Sorting Matrix

Vidushi Pathak (IIT2019027)
Shubham Netam (IIT2019028)
Puneet Singhwaiya (IIT2019029)

B.Tech 4th semester, Department of Information Technology
Indian Institute Of Information Technology, Allahabad

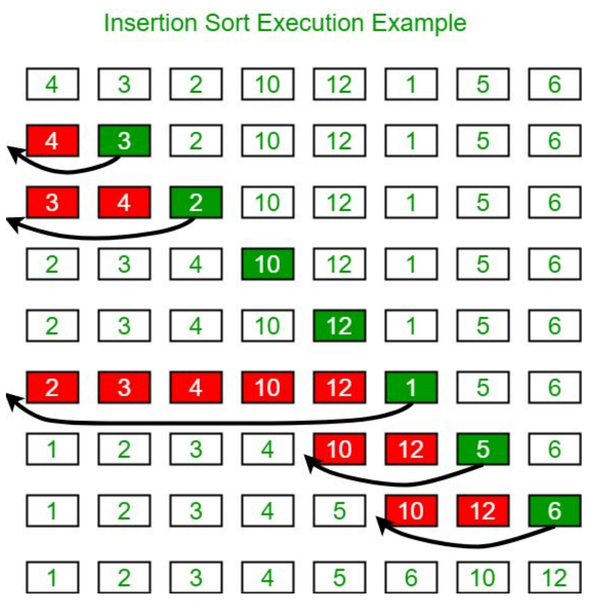
Abstract- ***In this paper we have discussed the algorithm to sort a matrix of positive integers using insertion and selection sort,their pros and cons,relationship between them and also did a complexity analysis on both sorting techniques.***

I. Introduction

A. Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

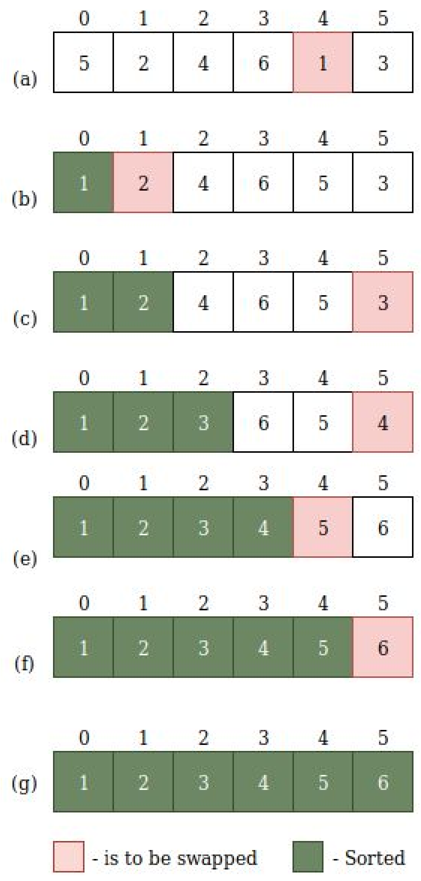
To sort an array of size n in ascending order we iterate the array from arr[1] to arr[n] and compare the current element(key) to its predecessor. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.



B. Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the array 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 array.

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.



Selection Sort

II. Pseudo code to sort a matrix

A. Using Insertion sort

1. We will treat the matrix as a 1D array to sort the matrix without using extra space. So for accessing the i^{th} element of the matrix the relationship can be defined as -
   I^{th} Element of the Matrix =
   mat[i / columns][i % columns]
   and size will be rows*columns.
   Here the starting index is 0.

2. Now we will start traversing from i=1 to i<size and will take mat[i/col][i%col] as key. Here col = number of columns.

3. We will define a variable j = i-1 and while j >=0 and mat[j/col][j%col]
   is greater than key we will keep moving the element one position up to make space for the swapped element i.e
   mat[(j+1)/col][(j+1)%col] = mat[j/col][j%col]

4. When we come out of the while loop we will place the key element at its correct position.
   mat[(j+1)/col][(j+1)%col] = key.

B. Using Selection sort

1. We will treat the matrix as a 1D array to sort the matrix without using extra space. So for accessing the i^{th} element of the matrix the relationship can be defined as -
   I^{th} Element of the Matrix =
   mat[i / columns][i % columns]
   and size will be rows*columns.

2. Now we will start traversing the array from i=0 to i<(size-1).

3. We will define a variable min = i and will traverse the array from j=i+1 to j<size and if we find a value mat[j] such that mat[j]<mat[min] we will assign

index j to min and will swap mat[i/col][i%col] value with the value present at mat[min/col][min%col].

III. Pros and Cons

A. *Insertion Sort*

Pros :-

i. Efficient for sorting small data.

ii. The insertion sort is an in-place sorting algorithm so the space requirement is minimal.

iii. Efficient for data that is almost sorted.

Cons :-

i. it is $O(n^2)$ on average, and so for large n it is usually very slow.

B. *Selection sort*

Pros :-

i. Simple and Easy to implement.

Cons :-

i. Inefficient for large lists, as it will always have a time complexity of $O(n^2)$.

IV. Relationship between Insertion and Selection sort

The logic for both algorithms is quite similar. They both have a partially sorted sub-array in the beginning of the array. The only difference is how they search for the next element to be put in the sorted array.

- **Insertion sort**: *inserts* the next element at the correct position;
- **Selection sort**: *selects* the smallest element and exchange it with the current item;

V. Complexity Analysis For Insertion sort

As we are treating our 2D array matrix as a 1D array of length=size (where, size = no. Of row * no. of column),so its time complexity and space complexity will be almost similar to the time and space complexity of an array i.e. mat[size].

So our analysis is based on pre-known complexity analysis of insertion sort of 1D array.

(i). Time Complexity

Worst case: When elements are in reversed sort arrangement (descending order).

$\mathbf{t}_{\mathbf{worst}}$= O(size*2)

Average cases often have the same complexity as the worst case.

$\mathbf{t}_{\mathbf{avg}}$= O(size*2)

Best case: When elements are arranged in a sorted manner.

$\mathbf{t}_{\mathbf{best}}$ = O(size)

(ii). Space Complexity

As we are only rearranging the elements of array by swapping so we didn't need any extra space.

Insertion sort uses O(1) auxiliary

space.

VI. Complexity Analysis For selection sort

As we are treating our 2D array matrix as a 1D array of length=size (where, size = no. Of row * no. of column),so its time complexity and space complexity will be almost similar to the time and space complexity of an array i.e. mat[size].

So our analysis is based on pre-known complexity analysis of selection sort of 1D array.

(i). Time Complexity

Selection sort takes similar time for any arrangement of data in an array so,

Worst case time=best case time= avg. case time

$\mathbf{t}_{\mathbf{worst}} = \mathbf{t}_{\mathbf{avg}} = \mathbf{t}_{\mathbf{best}}$ = O(size*2)

(ii). Space Complexity

As we are only rearranging the elements of the array by swapping so we didn't need any extra space.

Selection sort uses O(1) auxiliary

space.

VII. Conclusion

We concluded that both insertion and selection sort are quite similar as both are simple, in-place sorting algorithms and both are inefficient for large lists. However, insertion sort is better than selection sort as the time complexity of selection sort is O(n*2) in all cases i.e it has no best case scenario.

VIII. References

- Geeks for geeks.
- Tutorials point
- Image source: geeks for geeks