DAY-19

Sorting Algorithms

A sorting algorithm is used to rearrange a given array or list elements according to comparison operator on the elements.

→ the comparision operator is used to decide the new order of element in the respective data structure.

Example:-

input→ P i k a c h u

sorted→ P a c h i k u

* the character with lesser ASCII value will be placed first than the character with higher ASCII value

Quicksort:-

→ an in place sorting algoritum with worst case time complexity of $\boxed{n^2}$.

Implementation:-

→ can be implemented both iteratively and recursively.

But:→ recursive is more convenient, intuitive and simplistic. Not, iterative.

three parts:

1. partitioning the array (list) about the pivot
2. Passing the smaller arrays (lists) to the recursive calls.
3. joining the sorted lists that are returned from the recursive call and the pivot.

To-do:-

→ will use lets use the first element as pivot

→ Elements smaller than or equal to the pivot will go to the left of the frontier (imagine a frientier next to the pivot).

→ while greater ones will stay at right.

-> traverse till the end of the list

Now: -> swap the pivot with the
the element just before the frontier.

*-> left and right (to pivot)
arrays are passed to
the recursive call.

-> then, finally join the
sorted arrays (pivot will be
in between them.

Code:

```
def QuickSort(arr):
    elements = len(arr)
    if elements < 2 : # Base case
        return arr
    curr-pos = 0   # pos of the partition ele.
    for i in range(1, elements): # partition loop
        if arr[i] <= arr[0] :
            curr-pos + 1
            temp = arr[i]
            arr[i] = arr [curr_pos]
            arr[curr_pos] = temp

    temp = arr [0]
    arr [0] = arr [curr-pos]
    arr [curr_pos = temp

    left = QuickSort (arr[0:curr_pos])
    right = QuickSort(arr[cur-pos : 0])

    arr = left + [arr [curr-pos]] + right

    return arr
```