ASSIGNMENT NO : 6

Write a **python** program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.

QuickSort

Like Merge Sort, QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is **partition().** Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function :

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
   if (low < high)
   {
      /* pi is partitioning index, arr[pi] is now
         at right place */
      pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }
}
```

Code :

Python program for implementation of Quicksort Sort

```
# This function takes last element as pivot, places
# the pivot element at its correct position in sorted
# array, and places all smaller (smaller than pivot)
# to left of pivot and all greater elements to right
# of pivot
```

```
def partition(arr,low,high):
    i = ( low-1 )                 # index of smaller element
    pivot = arr[high]             # pivot

    for j in range(low , high):

        # If current element is smaller than the pivot

         if   arr[j] < pivot:

        # increment index of smaller element

            i = i+1
            arr[i],arr[j] = arr[j],arr[i]

            arr[i+1],arr[high] = arr[high],arr[i+1]
             return ( i+1 )


# The main function that implements QuickSort
# arr[] --> Array to be sorted,
# low  --> Starting index,
# high  --> Ending index

# Function to do Quick sort
def quickSort(arr,low,high):
    if low < high:

        # pi is partitioning index, arr[p] is now
        # at right place
        pi = partition(arr,low,high)

        # Separately sort elements before
        # partition and after partition
        quickSort(arr, low, pi-1)
```

```
        quickSort(arr, pi+1, high)

# Driver code to test above

arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr,0,n-1)
print ("Sorted array is:")
for i in range(n):
    print ("%d" %arr[i]),
```

Analysis of QuickSort

Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \Theta(n)$$

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.
The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.

$$T(n) = T(0) + T(n-1) + \Theta(n)$$

which is equivalent to

$$T(n) = T(n-1) + \Theta(n)$$

The solution of above recurrence is $\Theta(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

$T(n) = 2T(n/2) + \Theta(n)$

The solution of above recurrence is $\Theta(nLogn)$. It can be solved using case 2 of Master Theorem.

Average Case:

To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.

We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

$T(n) = T(n/9) + T(9n/10) + \Theta(n)$

For example: In the array {52, 37, 63, 14, 17, 8, 6, 25}, we take 25 as **pivot**. So after the first pass, the list will be changed like this.

{6 8 17 14 **25** 63 37 52}

Hence after the first pass, pivot will be set at its position, with all the elements **smaller** to it on its left and all the elements **larger** than to its right. Now 6 8 17 14 and 63 37 52 are considered as two separate sunarrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.