International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org, editorijaiem@gmail.com
Volume 2, Issue 5, May 2013 **ISSN 2319 - 4847**

Improving the performance of selection sort using a modified double-ended selection sorting

SurenderLakra[1], Divya[2]

[1&2]Haryana Institute of Technology, Haryana

ABSTRACT

Borrowing ideas from single selection sort, we propose a new selection sorting technique for double-ended selection sort. Both theoretical analysis and coding explanation show that the proposed sorting improves the performance of selection sort All the code of elementary sorting techniques are in the textbook, easily found on the web, but the use of "double sorting" is not available and offer speed improvement over the normal selection sort. With double-ended selection sort, the average number of comparison is slightly reduced. Here, idea of using two chosen elements simultaneously, applies to "selection sort", through possibility of enhance speed up 25% to 35%. Code for this sorting is written in dozen lines of C++ Code. So, easily within the reach of beginners who understand the basic concept of this new sorting. In addition, the concept is also cleared more how the N² sorts can be improved towards the NlogN sort. However, efforts have been made to improve the performance of selection sorting and implication will be faster.

Keywords: Algorithm, Selection and Double-Ended Selection Sort, Comparison, Swapping.

1. INTRODUCTION

To be a good programmer, an excellent programming practice must be done but this can possible if we know about related topic theoretically as well as practically. In starting when students learn to make program with array, they do exercise relevant to selection sort, and then by the help of required coding convert into code on their own. Many sorting algorithm we did in lab by the help of instructor and over there instructor work was always check students who grabbed the code off the web but effective coding can be possible only when we do own self. Sorting is a data structure operation, which is used for making searching and arranging of element or record. Here arrangement of sorting involves either into ascending or descending order. Everything in this world has some advantage and disadvantage, some sorting algorithms are problem specific means they work well on some specific problem not all the problem. It saves time and help searching data quickly. Sorting algorithm performance varies on which type of data being sorted, not easier to say that which one algorithm is better than another. Here, performance of different algorithm is according to the data being sorted [1]. Examples of some common sorting algorithms are the exchange or bubble sort, the selection sort, the insertion sort and the quick sort.

The Selection sort is a good one that use for finding the smallest element in the array and put in the proper place. Swap it with the value in the first position. Repeat until array is sorted (starting at the second position and advancing each time). It is very intuitive and simple to program, offer quit good performance for particular strength being the small number of exchanges needed. The selection sort always goes through a set number of comparisons, for a given no of data items [2]. But sometime question raise in front of us, is there any way through this sorting can be more effective and how to convert that algorithm into code. Then demonstrate a modification of this algorithm, and finally to assign the coding modification as a programming. This paper suggests one simple modification of sorting algorithm: Double Selection Sort. Some one can argue that the use of double sort for these small array partitions will provide a improvement to this critical algorithm.

Since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. For example, bubble sort was analyzed as early as 1956[3]. Although many consider it a solved problem, useful new sorting algorithms are still being invented (for example, library sort was first published in 2004). Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts, such as big O notation, divide and conquer algorithms, data structures, randomized algorithms, best, worst and average case analysis, time-space tradeoffs, and lower bounds.

To understand significant concepts and programming practice, a good programming exercise plays an important role i.e. for using double-ended selection sort instead of normal selection sorting technique that enhance the sorting skills. Well, there are two cases occurred at the time of making double sorting code for selection sort, when the size is odd or even.

An effort is done in positive direction and realizes coding technique for double sorting offer substantial improvements speed up to 25% to 35% over the single selection sorting [10].

2. SELECTION SORT

The selection sort is used for sorting the smallest unsorted item present in the given list, and then swaps it with item to fill the next position. The complexity of selection sort is O (N^2) and this sorting behaves like unwanted stepchild. It gives 60% better improved performance over the bubble sort, but in case of insertion sort acts twice as fast as the bubble sort and easy to implement same as the selection sort [6]. In short, if we want to use the selection sort for some reason we should implement selection sort instead of using other sorting like insertion sort. This sorting is helpful for small sorting lists; try to avoid more than 1000 items with it or repetitively sorting lists of more than a couple hundred items.

In pseudo code, the selection sort is described as [9]:

```
Set interchange count to zero (not required, but done just to keep track of the interchanges).
   For each element in the list from first to next- to-last,
     Find the smallest element from the current element being referenced to the last element by:
       Setting the minimum value equal to the current element
        Saving (sorting) the index of the current element.
          For each element in the list from the current element+1 to the last element in the list,
           If element [inner loop index] <minimum value,
            Set the minimum value = element [inner loop index].
           Save the index of the new found minimum value.
         End If.
       End For
     Swap the current value with the new minimum value.
   Increment the interchange count.
  End for.
Return the interchange count.
```

The coding for selection sorting algorithm, specifically an in place comparison sort. Here, large list making it inefficient because of its *O* (N^2*)* complexity. It generally performs worse than the similar insertion sort. The main purpose of using selection sort just for simplicity, and also has performance advantages over more complicated algorithms in certain situations. If loops depend on the data in the array, then it creates ambiguity to analyze compared to other sorting algorithms. Selecting the lowest element requires scanning all n elements (this takes $n - 1$ comparisons) and then swapping it into the first position. Finding the next lowest element requires scanning the remaining $n - 1$ element and so on, for $(n - 1) + (n - 2) + ... + 2 + 1 = n (n - 1) / 2$ ı (n2) comparisons. Each of these scans requires one swap for $n - 1$ elements (the final element is already in place). Selection sort's philosophy most closely matches human intuition. In other words, it also finds the largest element and put it in its place. Then search for next largest item and places it specific place and so on until the whole array is sorted. Arrange that element in its place; it trades positions with the element in that location. In last, the array will have a section that is sorted growing from the end of the array and take further step for the rest of the array that remained unsorted [4].

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.

It's clear that number of moves depend on the initial order of the values in the list .Because of simple and easier to implement selection sorting make number of moves maximum that must be made is N-1, where N is the no of elements in the list. Further, each move is a final move that results in an element residing in its final location in the sorted list. The main concept of selection sort is that N (N-1) comparisons are always required, regardless of the initial arrangement of the data. Here comparison of the number follows the last pass requires one comparison every time, the next to last pass also requires two comparisons and so on to the first pass which requires N-1 comparisons.

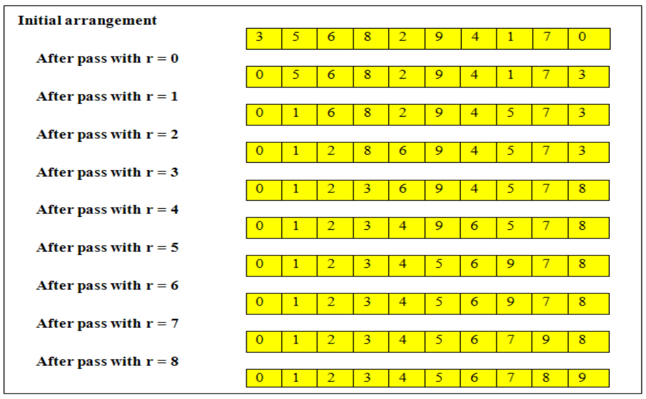

Figure 1. Selection Sort

After following this way of calculating total number or comparisons is [9]:

$$1+2+3+\ldots + N-1 = N(N-1)/2 = N^2/2 - N/2$$

For large values of N, the N^2 dominates, and the order of the selection sort is O (N^2).

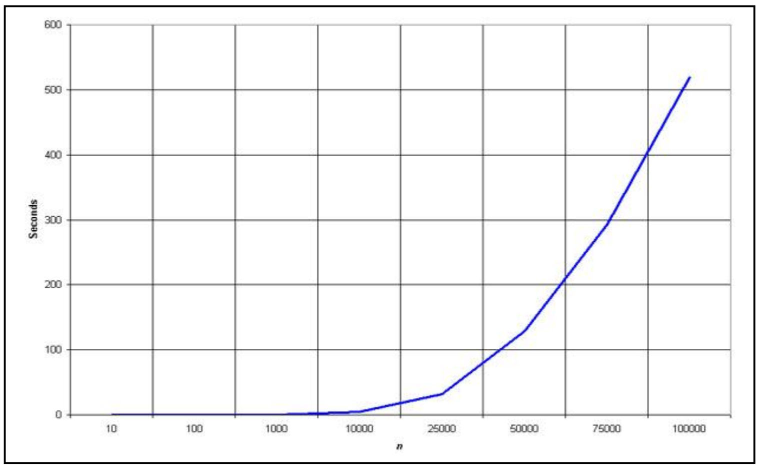

Figure 2. Selection Sort Efficiency

3. Related Work

Paper such as [11, 12] have worked to address related problems that insertion sort performs better on almost sorted arrays that other O (N^2) sorting algorithms. In this paper they developed a bi-partitioned insertion algorithm for sorting, which out beats all other O (N^2) sorting algorithms in general cases and also prove the correctness of the algorithm and give a detailed time complexity analysis of the algorithm. In [12] Ming Zhou, Hongfa Wang suggest an interesting approach to borrow ideas from one- dimensional array selection sorting algorithms, propose a sorting algorithm for two dimensional arrays. Furthermore, conversion of sorting one-dimensional arrays to that of two-dimensional (m*n) arrays and find the values of m and n that minimize the computation time.

While, [13] Oyelami Olufemi Moses worked on improving the performance of bubble sort using a modified diminishing increment sorting. This paper presents a Meta algorithm called Oyelami's Sort that combines the technique of Bidirectional Bubble Sort with a modified diminishing increment sorting. The results from the implementation of the algorithm compared with Batcher's Odd-Even Sort and Batcher's Bitonic Sort showed that the algorithm performed better than the two in the worst case scenario. The implication is that the algorithm is faster. Now, the double selection sort problem is a somewhat well-known problem as will be discussed in section 4.

4. DOUBLE SELECTION SORT

The double selection sort starts from two elements and searches the entire list until it finds the minimum value and maximum value. The sort places the minimum value in the first place and maximum value in the last place, selects the second and second last element and searches for the second smallest and largest element. This process continues until the complete list is sorted. In other words,a take on an elementary sorting algorithm that is designed to minimize the number of exchanges that are performed. It works by making N-1 passes over the shrinking unsorted portion of the array, each time selecting the smallest and largest value. Those values are then moved into their final sorted position with one exchange a piece.

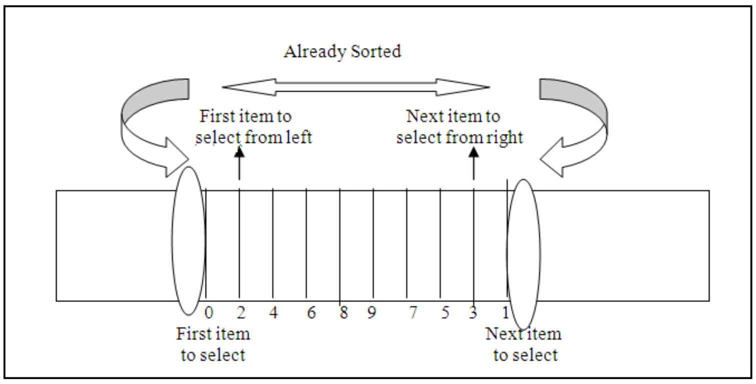

Figure 3. Double selection of the pair of elements

Here, schematic representation is showing that we are at the point where first pairs of item have been double selection sorted drawn with an oval in figure 2. Then check for the next first and next item selection from left to right. We first interchange the two elements of this pair if necessary. We first select two elements small and large, when that is done, interchange with first and last position. Same procedure for the next elements present in sorting list. This type of coding is needed to mitigate the number of exchange and one way in which the code is somewhat better than for straight selection sort. In other words, this algorithm generally succeeds in the goal that the larger element tend to be inserted from the large side of the array (e.g., the right side), and the smaller element from the small side (the left).
The basic operation of simple selection sort is to scan a list x1,…., xn to locate the smallest element and to position it at the beginning of the list. A variation of this approach is to locate both the smallest and the largest elements while scanning the list and to position them at the beginning and at the end of the list respectively. On the next scan, this process is repeated for the sub list x2,… x n-1 and so on. This is double-ended simple selection sort.
For example, suppose we have a list, which contains the elements: **3 5 6 8 2 9 4 1 7 0.** If we want to sort this list using the selection sort technique, the details of this technique are illustrated in Fig 4[8].

Initial arrangement	3	5	6	8	2	9	4	1	7	0
After pass with r = 0 and 9	0	5	6	8	2	3	4	1	7	9
After pass with r = 1 and 8	0	1	6	7	2	3	4	5	8	9
After pass with r = 2 and 7	0	1	2	5	6	3	4	7	8	9
After pass with r = 3 and 6	0	1	2	3	4	5	6	7	8	9

Figure 4. Double Selection Sort

In figure 4, the part 1 is generated during the first pass with r = 0 and 9, similarly pass 2 , pass3 and pass4 are produced after passing with r =1 and 8, r =2 and 7, r = 3 and 6. From the above representation in Figure 3, it is clear that the algorithm consumes most of its time in the inner loop. During the pass current of the outer loop size – current elements are examined to find the smallest remaining elements. This requires size – current – 1 comparisons. The number of comparisons for all passes is:

$$E(n)= \sum_{index=0}^{n-1} n - index - 1$$

Let K = n-index-1

K = n-1 at index = 0

K = 0 at index = n-1

$$E(n)= \sum_{k=0}^{n-1} k$$

$$=n(n-1)/2$$
$$= O(n^2)$$

Selection Sorts find the minimum, or maximum, of N elements in N-1 comparisons. The Max Min algorithm lets you fine the minimum and maximum of N elements in 3N/2 comparisons (keep the current min and max; take 2 more elements, compare them against each other; then compare the two largest and the two smallest). To convert this to a sorting algorithm, we reverse the "build direction" of the Double Selection Sort algorithm by sorting our array from the outside in. Find the Max & Min in array elements [0…..N-1], place them in the two outside positions. Then elements [0 … *N*–1], place them in the two outside positions, then iterate this algorithm with the remaining array elements [1 … *N*–2]. While Selection Sort requires $N^2/2 + O(N)$ comparisons, Double Selection Sort will require $3N^2/8 + O(N)$ comparisons, for a 25% speedup on this measure. Unfortunately, Selection Sort is used primarily when comparisons are much cheaper than assignments, since it uses only $N + O(1)$ assignments. Consequently, this improvement does not have as much of the "real-life value" that Double Selection Sort has. Nevertheless, it could still be a useful assignment, or for a class that's seen Double Selection Sort, it would make a useful comparison and an opportunity to demonstrate the Max Min algorithm.
In pseudo code, the double-ended selection sort is described as [9]:

```
Set interchange count from first to last (not required, but done just to keep track of the interchanges).
    For smallest element in the list from first to next- to-last and for largest element from last to second one,
     Find the smallest and largest elements from the current element being referenced to all elements by:
       Setting the minimum value equal to the first element and the maximum value equal to the last element
         Saving (sorting) the index of the first and last element.
           For each element in the list from the first element+1 and to the last element-1 in the list,
             If element [inner loop index] <minimum value,
               Set the minimum value = first element [inner loop index].
                Then element [inner loop index]>maximum value,
                 Set the maximum value = last element [inner loop index].
               Save the index of the new found minimum and maximum value.
             End If.
           End For
         Swap the current value with the new minimum value and maximum value.
       Increment the interchange count and decrement from other side of the list.
     End for.
   Return the interchange count
```

5. THE DOUBLE SELECT.CPP PROGRAM SEGMENT IMPLEMENTATION

```
#include<iostream.h>
#include<constream.h>
              template <class D>                  // use of single class template
      void DeSelSort(D arr[],int n)  //Normal template function declaration
       {
           int smallIndex,largeIndex;
           int leftPass=0,rightPass=n-1,i,j;
           D temp;                                // data member of template

         while (leftPass<=rightPass)
        {
            smallIndex=leftPass;
            largeIndex=rightPass;

    for (i=leftPass+1;i<rightPass;i++)
      if (arr[i]<arr[smallIndex])
          smallIndex=i;
      if (smallIndex!=leftPass)
        {
         temp=arr[leftPass];
        arr[leftPass]=arr[smallIndex];
        arr[smallIndex]=temp;
        }

        for (j=rightPass-1;j>leftPass;j--)

      if(arr[j]>arr[largeIndex])
         largeIndex=j;
      if(largeIndex!=rightPass)
        {
         temp=arr[rightPass];
        arr[rightPass]=arr[largeIndex];
        arr[largeIndex]=temp;
        }
    leftPass++;
    rightPass--;
   }
      }
```

```
  void main()
 {
  clrscr ();
    cout<<"Entered list for double end selection."<<endl;
        int i[8] = {4,6,2,0,3,7,9,5};
        float f[8] = {2.3,7.5,5.4,9.6,3.5,6.7,8.3,4.5};
          DeSelSort(i,8);//call to function
          DeSelSort(f,8);//call to function
        int leftPass = 0,rightPass = 0;
    cout<<"\n Integer array elements in ascending order :->";
 while(leftPass<8)
    cout<<i[leftPass++]<<" "; //display contents of integer array
    cout<<"\n";
    cout<<"\n Float array elements in ascending order :-> ";
while(rightPass<8)
   cout<<f[rightPass++]<<" "; //display contents of float array
    getch ();
}
```

6. CONCLUSION

Selection Sort is not known to be a good algorithm because it is an in-place comparison sort based sorting algorithm. However, efforts have been made to improve the performance of the algorithm particularly where auxiliary memory is limited. With Bidirectional Selection Sort, the average number of comparisons is slightly reduced and Double Selection sort similar to simple Selection Sort also performs significantly better and carrying out comparisons in a novel way so that no propagation of exchange is necessary. This improvised version of sorting based on using the technique of Double Selection Sort and a modified diminishing increment sorting. The experimentation of the proposed algorithm has shown more efficient instead of using previous version. The algorithm is recommended for all sizes of elements to be sorted but much more efficient as the elements to be sorted increases.

References

[1.] Y.Han “Deterministic sorting in O(nlog logn) time and linear space”, Proceedings of the thirty – fourth annual ACM symposium on Theory of computing , Montreal, Quebec, Canada,2002, p.602-608.
[2.] M.Thoroup ” Randomized Sorting in O(nlog logn) Time and Linear Space Using Addition, shift, and Bit-wise Boolean Operations”, Journal of Algorithms, Volume 42, Number 2,February 2002, p.205-230.
[3.] Wikipedia. Address: http://www.wikipedia.com
[4.] Available at: http://webspace.ship.edu/cawell/Sorting/selintro.htm
[5.] Available at: http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm
[6.] Availableat:http://72.14.235.104/search?q=cache:oRR7xyvYMJ:linux.wku.edu/~lamonml/algor/sort/selection.html+selection+sort&hl=en&ct=clnk&cd=3&gl=in, accessed on 20 May, 2007.
[7.] Sartaj S (2000). Data Structures, Algorithms and Applications in Java. McGraw-Hill.
[8.] Expert Data Structures with C (Third Edition, 2008), R B Patel.
[9.] Program Development and Design using C++ (Second Edition), Gary J.Bronson.
[10.] Darrah P.Chavey “Double Sorting: Testing Their Sorting Skills”.
[11.] Tarun Tiwari, Sweetesh Singh, Rupesh Srivastava and Neerav Kumar “ A Bi-partitioned Insertion Algorithm Sorting” Motilal Nehru National Institute of Technology, India
[12.] Ming Zhou, Hongfa Wang “An Efficient Selection Sorting Algorithm for Two-Dimensional Arrays “Zhejiang. Water Conservancy and Hydropower College Hangzhou, China
[13.] Oyelami Olufemi Moses “ Improving the performance of bubble sort using a modified diminishing increment sorting” Covenant University, P. M. B. 1023, Ota, Ogun State, Nigeria

AUTHOR

Surender Lakra received the B.Tech and M.Tech. degrees in Computer Science & Engineering from Singhania University, Rajasthan and Amity University, Haryana in 2010 and 2012, respectively. Currently he is working in HIT, Haryana from last one year. He has published Research papers in 2 National/ International Journals and Conferences. His area of interest is Programming Language and Artificial Intelligence.

Divya received the B. Tech degree in Information Technology from R.G.E.C Meerut, U.P (U.P.T.U) in 2010 and M.Tech degree in Computer Science & Engineering from Lingaya's University in 2012, respectively. Currently, she is working in Haryana Institute of Technology, Haryana from last one year. She has published Research papers in 7 National/ International Journals and Conferences. Her area of interest is system and network security