Mirza Abdulla, International Journal of Computer Science and Mobile Applications,
Vol.3 Issue. 12, December- 2015, pg. 15-21 **ISSN: 2321-8363**

Marrying Inefficient Sorting Techniques Can Give Birth to a Substantially More Efficient Algorithm

Mirza Abdulla, Ph.D.,
Computer Science Department, AMA international University Bahrain, mirza999@hotmail.com

Abstract

The classical sorting techniques such as Selection Sort and Insertion sort have a poor worst case time performance of $O(n^2)$. However, these techniques can be useful for relatively small data sets. In this paper we look at the interesting case of applying the ideas of both techniques on a set of *n* elements. We show that it is possible to use both techniques together to produce a more efficient hybrid sorting algorithm. This hybrid algorithm runs in $O(n\sqrt{n})$ and therefore, improves the worst case time efficiency by a factor of $\sqrt{n}$. The asymptotic time performance of the hybrid algorithm is the same for the best, average and worst case scenarios.

Keywords: sorting algorithms, Selection sort, Insertion sort, Worst case performance.

1. Introduction

Sorting is one of the most performed operations in computing. Sorting problem and algorithms were known for quite some time. Probably one of the most well known of the oldest sorting algorithm is that of the Hollerith sorter [Anon]. One of the best references to this subject is [Knuth]. There are many sorting algorithms with some being more appropriate for certain tasks. For example, some of the criteria for the choice of a good sorting algorithm is how adaptive is the algorithm is (for nearly sorted data), or its stability (changing of relative order of elements with equal keys), or whether it can sort in situ (in place) or not, as well as, of course, how efficient the algorithm is. Two of the most well known sorting algorithms are Selection sort and Insertion sort. They are usually taught in first undergraduate courses on data structures and algorithms and have been well studied in the computer literature with many variations as indeed they provide a tantalizing contrast between theoretical and empirical analysis. Both techniques work on having two sets: one sorted and the other unsorted set. Selection sort works by starting with an initially empty sorted set and the rest of the data in the unsorted list. We repeatedly select the minimum of the unsorted list and append it to the end of the sorted list until the unsorted list becomes empty. Insertion sort on the other hand starts with a sorted list with a single element initially, and the rest of the data being in the unsorted lists. We repeatedly take an element from the unsorted list and insert it in the right position in the sorted list, moving data element when necessary to make room for the newly inserted data item. This process is repeated until the unsorted list is empty. Both techniques sort in situ and take $O(n^2)$ time to sort *n* data items in the worst case. These sorting techniques are not considered as efficient sorting techniques since one can show that sorting of *n* elements by comparison can be performed in $\Theta(nlogn)$ time in the worst case using for example Heap sort or Merge sort.

In this paper we study the effect of combining ideas from both techniques and analyze the performance of this hybrid technique. We show that combining the ideas of selection and insertion sorts gives an improved in situ sorting algorithm that can sort in $O(n\sqrt{n})$ time which provides a significant improvement over the performance of both sorting algorithms.

2. Analysis of Selection and Insertion Sort

I. SELECTION SORT.

Selection Sort and as mentioned earlier is one of the simplest sorting techniques and is useful when data is small. It requires O(n^2) comparisons in the worst case, but O(n) data movement in the worst case, making it an appropriate sorting technique for small datasets but with large size of records. It repeatedly finds the minimum of remaining unsorted items and swaps that element with the j^{th} element on the j^{th} iteration, j=1, 2, 3,…, n-1.

Pseudo code

Algorithm SelectionSort (X, n)

X[1..n]

S. No	statement	Cost	Worst Case Times
1	***for*** $i \leftarrow$ 1 TO n-1	$C1$	n
2	$iMin \leftarrow i$	$C2$	$n-1$
3	***for*** $j \leftarrow i$ +1 TO n	$C3$	$\sum_{j=i+1}^{n}(1+\sum_{i=1}^{n-1}1)=\frac{n(n+1)}{2}$
4	***if*** (X[j]>X[$iMin$])	$C4$	$\sum_{j=i+1}^{n}(\sum_{i=1}^{n-1}1)=\frac{n(n-1)}{2}$
5	$iMin \leftarrow j$	$C5$	$\sum_{j=i+1}^{n}(\sum_{i=1}^{n-1}1)=\frac{n(n-1)}{2}$
6	$min \leftarrow$X[$iMin$]	$C6$	$n-1$
7	X[$iMin$] $\leftarrow$ X[i]	$C7$	$n-1$
8	X[i] $\leftarrow$ min	$C8$	$n-1$

Note: in steps 1 and 3 we add an extra comparison step for the case when the test to exit the do loop is true.

Analysis of Worst - Case Time Complexity

The overall cost of operations performed in the worst case occurs when we always perform step 5 (when the original data set is already sorted by in reverse order). In such case we have:

$$T(n) = C1*n + C2*(n-1) + C3*\frac{n(n+1)}{2} + C4*\frac{n(n-1)}{2} + C5*\frac{n(n-1)}{2} + (C6+C7+C8)(n-1)$$

Rearranging the terms we get

$$T(n) = C3*\frac{n^2}{2} + C4*\frac{n^2}{2} + C5*\frac{n^2}{2} + \left(C2 + \frac{C3}{2} + \frac{-C4}{2} + \frac{-C5}{2} + C6 + C7 + C8\right)*n - (C2 + C6 + C7 + C8)$$

Thus we have

$$T(n) = an^2 + bn + c = O(n^2)$$

The best case scenario on the other hand occurs when the condition on step 4 is always false and thus step 5 is never executed. In such a case we have.

$$T(n) = C1*n + C2*(n-1) + C3*\frac{n(n+1)}{2} + C4*\frac{n(n-1)}{2} + C5*(0) + (C6+C7+C8)(n-1)$$

Rearranging the terms we get

$$T(n) = C3*\frac{n^2}{2} + C4*\frac{n^2}{2} + + \left(C2 + \frac{C3}{2} + \frac{-C4}{2} + C6 + C7 + C8\right)*n - (C2 + C6 + C7 + C8)$$

Thus we have

$T(n) = an^2 + bn + c = O(n^2)$ in the best case as well.

From the above results one can easily conclude that selection sort requires $O(n^2)$ operations in the average case too.

II. INSERTION SORT.

Insertion Sort

Insertion sort algorithm is also one of oldest, easiest and most useful sorting algorithms for dealing with modicum of data set. If the first few objects are already sorted, an unsorted object can be inserted in the sorted set in proper place [10].

B. Pseudo Code & Execution Time of Individual Statement of Insertion Sort

Algorithm InsertionSort (X, 1, n)

Given *X*[1..n]

S.No	Iteration	Cost	Worst Case Times
1	***for*** $j \leftarrow$ 2 TO length[X] {	$C1$	n
2	$key \leftarrow X[j]$	$C2$	$n-1$
3	// Put X[j] into the sorted sequence X[1 . . $j-1$]		
4	$i \leftarrow j-1$	$C3$	$n-1$
5	***while*** $i > 0$ and X[i] > key {	$C4$	$\sum_{j=2}^{n}(1+\sum_{i=1}^{j-1} t_i) = n-1+\frac{n(n-1)}{2}$
6	X[i +1] $\leftarrow$ X[i]	$C5$	$\sum_{j=2}^{n}\sum_{i=1}^{j-1} t_i = \frac{n(n-1)}{2}$
7	$i \leftarrow i-1$ }	$C6$	$\sum_{j=2}^{n}\sum_{i=1}^{j-1} t_i = \frac{n(n-1)}{2}$
8	X[i + 1] $\leftarrow$ key }	$C7$	$n-1$

Analysis of Worst - Case Time Complexity

The number of iterations is maximum when step is TRUE for all values of $i > 0$, and thus the term t_i (which can be either 0 or 1 depending on whether the statement is true or not) will always be 1 for these values of i. in such a case we have

$$T(n) = C1*n + C2*(n-1) + +C3*(n-1) + C4*\left(n-1+\frac{n(n-1)}{2}\right) + C5*\frac{n(n-1)}{2} + C6*\frac{n(n-1)}{2} + C7*(n-1)$$

Rearranging the terms we get

$$T(n) = (C4+C5+C6)*\frac{n^2}{2} + \left(C1+C2+C3+\frac{C4}{2}-\frac{C5}{2}-\frac{C6}{2}+C7\right)*n - (C2+C3+C4+C7)$$

Thus we have

$$T(n) = an^2 + bn + c = O(n^2)$$

The number of iterations is minimum when step is FALSE for all values of $i>0$, and thus the term t_i (which can be either 0 or 1 depending on whether the statement is true or not) will always be 0 for these values of i. in such a case we have

$$T(n) = C1*n + C2*(n-1) + +C3*(n-1) + C4*(n-1) + C5*(0) + C6*(0) + C7*(n-1)$$

Thus

$$T(n) = (C1+C2+C3+C4+C7)*n - (C2+C3+C4+C7)$$

From which one can conclude that

$$T(n) = a*n + b = O(n)$$

However, the average case analysis yields a $O(n^2)$ time bound, but we will omit the proof.

3. Hybrid InsertionSelection Sort Technique

Our technique of marrying both the selection and insertion sort ideas can be summed up as follows:

1. Given an array *X* with *n* data element indexed from 1 to *n*, one can view the array as consisting of $\sqrt{n}$ smaller adjacent arrays *X*[1.. $\sqrt{n}$], *X*[1+$\sqrt{n}$..2$\sqrt{n}$], …, *X*[*n*+1-$\sqrt{n}$..*n*] as shown in Figure 1. we shall refer to these sub-arrays as: $X_1, X_2, \ldots, X_i, \ldots, X_{\sqrt{n}}$.

2. Sort each sub-array X_i , $1 \le i \le \sqrt{n}$, using insertion sort.
3. For each element *X*[*j*], *j* = 1, 2, 3, …, *n*, of the array *X*, perform the following variant of Selection sort:

   If *X*[*j*] falls in the sub-array X_k, then for all sub-arrays X_{k+1} … $X_{\sqrt{n}}$ perform selection of minimum of all of these sub-arrays as follows:

   i. Find the minimum, and its location of only the first elements of each sub-array
   ii. Compare this minimum and the element *X*[*j*], and if this minimum is less then swap with X[*j*], in which case we perform insertion sort iteration on the sub-array from which the element *X*[*j*] was moved to in order to make sure that the sub-array remains sorted.

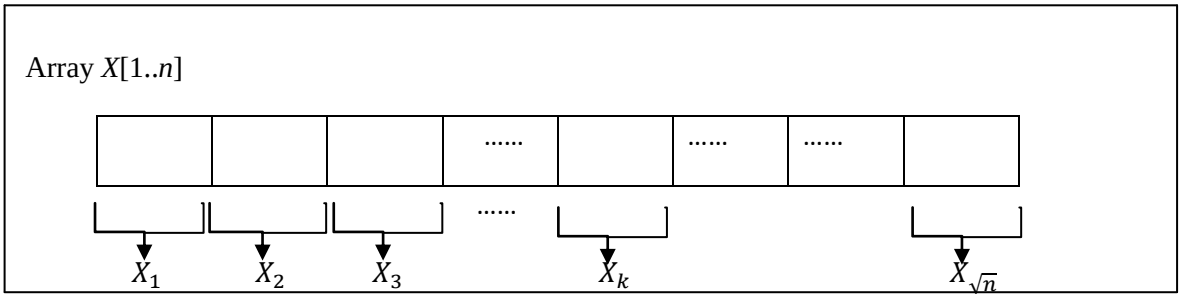

Figure 1

Lemma 1. The hybrid technique sorts the array *X* in situ.

Proof.

The correctness of the technique follows by induction:

The first position of the array *X* will hold the smallest of the elements in the first position of all the sub-arrays. Since each sub array was already sorted using insertion sort, it follows that the first position of each sub-array contains the smallest element in that sub-array and hence the first position of *X* will contain the smallest element in the whole array.

Now assume that we are currently in the *j*+1st iteration of step 3 and all the elements up to and including the *j*th element of array *X* are indeed the smallest *j* elements of the array and are in sorted order.

Let $k = \left\lceil \frac{j+1}{\sqrt{n}} \right\rceil$ be the value of the quotient of division of *j*+1 over $\sqrt{n}$ with the fraction rounded up to the next integer, and let $i = j + 1 - (k-1) * \sqrt{n}$.

Thus the *j*+1st of the array *X* is also the *i*th element of the X_k sub-array.

Step 3.i performs the operation of selection sort by finding the minimum of the elements in locations *j*+1..*n*. it does so by finding the smallest of the smallest element of the remaining sub-arrays $X_{k+1}..X_{\sqrt{n}}$ and comparing that smallest value with the current element in (*j*+1) position of the array *X* to place the smallest in that location. Thus location *j*+1 of the array *X* will hold the smallest *j*+1st element of the array after the iteration is completed.

It follows therefore that the technique does indeed sort the array X in place. QED

From the discussion above one can see that we avoided the selection of the minimum of the unsorted data items in the array by selecting only the minimum element of the smallest value of the sub-arrays. Therefore, now we need only to inspect $\sqrt{n}$ elements of the array X to find the minimum of the remaining elements of the array. It is here where we gain the benefit of reducing the asymptotic time complexity of the classical Selection sort by a factor of$\sqrt{n}$.

The Hybrid Algorithm

Given array $X[1..n]$ of integer values

S#	statement	cost	times
1	for i = 1 to $\sqrt{n}$ {	$C1$	$1+\sqrt{n}$
2	$lb = (i\text{-}1)* \sqrt{n} +1$	$C2$	$\sqrt{n}$
3	$ub = (i)* \sqrt{n}$	$C3$	$\sqrt{n}$
4	InsertionSort (X [lb.. ub]) }	$C4$	$\sqrt{n}(\sqrt{n})^2 = n\sqrt{n}$
5	for i = 1 to n-1 {	$C5$	n
6	$j = \lceil i/\sqrt{n} \rceil$	$C6$	n-1
7	$lb = j* \sqrt{n} +1$	$C7$	n-1
8	min = X [lb]	$C8$	n-1
9	$minj = lb$	$C9$	$n-1$
10	$k = minj$	$C10$	n-1
11	while ($k < -\sqrt{n}$) {	$C11$	$1+(n-1)\sqrt{n}$
12	if X [k] < X [$minj$]	$C12$	$(n-1)\sqrt{n}$
13	$minj$= k	$C13$	$(n-1)\sqrt{n}\ t_k$
14	$k = k + \sqrt{n}$}	$C14$	$(n-1)\sqrt{n}$
15	if X [$minj$] < X [lb] {	$C15$	$n-1$
16	swap(X [k], X [$minj$])	$C16$	$(n-1)\ s_k$
17	InsertionSort (X [$minj$ .. $minj$ - 1+$\sqrt{n}$]) }	$C17$	$(n-1)\sqrt{n} * s_k = n\sqrt{n}\ s_k$
18	i=i+1 }	$C18$	$n-1$

Analysis of the hybrid algorithm.

Steps 1 to 4 of the algorithm sort the sub-arrays $X_1, X_2, \ldots, X_i, \ldots, X_{\sqrt{n}}$ of the array X. each sub array consists of only $\sqrt{n}$ data items. Thus step 4 would take $C4*(\sqrt{n})^2$ to sort each sub array, from which it follows that the overall time for step 4 is $C4* n\sqrt{n}$. In steps 5 to 8 we iterate over the array X starting from the first data element up to the last. In step 6 we find the sub-array the current element belongs to, and steps 7 and 8 we find the lower and upper bounds of the next adjacent sub-array. In steps 9 to 15 we select the minimum of the remaining unsorted data items of X by only going thru the first element of each sub-array X_i. We compare that element with item currently under consideration and swap the values if the current value in location j of the array X is not the minimum. In step 17 we make sure that the sub-array from which the swapped item came from remains sorted by calling InsertionSort again. Such a step can only take $C17*\sqrt{n}$ time in the worst case in, since the sub-array was sorted before and now we have at most one item of the sub-array not in its right position. Moreover, since we iterate over steps 5 to 18 n-1 times the extra factor of n-1 for each of these steps follows.

Note that the when steps 13 or 15 and 16 are performed the value of t_k and s_k are equal to 1, and 0 otherwise. Thus we have as a worst case time complexity:

$$T(n) = C1*(1+n) + C2*(\sqrt{n}) + +C3*(\sqrt{n}) + C4*(n\sqrt{n}) + C5*(n+1) + + (C6+C7+C8+C9+C10+C15+C16+C18)*(n-1) + C11*(1 + (n-1)\sqrt{n}) + (C12+C14)*(n-1)\sqrt{n} + C13*(n-1)\sqrt{n} + C17*n\sqrt{n}$$

Rearranging we get

$$T(n) = (C4+C11+C12+C13+C14+C17)*n\sqrt{n} + (C5+C6+C7+C8+C9+C10+C15+C16+C18)*n + (C2+C3-C11-C12-C13-C14)*\sqrt{n} + (C1+C5+C11-C6-C7-C8-C9-C10-C15-C16-C18)$$

Thus $T(n) = an\sqrt{n} + bn + c\sqrt{n} + d)$

Where $a = (C4+C11+C12+C13+C14+C17)$

$b = (C5+C6+C7+C8+C9+C10+C15+C16+C18)$

$c = (C2+C3-C11-C12-C13-C14)$

And $d = (C1+C5+C11-C6-C7-C8-C9-C10-C15-C16-C18)$

It follows therefore, that $T(n) = an\sqrt{n} + bn + c\sqrt{n} + d) = O(n\sqrt{n})$ giving the substantial improvement of a factor of $\sqrt{n}$ over the performance of either insertion sort or selection sort.

Best case performance of the Hybrid algorithm

The best case time complexity performance of the algorithm occurs when steps 13 or 15 and 16 are never executed the value of t_k and s_k are then equal to 0. This occurs when the data in array *X* were already in required sorted order. In such a case step 4 would take $C4*n$ time, and as mentioned earlier, steps 13, 15, and 16 are never executed. However, the complexity remains $O(n\sqrt{n})$ since steps 11, 12, and 14 are executed that many times. This follows from the fact that Selection sort would perform $O(n^2)$ time in the best case too.

Thus we have:

Theorem. The Hybrid Selection Insertion algorithm best, average, and worst case performance are all$O(n\sqrt{n})$.

4. Conclusion

In this paper we presented an algorithm built using the ideas of classical selection and insertion sorting. We proved that the new hybrid algorithm has a worst, best, and average case time complexity of $O(n\sqrt{n})$. The idea was to reduce the time it takes selection sort to find the minimum element from $O(n)$ to just $O(\sqrt{n})$. This was achieved by treating the input array as constituting of $\sqrt{n}$ sorted sub-arrays each of size $\sqrt{n}$. The use of sub-arrays reduced the time a data item is moved from its current position to its correct position from n in the classical insertion sort technique to just $\sqrt{n}$ places in the hybrid technique.

References

[1]Anon., 1891, Untitled reports on Hollerith sorter, J. American Stat. Assoc. 2, 330-341

[2]Shaw M., 2003, Writing good software engineering research papers, *In Proceedings of 25th International Conference on Software Engineering*, pp.726-736

[3] Flores I. ,Oct 1960, "Analysis of Internal Computer Sorting". J.ACM 7,4, 389- 409.

[4] Knuth, D., 1998"The Art of Computer programming Sorting and Searching", 2nd edition, vol.3. Addison- Wesley.

[5] Soubhik Chakraborty, Mausumi Bose, and Kumar Sushant, A Research thesis, On Why Parameters of Input Distributions Need be Taken Into Account For a More Precise Evaluation of Complexity for Certain Algorithms.

[6] D.S. Malik, 2002, C++ Programming: Program Design Including Data Structures, Course Technology(Thomson Learning), www.course.com.
[7] J. L. Bentley and R. Sedgewick, 1997, "Fast Algorithms for Sorting and Searching Strings", ACM-SIAM SODA ˝97, 360-369.
[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, 2001,."Introduction to Algorithms". MIT Press, Cambridge, MA, 2nd edition.
[9] Flores, I., Jan 1961, "Analysis of Internal Computer Sorting". J.ACM 8, 41-80.
[10] V.Estivill-Castro and D.Wood, 1992,"A Survey of Adaptive Sorting Algorithms", Computing Surveys, 24:441-476.
[11] Cocktail Sort Algorithm or Shaker Sort Algorithm, http://www.codingunit.com/cocktail-sort-algorithm-or-shaker-sort-algorithm.
[12] S. Jadoon, S. F. Solehria, S. Rehman and H. Jan, "Design and Analysis of Optimized Selection Sort Algorithm", International Journal of Electric & Computer Sciences (IJECS-IJENS), vol. 11, no. 01, pp. 16-22.
[13] S. Chand, T. Chaudhary and R. Parveen, 2011, "Upgraded Selection Sort", International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397, vol. 3, no. 4, pp. 1633-1637.
[14] J. Alnihoud and R. Mansi, 2010, "An Enhancement of Major Sorting Algorithms", International Arab Journal of Information Technology, vol. 7, no. 1, pp. 55-62.
[15] O. O. Moses, 2009, "Improving the performance of bubble sort using a modified diminishing increment sorting", Scientific Research and Essay, vol. 4, no. 8, pp. 740-744.
[16] H. W. Thimbleby, 1989, "Using Sentinels in Insert Sort", Software-Practice and Experience, vol. 19, no. 3, pp. 303-307.
[17] T. S. Sodhi, S. Kaur and S. Kaur, 2013, "Enhanced Insertion Sort Algorithm", International Journal of Computer Applications, vol. 64, no. 21, pp. 35-39.

A Brief Author Biography

Mirza Abdulla – A full time faculty member of the Computer Science Department in the AMA International University, Bahrain.

