REVIEW ON SORTING ALGORITHMS
A comparative study on two sorting algorithms

By
Pooja Adhikari

A Term Paper
Submitted to the Faculty of Dr. Gene Boggess
Mississippi State University
In the Department of Computer Science & Engineering

Mississippi State, Mississippi

04 2007

ABSTRACT

Any number of practical applications in computing requires things to be in order. The performance of any computation depends upon the performance of sorting algorithms. Like all complicated problems, there are many solutions that can achieve the same results. One sort algorithm can do sorting of data faster than another. A lot of sorting algorithms has been developed to enhance the performance in terms of computational complexity, memory and other factors. This paper choose two of the sorting algorithms among them selection sort and shell sort and compares the various performance factor among them.

1. INTRODUCTION

Sorting is the rearrangement of things in a list into their correct lexicographic order. A number of sorting algorithms have been developed like include heap sort , merge sort, quick sort, selection sort all of which are comparison based sort .There is another class of sorting algorithms which are non comparison based sort. This paper gives the brief introduction about sorting algorithms [2] where it discuss about the class of sorting algorithms and their running times. It mainly analyses the performance between two sorting algorithms .Selection sort [3] and Shell sort [5].
Selection sort is the simple sorting method with a very simple sorting algorithm [3]. However the running time of this algorithm is not that optimal as compared to performance of best sorting algorithm .The analysis of the running time [4] concludes this thing.

Shell sort is another comparison based sorting algorithm which is better than selection sort [5] and works in a gap sequence [7]. The running time and the performance analysis [6] shows that although its performance is better than the selection sort, there are other various factor that needs to be keep in mind.

The advantages of selection sort contain within it a lot of disadvantage and so are shell sort [8].However there are equally important and is best for a particular purpose.

2. SORTING

Sorting is one of the basic operations that are performed by many computers since handling the data in a certain order is more efficient than handling the randomized data.

There are different kinds of sorting algorithms .Two common sorting algorithms among them are: Selection Sort and Shell Sort

3. SELECTION SORT

Selection sort is one of the simplest algorithms. It is actual an improvement in performance of bubble sort. This algorithm is called selection sort because it works by selecting a minimum element in each step of the sort. This algorithm, iterating through a list of n unsorted items, has a worst-case, average-case, and best-case run-time of $\Theta(n^2)$, assuming that comparisons can be done in constant time. The Selection sort spends most of its time trying to find the minimum element in the "unsorted" part of the array. The brief algorithm for selection sort is given below.

SELECTION_SORT (A)	Cost of Each Step
1. for i ← 0 to n-1 do	O(n)
2. min ← i;	O(1)
3. for j ← i + 1 to n do	$O(n^2)$
4 If A[j] < A[min] then	O(1)
5. min←j	O(1)
6. swap(A[i] ,A[min])	O(1)

Total run time Cost== $\Theta(n^2)$ asymptotically

It works as follows:

1. Find the smallest element using a linear scan.
2. Swap the element with the element in first position.
3. Find the second smallest element in the remaining array.
4. Then swap to the second position.
5. Continue the steps until a sorted list is obtained.

4. PERFORMANCE ANALYSIS OF SELECTION SORT

For each value of element until n it needs to compare and check whether there is any value smaller than it and needs (n-1) comparisons. Thus it can be easily seen that run time complexity of selection sort is $\Theta (n^2)$ due to the line no.3.But since it just requires the Θ (n) swaps means linear order writes to the memory which is optimal to any sorting algorithm .

Table 1. further clarifies the above statement. The running time is obtained by calculating how many times the line 4 of the above algorithm got executed on the specified array size.

Number of Array Elements	Running Time
8	28
16	120
32	496
64	2016
128	8128
256	32640

Table 1. Running time of selection sort

From the above table it is clear that this naive approach is very good for small number of elements. But it requires the comparison of every element to every other element which will lead to a great deal of work for large data sets.

The worst case occurs if the array is already sorted in descending order. Nonetheless, the time require by selection sort algorithm is not very sensitive to the original order of the array to be sorted: the test "if A[j] <A[min]" is executed exactly the same number of times in every case.

5. SHELLL SORT

Shell sort is yet another comparison based sort whose worst case running time is $\Theta(n^2)$. It was invented by Donald Shell in 1959. It is an insertion sort working over the gap sequence. Since insertion sort is efficient if the input is "almost sorted" and it is inefficient, on average, because it moves values just one position at a time, shell sort is using the advantages and removing the disadvantage of insertion sort.

Shell sort performs insertion sort on elements separated by a gap of several positions. This lets an element take "bigger steps" toward its expected position.

For Example: Let us suppose that we have a sequence of 15 numbers to be sorted

15 3 8 12 9 7 1 6 4 13 10 2 5 11

Let us first do the sorting between elements which are in gap 4 to each other.

So in First Step

15 9 4 5 will be sorted

3 7 13 11 will be sorted

8 1 10 will be sorted

12 6 2 will be sorted

The result of this step: 4 3 1 2 5 7 8 6 9 11 10 12 15 13

Similarly in second Step elements which are in gap less than the above are sorted. The final step is when the gap is 1.It is then a native insertion sort. Since the elements will be almost sorted till this point it will not take much time Thus the worst case condition of insertion sort is avoided.

This method can be complex and inefficient to large amount of data sets, it is much faster than the insertion sort for sorting small number of elements (less than 1000 or so elements).

After the first step the data sequence will be partly sorted and this will increase in the subsequent steps. The algorithm of shell sort is given below:

SHELLSORT (A)

1. step←m;
2. while step > 0

```
3.              for (i←0 to n with increment 1)
4.               do  temp←0;
5.               do     j←i;
6.                      for (k←j+step to n with increment step)
7.                      do      temp←A[k];
8.                      do      j←k-step;
9.                              while (j>=0 && A[j]>temp)
10.                             do      A[j+step]=A[j];
11.                             do        j←j-step;
12.                     do      Array[j]+step←temp;
13.         do   step←step/2;
```

The main focus of the shell sort lies in line 6. Line 6-12 is algorithm for insertion sort. But insertion sort is performed between the elements between some specified gaps. Since the sorted order of elements get increased with the completion of each step these lines gives a constant value less than the value of n. Thus in the best case of this is almost O(n).But there may be the worst case condition for large number of inputs where computational complexity where the gap insertion sort can give almost n steps resulting into running time of $\Theta(n^2)$.

6. PERFORMANCE ANALYSIS OF SHELL SORT

The following table gives the running time of shell sort .It is obtained by running the above algorithm in the different input elements. Running time is obtained by counting the number of times line 10 and 11 got executed.

Number of elements	Running Time
8	10
16	38
32	125
64	386
128	1682
256	6179

Table 2. Running time of Shell sort for the various inputs

Thus from the above table it is clear that it is not exactly Θ (n^2). The increment in the performance of shell sort is because it tries to implement the best part of insertion sort. After each gap step the array elements will get partly sorted which makes the performance increase for the shell sort.

7. GAP SEQUENCE

The gap sequence is an integral part of the shell sort algorithm. Any increment sequence will work, so long as the last element is 1. The algorithm begins by performing a gap insertion sort, with the gap being the first number in the gap sequence. It continues to perform a gap insertion sort for each number in the sequence, until it finishes with a gap. The gap value should be decreased when a one phase of insertion sort on elements on the gap.

8. PERFORMANCE COMPARISION

Selection sort and Shell sort are both comparisons based sorting algorithms which uses two different perspectives. One operates by selecting smallest element of the unsorted part of the list each time. Another operates by performing insertion sort in a gap sequence. The following graph compares the running time of these algorithms based on Table 1 and Table 2.

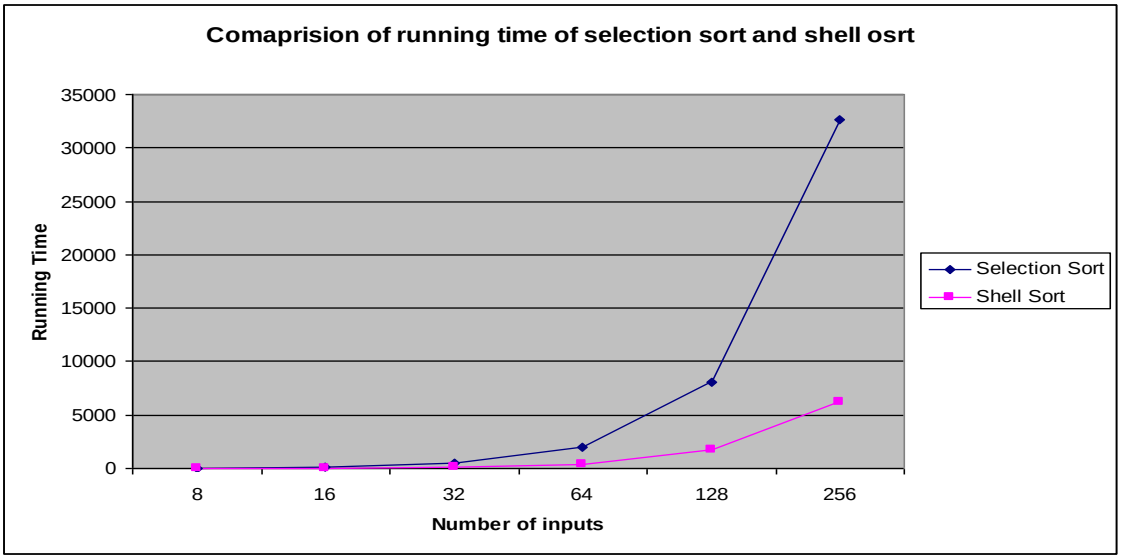

Fig: Running time of selection sort and shell sort

From the above analysis also it is clear that performance of selection sort is worse than shell sort. Selection Sort is fairly simple algorithm and can be easily implemented. The algorithm of shell sort is rather complex and it will give very poor running time in case of large sets of data. But the performance of both algorithms is worse than the lower bound run time of comparison sort O (nlogn).These algorithms can be implemented for small amount of data but not for large amount of data.

8. CONCLUSION

This paper discuss two comparison based sorting algorithms .It analyses the performance of these algorithms for the same number of elements .It then concludes selection sort shows the poor performance than shell but being the simple structure selection sort is more preferred and widely used. Although shell sort gives the better performance because of its complicated structure it cannot be used for large arrays. Selection sort is also not that popular for the large arrays. Both have the upper bound running time O (n^2).

9. REFERENCES

[1] Thomas H. Cormen,Charles E. Leiserson,Ronald L. Rivest,Clifford Stein,*Introduction to Algorithms*,Second Edition,Prentice-Hall New Delhi,2004

[2] Ananth Grama,Anshul Gupta,George Karypis,Vipin Kumar,*Intoduction to Parallel Computing*,Second Edition,Addison-Wesley

[3] Wikipedia, *Sorting Algorithm*, http://en.wikipedia.org/wiki/Sorting_algorithm, Current: 2007-04-24

[4] Michael Lamont,*Sorting Algorithms*, http://linux.wku.edu/~lamonml/algor/sort/sort.html, Cuttent :2007-04-24

[5] Wikipedia, *Selection Sprt*, http://en.wikipedia.org/wiki/Selection_sort , Current: 2007-04-24

[6] Wikipedia, *Shell Sort*, http://en.wikipedia.org/wiki/Shell_sort , Current: 2007-04-24

[7] Robert Lafore,Data Structures and Algorithmsin Java,Second Addition,2002.