Quick Sort

Quick Sort is one of the different Sorting Technique which is based on the concept of Divide and Conquer, just like merge sort. But in quick sort all the heavy lifting (major work) is done while dividing the array into subarrays, while in case of merge sort, all the real work happens during merging the subarrays. In case of quick sort, the combine step does absolutely nothing.

This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.

Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer: Recursively, sort two subarrays with Quicksort.

Combine: Combine the already sorted array.

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

It is also called partition-exchange sort. This algorithm divides the list into three main parts:

1. Elements less than the Pivot element

2. Pivot element(Central element)

3. Elements greater than the pivot element

Pivot element can be any element from the array, it can be the first element, the last element or any random element. Here, we will take the rightmost element or the last element as pivot.

Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows -

1. Pivot can be random, i.e. select the random pivot from the given array.

2. Pivot can either be the rightmost element or the leftmost element of the given array.

3. Select median as the pivot element.

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 subarrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.

- **How Quick Sorting Works?**

Following are the steps involved in quick sort algorithm:

After selecting an element as pivot, which is the last index of the array in our case, we divide the array for the first time.

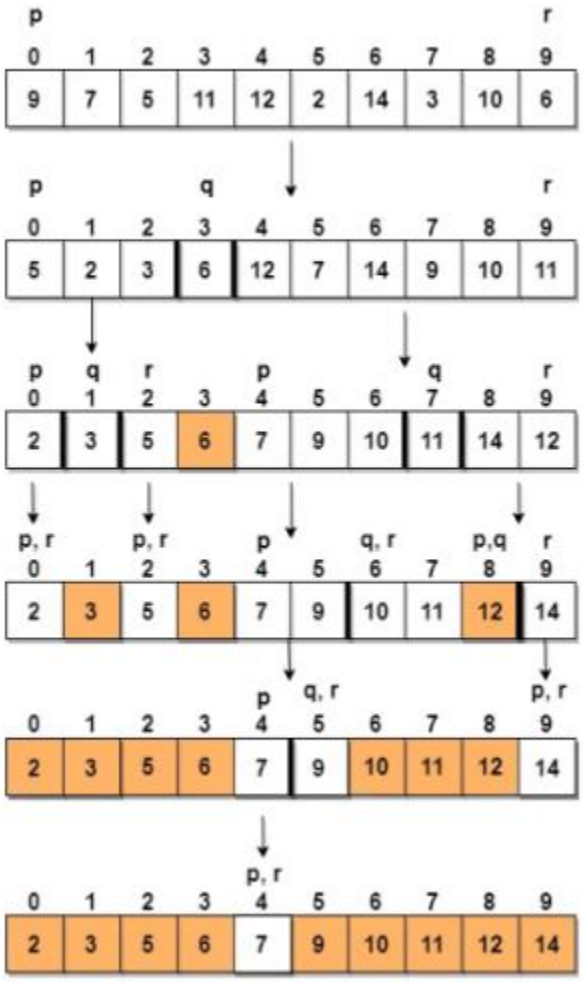
In quick sort, we call this partitioning. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the pivot will be on the left side of the pivot and all the elements greater than the pivot will be on the right side of it.

And the pivot element will be at its final sorted position.

The elements to the left and right, may not be sorted.

Then we pick subarrays, elements on the left of pivot and elements on the right of pivot, and we perform partitioning on them by choosing a pivot in the subarrays.

Let's consider an array with values {9, 7, 5, 11, 12, 2, 14, 3, 10, 6}
Below, we have a pictorial representation of how quick sort will sort the given array.



In step 1, we select the last element as the pivot, which is 6 in this case, and call for partitioning, hence re-arranging the array in such a way that 6 will be placed in its final position and to its left will be all the elements less than it and

to its right, we will have all the elements greater than it.

Then we pick the subarray on the left and the subarray on the right and select a pivot for them, in the above diagram, we chose 3 as pivot for the left subarray and 11 as pivot for the right subarray.

And we again call for partitioning.

- **Program of Quick sort**

```
#include <iostream>
using namespace std;

class Quick {
public:
   int partition(int a[], int start, int end) {
      int pivot = a[end];
      int i = start - 1;

      for (int j = start; j <= end - 1; j++) {
         if (a[j] < pivot) {
            i++;
            int temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }

      int temp = a[i + 1];
      a[i + 1] = a[end];
      a[end] = temp;

      return (i + 1);
   }

   void quick(int a[], int start, int end) {
      if (start < end) {
         int p = partition(a, start, end);
```

```
            quick(a, start, p - 1);
            quick(a, p + 1, end);
        }
    }

    void printArr(int a[], int n) {
        for (int i = 0; i < n; i++)
            cout << a[i] << " ";
    }
};

int main() {
    int a[] = {13, 18, 27, 2, 19, 25};
    int n = sizeof(a) / sizeof(a[0]);

    Quick q1;
    cout << "Before sorting array elements are - " << endl;
    q1.printArr(a, n);

    q1.quick(a, 0, n - 1);

    cout << "\nAfter sorting array elements are - " << endl;
    q1.printArr(a, n);
    cout << endl;

    return 0;
}
```

Note: Quick sort is not a stable sorting technique, so it might change the occurrence of two similar elements in the list while sorting.

- **Program of Quick sort (Strings)**

```
#include <stdio.h>
#include <string>
#include <iostream>
using namespace std;
```

```
void quickSort(std::string & str, int s, int e) {
  int start = s, end = e;
  // Set the pivot at mid.
  int pivot = str[(start + end) / 2];

  do {
    while (str[start] < pivot)
      start++;

    while (str[end] > pivot)
      end--;

    if (start <= end) {
      swap(str[start], str[end]);
      start++;
      end--;
    }
  } while (start <= end);

  /* recursion */

  if (s < end) {
    quickSort(str, s, end);
  }

  if (start < e) {
    quickSort(str, start, e);
  }
}

int main() {
  string str;
  cout << "Enter a string : ";
  cin >> str;
  quickSort(str, 0, (int)str.size() - 1);
  cout << "The sorted string is: " << str;
}
```