An Analytical Study of Various Sorting Algorithms

Vimal A.Vaiwala
Assistant Professor
SDJ International College
Surat, Gujarat

ABSTRACT
An algorithm is accurately specified by a sequence of instructions to be carried out in order to solve a given problem. Sorting is considered by computer science as an intermediate step in many operations. Sorting is the process of putting a list of components in a specific order. According to their primary values, the elements are ordered in ascending or descending order. This study offers various data structure sorting algorithms, such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort and Quick Sort and also analyses how well they perform in terms of time complexity. These algorithms are crucial and have been a focus for a while, but the issue is remain the same of "which algorithm is use to when?" which is the important aspect for this study. This research paper gives an in-depth analysis of these algorithms workings and draws comparisons between them based on a number of different characteristics to reach solution.

Keywords— Algorithm, Sorting, Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Complexity

Introduction

In computer science, a sorting algorithm is an efficient algorithm which performs an important task that puts elements of a list in a certain order or arranges a collection of items into a particular order. Sorting data has been developed to arrange the array values in various ways for a database. Sorting will order an array of numbers from lowest to highest or from highest to lowest, or arrange an array of strings into alphabetical order. It sorts an array into increasing or decreasing order[1]. Most simple sorting algorithms involve two steps which are compare two items and swap two items or copy one item. It continues executing over and over until the data is sorted. Sorting is nothing but storing data in the sort order; it may be in ascending or descending order. Sort phrase highlighted with the keyword. There are so many things in our real life; we have to look for a specific entry in the database for songs ride in the merit list, a particular phone number, a page in a book, etc. Sort organizes the data in a sequence that makes searching easier. Each entry will be when you have a key. Sorting is the process of arranging the elements in some logical order. Sorting is classified into following categories:

External sorting: It deals with sorting of the data stored in data files. This method is used when the volume of data is very large and cannot be held in computer main memory.

Internal sorting: It deals with sorting the data held in memory of the computer.

System complexity: In the context of computation. On the basis of performance, this algorithm can be divided into worst case, average case, and best case scenarios.

Computational complexity: Based on the number of swaps. Each sorting algorithm uses a different number of swaps to complete the operation.

WORKING OF ALGORITHMS

Bubble Sort[2]

In bubble sort, two elements are compared and if they are out of order then they will be interchanged. This method will cause records with small keys to move or "bubble up". Assume there are R1, R2, R3, R4……Rn elements to sort in a table [2]. During the first pass R1 and R2 will be compared and if they are out of order, they will be interchanged. This process is repeated for R2 and R3 and R4 and so on. This sorting technique is not efficient for large tables, for small tables it works fine.

For Example, Consider we want to sort the following data:

40, 30, 50, 20, 10

40	30	30	30	30		30	30	30	30	30
30	40	40	40	40		40	40	20	20	20
50	50	50	20	20		20	20	40	10	10
20	20	20	50	10		10	10	10	40	40
10	10	10	10	50		50	50	50	50	50
PASS-1						**PASS-2**				
30	20	20	20	20		20	10	10	10	10
20	30	10	10	10		10	20	20	20	20
10	10	30	30	30		30	30	30	30	30
40	40	40	40	40		40	40	40	40	40
50	50	50	50	50		50	50	50	50	50
PASS-3						**PASS-4**				

Working on Bubble Sort

Advantage: Simplicity and ease-of-implementation.
Disadvantage: Code inefficient.

Algorithm

In this algorithm, n is the size of the table, arr is the array of n elements that we want to sort.

Step 1: [Initialization] i=0.
Step 2: Repeat through step 4 while i<n.
Step 3:Repeat through for j = 0 to n – i
Step 4 :if arr[j]>arr[j+1] then
    Temp=a[j]
    a[j]=a[j+1]
    a[j+1]=Temp
    End if
    End of Step 3
    End of Step 2
Step 5: Exit

Selection Sort[2]

It is one of the easiest ways to sort a table. Beginning with the first record in the table, a search is performed to locate the element which has the smallest key. When this element is found, it interchanged with the first record table [2]. This interchange places the record with smallest key in the first position of the table. Then the second smallest element will be searched by examining the keys from second position onwards. This process is continued until all records have been sorted in ascending order.

Assume we have to sort data in ascending order:
For Example 40,30,50,20,10

40	10	10	10	10
30	30	20	20	20
50	50	50	30	30
20	20	30	50	40
10	40	40	40	50
No of Passes	(1)	(2)	(3)	(4)

Working on Selection Sort

Advantage: Simple and easy to implement
Disadvantage: Inefficient for large lists, so similar to the more efficient insertion sort, the insertion sort should be used in its place.

Algorithm
In this algorithm, n size of the table, arr is the array of n elements that we want to sort.
Step 1: [Initialization] i=0.
Step 2: Repeat through step 7 while i < n.
Step 3: j=i+1
Step 4: Repeat through Step 6 while j < n
Step 5: if arr[i] > arr[j] then
Temp=arr[i]
arr[i]=arr[j]
arr[j]=Temp
End if
Step 6: j=j+1
Step 7: i=i+1
Step 8:Exit

Insertion Sort[2]

Insertion sort arranges the data in order at the time of insertion. It scans the list from 0 to n – 1 , inserting each element into proper position in the previously sorted list. Among simple sort algorithms, insertion sort is one of the best[2]. It uses the less number of comparisons then bubble sort and selection sort. It is quite efficient for small sized table. For large tables, it is extremely slow.
For Example 35,42,23,47,55,30,50,37,57,53

35	23	23	23	23	23	23	23	23	23
42	35	35	35	30	30	30	30	30	30
23	42	42	42	35	35	35	35	35	35
47	47	47	47	42	42	37	37	37	37
55	55	55	55	47	47	42	42	42	42
30	30	30	30	55	50	47	47	47	47
50	50	50	50	50	55	50	50	50	50
37	37	37	37	37	37	55	55	53	53
57	57	57	57	57	57	57	57	55	55
53	53	53	53	53	53	53	53	57	57

Working on Insertion Sort

Advantage: Relative simple and easy to implement. Twice faster than bubble sort.

Disadvantage: Inefficient for large lists.

Algorithm

In this algorithm, n is the size of the table, arr is the array of n elements that we want to sort and no is the number that we want to enter into array.

Step 1: [Initialization] count = 1, j=0
Step 2: Repeat through step 3 to 5 while count <= n.
Step 3: Enter number and store it into no.
count = count+1
i = j – 1
Step 4: Repeat step 4 until no < arr[i] and i >=0
arr[i+1] = arr[i]
i = i – 1
End of step 4
Step 5: arr [i + 1] = no
j = j + 1
End of step 2
Step 6 : Exit

Quick Sort[2]

The Quick Sort is an in-place divide and conquer, massively recursive sort. It is also known as Partition-Exchange sorting technique [2]. It performs very well on large tables.

It has two phases:

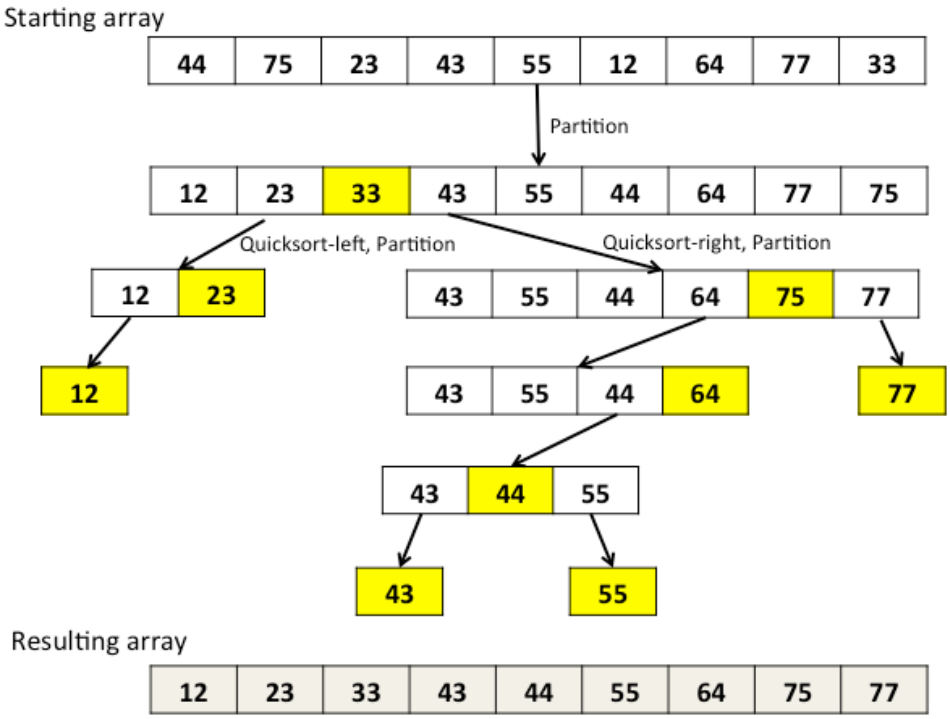
1. The partition phase
2. The sort phase.

Quick Sort treats an array as a list of elements when sort begins it select the list middle element as the list pivot. It divides the list into two sub lists.

1. With elements that are less than the list pivot
2. List with the elements greater than or equal to the list pivot.

It recursively invokes itself with both list. Each time when the sort is invoked, it further divides the elements into smaller sub lists.

For example :- 44, 75, 23, 43, 55, 12, 64, 77, 33



Working on Quick Sort

Advantage: Fast and efficient

Disadvantage: Show horrible result if list is already sorted.

Algorithm

Q_sort (list, first, last)

list →Represents the list of elements.

first→Represents the position of the first element in the list (only the starting point, it's value changes during the execution of the function).

last→Represents the position of the last element in the list (Only at starting point the value of it changes during the execution of the function).

Step 1: [Initially]
low = first
high=last
pivot=list[(low + high)/2] [Middle element of the element of the list]

Step 2: Repeat through Step 7 while (low < = high)

Step 3: Repeat through Step 4 while (list [low] < pivot)

Step 4: low = low + 1

Step 5: Repeat through Step 6 while (list [high] > pivot)

Step 6: high = high – 1

Step 7: if (low < = high)

I. Temp=list[low]
II. List[low]=list[high]
III. List[high]=temp
IV. low = low+1
V. high = high – 1

Step 8: if (first < high) Q_Sort (list, first, high)

Step 9: if (low <last) Q_Sort (list, low, last)

Step 10: exit

Merge Sort[2]

Using merge sort, we can merge sort two sorted tables into single sorted table[2]. In merge sort, it is compulsory that the tables that we want to merge, should be in sorted order.

For Example to sort following tables:

Table 1: 12, 19, 21, 24, 24, 39, 44

Table 2: 13, 16, 27, 29, 43

First we have to compare 1st two elements of two tables. Whichever is less, is taken into a third table. Now the pointer of that table (which has lesser element) and the third table is incremented by 1. Again two elements of two tables are compared. The lesser is taken into third table, pointer of that table (which has lesser element) and the third table is incremented by one. This process is repeated until all the elements of two tables are sorted and merged in to third table. This process is shown in Fig.

Table-1	12	19	21	24	39	44					
Table-2	13	16	27	29	42						
Table-3	12	13	16	19	21	27	24	29	39	43	44

Working on Merge Sort

Advantage*:* Well suited for large data set.

Disadvantage*:* At least twice the memory requirements than other sorts

Algorithm

Merge_Sort(a1, n1, a2, n2)

n1 and n2 are the sizes of two sorted tables a1 and a2 respectively. a1 and a2 are two sorted tables, which contains n1 and n2 elements respectively. a3 is the array of n1 + n2 elements. i, j and k are the pointer for the arrays a1, a2 and a3 respectively.

```
Step 1: [Initialization] i = j = k = 0
Step 2: Repeat step 3 to 4 while k < (n1 + n2)
Step 3: if a1[i] < a2[j] then
            a3[p]=a1[i]
            p=p+1
            i=i+1
        else if a1[i] > a2[j] then
            a3[p]=a2[j]
            p=p+1
            i=i+1
        else (both are equal)
            a3[p]=a1[i]
            p=p+1
            i=i+1
            j=j+1
        end if
Step 4: if (i=n1) or (j=n2) then
        Break
Step 5: Repeat while i ! = n1
        a3[p] = a1[i]
        p=p+1
        i=i+1
Step 6: Repeat while j ! = n2
```

```
        a3[p] = a1[j]
        p=p+1
        j=j+1
Step 7 Return
```

COMPARISON BETWEEN DIFFERENT SORTING TECHNIQUES

Sort	Best Case	Average case	Worst Case	Memory	Comments
Bubble	n–1 comparison	n(n-1)/2 comparison	n(n-1)/2 comparison	Constants	A very simple algorithm, to code and one to explain, but very slow. It is very well for short table array.
Selection	n^2 comparison	n^2 comparison	n^2 comparison	Constants	Even a perfectly sorted input requires scanning the entire array.
Insertion	n–1 comparison	n(n-1)/2 comparison	n(n-1)/2 comparison	Constants	In the best case (already sorted), every insert requires constant time. Inefficient for large lists.
Quick	n^2 comparison	n log n comparison	n log n comparison	Constants	It is fast and efficient algorithm but Show horrible result if list is already sorted.
Merge	n *log n comparison	n *log n comparison	n *log n comparison	Depends	At least twice the memory requirements than other sorts. It is used divide and conquer method.

COMPARISON BY USING CODE WRITTEN IN C SHARP LANGUAGE

I will determine the effectiveness of the various sorting algorithms according to the time by using randomized trials. The build setting will be built using the C# language in Asp.Net Framework. I will discuss and implement several sorting algorithms such as bubble sort, selection sort, and insertion sort and will also include complexity sort such as quick sort and merge sort. I will represent these algorithms as a way to sort an array or integers and run random trails of length.

To examine, I create a namespace called "ConsoleApplication1" which contains one class —"SortComparison". This class contains various Functions for Selection Sort, Insertion Sort, Quick Sort, Bubble Sort and Merge Sort. In Main () function I will be using Random Number Generator for generating the number of elements. I will be using the StopWatch Class[11] of the System.Diagnostics Namespace which will help me to find the running time of the algorithm in microseconds.

```
int[] arr_selection = new int[10000]; //Number of elements are 10000 int[] arr_insertion = new int[10000];
int[] arr_bubble = new int[10000];
 int[] arr_merge = new int[10000];
Random rn = new Random();
for (int i = 0; i < arr_selection.Length; i++)
{
        arr_selection[i] = rn.Next(1, 10000)
        // Random Number for generating 10000 elements
}
```

Similarly we can generate random numbers for different sorting algorithms System.Diagnostics.Stopwatch

```
sw = new System.Diagnostics.Stopwatch();
sw.Start();
        // Sorting Function to be called
sw.Stop();
long timeselection = sw.ElapsedTicks/(System.Diagnostics.Stopwatch.Frequency/(1000L * 1000L));
```

Time election is the time in the microseconds. I will be calling each sorting function to find the running time of that sorting algorithm so that I can compare the running time of the algorithms. For this I passed different number of elements (N=10, 100, 1000, 10000) to the sorting Functions. I ran the program three times for each value of N (i.e. 10 or 100 or 1000 or 10000) and tried to find the running time of each sorting algorithm).
Table shows the running time of each algorithm for first, second, and third run. I have also calculated the average running time (In Microseconds) based upon the running time.

First Run(Time in Microseconds)					
N	Selection Sort	Insertion Sort	Bubble Sort	Merge Sort	Quick Sort
10	267	223	208	535	318
100	298	250	290	598	337
1000	3686	3286	8393	3416	605
10000	348511	263315	809311	91848	3316
Second Run(Time in Microseconds)					
10	307	219	206	543	320
100	563	442	519	1073	609
1000	3924	2805	8308	4410	617
10000	343363	264876	809943	103267	3366
Third Run(Time in Microseconds)					
10	265	318	223	601	337
100	337	245	291	598	335
1000	3728	2939	8402	4005	609
10000	336569	294503	802286	142748	3335
Average					
10	279.66	253.33	212.33	559.66	325.00
100	399.33	312.33	366.66	756.33	427.00
1000	3779.33	3010.00	8367.66	3943.66	610.33
10000	342814.33	274231.33	807180.00	112621.00	3339.00

CONCLUSION

In this study I have studied about various sorting algorithms and their comparison. There is advantage and disadvantage in every sorting algorithm. To find the running time of each sorting algorithm I used one program for comparing the running time (in Microseconds). After running the same program on three different runs (for each different value of N=10, 100, 1000, 10000), I calculated the average running time for each algorithm. From the time I can conclude that Quick Sort is the most efficient algorithm.

REFERENCES

[1] Sareen Pankaj(March 2013), *Comparison of Sorting Algorithms (On the Basis of Average Case)*, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3.

[2]Morena R. , Tailor P., Dindoliwala V. , (2011) *Data Structure* , Nirav Prakashan ISBN No.:978-93-81060-44-5, First edition.

[3] Sedgewick, *Algorithms in C++*, pp.96-98, 102, ISBN 0-201- 51059-6,Addison-Wesley , 1992

[4] http://www.dotnetperls.com/stopwatch