Find and Place Sorting Technique for Unique Numbers

Vishweshwarayya C Hallur,
Lecturer
Angadi Institute of Technology & Management, Belgaum

Ravindra S Hegadi, PhD.
Associate Professor
School of Computational Science,
Solapur University, Solapur

S. P. Sajjan
Lecturer
Karnataka University
Dharwad

ABSTRACT

To perform searching operation we have different kinds of searching techniques. These all searching algorithms works on data, which are previously sorted. An efficient algorithm is required to make searching fast and efficient. This paper presents a new sorting algorithm named as "Find and Place Sorting Technique for Unique Numbers (FPSTUN)". This FPSTUN is designed to perform sorting quickly and easily and also efficient as existing algorithms in sorting.

Key Words: Algorithm, Sorting, FPSTUN.

1. INTRODUCTION

Using a computer to solve problem involves directing it on what step it must follow to get the problem to be solved. The step it must follow is called an algorithm.

There is a direct correlation between the complexity and effectiveness of an algorithm [1].

The complexity of an algorithm is generally written in the form of Big O(n) notation form, where Big O represents the complexity of the algorithm and value n represents the size of the list. The two groups of sorting algorithm are O(n2), which includes selection sort, shell sort, bubble sort, and insertion sort. And O(n log(n)) which includes the quick sort, heap sort, and merge sort[2].

Since the drastic advancement in computing, most of the research is done to solve the sorting problem, perhaps due to the complexity of solving it efficiently despite its simple and familiar statement. It is always very much difficult to say that one sorting technique is better than another. Performance of the various sorting algorithms depends upon the data being sorted. Sorting is very much important and is used in most of the applications and there have been plenty of performance analyses [3][4].

There has been growing interest on enhancements to sorting algorithms that do not have an effect on their asymptotic complexity but rather tend to improve performance by enhancing data locality[5].

2. PROPOSED SYSTEM

In FPSTUN technique the first number will be compared with all the elements in the list, at the end of each pass the selection of proper index in new list is done and then element is placed it to that position in the new list. And this step will be repeated for n number of times to get complete sorted list.

The best case time complexity of FPSTUN is Omega Ω(n2), the average case of the FPSTUN is theta Θ(n2) and worst case of FPSTUN is Big O(n2).

Diagrammatic representation of ISSA:

Unsorted List with size a[5]

20	5	49	23	2
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 1: Unsorted Array

New List i.e. b[5]

a[0]	a[1]	a[2]	a[3]	a[4]

Fig 2: Empty Array

After the first pass, the index of 20 is identified and then 20 is placed in to that position in the new list. Therefore index of 20 is 2. Hence,

		20		
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 3:

After the second pass, the index of the 5 is identified and then 5 is placed into that position in the new list. Therefore index of 5 is 1. Hence,

	5	20		
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 4:

After the 3rd pass, the index of 49 is identified and then 49 is placed into that position in new list. Therefore index of 49 is 4. Hence,

	5	20		49
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 5:

After the 4th pass, the index of 23 is identified and then 23 is placed into that position in the new list. Therefore index of 23 is 3. Hence,

	5	20	23	49
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 6:

After the 5th pass, the index of 2 is calculated and then 2 is placed into that position in the new list. Therefore index of 2 is 2. Hence,

2	5	20	23	49
a[0]	a[1]	a[2]	a[3]	a[4]

Fig 7:

3. ALGORITHM

Algorithm FPSTUN(a,b,n)

for i ← 0 to n-1

```
k ← 0
for j ← 0 to n-1
        if (a[i] > a[j]) then
                increment k
b[k] ← a[i]
```

4. COMPARISONS OF FPSTUN WITH OTHER SORTING TECHNIQUE

Below there are some tables representing the calculated running time for n input values and their respective graphs in best case, average case and worst case.

a) Best Case:

Table 1. Best Case Time Complexity

Best Case				
	Insertion	Quick	Shell	FPSTUN
10	10	10	10	100
50	50	84.94	144.32	2500
100	100	200	400	10000
150	150	326.41	710.30	22500
200	200	460.20	1058.94	40000

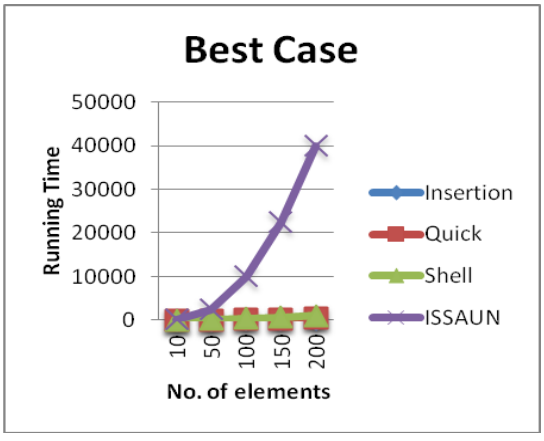

Fig 8: Best Case

b) Average Case:

Table 2. Average Case Time Complexity

Average Case				
	Insertion	Quick	Shell	FPSTUN
10	100	10	17.78	100
50	2500	84.94	132.95	2500
100	10000	200	316.22	10000
150	22500	326.41	524.94	22500
200	40000	460.20	752.12	40000

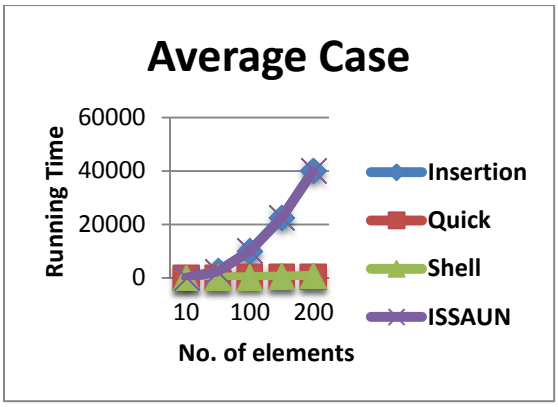

Fig 9: Average Case

c) Worst Case:

Table 3. Worst Case Time Complexity

Worst Case				
	Insertion	Quick	Shell	FPSTUN
10	100	100	31.62	100
50	2500	2500	353.55	2500
100	10000	10000	1000	10000
150	22500	22500	1837.11	22500
200	40000	40000	2828.42	40000

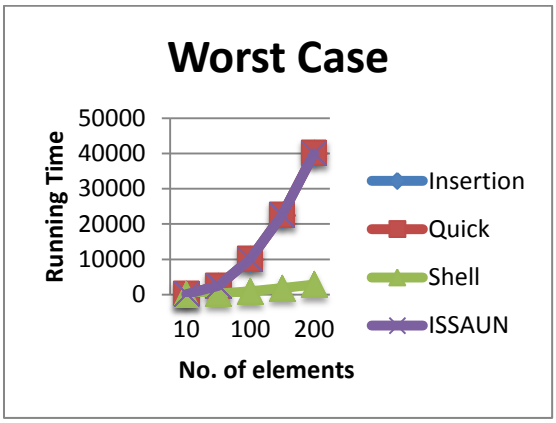

Fig 10: Worst Case

From the above graphs for various cases complexities one can easily say that FPSTUN takes same time like selection sorting and FPSTUN is easier to implement as compare to selection sorting and in worst case it takes same time like other sorting algorithms except quick sort technique.

5. CONCLUSION

Logic of FPSTUN is based on the logics of well known sorting algorithms like selection sorting technique and insertion sorting technique. Hence implementation is very much easier than those two sorting techniques. And also we can easily understand the concept as we don't do any swapping in this technique. In future we will do it for duplicate numbers also.

6. REFERENCES

[1]. Hore, C.A.R. "Algorithm 64: Quick sort". Comm.ACM 4,7(July 1961), 321.

[2]. Soubhik Chakraborty, Mousami Bose, and Kumar Sushant, A research thesis, On way Parameters of input Distributions Need be Taken Into Account For a more precise Evalution of complexity for certain Algorithms.

[3]. J.L. Bentley and R. Sedgewick. "Fast Algorithm for Sorting and Searching Strings" ACM-SIAM SODA 97, 360-369,1997.

[4]. D.S.Malik, C++ Programming: Program Design including Data Structures, Course Technology (Thomson Learning), 2002, www.course.com

[5]. D.Jim enez-Gonz'alez, J. Navarvo, and Larriba-Pay. CC-Radix: "A catch conscious sorting based on Radix sort". In Euromicro Conference on Parallel Distributed and Network based Processing. Pages 101-108, February 2003.