

Objective: Implementation of Quick Sort

Quick Sort

Quicksort is an algorithm based on divide and conquer approach in which the array is split into subarrays and these sub-arrays are recursively called to sort the elements.

Quick Sort Algorithm:

```
quickSort(array, leftmostIndex, rightmostIndex)

  if (leftmostIndex < rightmostIndex)

    pivotIndex <- partition(array, leftmostIndex, rightmostIndex)

    quickSort(array, leftmostIndex, pivotIndex)

    quickSort(array, pivotIndex + 1, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)

  set rightmostIndex as pivotIndex

  storeIndex <- leftmostIndex - 1

  for i <- leftmostIndex + 1 to rightmostIndex

    if element[i] < pivotElement

      swap element[i] and element[storeIndex]

      storeIndex++

  swap pivotElement and element[storeIndex+1]

  return storeIndex + 1
```

Code:

```
def Partition(array, low, high):

    p = array[high]

    i = low - 1

    print(p, " is the pivot.")

    for j in range(low, high):

        if array[j] <= p:

            i = i + 1

            (array[i], array[j]) = (array[j], array[i])

    (array[i + 1], array[high]) = (array[high], array[i + 1])

    print(array)

    return i + 1


def QuickSort(array, low, high):

    if low < high:

        pi = Partition(array, low, high)

        QuickSort(array, low, pi - 1)

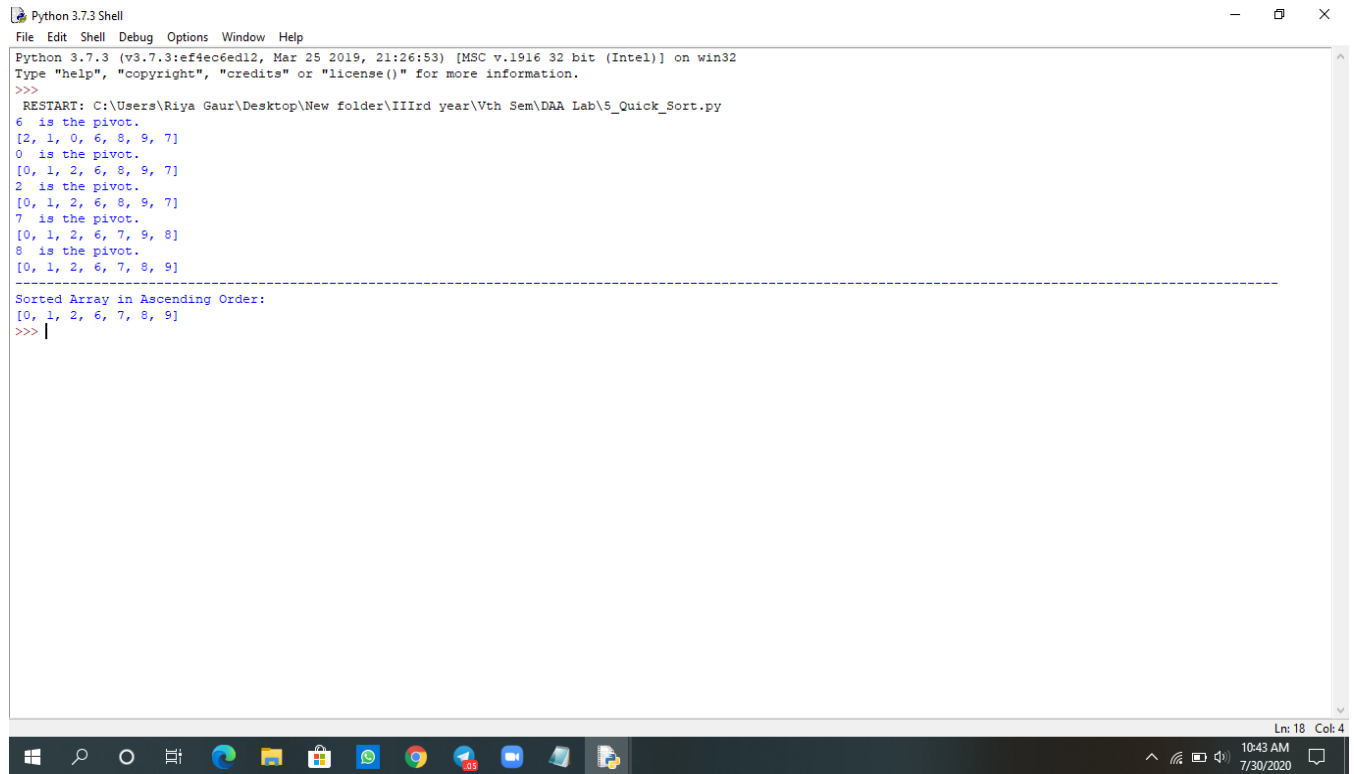
        QuickSort(array, pi + 1, high)


data = [8, 7, 2, 1, 0, 9, 6]

size = len(data)

QuickSort(data, 0, size - 1)
```

```
print("-----")
print('Sorted Array in Ascending Order:')
print(data)
```



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:\Users\Riya Gaur\Desktop\New folder\IIIrd year\Vth Sem\DAA Lab\5_Quick_Sort.py
6 is the pivot.
[2, 1, 0, 6, 8, 9, 7]
0 is the pivot.
[0, 1, 2, 6, 8, 9, 7]
2 is the pivot.
[0, 1, 2, 6, 8, 9, 7]
7 is the pivot.
[0, 1, 2, 6, 7, 9, 8]
8 is the pivot.
[0, 1, 2, 6, 7, 8, 9]
-----
Sorted Array in Ascending Order:
[0, 1, 2, 6, 7, 8, 9]
>>> |
```

Output:

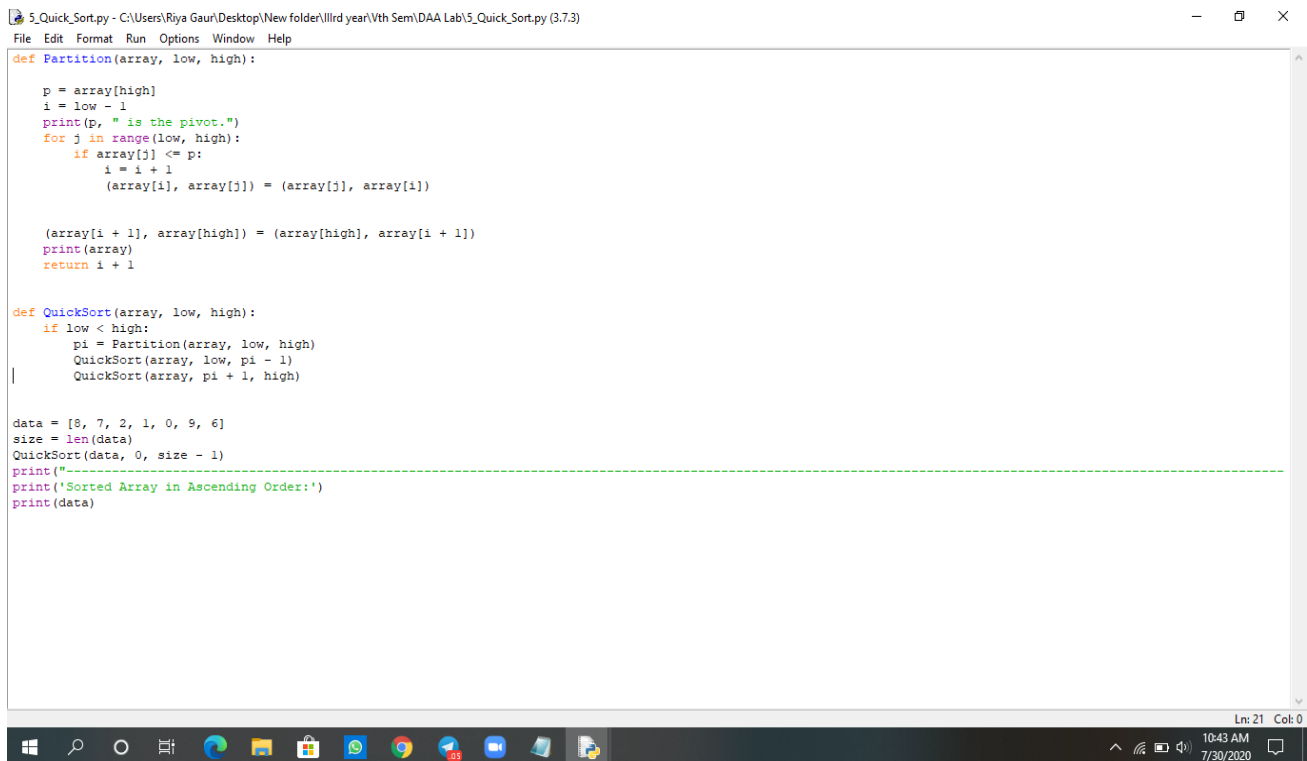
```
6 is the pivot.
[2, 1, 0, 6, 8, 9, 7]
0 is the pivot.
[0, 1, 2, 6, 8, 9, 7]
2 is the pivot.
[0, 1, 2, 6, 8, 9, 7]
7 is the pivot.
[0, 1, 2, 6, 7, 9, 8]
```

8 is the pivot.

[0, 1, 2, 6, 7, 8, 9]

Sorted Array in Ascending Order:

[0, 1, 2, 6, 7, 8, 9]



```
5_Quick_Sort.py - C:\Users\Riya Gaur\Desktop\New folder\lllrd year\lVth Sem\DAA Lab\5_Quick_Sort.py (3.7.3)
File Edit Format Run Options Window Help

def Partition(array, low, high):
    p = array[high]
    i = low - 1
    print(p, " is the pivot.")
    for j in range(low, high):
        if array[j] <= p:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])

    (array[i + 1], array[high]) = (array[high], array[i + 1])
    print(array)
    return i + 1

def QuickSort(array, low, high):
    if low < high:
        pi = Partition(array, low, high)
        QuickSort(array, low, pi - 1)
        QuickSort(array, pi + 1, high)

data = [8, 7, 2, 1, 0, 9, 6]
size = len(data)
QuickSort(data, 0, size - 1)
print("-----")
print('Sorted Array in Ascending Order:')
print(data)
```

Time Complexities:

- **Worst Case Complexity [Big-O]:** $O(n^2)$

It occurs when the pivot element picked is either the greatest or the smallest element.

This condition leads to the case in which the pivot element lies in an extreme end of the sorted array. One sub-array is always empty and another sub-array contains $n -$

1 elements. Thus, quicksort is called only on this sub-array.

However, the quick sort algorithm has better performance for scattered pivots.

- **Best Case Complexity [Big-omega]:** $O(n \log n)$

It occurs when the pivot element is always the middle element or near to the middle element.

- **Average Case Complexity [Big-theta]:** $O(n \log n)$

It occurs when the above conditions do not occur.

Quick Sort Applications

Quicksort is implemented when

- the programming language is good for recursion
- time complexity matters
- space complexity matters