SELECTION SORT AND ITS ANALYSIS

❖ INTRODUCTION

Selection sorting is conceptually the simplest sorting algorithm. It is an ***in-place*** (i.e. not needing any auxiliary storage), ***comparison-based*** sorting technique.

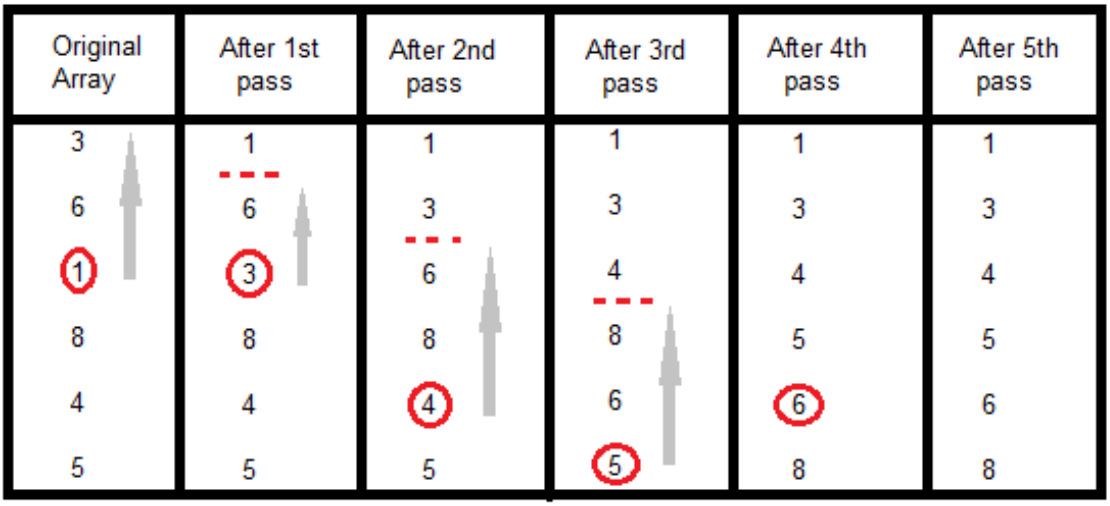
This algorithm first finds the smallest element in the array and exchanges it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continues in this way until the entire array is sorted. Basically it means that the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

❖ ILLUSTRATIVE EXAMPLE

Input Array: [3 6 1 8 4 5]

Output Array: [1 3 4 5 6 8]

In the first pass, the smallest element found is 1, so it is placed at the first position, then leaving first element, smallest element is searched from the rest of the elements, 3 is the smallest, so it is then placed at the second position. Then we leave 1 and 3, from the rest of the elements, we search for the smallest and put it at third position and keep doing this, until array is sorted.



❖ SELECTION SORT ALGORITHM

1. Repeat For J = 0 to N-1
2. Set MIN = J
3. Repeat For K = J+1 to N
4. If (A[K] < A[MIN]) Then
5. Set MIN = K

[End of If]
[End of Step 3 For-Loop]
6. Interchange A[J] and A[MIN]

[End of Step 1 For-Loop]
7. Exit

❖ C PROGRAM CODE

```
void selectionSort(int a[], int size)
{
    int i, j, min, temp;
    for(i=0; i < size-1; i++ )
  {
     min = i; //setting min as i
     for(j=i+1; j < size; j++)
    {
      if(a[j] < a[min])    //if element at j is less than element at min position
     {
       min = j; //then set min as j
      }
     } // end of inner loop
     temp = a[i];
     a[i] = a[min];
     a[min] = temp;
    } // end of outer loop
 }// end of function
```

❖ COMPLEXITY ANALYSIS

Space Complexity

Selection sort is an in-place algorithm. It performs all computation in the original array and no other array is used. Hence, the space complexity works out to be **O(1)**.

Time Complexity

The number of comparison in the selection sort algorithm is independent of the original order of the element. This makes it a ***data-insensitive sorting technique***. Selection sort algorithm consists of two nested loops.

There are n-1 comparison during PASS 1 to find the smallest element, there are n-2 comparisons during PASS 2 to find the second smallest element, and so on. Accordingly, $T(n)=(n-1)+(n-2)+\ldots\ldots\ldots\ldots\ldots\ldots\ldots+2+1 = n(n-1)/2 = \mathbf{O(n^2)}$

Algorithm	Worst Case	Average Case	Best Case
Selection Sort	$n(n-1)/2 = O(n^2)$	$n(n-1)/2 = O(n^2)$	$O(n^2)$

Conclusion

Selection sort is a sorting algorithm that has a quadratic running time complexity of $O(n^2)$, thereby making it inefficient to be used on large lists. Although selection sort performs worse than Insertion Sort algorithm, it is noted for its simplicity and also has performance advantages over more complicated algorithms in certain situations. Selection sort is generally used for sorting files with very large objects (records) and small keys.

❖ ADVANTAGES OF SELECTION SORT

1. It is simple and easy to implement.
2. It can be used for small data sets.
3. It is 60 per cent more efficient than Bubble Sort.
4. Selection sort uses minimum number of swap operations O(n) among all the sorting algorithms. One thing which distinguishes selection sort from other sorting algorithms is that it makes the minimum possible number of swaps, $n - 1$ in the worst case.

❖ QUESTION OF STABILITY

Stable sorting algorithms maintain the relative order of records with equal keys (i.e. values). That is, a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

Selection Sort is not a stable sorting algorithm, because it swaps non-adjacent elements. The most pertinent example being: Given [2, 2, 1], the '2' values will not retain their initial order in the output sorted array.

However, Selection sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e. by placing the number in its position by pushing every element one-step forward. In simple terms use a technique like insertion sort which means inserting element in its correct place. However, this modification either requires a data structure that supports efficient insertions or deletions, such as a linked list.

❖ OPTIMIZED SELECTION SORT

An improvement can be made on selection sort. As we know that the selection sort works by taking either the minimum or maximum element from the array and placing that element at its correct position. Here the idea is to take the *max* and *min* simultaneously, then sort the array from two ends. This is much faster than normal Selection Sort. It sorts the data in half the number of iterations as it sorts two data elements at a time. Therefore, it minimizes the sorting time by up to 50%.

References –

1. Data Structures – Seymour Lipschutz
2. Data Structures – Srivastava & Srivastava
3. Data Structures – Reema Thareja