TECHNO INDIA UNIVERSITY
WESTBENGAL

Data Structures & Algorithms
(PCC-CS 301)

Dr. Debashis Das
Associate Professor
Department of CSE
Techno India University, Kolkata

Topics Covered

1. Divide-and-Conquer based Sort
    1.1. Merge sort
    1.2. Quick sort

Divide & Conquer

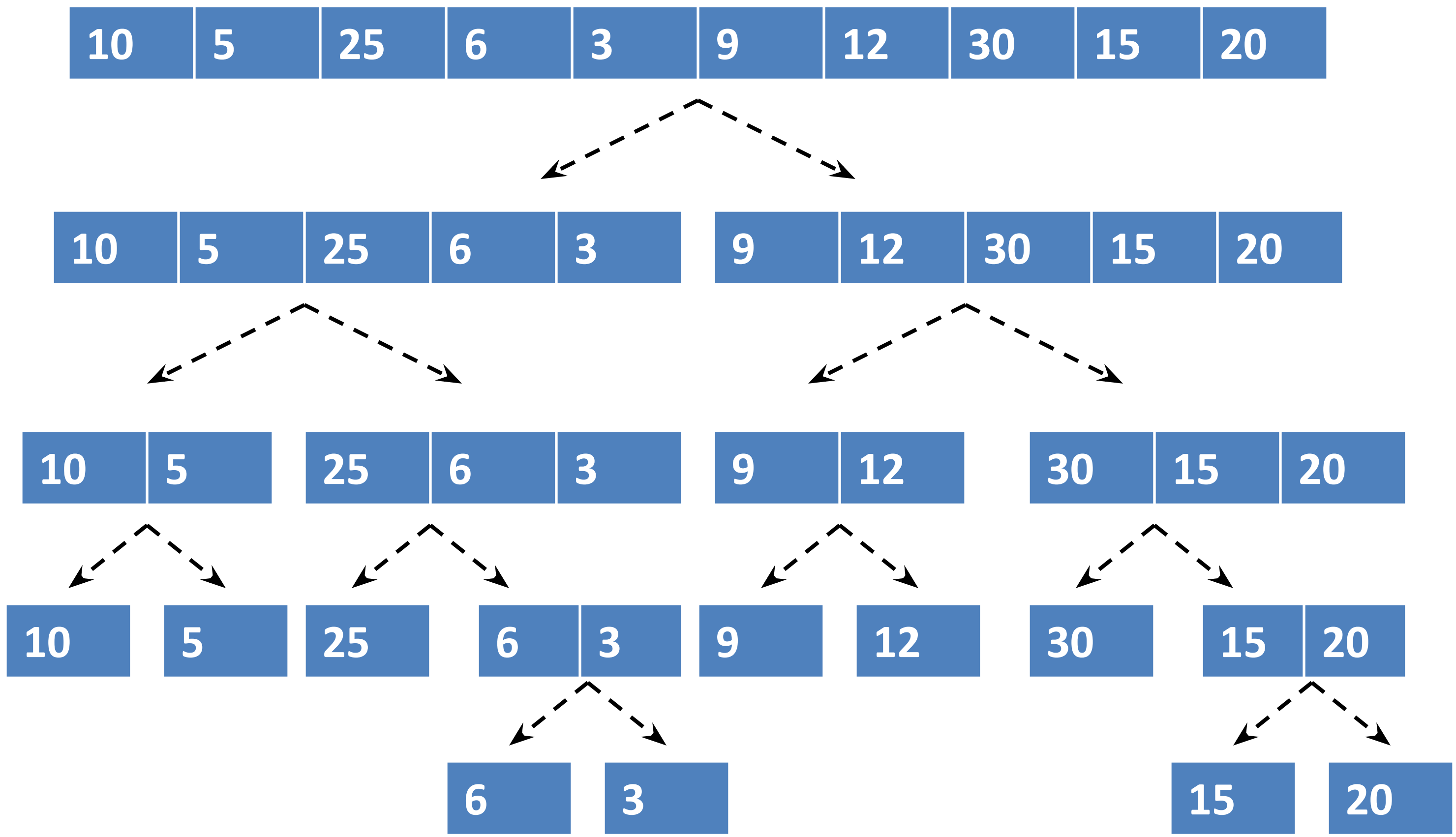
- Introduction
  - This is an algorithmic approach to solve various problems
  - It involves 3 steps to solve a problem
    - Divide : it divides the entire problem into smaller sub-problems that can be solved easily
    - Recursive solution: solve each of the sub-problems recursively
    - Combine: combine all sub-solutions to obtain the final solution
  - Data sorting is one of such problems that can be solved using this approach
    - Merge sort (recursive solve and combine performed simultaneously)
    - Quick sort (there is no specific combine step visible separately)

TECHNO INDIA UNIVERSITY
WESTBENGAL

Merge Sort

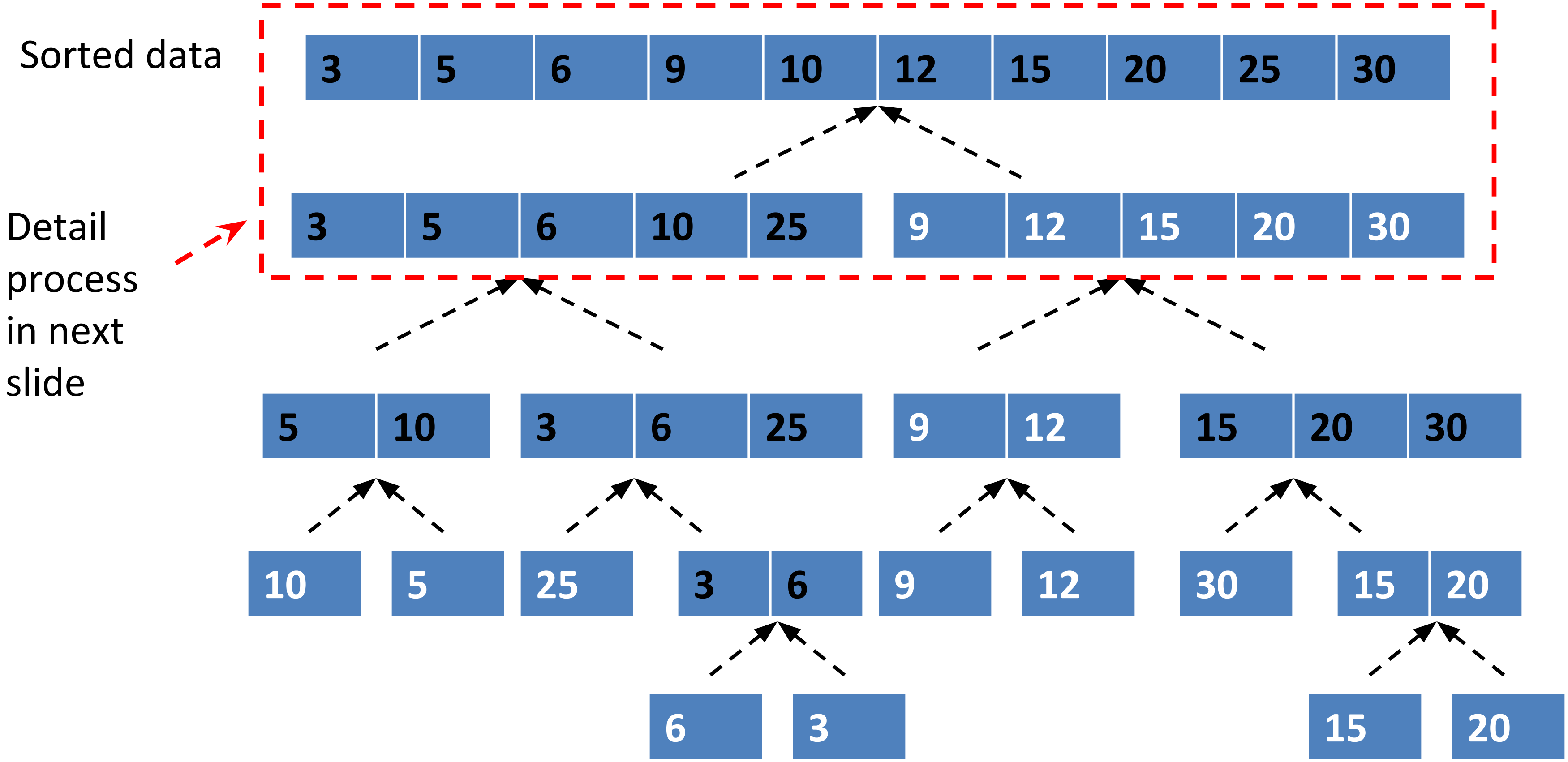
- Divide

Input data set



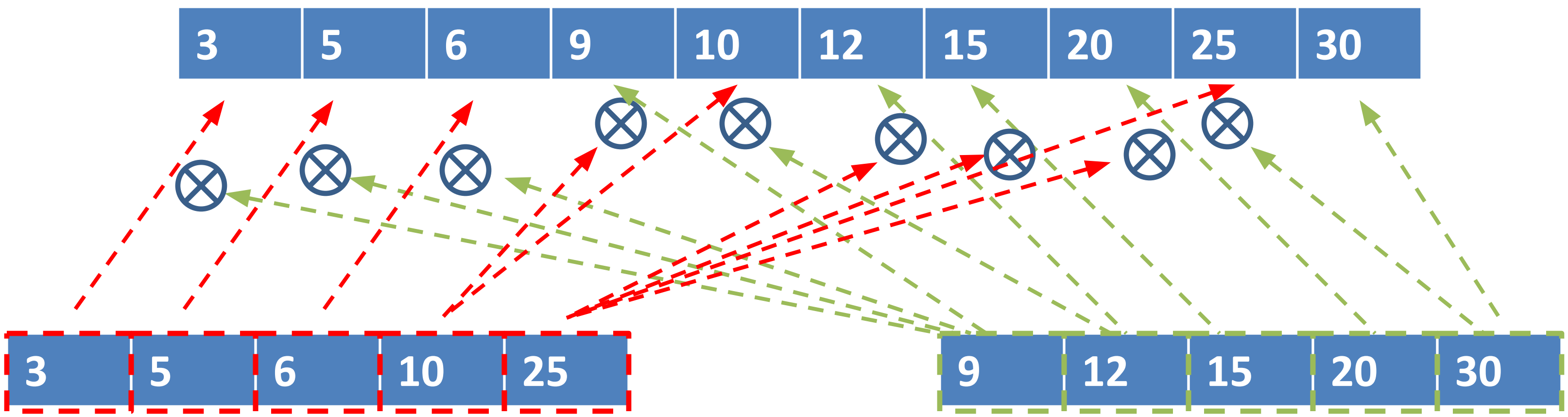
Merge Sort

- Recursive solve & Combine



Merge Sort

- Recursive solve & Combine (process)



Merge Sort

- Algorithm

```
Merge_sort( A, p, r)   // A is the array
{
  if p > r
     return
  q = (p+r)/2       // finding middle position
  Merge_sort(A, p, q)
  Merge_sort(A, q+1, r)
  Merge(A, p, q, r)
}
```

```
Calling_function
{
  Merge_sort( A, 1, len(A)) // A is the array
}
```

```
Merge( A, p, q, r)
{
  set Left array L with elements A[p] to A[q]
  set Right array R with A[q+1] to A[r]
  set S as sorted list
  while i < length(L) and j< length(R)
     if L(i) < R(j)
        S(k) := L(i)
        i:=i+1
     else
        S(k) := R(j)
        j:=j+1
   if i < length(L)
      S(k) := L(i)
   if j< length(R)
      S(k) := R(j)
}
```

Merge Sort

- Complexity

Divide step: $O(\log_2(n))$
[in each step data set are becoming half in size. If we consider the entire division tree, height of the tree will be $\log_2(n)$]

Solve & Combine: $O(n)$

- All cases: complexity

Time Complexity		
Best case	Average case	Worst case
$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$

Quick Sort

- It is another problem that uses Divide-and-Conquer approach to produce solution

- **Divide**: partition the input array ***A[p … r]*** into two sub-arrays ***A[p … q-1]*** and ***A[q+1 … r]*** such that each element of the left sub-array is less than or equal to ***A[q]*** and right sub-array is greater than or equal to ***A[q]***. This step finds the index ***q*** in each iteration
- **Conquer**: Sort the two sub-arrays A[p … q-1] and A[q+1 … r] by recursive call to quick sort
- **Combine**: As sub-arrays are already sorted, no work is performed to combine them

TECHNO INDIA UNIVERSITY
WESTBENGAL

Quick Sort

- Algorithm:
  - In each iteration, it considers an element as the ***pivot*** element
  - After each iteration, the entire array is partitioned into two sub-arrays based on the ***pivot*** element
  - At the end of a particular iteration, ***pivot*** element will be placed in it's proper place

```
Quick_sort(A,p,r) // p: start index, r: end index
 if  p<r
   q=Partition(A,p,r)
   Quick_sort(A,p,q-1)
    Quick_sort(A,q+1,r)
```

```
Partition(A,p,r)
  x=A[r]   // x is pivot element
  i=p-1
  for  j=p to r-1
     if  A[j] <= x
        i=i+1
        swap A[i] and A[j]
  swap A[i+1] and A[r]
  return i+1
```

Quick Sort

- Example:

2 8 7 1 3 5 6 4 *pivot*

i p r
2 8 7 1 3 5 6 4
j

A[j] <= x :: 2<=4
i=i+1
Swap A[i] and A[j]

P, i r
2 8 7 1 3 5 6 4
j

P, i r
2 8 7 1 3 5 6 4
j

A[j] <= x :: 8<=4 ?
No…Next iteration

P, i r
2 8 7 1 3 5 6 4
j

A[j] <= x :: 7<=4 ?
No…Next iteration

P, i r
2 8 7 1 3 5 6 4
j

P i r
2 8 7 1 3 5 6 4
j

Last step

Placed in it's actual position

p r
2 1 3 8 7 5 6 4
i

Quick Sort

- All cases: complexity

Time Complexity		
Best case	Average case	Worst case
$O(n \log(n))$	$O(n \log(n))$	$O(n^2)$

Queries?