QUICK_SHORT

BASIC IDEA:

The key process in **quickSort** is a **partition()**. The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.

Time Complexity:

- **Best Case**: Ω (N log (N))
  The best-case scenario for quicksort occur when the pivot chosen at the each step divides the array into roughly equal halves.
  In this case, the algorithm will make balanced partitions, leading to efficient Sorting.
- **Average Case: θ (N log (N))**
  Quicksort's average-case performance is usually very good in practice, making it one of the fastest sorting Algorithm.
- **Worst Case: O(N2)**
  The worst-case Scenario for Quicksort occur when the pivot at each step consistently results in highly unbalanced partitions. When the array is already sorted and the pivot is always chosen as the smallest or largest element. To mitigate the worst-case Scenario, various techniques are used such as choosing a good pivot (e.g., median of three) and using Randomized algorithm (Randomized Quicksort) to shuffle the element before sorting.
- **Auxiliary Space:** O(1), if we don't consider the recursive stack space. If we consider the recursive stack space then, in the worst case quicksort could make $O(N)$.

Algorithm:

```
QUICKSORT (array A, start, end)
{
 1  if (start < end)
 2  {
3  p = partition(A, start, end)
4  QUICKSORT (A, start, p - 1)
5  QUICKSORT (A, p + 1, end)
6  }
 }
```

CODE:

```
#include<stdio.h>
void swap( int *a, int *b)
{
```

```
int temp;
temp=*a;
*a=*b;
*b=temp;
}

int partition(int arr[],int p,int r){
int x,i,j;
x=arr[r];
i=p-1;
for(j=p;j<=r-1;j++){
if(arr[j]<=x){
i++;
swap(&arr[i],&arr[j]);
}
}
swap(&arr[i+1],&arr[r]);
return i+1;
}

void quick_Sort(int arr[],int p,int r){
int q;
if(p<r){
q=partition(arr,p,r);
quick_Sort(arr,p,q-1);
quick_Sort(arr,q+1,r);
}

}

int main(){
int n,i,p,r;
printf("Enter the toal number of elements:");
scanf("%d",&n);
int arr[n];
for (i=0; i<n; i++) {
printf("arr[%d]:",i);
scanf("%d",&arr[i]);
}

p=0;
r=(sizeof(arr)/sizeof(int))-1;
quick_Sort(arr,p,r);
for(i=0;i<=r;i++){
```

```
printf("%d\t",arr[i]);
}
return 0;
}
```

OUTPUT: