What is QuickSort in Python?

Quicksort is a popular sorting algorithm that sorts an array using a divide-and-conquer strategy, which implies that it sorts an array by repeatedly dividing it into smaller subarrays and then sorting the subarrays recursively.

It is one of the most effective data-sorting algorithms available. It is typically faster than other sorting algorithms such as merge sort for large arrays. In practical applications, it outperforms other sorting algorithms such as Bubble Sort and Insertion Sort despite having a worst-case time complexity of O(n^2).

The basic idea behind the QuickSort algorithm is to divide an array into two subarrays, one containing all the elements smaller than the pivot element and the other containing all the elements larger than the pivot element.

Frequently, the median of the array serves as the pivot point. After the array has been partitioned, the QuickSort algorithm sorts the two subarrays recursively. All subarrays are emptied sequentially until the entire array is sorted.

QuickSort Algorithm

Listed below are the components of the QuickSort algorithm:

1. A pivot element is used to divide a larger array into subarrays.
2. When dividing the array in half, the pivot should be positioned such that elements smaller than the pivot are maintained on the left and elements larger than the pivot are maintained on the right.
3. The same technique is used to separate the left and right subarrays.
4. This procedure is repeated until every subarray contains a single value.
5. Currently, the components have been sorted. The items are finally combined into a single sorted array.

1. Time Complexities

- **Worst Case Complexity [Big-O]**: `O(n²)`
  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 quicksort 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.

2. Space Complexity

The space complexity for quicksort is `O(log n)`.

```python
def quicksort(arr):
    if len(arr) <= 1:
     return arr
    else:
        pivot = arr[0]
        left = [x for x in arr[1:] if x < pivot]
        right = [x for x in arr[1:] if x >= pivot]
        return quicksort(left) + [pivot] + quicksort(right)
arr = [5, 1, 8, 3, 2]
print("original array:", arr)
sorted_arr = quicksort(arr)
print("Sorted array:", sorted_arr)
```

```python
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        left = [x for x in arr[1:] if x < pivot]
        right = [x for x in arr[1:] if x >= pivot]
        return quicksort(left) + [pivot] + quicksort(right)
user_input = input("Enter the array elements separated by spaces: ")
arr = [int(x) for x in user_input.split()]
sorted_arr = quicksort(arr)
print("Sorted array:", sorted_arr)
```