DAA Assignment 1

Group - 9

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

Problem Statement

Sort a matrix of positive integers using insertion and selection sort. Show how they are related and pros and cons.

Introduction

INSERTION SORT

Insertion sort is a simple 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.

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.

Pseudo code to sort matrix

We will treat the matrix as 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]
- Size = rows*columns

Insertion Sort

```
for i=1 to i<size
    key = mat[i/col][i%col]
    j=i-1
    while j>=0 and mat[j/col][j%col] > key
        mat[(j+1)/col][(j+1)%col] = mat[j/col][j%col]
        j=j-1
    end while
    mat[(j+1)/col][(j+1)%col] = key
end for
```

Time Complexity Analysis

- In the pseudo code we could see that there are 7 operations under this algorithm.
- So we will find the time complexity of each operation and then we will add them to obtain the total time complexity of our algorithm.
- We assume the cost of each operation i as C_i where i belongs to (1,2,3,4,5,6,8)

```
   for i=1 to i<size
2.      key = mat[i/col][i%col]
3.      j=i-1
4.      while j>=0 and mat[j/col][j%col] > key
5.              mat[(j+1)/col][(j+1)%col] =
mat[j/col][j%col]
6.              j=j-1
7.      end while
8.      mat[(j+1)/col][(j+1)%col] = key
9. end for
```

COST OF LINE	NO. OF TIMES IT IS RUN
C_1	n
C_2	n - 1
C_3	n - 1
C_4	$\sum_{j=1}^{n-1}(t_j)$
C_5	$\sum_{j=1}^{n-1}(t_j - 1)$
C_6	$\sum_{j=1}^{n-1}(t_j - 1)$
C_8	n - 1

```
1. for i=1 to i<size
2.        key = mat[i/col][i%col]
3.        j=i-1
4.        while j>=0 and mat[j/col][j%col] > key
5.                mat[(j+1)/col][(j+1)%col] =
mat[j/col][j%col]
6.                j=j-1
7.        end while
8.        mat[(j+1)/col][(j+1)%col] = key
9. end for
```

Total running time of Insertion sort

$(T(n)) = C_1 * n + (C_2 + C_3) * (n - 1) + C_4 * \Sigma^{n-1}_{j=1}(t_j) + (C_5 + C_6) * \Sigma^{n-1}_{j=1}(t_j)-1 + C_8 * (n - 1)$

- Best Case Analysis-

In best case array is already sorted therefore $t_j = 1$
So on solving we will get time complexity = O(n)

- Average Case Analysis-

Time complexity = $O(n^2)$

- Worst Case Analysis-

In worst case array is in descending order therefore $t_j = j$
So on solving we will get time complexity = $O(n^2)$

Selection Sort

```
for i=0 to i< size-1
    min = i
    for j =i+1 to j<n
        if mat[j/col][j%col]<mat[min/col][min%col]
            min = j
    Swap mat[i/col][i%col] with mat[min/col][min%col]
```

Time Complexity Analysis

- In the pseudo code we could see that there are 6 operations under this algorithm.
- So we will find the time complexity of each operation and then we will add them to obtain the total time complexity of our algorithm.
- We assume the cost of each operation i as Ci where i belongs to (1,2,3,4,5,6)

```
1.for i=0 to i< size-1
2.     min = i
3.     for j =i+1 to j<n
4.             if
mat[j/col][j%col]<mat[min/col][min%col]
5.                     min = j
6.     Swap mat[i/col][i%col] with
mat[min/col][min%col]
```

COST OF LINE	NO. OF TIMES IT IS RUN
C_1	n
C_2	n -1
C_3	n(n-1)/2
C_4	n(n-1)/2
C_5	n(n-1)/2
C_6	n

```
1.for i=0 to i< size-1
2.      min = i
3.      for j =i+1 to j<n
4.              if
mat[j/col][j%col]<mat[min/col][min%col]
5.                      min = j
6.      Swap mat[i/col][i%col] with
mat[min/col][min%col]
```

Total running time of selection sort

$T(n) = C1*n + C2*(n-1) + C3*(n(n-1)/2\ ) + C4*(n(n-1)/2) + C5*(n*(n-1)/2) + C6*n$

$T(n) = O(n^2)$

Time complexity for best, average and worst case $= O(n^2)$

Pros and Cons

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.

Selection Sort

Pros :-

i. Simple and Easy to implement.

Cons :-

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

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.