22/10/2023

* Quick Sort →

Algorithm →

Step 1: Write partition logic to find left and right part of the array. Pivot index divides the array into two.

Step 2: Make a recursive call for left part. left part is from s → pivot − 1.

Step 3: Make a recursive call for right part. Right part is from pivot + 1 → e.

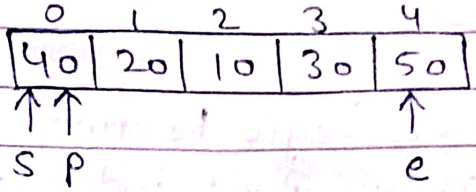
```
quickSort(arr, s, e)
{
// Base case
1. Break into left
and right using
pivot. Let pivot
be starting index.
2. Make recursive
call for left and
right.
}
```

```
partition (arr, s, e)
{
1. Take starting index as pivot.
2. Count all elements which
   are less than pivot element.
3. Place pivot element at the
   position it must be.
4. Update pivot index.
5. Place elements less than
   pivot element to left of
   pivot index and elements
   greater than pivot element
   to right of pivot index.
6. Return new pivot index.
```

__/__/__

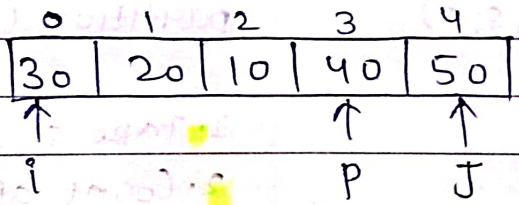
Dry Run →

array = {40, 20, 10, 30, 50}



- p = 0
- count = 3 (Count of elements less than pivot element.)
- rightIndex = s + count
  = 0 + 3
  = 3
  (rightIndex indicates right position of pivot element)

- swap(arr[p], arr[rightIndex])

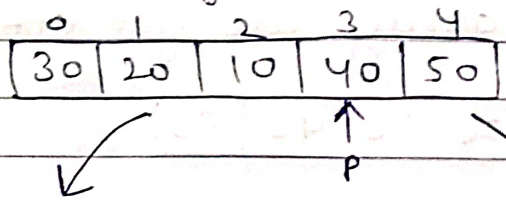


(new pivot index = rightIndex = 3)

- All elements less than pivot element are placed left to pivot element and elements greater than pivot elements are placed right to pivot element. So, no need to swap here.

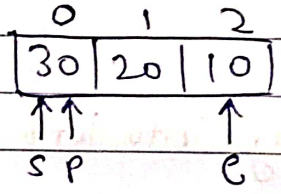
- pivotIndex = 3

Now, the array is →

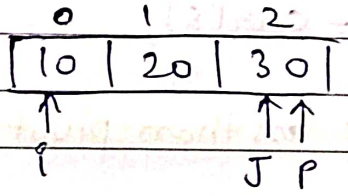


Call for left →

s=0, e=p-1=2



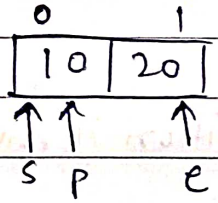
- p=0
- count=2
- rightIndex=2
- swap(arr[p], arr[rightIndex])



(new pivot index = right index = 2)

Call for left →

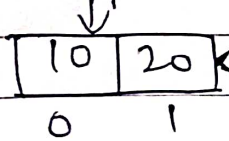
s=0, e=p-1=1



p=0, count=0

rightIndex = 0

swap



Call for right →

s=p+1, e=e

There is no such element

So, s>e (Base case) return.

Call for right → s=p+1=4, e=4

50 → Base case

So, return

Call for left → s=p, e=p-1

(s>e) return.

Call for right ⇒ s=p+1=1, e=1

20 → s==e, return

So, the final array we got after sorting is →

10	20	30	40	50

Code →

```
#include <bits/stdc++.h>
using namespace std;

// this function divides the array into left and
// right using pivot

    int partition (int arr[], int s, int e){
// let's take starting index as pivot
        int pivotIndex = s;
        int pivotElement = arr[s];

// count all elements less than pivot element
        int count = 0;
        for (int i= s+1; i<=e; i++){
            if (arr[i] <= pivotElement) {
                count++;
            }
        }

// place pivot element at the position it must be have
        int rightIndex = s + count;
        swap ( arr[pivotIndex], arr[rightIndex]);
        // update pivot index
        pivotIndex= rightIndex;
```

```
// pivot element se chotte uske left mai and
// bade right mai krdo
    int i = s;
    int j = e;
    while (i < pivotIndex && j > pivotIndex){
        while (arr[i] <= pivotElement){
            i++;
        }
        while (arr[j] > pivotElement){
            j--;
        }
        if (i < pivotIndex && j > pivotIndex){
            swap (arr[i], arr[j]);
        }
    }
    return pivotIndex;
}

void quickSort (int arr[], int s, int e){
    // base case
    if (s >= e){
        // in case of empty array & single element
        return;
    }

    // partition logic
    int pivot = partition (arr, s, e);

    // recursive call for left ->
    quickSort (arr, s, pivot-1);
    // recursive call for right ->
    quickSort (arr, pivot+1, e);
}
```

```
int main(){
    int arr[]={40, 20, 10, 30, 50};
    int size = 5;
    int s=0;
    int e = size-1;
    quickSort(arr, s, e);
    for(int i=0; i<size; i++){
        cout<<arr[i]<<" ";
    }
}
```

Time complexity → (worst case)

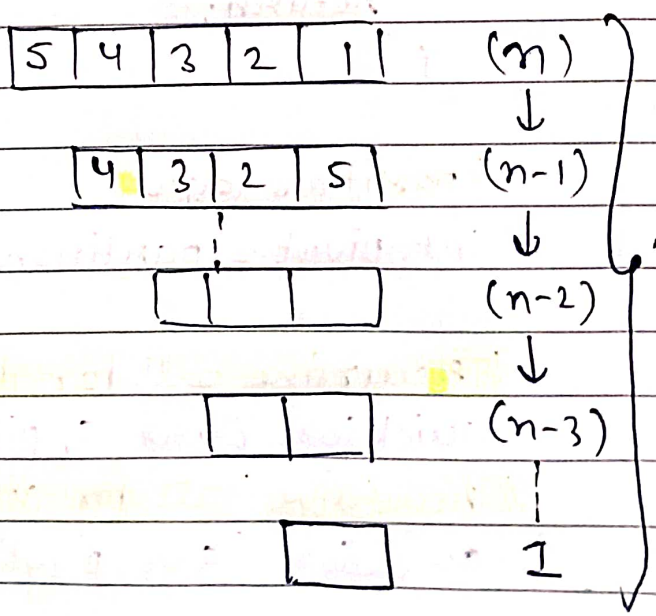
for partitioning logic →

O(n) → Comparing whole elements in array with pivot elements. That's why traversing whole array.

O(n) → for arranging elements to the left and right of the pivot element.

In worst case, elements are arranged in descending order →

for instance,



So, partitioning of array is occuring n times and every partition takes n time.

That's why $T(n) = O(n^2)$

Space complexity → O(n) [stack space]

* Another approach → Quick Sort

Code →

```
#include <bits/stdc++.h>
using namespace std;

void quickSort(int arr[], int start, int end){
    // base case
    if (start >= end) return;

    int i = start-1;
    int j = start;

    // let's take end as pivot
    int pivot = end;
    while (j < pivot){
        if (arr[j] < arr[pivot]){
            i++;
            swap(arr[i], arr[j]);
        }
        j++;
    }
    i++;
    swap(arr[i], arr[pivot]);
```

```
// recursive call for left half →
    quickSort ( arr, start, i-1);
// recursive call for right half →
    quickSort (arr, i+1, end);
}
int main() {
    int arr[] = {9, 2, 7, 4};
    int size = 4;
    int s = 0;
    int e = size - 1;
    quickSort (arr, s, e);
    for (auto num: arr){
        cout <<num << " ";
    }
}
```