

COL106: Data Structures and Algorithms

Lecture 7

Keerti Choudhary
Department of Computer Science, IIT Delhi

Announcements

- Java Doubt clearing session this Saturday 3-4 PM, LT 1 Block VI.

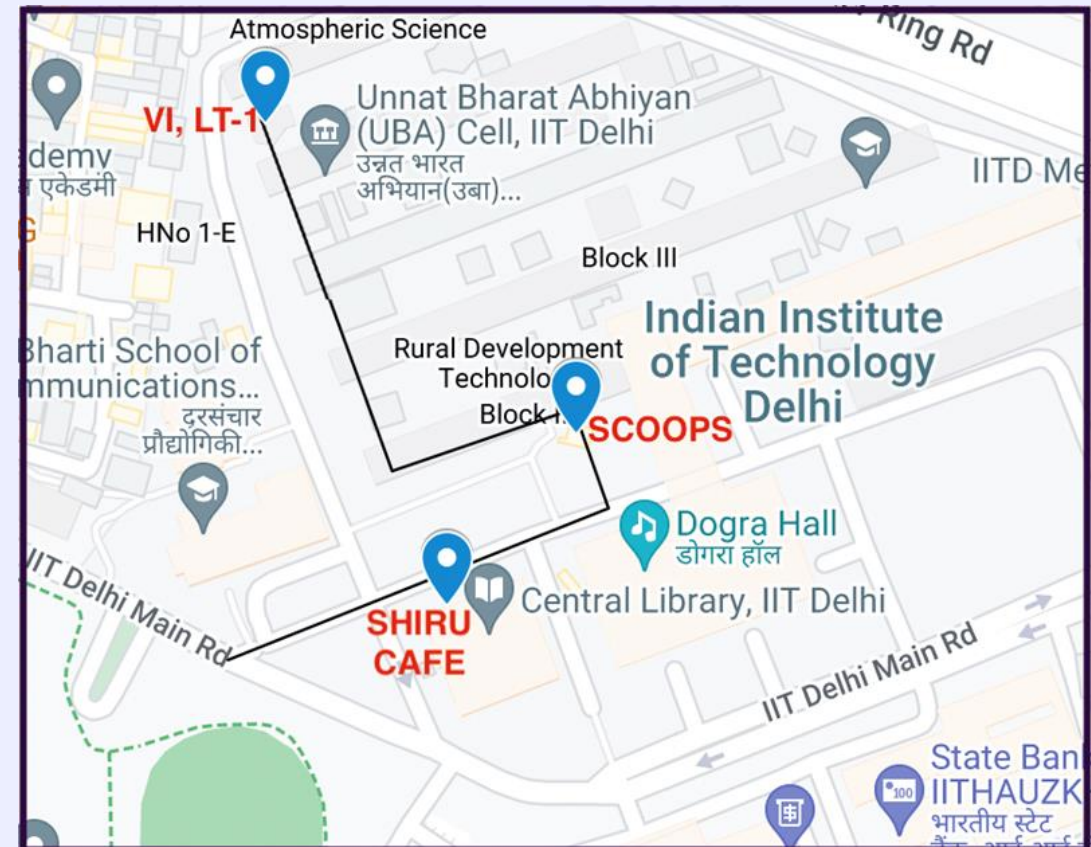
Who should attend?

Students facing difficulty in Java programming language.

By whom?

Rishi Sarraf and Raunak Jain

- First Quiz - Last week of January



Today's Lecture

Sorting algorithms and their analysis:

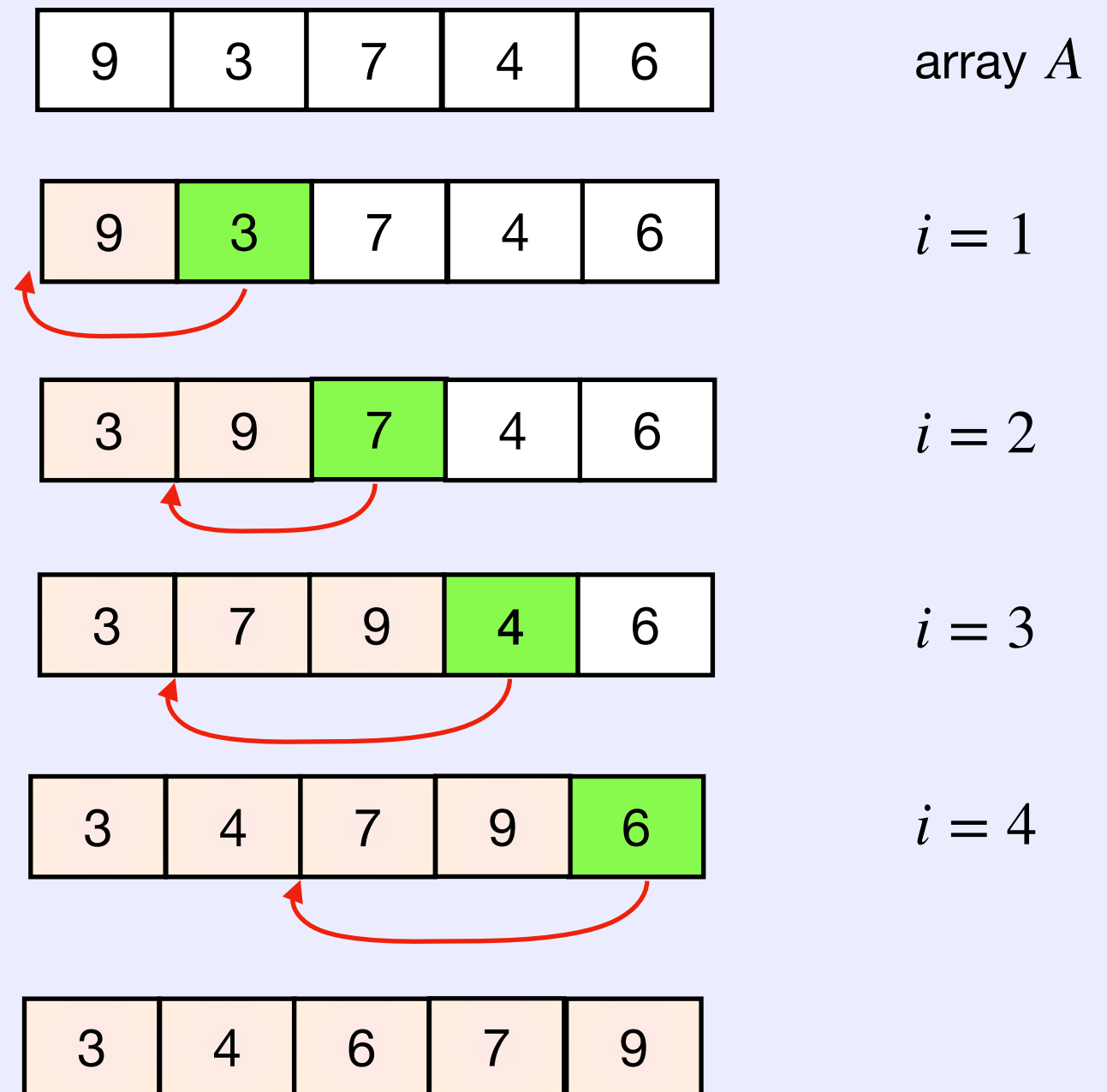
1. Insertion Sort
2. Merge Sort
3. Quick Sort

Insertion Sort

Iterative algorithm

After i^{th} round:

Ensure that elements $A[0], \dots, A[i]$ are sorted.



Insertion Sort

Goal: To sort an array A containing n integers.

InsertionSort(A):

For $i = 1$ to $n - 1$ **do**

/ Insert $A[i]$ at proper location to ensure $A[0, i]$ is sorted */*

j = smallest index that satisfies $A[j] \geq A[i]$

$temp = A[i]$

Move contents of $A[j, i - 1]$ to sub-array $A[j + 1, i]$

Set $A[j] = temp$

Return A

Remark: 1. Moving the contents of subarray $A[j, i - 1]$ takes $O(i)$ time in worst-case.
2. However, index j can be computed in $O(\log n)$ time using Binary Search.

Insertion Sort Analysis

Let $T(n)$ = number of steps taken by the algorithm

Then, $T(n) \leq (n - 1) + (n - 2) + \cdots + (3) + (2)$

So the time complexity is: $T(n) = O(n^2)$

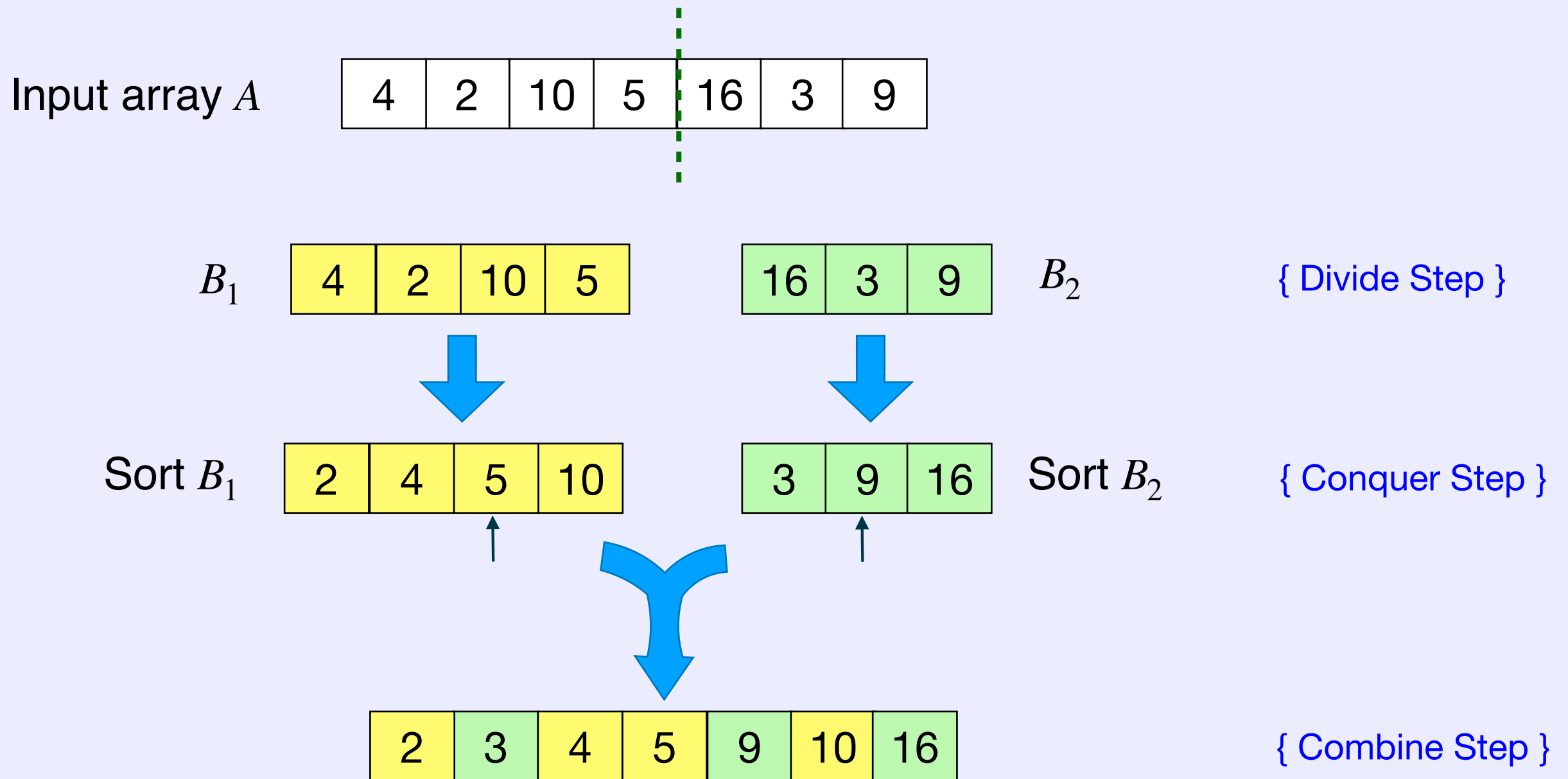
Remark:

If A is already sorted then the time taken by Insertion sort is just $O(n)$

If A is reverse-sorted then the time taken is $\Omega(n^2)$

Merge Sort

A sorting algorithm based on divide-and-conquer approach:



Merge Sort

Merge-Sort(B_1)

{ Create two new arrays $B_1 = A[0, \frac{n}{2}]$ and $B_2 = A[\frac{n}{2} + 1, n - 1]$

{ Merge-Sort(B_1)

{ Merge-Sort(B_2)

{ Set $i, j = 0$

for $k = 0$ to $n - 1$ do

if $i < \text{size}(B_1)$ and $B_1[i] \leq B_2[j]$ then

Set $A[k] = B_1[i]$ and increment i by 1 */* Copy $B_1[i]$ at k -th index of A */*

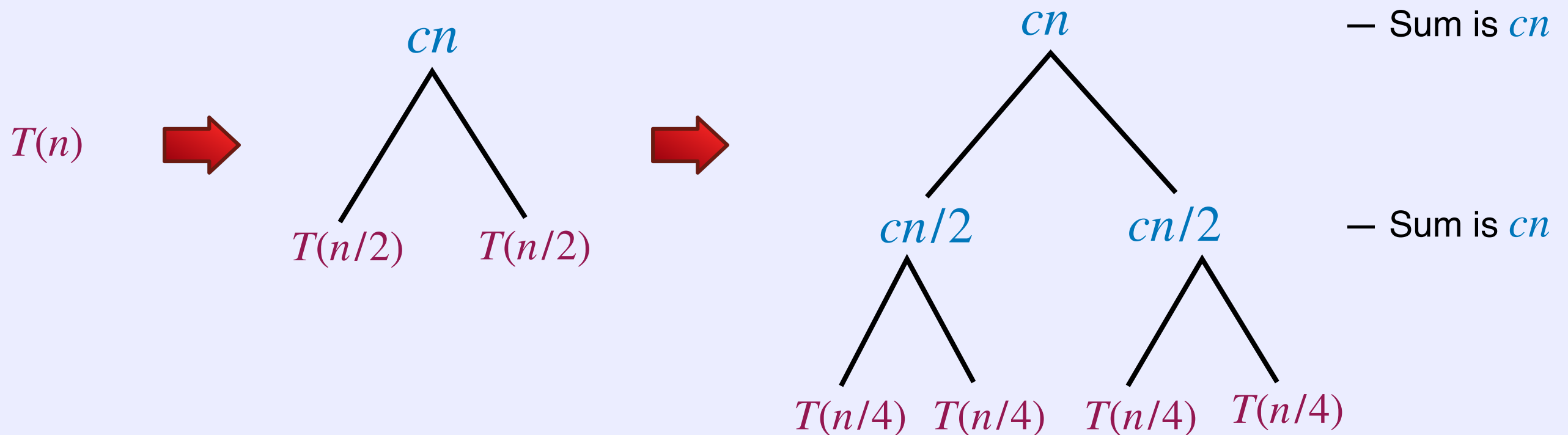
else

Set $A[k] = B_2[j]$ and increment j by 1 */* Copy $B_2[j]$ at k -th index of A */*

return A

Merge Sort Analysis

Let $T(n)$ = number of steps taken by the algorithm. Then, $T(n) \leq 2 T\left(\frac{n}{2}\right) + cn$



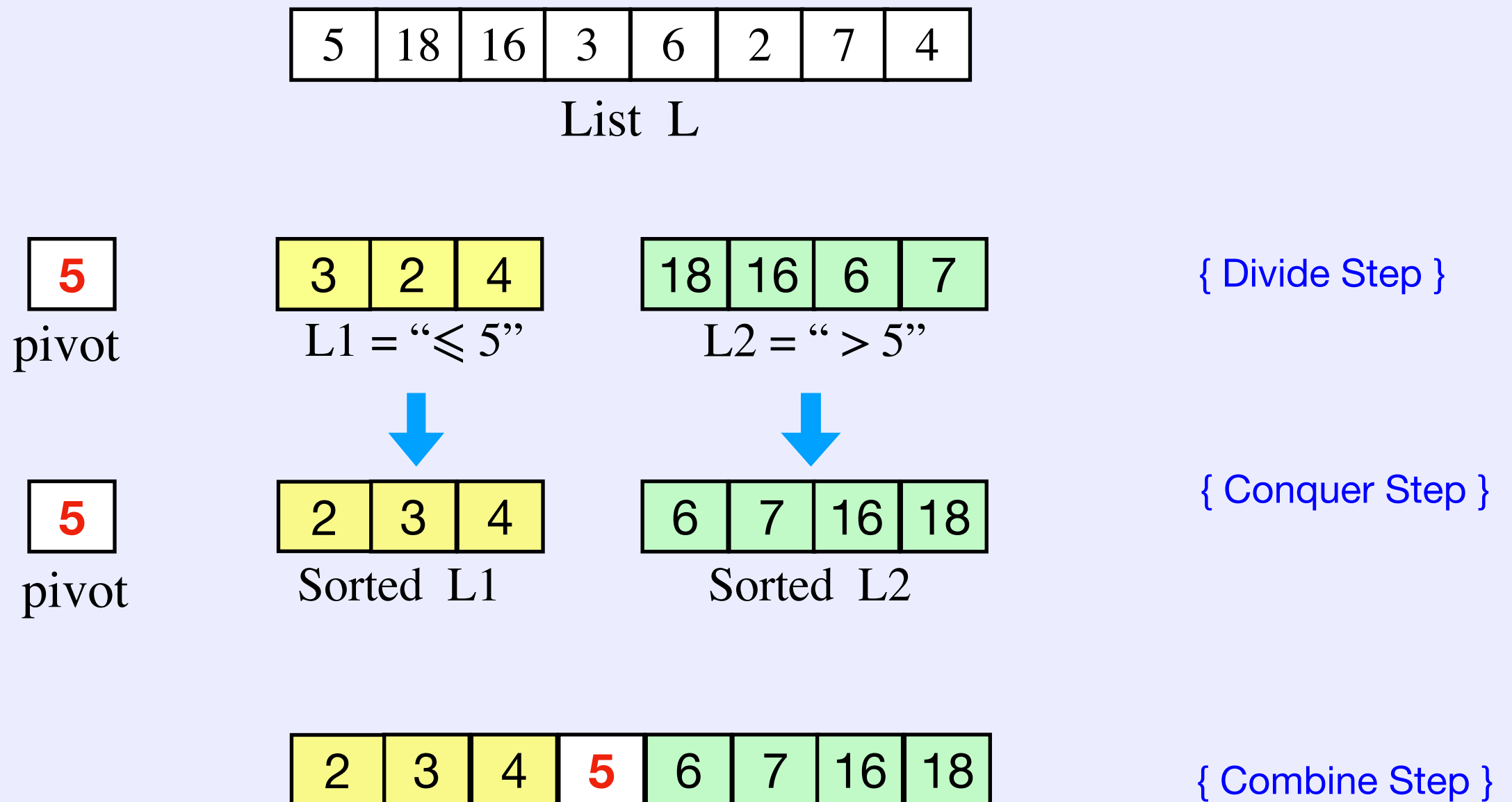
On expanding we get that tree will have $\log_2 n$ levels.

The sum of terms in each level is cn .

So, the total **time** complexity is $(cn \times \text{no. of levels}) = O(n \log n)$.

Quick Sort

Another sorting algorithm based on divide-and-conquer approach:



Quick Sort

Goal: To sort a list L containing n integers.

QuickSort(L):

pivot = Random element of list L

$L_1, L_2 \leftarrow$ empty lists

For each ($x \in L \setminus \textit{pivot}$) do

 If ($x \leq \textit{pivot}$) : $L_1.\text{append}(x)$

 If ($x > \textit{pivot}$) : $L_2.\text{append}(x)$

Return *QuickSort*(L_1) $\circ$ *pivot* $\circ$ *QuickSort*(L_2)

Question: If $T(n)$ = expected time to sort n elements, then what is recurrence relation to solve $T(n)$?

Quick Sort Analysis

Let $T(n)$ = expected number of steps taken by the algorithm. Then,

$$T(n) = \frac{1}{n} \sum_{i=1}^n \left(T(i-1) + T(n-i) \right) + n$$

Probability that pivot is
 i^{th} smallest element

Expected Time to
sort L_1 when its
size is $i-1$

Expected Time to
sort L_2 when its
size is $n-i$

Time to perform
split and merge
operations

Homework

- Solve the above recurrence relation.
- Obtain an implementation for quick sort that uses only $O(1)$ extra space.