Algorithm Analysis and Design

Divide and Conquer strategy
(Merge Sort)

Lecture -13

Overview

- Learn the technique of "divide and conquer" in the context of merge sort with analysis.

A Sorting Problem
(Divide and Conquer Approach)

- **Divide** the problem into a number of sub problems.
- **Conquer** the sub problems by solving them recursively.
  - ***Base case:*** If the sub problems are small enough, just solve them by brute force.
- **Combine** the sub problem solutions to give a solution to the original problem.

Merge sort

- A sorting algorithm based on divide and conquer. Its worst-case running time has a lower order of growth than insertion sort.
- Because we are dealing with sub problems, we state each sub problem as sorting a sub array $A[p\,.\,.\,r]$.
- Initially, $p = 1$ and $r = n$, but these values change as we recurse through sub problems.

To sort $A[p\,.\,.\,r]$:

- **Divide** by splitting into two sub arrays $A[p\,.\,.\,q]$ and $A[q + 1\,.\,.\,r]$, where q is the halfway point of $A[p\,.\,.\,r]$.
- **Conquer** by recursively sorting the two sub arrays $A[p\,.\,.\,q]$ and $A[q + 1\,.\,.\,r]$.
- **Combine** by merging the two sorted sub arrays $A[p\,.\,.\,q]$ and $A[q + 1\,.\,.\,r]$ to produce a single sorted sub array $A[p\,.\,.\,r]$. To accomplish this step, we'll define a procedure MERGE(A, p, q, r).

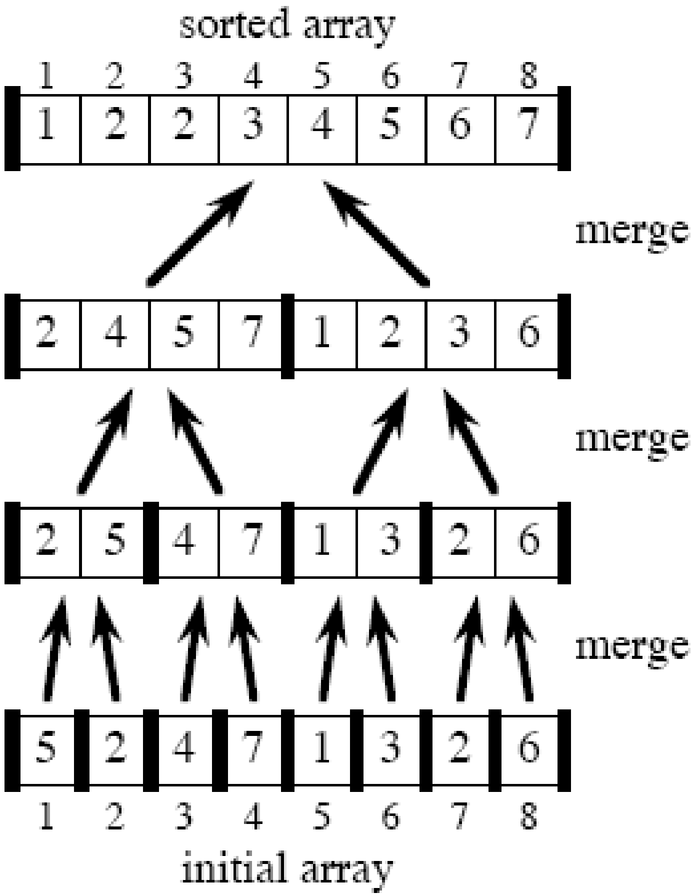
Merge Sort (Algorithm)

The recursion bottoms out when the subarray has just 1 element, so that it's trivially sorted.

```
MERGE-SORT(A, p, r)
if p < r                                 ▷ Check for base case
  then q ← ⌊(p + r)/2⌋                   ▷ Divide
       MERGE-SORT(A, p, q)               ▷ Conquer
       MERGE-SORT(A, q + 1, r)           ▷ Conquer
       MERGE(A, p, q, r)                 ▷ Combine
```

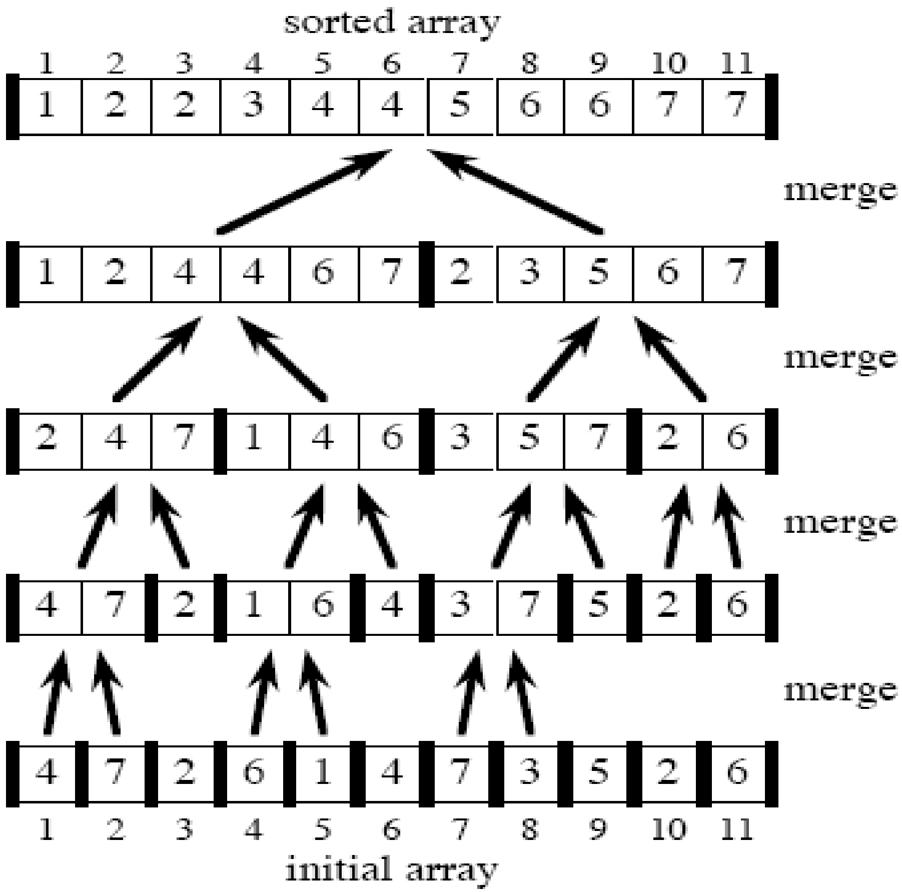
Example

Bottom-up view for n = 8: [Heavy lines demarcate subarrays used in subproblems.]



Example

Bottom-up view for n = 11: [Heavy lines demarcate subarrays used in subproblems.]



Merging

Input: Array A and indices p, q, r such that

- $p \le q < r$.
- Subarray $A[p\,.\,.\,q]$ is sorted and subarray $A[q+1\,.\,.\,r]$ is sorted. By the restrictions on p, q, r , neither subarray is empty.

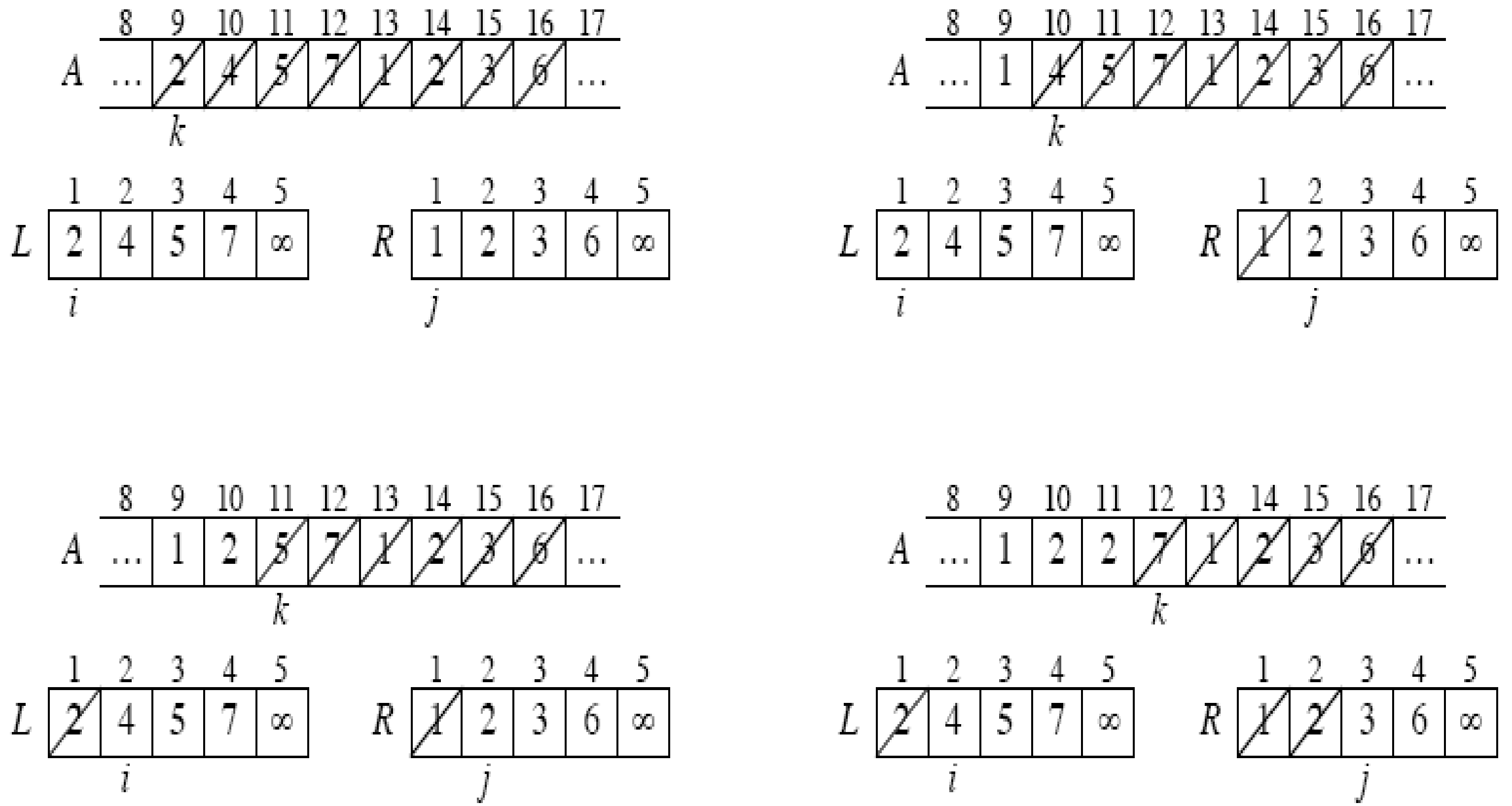
Output: The two subarrays are merged into a single sorted subarray in $A[p\,.\,.\,r]$.

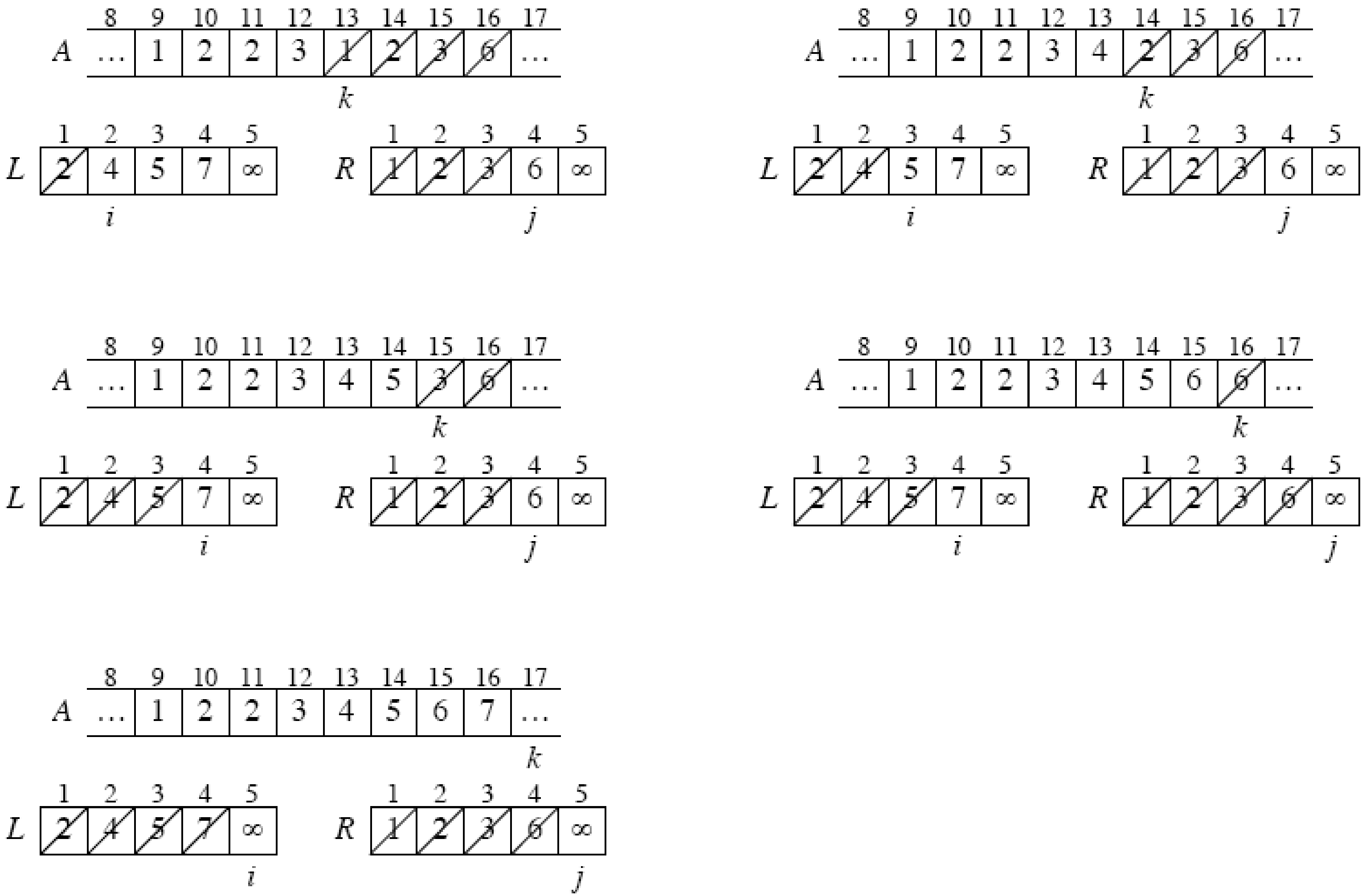
We implement it so that it takes $\Theta(n)$ time, where $n = r - p + 1$ = the number of elements being merged.

Pseudocode (Merging)

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
     do L[i] ← A[p + i − 1]
for j ← 1 to n2
     do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
     do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
```

Example [A call of MERGE(9, 12, 16)**]**





8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 7 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

Analyzing divide-and-conquer algorithms

Use a ***recurrence equation*** (more commonly, a ***recurrence***) to describe the running time of a divide-and-conquer algorithm.

Let $T(n)$ = running time on a problem of size n.

- If the problem size is small enough (say, $n \le c$ for some constant c), we have a base case. The brute-force solution takes constant time: $\Theta(1)$.
- Otherwise, suppose that we divide into a subproblems, each $1/b$ the size of the original. (In merge sort, $a = b = 2$.)
- Let the time to divide a size-n problem be $D(n)$.
- There are a subproblems to solve, each of size n/b $\Rightarrow$ each subproblem takes $T(n/b)$ time to solve $\Rightarrow$ we spend $aT(n/b)$ time solving subproblems.
- Let the time to combine solutions be $C(n)$.
- We get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c\,, \\ aT(n/b) + D(n) + C(n) & \text{otherwise}\,. \end{cases}$$

Analyzing merge sort

For simplicity, assume that n is a power of 2 $\Rightarrow$ each divide step yields two sub-problems, both of size exactly $n/2$.

The base case occurs when $n = 1$.

When $n \geq 2$, time for merge sort steps:

Divide: Just compute q as the average of p and $r \Rightarrow D(n) = \Theta(1)$.

Conquer: Recursively solve 2 subproblems, each of size $n/2 \Rightarrow 2T(n/2)$.

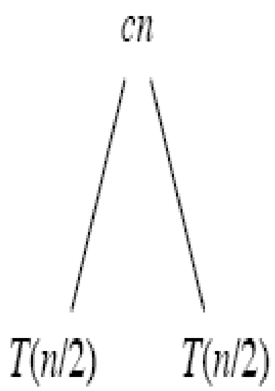
Combine: MERGE on an n-element subarray takes $\Theta(n)$ time $\Rightarrow C(n) = \Theta(n)$.

Since $D(n) = \Theta(1)$ and $C(n) = \Theta(n)$, summed together they give a function that is linear in n: $\Theta(n) \Rightarrow$ recurrence for merge sort running time is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

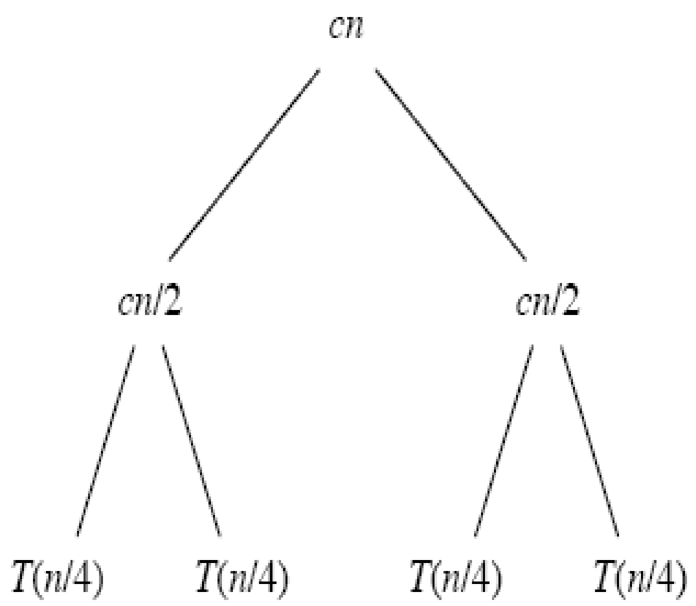
Recursion tree (Step 1)

- For the original problem, we have a cost of cn, plus the two subproblems, each costing $T(n/2)$:



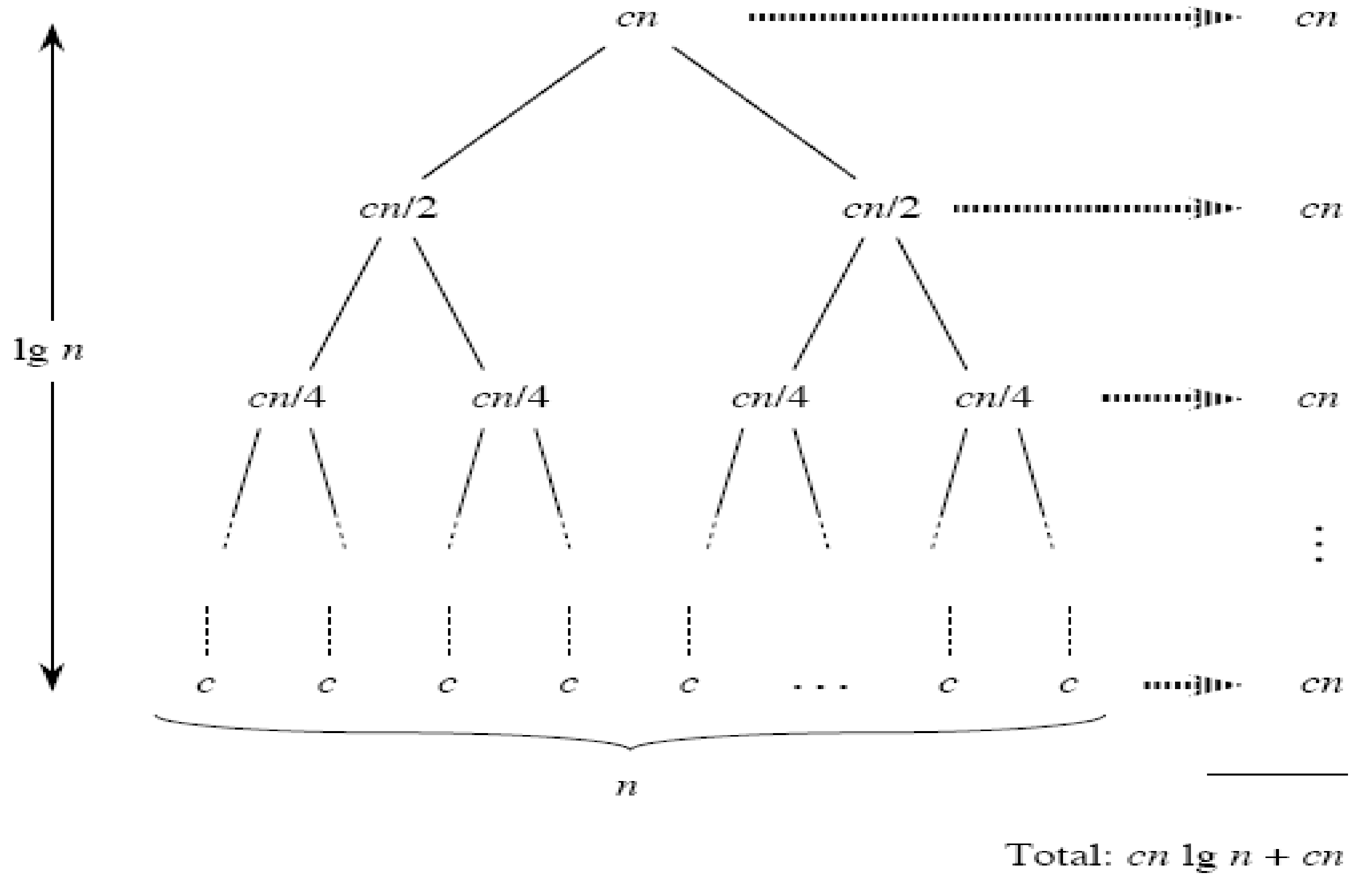
Recursion tree (Step 2)

- For each of the size-$n/2$ subproblems, we have a cost of $cn/2$, plus two subproblems, each costing $T(n/4)$:



Recursion tree (Step n)

- Continue expanding until the problem sizes get down to 1:



Home Assignment

- Solve the Recurrence of Merge Sort with the help of Master method.

Thank u