head of Faculty of Computer science).

**References**


**Table 7. Running time (in Sec) of sorting algorithms**

<table>
<thead>
<tr>
<th>No. of elements</th>
<th>Bubble Sort</th>
<th>Merge Sort</th>
<th>Quick Sort</th>
<th>Insertion Sort</th>
<th>Selection Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.0053</td>
<td>0.006</td>
<td>0.007</td>
<td>0.11614</td>
<td>0.010230</td>
</tr>
<tr>
<td>500</td>
<td>0.0356</td>
<td>0.303</td>
<td>0.032</td>
<td>0.02963</td>
<td>0.046357</td>
</tr>
<tr>
<td>1000</td>
<td>0.0828</td>
<td>0.653</td>
<td>0.063</td>
<td>0.05793</td>
<td>0.050502</td>
</tr>
<tr>
<td>2000</td>
<td>0.0994</td>
<td>0.092</td>
<td>0.109</td>
<td>0.12958</td>
<td>0.068162</td>
</tr>
<tr>
<td>3000</td>
<td>0.1485</td>
<td>0.141</td>
<td>0.203</td>
<td>0.16018</td>
<td>0.168929</td>
</tr>
<tr>
<td>4000</td>
<td>0.2063</td>
<td>0.177</td>
<td>0.165</td>
<td>0.18979</td>
<td>0.158064</td>
</tr>
<tr>
<td>5000</td>
<td>0.2541</td>
<td>0.168</td>
<td>0.180</td>
<td>0.19998</td>
<td>0.199698</td>
</tr>
</tbody>
</table>

**Figure 2. Sorting algorithm performance test**

**4. Conclusion**

This paper compares the performance of five algorithms and data sequences. Since there is no specific algorithm that can solve any problem, it is necessary to make more comparisons of sorting algorithms to work with them more effectively. As survey above, each of the sorting algorithms has runtime of data sequences. In addition, it is easy to implement and suitable for small data sets. Input type and comparative performance of inputs vary on different types of inputs. For example, insertion sort is more effective than selecting data items and exchanging string data items. Future work requires the study of parallel transformation concepts to implement parallel processing and the implementation of a sorting algorithm.

**Acknowledgement**

We would like express our respect and thanks to Pro-Rector Dr. May Phyo Oo, Editor and Chief at the University of Computer studies (Bannmaw). And also we