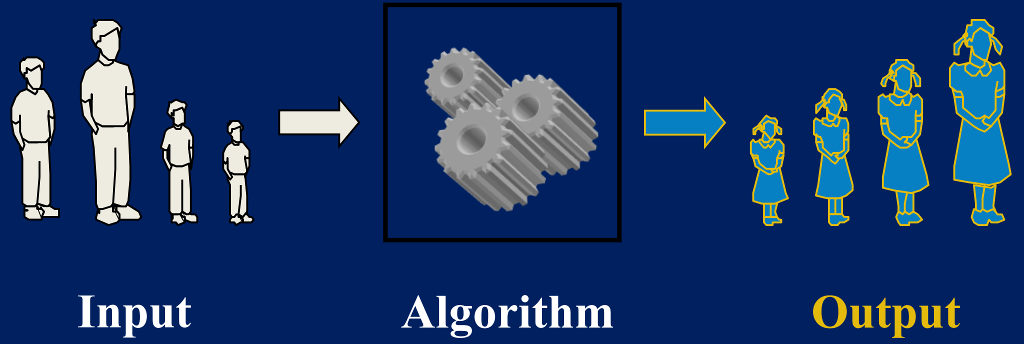
Chapter 5: Divide and Conquer

Quick Sort



Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

Divide and Conquer

- Sorting Algorithms
  - Bubble sort
  - Selection sort
  - Insertion Sort
  - **Quick Sort**
  - **Merge Sort**

Divide-Conquer Approach

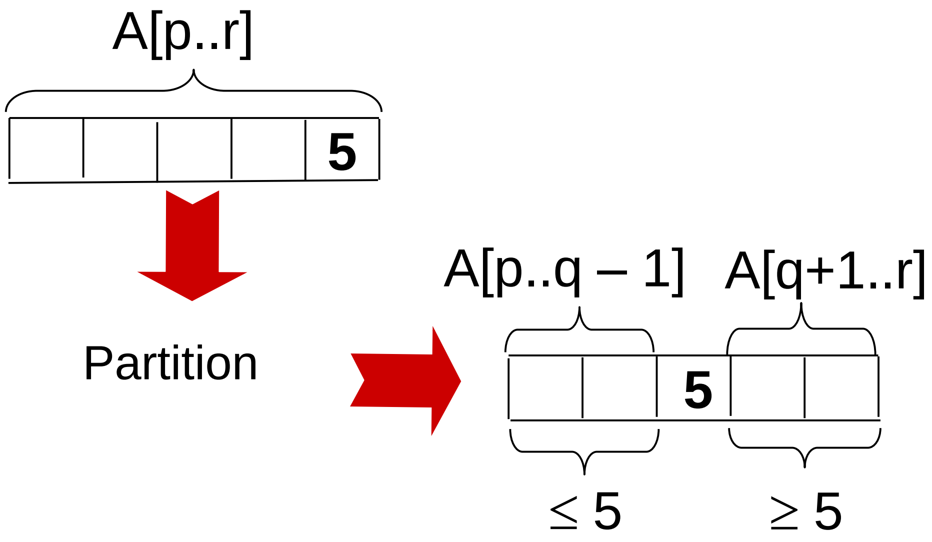
Divide and Conquer: Quick Sort

- Follows the **divide-and-conquer** paradigm.
- ***Divide***: Partition (separate) the array $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$.
  - Each element in $A[p..q-1] < A[q]$.
  - $A[q] <$ each element in $A[q+1..r]$.
  - Index q is computed as part of the partitioning procedure.
- ***Conquer***: Sort the two subarrays by recursive calls to quicksort.
- ***Combine***: The subarrays are sorted in place – no work is needed to combine them.
- **How do the divide and combine steps of quicksort compare with those of merge sort?**

Divide and Conquer: Quick Sort ...

<u>Quicksort(A, p, r)</u>

```
Quicksort(A, p, r)
    if p < r then
        q := Partition(A, p, r);
        Quicksort(A, p, q – 1);
        Quicksort(A, q + 1, r)
```




```
Partition(A, p, r)
    x := A[r], i:= p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
        {
            i := i + 1;
            A[i] ↔ A[j];
        }
    A[i + 1] ↔ A[r];
    return i + 1
```

Divide and Conquer: Quick Sort ...

```
                 p                                r
initially:       2  5  8  3  9  4  1  7  10  6
               i j
```

note: pivot (x) = 6 (Last No.)

```
next iteration:  2  5  8  3  9  4  1  7  10  6
                 i  j

next iteration:  2  5  8  3  9  4  1  7  10  6
                    i  j

next iteration:  2  5  8  3  9  4  1  7  10  6
                    i     j

next iteration:  2  5  3  8  9  4  1  7  10  6
                       i     j
```

```
Partition(A, p, r)
    x := A[r], i:= p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
        {
            i := i + 1;
            A[i] ↔ A[j];
        }
    A[i + 1] ↔ A[r];
    return i + 1
```

Divide and Conquer: Quick Sort ...

next iteration:	2	5	3	8	9	4	1	7	10	**6**
			i		j					
next iteration:	2	5	3	8	9	4	1	7	10	**6**
			i			j				
next iteration:	2	5	3	4	9	8	1	7	10	**6**
				i			j			
next iteration:	2	5	3	4	1	8	9	7	10	**6**
					i			j		
next iteration:	2	5	3	4	1	8	9	7	10	**6**
					i				j	
next iteration:	2	5	3	4	1	8	9	7	10	**6**
					i					j
after final swap:	2	5	3	4	1	**6**	9	7	10	8
					i					j

Partition(A, p, r)
x := A[r], i:= p – 1;
for j := p **to** r – 1 **do**
if A[j] $\leq$ x **then**
{
i := i + 1;
A[i] $\leftrightarrow$ A[j];
}
A[i + 1] $\leftrightarrow$ A[r];
return i + 1

Divide and Conquer: Quick Sort ...

Analysis of quicksort—best case

- Algorithm always chooses best pivot and splits sub-arrays in half at each recursion
  - T(0) = T(1) = O(1)
    - constant time if 0 or 1 element
  - For N > 1, 2 recursive calls plus linear time for partitioning
  - T(N) = 2T(N/2) + O(N)
    - Same recurrence relation as Mergesort
  - T(N) = O(N log N)

Divide and Conquer: Quick Sort ...

Analysis of quicksort - Worst Case

- Algorithm always chooses the worst pivot – one sub-array is empty at each recursion
  - $T(N) \leq a$ for $N \leq C$
  - $T(N) \leq T(N-1) + bN$
  - $\leq T(N-2) + b(N-1) + bN$
  - $\leq T(C) + b(C+1) + \ldots + bN$
  - $\leq a + b(C + (C+1) + (C+2) + \ldots + N)$
  - $T(N) = O(N^2)$
- Fortunately, *average case performance* is O(N log N)

Quick Sort: Summary

Some of the important properties of Quick sort algorithm are-

- Quick sort is **not a stable sorting** algorithm because the swapping of elements is done according to pivot's position (without considering their original positions)
- Quick sort is an **in-place sorting** algorithm.
- The worst case time complexity of Quick sort algorithm is **$O(n^2)$**.
- The space complexity of Quick sort algorithm is **$O(n)$**.
- Quick sort is faster than the merge sort.

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in