International Journal of Computer Applications Technology and Research
Volume 5–Issue 11, 721-724, 2016, ISSN:-2319–8656

# A Few Improvements for Selection Sort Algorithm

Prof. T.S.V. Mani S. Divyaa Devi Dr. S. Sakthivel

**Abstract :**

Sorting refers to arranging data in a particular format. In this paper three modifications for the selection sort algorithm are presented. The modifications are mainly based on the programming methodologies used. In the first algorithm smallest and biggest element are found out in a single loop and exchanging first element with the smallest and last element with the biggest. This is repeated for all the elements. In the second algorithm first smallest and second smallest element are found out in a single loop and these values are moved as the first two elements. This is repeated for the rest of the elements. In the third algorithm first biggest and second biggest element are found out in a single loop and these are carried to the last positions. This is repeated for the remaining elements. The advantages of the modifications are then analyzed.

**Keywords:** Selection Sort, Algorithm, Comparison, Exchange

## 1. INTRODUCTION

Ordering the data in an increasing or decreasing fashion according to some data item is called sorting. Selection sort is a simple sorting algorithm. Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array. This process continues moving unsorted array boundary by one element to the right. The average and worst case complexity are of O(n2) where n are no. of items.

## 2. RELATED WORK

[1] Both ended Sorting Algorithm. The algorithm comprises of two phases, in first phase, one element from the front end and one element from rear end are compared. If the front element is greater than rear element, then swap the elements. In the second phase, two consecutive elements from the front are taken and compared. Replacing is done if required.

[2] Improving the performance of Selection sort. The algorithm is developed by finding the smallest element in the data list and replacing with first element is done in one loop and finding the largest element in the data list and replacing with last element is done in another loop.

## 3. SELECTION SORT

Selection sort is a simple sorting algorithm. In this sorting algorithm the list is divided into two parts, sorted part at left end and unsorted part at right end. Initially sorted part is empty and unsorted part is entire list. Smallest element is selected from the unsorted array and swapped with the leftmost element and that element becomes part of sorted array.

### 3.1 Algorithm

```
Procedure Selection_Sort(K,N)

1. [Loop on Pass Index]
        Repeat thru step 4 for Pass = 1,2,...,N-1
2. [Initialize Minimum Index]
        MIN_INDEX ← Pass
3. [Make a Pass and obtain smallest element]
        Repeat for I = Pass+1, Pass+2,..., N
        If K[I] < K[MIN_INDEX]
        then MIN_INDEX ← I
4. [Exchange elements]
        If MIN_INDEX ≠ Pass
        then K[Pass] ‹—› K[MIN_INDEX]
5. [Finished]
        Return
```

This process continues moving unsorted array boundary by one element to the right. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n are no. of items.

### 3.2 Pseudo Code

```
for (i = 0; i < count - 1; i++)
{
          minimum = i;
         for (j = i + 1; j < count; j++)
{
          if (data[minimum] > data[j])
         {
                    minimum = j;
         }
}
 swap data[i] and data[minimum]
}
```

## 4. FIRST SMALLEST FIRST BIGGEST

This algorithm finds the first smallest and the first biggest element in a single loop and exchanging first element with the smallest and last element with the biggest. This is repeated with other elements.

### 4.1 Algorithm

```
Procedure Sort1(K,N)

1. [ Loop on Pass Index]
        Repeat thru step 5 for Pass = 1,2,...N
2. [Initialize Minimum & Maximum Index]
        MIN_INDEX ← K[Pass]
        MAX_INDEX ← K[N]
3.[Make a Pass and obtain Smallest and Biggest element]
        Repeat for I = Pass+1, Pass+2....N
        If MIN_INDEX > K[I]
        then MIN_INDEX ← K[I]
                POS1 ← I
        If MAX_INDEX < K[I]
        then MAX_INDEX ← K[I]
                POS2 ← I
4. [Exchange elements]
        K[Pass]  ⟷ POS1
        K[N] ⟷ POS2
5. [Decrement N]
        N  ←  N-1
6. [Finished]
        Return
```

### 4.2 Pseudo Code

```
for (p=1;p<=n;++p)
{
        s = a[p], b = a[n];
        for(j=p;j<=n;++j)
        {
                if (s > a[j])
                {           s = a[j];
                        x = j;
                }
                else if (b < a[j])
                {
                        b = a[j];
                         y = j;
                }
Swap elements in position x and y with first and last element
n = n-1;
}
```

### 4.3 Working

This algorithm first finds the smallest as well as biggest number in a single loop. Once the smallest number is found, its position in the list is stored in the variable x, simultaneously the position of the biggest number is stored in variable y. Swap the first element and the element in position x and also swap the last element with the element in position y. The above process is continued until all the elements are attained its position. The loop process is done in n/2 times where n is the total no. of data elements in the list.

## 5. FIRST SMALLEST SECOND SMALLEST

This algorithm finds the first smallest and the second smallest element in a single loop and exchanging first element with the first smallest and the second element with the second smallest element.

### 5.1 Algorithm

```
Procedure Sort2(K,N)

1. [ Loop on Pass Index]
        Repeat thru step 4 for Pass = 1,2,...N
2. [Initialize First & Second Smallest Index]
        SMALL1 ← K[Pass]
        SMALL2 ← K[Pass+1]
3.[Make a Pass and obtain First Smallest and Second Smallest element]
        Repeat for I = Pass+1, Pass+2....N
        If SMALL1 > K[I]
        then SMALL2 ← SMALL1
                SMALL1 ← K[ I]
                POS1 ← I
        ElseIf SMALL2 < K[I]
        then SMALL2 ← K[I]
                POS2 ← I
4. [Exchange elements]
        K[Pass]  ⟷ POS1
        K[Pass+1] ⟷ POS2
5. [Finished]
        Return
```

### 5.2 Pseudo Code

```
for (p=1;p<=n;++p)
{
    small1=a[p]; small2 = a[p+1];
    for(j=p+1;j<=n;++j)
    {
         if (small1 > a[j])
        {
            small2 = small1;
            small1 = a[j];
            y = x;
            x = j;
        }
        else if (small2 > a[j])
        {       small2 = a[j];
                y = j;
        }
    }
    Swap elements in position x & y with p and p+1
    ++p;
}
```

### 5.3 Working

This algorithm first finds the first smallest as well as second smallest numbers in a single loop. Once the first smallest number is found, its position in the list is stored in the variable x, simultaneously the position of the second smallest number is stored in variable y. Swap the first element and the element in position x and also swap the second element with the element in position y. The above process is continued until all the elements are attained its position. The loop process is done in n/2 times where n is the total no. of data elements in the list.

## 6. FIRST BIGGEST SECOND BIGGEST

This algorithm finds the first biggest and the second biggest element in a single loop and exchanging the first element with the first biggest and the second element with the second biggest.

### 6.1 Algorithm

```
Procedure Sort3(K,N)

1. [ Loop on Pass Index]
        Repeat thru step 4 for Pass = 1,2,...N
2. [Initialize First & Second Biggest Index]
        BIG1 ← K[Pass]
        BIG2 ← K[Pass+1]
3.[Make a Pass and obtain First Biggest and Second Biggest element]
        Repeat for I = Pass+1, Pass+2....N
        If BIG1 > K[I]
        then BIG2 ← BIG1
                BIG1 ← K[ I]
                POS1 ← I
        ElseIf BIG2 < K[I]
        then BIG2 ← K[I]
                POS2 ← I
4. [Exchange elements]
        K[Pass] ⟷ POS1
        K[Pass+1] ⟷ POS2
5. [Finished]
        Return
```

### 6.2 Pseudo Code

```
for (p=1;p<=n;++p)
{
    x = 0; y = 0; big1= a[p]; big2 = a[p+1];
    for(j=p+1;j<=n;++j)
    {
         if (big1 < a[j])
        {       big2 = big1;
                big1 = a[j];
                y = x;
                x = j;
        }
        else if (big2 < a[j])
        {       big2 = a[j];
                y = j;
        }
    }
    Swap elements in position x & y with p and p+1
    ++p;
}
```

### 6.3 Working

This algorithm first finds the first biggest as well as second biggest numbers in a single loop. Once the first biggest number is found, its position in the list is stored in the variable x, simultaneously the position of the second biggest number is stored in variable y. Swap the first element and the element in position x and also swap the second element with the element in position y. The above process is continued until all the elements are attained its position. The loop process is done in n/2 times where n is the total no. of data elements in the list.

## 7. COMPARISON

### 7.1 Comparison Table

| No. of Elements | A | B | C | D |
|---|---|---|---|---|
| 50 | 1275 | 1236 | 1180 | 1177 |
| 100 | 5050 | 5027 | 4834 | 4810 |
| 150 | 11325 | 11257 | 10986 | 10985 |
| 200 | 20100 | 19959 | 19591 | 19618 |
| 300 | 45150 | 44896 | 44386 | 44320 |

A → Selection Sort
B → First Smallest First Biggest
C → First Smallest Second Smallest
D → First Biggest Second Biggest

Above programs were executed with sample input data of 50, 100, 150, 200, 300 and no. of comparisons ( IF conditions ) are calculated.

## 8. CONCLUSION

From the comparisons table it is found that the new algorithms (First Smallest & First Biggest, First Smallest & Second Smallest, First Biggest & Second Biggest) the no. of comparisons made is lesser than the no. of comparisons made in selection sort.

## 9. REFERENCES

[1] Anuradha Brijwal, Both Ended Sorting Algorithm & Performance comparison With Existing Algorithm, IJIEASR, Vol 3, No.6, June 2014.

[2] Surendra Lakra, Divya, Improving the performance of selection sort using a modified double ended selection sorting, IJAIEM, Vol2, Issue 5, May 2013.

[3] Jean Paul Tremblay, Paul G. Sorenson, An Introduction to Data Structures with Applications, Second Edition, Tata McGraw Hill Publishing Company Limited, 1991.

[4] Byron Gottfried, Jitender Kumar Chhabra, Programming with C, Third Edition, Schaum’s Outlines Series, 2011.