Quick Sort Algorithm:

By,

Prof. Pramod Nath

Assistant Professor

IT Department

KIET Ghaziabad

Contents:

- Introduction
- Algorithm
- Questions
- C program with output
- Efficiency
- Applications

Introduction:

- Fastest algorithm
- Developed by ***C.A.R. Hoare***
- Also known as ***Partition Exchange Sort***
- Internal sorting algorithm
- With time complexity O(n log n)
- Uses partitioning and recursion technique
- Makes use of a Pivot element for comparison purpose each time

Quick Sort:

- This sorting algorithm uses the idea of divide and conquer.
- It finds the element called **pivot which divides** the array into two halves in such a way that elements in the left half are smaller than pivot and elements in the right half are greater than pivot.

Quick Sort: (Basic Idea)

Three steps

- Find pivot that divides the array into two halves.
- Apply *Quick sort* in the left half.
- Apply *Quick sort* in the right half.

Quick Sort(Rule):

Remember this rule:

- All elements to the **RIGHT of pivot are GREATER than pivot.**
- All elements to the **LEFT of pivot are SMALLER than pivot.**

Algorithm:

```
void quick_sort(int a[ ],int p,int q)
{
    int m;
    if(p < q)
    {
        m = partition(a, p, q);
        quick_sort(a, p, m-1);
        quick_sort(a, m+1, q);
    }
}
```

Algorithm(Partition):

```
int partition(int a[ ],int p,int q)
{
    int pivot, i, j;
    pivot = a[p];
    i = p;
    for(j = p+1; j <= q; j++)
    {
          if (a[j] <= pivot)
                    {
                    i = i + 1;
                    swap(&a[i], &a[j]);
                    }
      }
    swap (&a[p], &a[i]);
    return (i);
}
```

Questions:

Q1. Sort 11, 2, 9, 13, 57, 25, 17, 1, 90, 3.

Q2. Sort 27, 35, 48, 64, 15, 5, 58, 62, 20, 19.

Q3. Sort 70, 39, 45, 88, 94, 27, 68, 77, 24, 120, 130.

```
#include <stdio.h>

void quick_sort(int[ ], int, int);
int partition(int[ ], int, int);
int swap(int*, int*);

int main( )
{
        int a[50], n, i;
        printf("How many elements?");
        scanf("%d", &n);
        printf("\nEnter array elements:");

        for(i=0; i<n; i++)
                scanf("%d",&a[i]);

        quick_sort(a, 0, n-1);
        printf("\nArray after sorting:");

        for(i=0; i<n; i++)
                printf("%d ",a[i]);

        return 0;
}
```

```
void quick_sort(int a[ ],int p,int q)
{
        int m;
        if(p < q)
        {
                m=partition(a, p, q);
                quick_sort(a, p, m-1);
                quick_sort(a, m+1, q);
        }
}
```

```
int partition(int a[ ],int p,int q)
{
        int pivot,i,j;
        pivot = a[p];
        i = p;
        for(j = p+1; j <= q; j++)
        {
                if (a[j] <= pivot)
                        {
                        i = i + 1;
                        swap(&a[i], &a[j]);
                        }
        }
        swap (&a[p], &a[i]);
        return (i);
}
```

```
int swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

```
How many elements?10

Enter array elements:11 2 9 13 57 25 17 1 90 3

Array after sorting:1 2 3 9 11 13 17 25 57 90
--------------------------------
Process exited after 20.79 seconds with return value 0
Press any key to continue . . .
```

Complexity and Efficiency:

Average Case	Best Case	Worst Case
O(n log n)	O(n log n)	$O(n^2)$
Efficiency of Quick sort depends upon the Element that is chosen as the ***Pivot***		
Worst case------ occurs when ***1. Array is already sorted either in ascending or descending order*** ***2. Left-most element is chosen as Pivot***		

Advantages & Applications:

- Faster than bubble, selection and insertion sort techniques
- Used to sort-------arrays of small size, medium size or large size

THANK YOU !