ITW Experiment 8

Saif Alam, 61

Aim: Create .m file and create quicksort algorithm in MATLAB

Theory:

QuickSort is a widely used, efficient, and comparison-based sorting algorithm. It's a divide-and-conquer algorithm that works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

Here's a high-level overview of the QuickSort algorithm:

1. *Select a Pivot:* Choose a pivot element from the array. Common choices include the first, last, or a randomly selected element.

2. *Partitioning:* Rearrange the array so that all elements less than the pivot come before it, and all elements greater than the pivot come after it. The pivot element is now in its final sorted position.

3. *Recursion:* Recursively apply the QuickSort algorithm to the sub-arrays on either side of the pivot.

4. *Combine:* The sorted sub-arrays are combined, with the pivot in its correct position, resulting in the sorted array.

QuickSort has an average and best-case time complexity of O(n log n), making it very efficient for large datasets. However, its worst-case time complexity is O(n^2), which occurs when a poorly chosen pivot consistently divides the array into imbalanced sub-arrays. To mitigate this, various strategies, such as choosing the median as the pivot, are used to make QuickSort more reliable in practice.

Code:

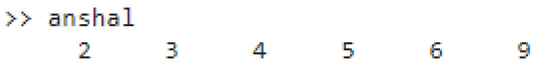
```
unsortedArray = [4, 2, 9, 6, 5, 3];
sortedArray = quicksort(unsortedArray);
disp(sortedArray);


function sortedArray = quicksort(arr)
    if numel(arr) <= 1
        sortedArray = arr;
        return;
    end

    pivot = arr(1);
    less = arr(arr < pivot);
    equal = arr(arr == pivot);
    greater = arr(arr > pivot);

    sortedArray = [quicksort(less), equal, quicksort(greater)];
end
```

Output:



```
>> anshal
     2     3     4     5     6     9
```

Conclusion: Thus, we have learned to create .m files and implemented quicksort.