

# Analysis on Selection Sort Algorithm based on 2-Element

Aman Akarshan Sinha<sup>1</sup> Ashish Kumar Kaushal<sup>2</sup> Aman Kumar<sup>3</sup> Ajit Tiwari<sup>4</sup>

Mr. Abhishek Malviya<sup>5</sup>

<sup>5</sup>Assistant Professor & Head of the Department

<sup>1,2,3,4,5</sup>Department of Computer Science and Engineering

<sup>1,2,3,4,5</sup>United Institute of Technology, Prayagraj, India

**Abstract**— This research paper shows improved version of selection sort algorithm which is based on 2-element, also gives the approach for design of algorithm and program in C++ language. This improved sorting technique is simple to implement. After that study compare the space complexity and time complexity of the algorithms with the basic selection sort and also discusses the advantages and disadvantages of the selection sort based on 2 element method and the original one.

**Keywords:** Sorting Algorithm, Selection Sorting

## I. INTRODUCTION

Sorting algorithms are the method for reordering the number of element from array in a specified order, such as greatest to smallest, or vice versa. These algorithms work like: they take an input array, process it (i.e, perform some operations on it), and return the sorted array as output.

Some considerations must be considered while selecting a sorting algorithm: How large is the collection to be sorted? What is the available memory size? Is there a need for expansion of the collection?

The answers to these questions may influence which algorithm is best suited to each case. Some algorithms, such as merge sort, may take a large amount of space or memory to perform, whereas insertion sort is not necessarily the quickest but does not require as many resources to run.

Before settling on a sorting algorithm, you should assess your requirements and take into account the limits of your system.

Along with the minimizing the time complexity and space complexity, it is beneficial to minimize the number of writes especially when the writing of dataset is costly like flash memory.

Data structure and algorithm includes lots of Sorting algorithms, among all of them Selection Sort requires the fewest amount of writes in array i.e it performs  $O(n)$  swaps. That's why our research starts with Selection Sort Algorithm.

## II. THE BASIC SELECTION SORT ALGORITHM

The Basic Selection sort technique is based on the idea that in each iteration, it identifies the smallest or largest element in an unordered Array and then swap it with 1st element of unordered Array.

### A. Algorithm Design Approach

The following is the main design approach of the basic selection sort algorithm.

In Selection Sort algorithm, list is divided logically into two subarrays, the ordered list at the left side and the unsorted list at the right side. Firstly, sorted list is of size 0 and unsorted list is of size N (size of input list). Then find the smallest element from unordered list and swapped with the

leftmost element of unordered list, and size of unsorted list is decreased by 1 or we can say element becomes a part of the sorted Array. The process continues until the size of unsorted list is become 0.

### B. Algorithm Description in C++

```
void selectionSort(int Array[], int n)
{ int u, v, minIndex;
// In each iteration decrease size of unsorted array by one
  for (u=0;u< len-1;u++)
    { minIndex = u;
// Finding minimum element from unsorted array
      for (v=u+1; v< n;v++) {
        if (Array [v] < Array [minIndex])
          minIndex = v;
      }
// Swapping the minimum element from the unsorted array
      with the first element of unsorted array
      int tempVar = Array [u];
      Array[u] = Array[minIndex];
      Array[minIndex] = tempVar;
    }
}
```

## III. SELECTION SORT ALGORITHM BASED ON 2-ELEMENT

As the basic selection sort method can only place one element per sort iteration at the beginning, so we can name it as the selection sort based on 1-element. Considering the 2-element selection sort method, that place 2 items in each iteration one at the beginning and the other at the end. and so it minimizes the number of iterations of the loop and comparison of elements require for sorting.

### A. Algorithm Design Approach

The basic design approach of the Selection Sort based on 2-element Algorithm is that in each iteration find the maximum element and minimum element and put that maximum element at the end and minimum element at the beginning of unsorted array and then size of Array in each iteration is decreased to  $n-2$ .

The concrete steps in the  $i$ -th sort iteration that runs  $n/2$  times are as follows:

Step 1: Take two variable low and high, low represent low index of unsorted Array and high represent upper index of unsorted Array. Initially store 0 in low and size of Array i.e n in high.

Step 2: Run while loop until low is greater than equal to high. In each iteration, find index of minimum element and maximum element from the unsorted Array and store in MinIndex and MaxIndex respectively.

Step 3: After that based on some condition change the element at index low, high, MinIndex, and MaxIndex of the Array.

Step 4: If MinIndex is equal to high and MaxIndex is equal to low then simply swap elements of MinIndex and MaxIndex.

Step 5: If MaxIndex is equal to low then store element at index MaxIndex into Array[low], store element at index MinIndex into Array[MaxIndex] and store element at index high into Array[MinIndex].

Step 6: If MinIndex is equal to high then store element at index MinIndex into Array[low], store element at index MaxIndex into Array[MinIndex] and store element at index low into Array[MaxIndex].

Step 7: If none of them condition happen then swap element at index low with MinIndex and swap element at index high with MaxIndex.

#### B. Algorithm Description in C

```
int* selectionSortOptimized (int Array[], int n)
{
    int j, MinIndex, MaxIndex;
    int lowIndex=0, highIndex=n-1;

    while ( lowIndex <= highIndex ) {
        MinIndex = lowIndex;
        MaxIndex = lowIndex;

        for (j = lowIndex+1; j <=highIndex; j++) {
            // Find the minimum element and maximum element from
            // unsorted Array
            if (Array[j] < Array[MinIndex])
                MinIndex = j;

            if (Array[j] > Array[MaxIndex])
                MaxIndex = j;
        }

        if (MinIndex == highIndex && MaxIndex == lowIndex) {
            int tempVar1 = Array[MinIndex];
            Array[MinIndex] = Array[MaxIndex];
            Array[MaxIndex] = tempVar1;
        }

        else if (MaxIndex == lowIndex) {
            int tempVar2 = Array[highIndex];
            Array[highIndex] = Array[MaxIndex];
            Array[MaxIndex] = Array[MinIndex];
            Array[MinIndex] = tempVar2;
        }

        else if (MinIndex == highIndex) {
            int tempVar3 = Array[lowIndex];
            Array[lowIndex] = Array[MinIndex];
            Array[MinIndex] = Array[MaxIndex];
            Array[MaxIndex] = tempVar3;
        }
        else {
            // swap element at index low with MinIndex and swap
            // element at index high with MaxIndex.

            swap(&Array[MinIndex], &Array[lowIndex]);

```

```
swap(&Array[MaxIndex], &Array[highIndex]);
    }

    highIndex--;
    lowIndex++;
}

return Array;
}
```

#### IV. TEST AND ANALYSIS OF ALGORITHM

Two more functions Main() and Print() which is output function are added to program to compare function selectionSortOptimized() with selectionSort(). Both algorithms are compared on the same elements of unordered list. And tested with the number of test cases ranging from small to large number of unordered elements in list.

##### A. Analysis on both Algorithm on the basis of time and space complexity

Considering the basic Selection Sort algorithm, Array is divided logically into two subarrays: left subarray consists ordered elements and next subarray contains unordered elements, initially size of unsorted Array is 0. Iterate it (n-1) times and in each iteration we find minimum element from unordered array and swap it with 1st element of unordered Array.

So in the 1st iteration we perform (n - 1) comparisons, in the second (n-2), in the third (n-3) and this continue until comparison is only one that is also our last iteration. Therefore, the total number of comparisons is (n-1) + (n-2) + (n-3) + (n-4) + ... + 1 = n(n-1)/2 times. So the overall time complexity is quadratic i.e O(n<sup>2</sup>) for best, average and worst case. In basic Selection sort, we are not using any extra memory space except loop variables u and v and the array length n, minIndex, and min. Hence Space complexity of Basic Selection Sort is constant i.e S(n) = O(1).

Considering the Selection Sort Algorithm based on 2-element, the function selectionSortOptimized() consist of nested double loops. Outer loop runs while low is less than equal to high and in each iteration low is increased by 1 and high is decreased by 1, it means outer loop runs for n/2 times where n is size of Array.

Inner loop finds minIndex and maxIndex which also runs for n/2 times. Inside outer loop, there is 4 if-else statement, which execute sequentially after the inner loop, that also run for n/2 times. So in the first iteration of outer loop, we perform (n-1) comparisons, in the second (n-3), in the third (n-5) and this continue until comparison is only one that is also our last iteration. Therefore, the total number of comparisons is (n-1) + (n-3) + (n-5) + ... + 1 = n(n-2)/4 times. And each if-else statement runs for constant time. So the overall time complexity is quadratic i.e O(n<sup>2</sup>) for best, average and worst case. Despite the fact that optimized algorithm requires two more auxiliary variable maxIndex and high than selectionSort(), but still its space complexity is constant i.e. S(n) = O(1). Thus, we can say that both algorithm has equal time and space complexity. But the optimised algorithm is more time efficient than the basic selection sort algorithm.

## V. CONCLUSION

We can conclude that Selection Sort based on 2-element Algorithm is an improved algorithm design idea which is based on the basic/traditional selection sort algorithm.

After comparing the time and space complexity of the both algorithms, the following results were achieved:

The Selection Sort Algorithm based on 2-element and basic Selection Sort Algorithm has same average space and time complexity. But the optimised algorithm is more time efficient than the basic one especially for large dataset that is positive result. It uses one more auxiliary space and 3 extra *if-else* statement but still have same space complexity. Furthermore, we can say optimized algorithm is little bit complicated than the basic one.

The purpose of this study is to present an alternative idea of the selection sort algorithm. so that a theoretical framework for the fundamental selection sort algorithm optimization may be established, and to serve as a teaching guide for the relevant chapters in the "Data Structure and Algorithm" course.

## REFERENCES

- [1] Rao, D. T. V. D., & Ramesh, B. (2012). Experimental based selection of best sorting algorithm. *Int. J. Mod. Eng. Res.*, 2(4), 2908-2912.
- [2] Jadoon, S., Solehria, S. F., Rehman, S., & Jan, H. (2011). Design and analysis of optimized selection sort algorithm. *International Journal of Electric & Computer Sciences (IJECS-IJENS)*, 11(01), 16-22.
- [3] Hayfron-Acquah, J. B., Appiah, O., & Riverson, K. (2015). Improved selection sort algorithm. *International Journal of Computer Applications*, 110(5).
- [4] Khan, M. (2012). Proposal of a two way sorting algorithm and performance comparison with existing algorithms.
- [5] Kumar, M., Malhotra, M., & Ahuja, D. (2015). Minimizing the execution time of selection sort algorithm. *International Journal of Engineering and Computer Science*, 6(4), 21-27.
- [6] Gill, S. K., Singh, V. P., Sharma, P., & Kumar, D. (2019). A comparative study of various sorting algorithms. *International Journal of Advanced Studies of Scientific Research*, 4(1).