DIVIDE AND CONQUER APPROACH

→ In divide-and-conquer approach, we break the problem into several subproblems that are similar to the original problem but smaller in size, solve the problems recursively and then combine these solutions to create a solution to the original problem.

→ Recursive algorithms follow divide-and-conquer approach.

→ The divide-and-conquer approach involves three steps at each level of the recursion.

1. DIVIDE: The problem is divided into a number of subproblems.
2. CONQUER: Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, then solve the subproblems in a straightforward manner.
3. COMBINE: Combine the solutions to the subproblems into the solution for the original problem.

→ There are different algorithms which follow divide-and-conquer approach:

1. Merge sort.
2. Quick sort.
3. Binary search.

MERGE SORT

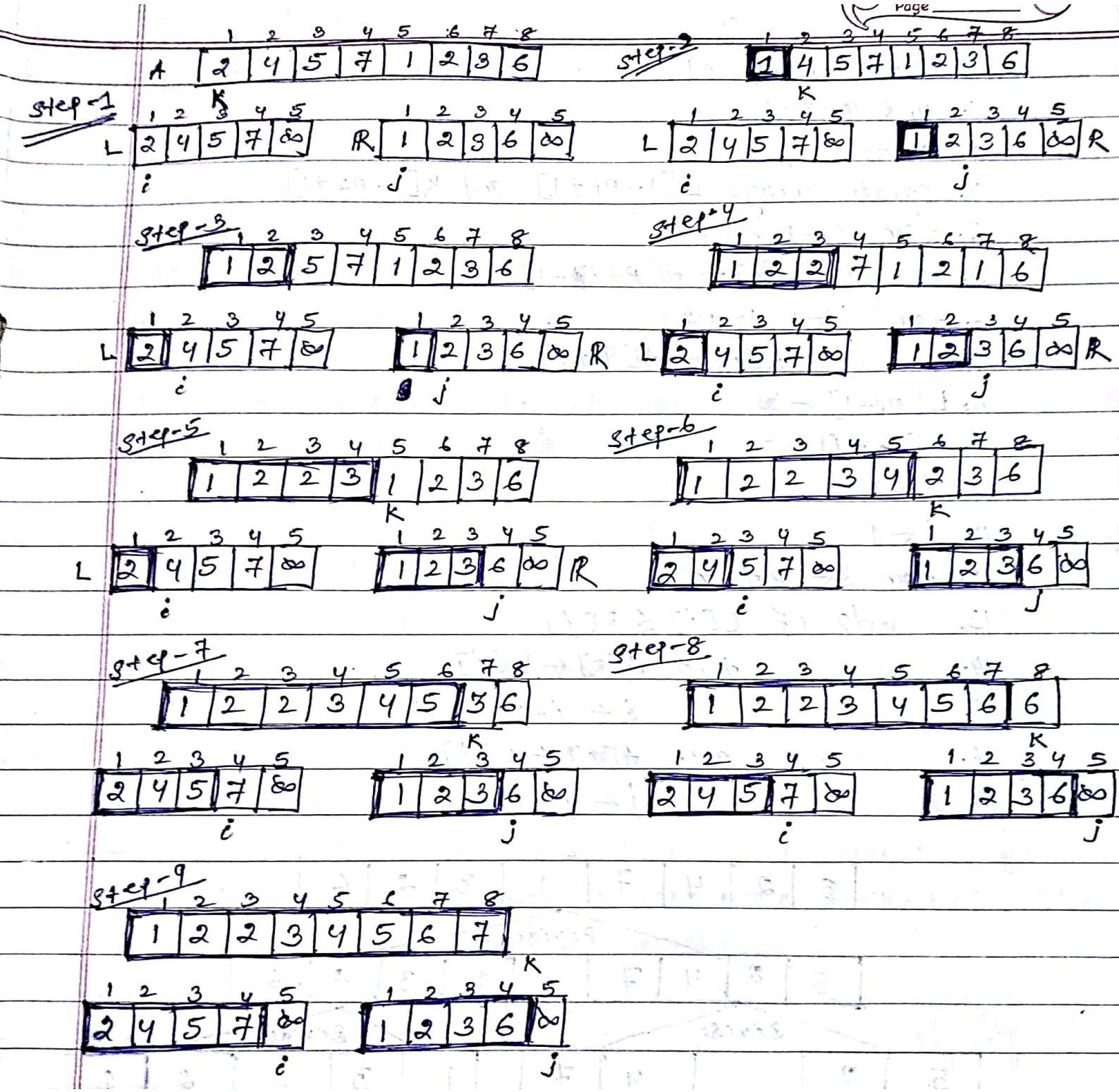
The merge sort algorithm follows the divide-and-conquer paradigm:

DIVIDE: Divide the n-element sequence to be sorted into two subsequences of n/2 elements each.

CONQUER: Sort the two subsequences recursively using merge sort.

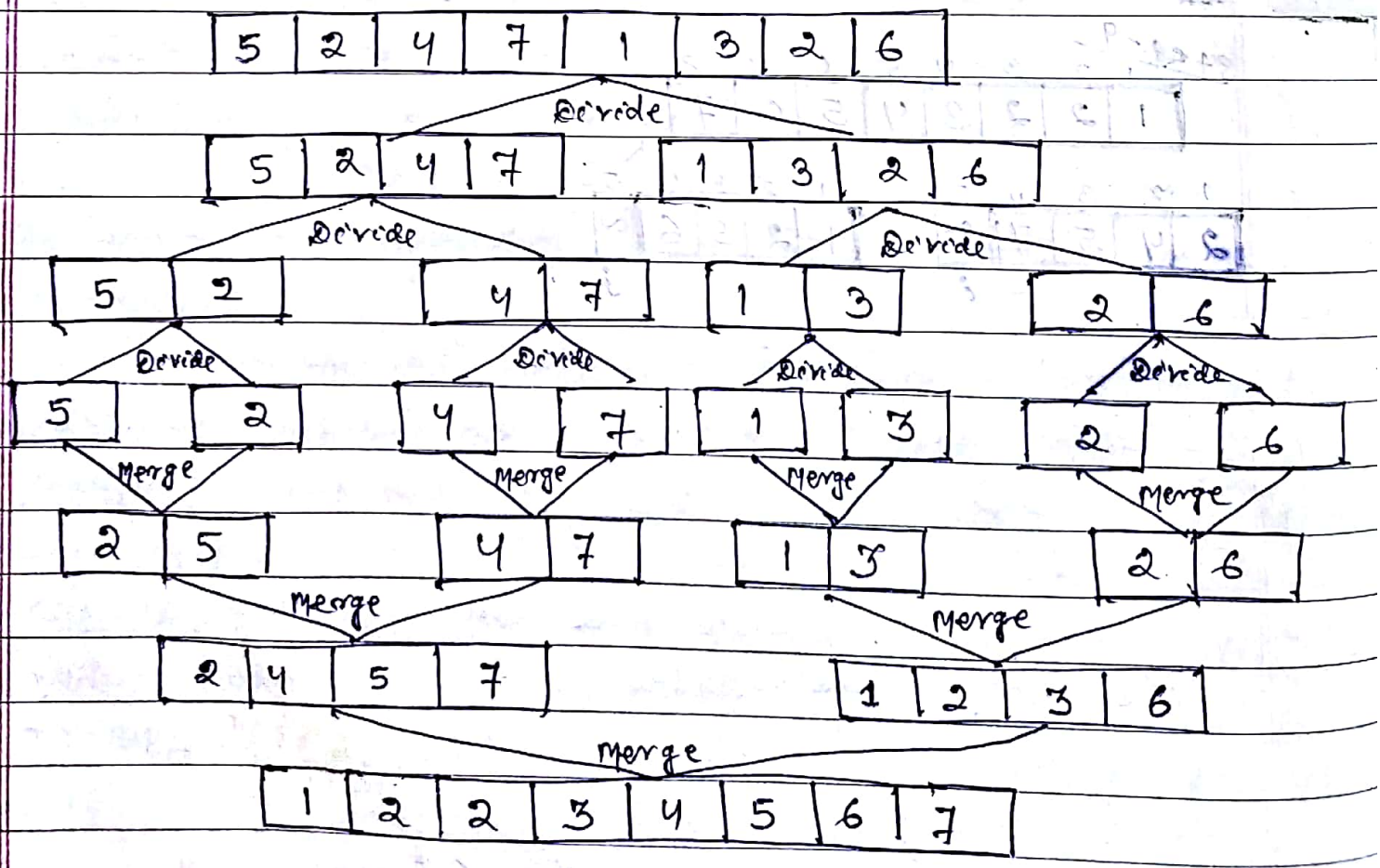
COMBINE: Merge the two sorted subsequence to produce the sorted answer.

```
MERGE_SORT(A, p, r)
1. if p < r
2.     then q ← ⌊(p+r)/2⌋
3.     MERGE_SORT(A, p, q)
4.     MERGE_SORT(A, q+1, r)
5.     MERGE(A, p, q, r)
```



MERGE (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1..n_1+1]$ and $R[1..n_2+1]$
4. for $i \leftarrow 1$ to n_1
5. do $L[i] \leftarrow A[p+i-1]$
6. for $j \leftarrow 1$ to n_2
7. do $R[j] \leftarrow A[q+j]$
8. $L[n_1+1] \leftarrow \infty$
9. $R[n_2+1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. for $k \leftarrow p$ to r
13. do if $L[i] \leq R[j]$
14. then $A[k] \leftarrow L[i]$
15. $i \leftarrow i+1$
16. else $A[k] \leftarrow R[j]$
17. $j \leftarrow j+1$



Analysis Of Merge Sort :

→ The worst-case running time of merge sort on n numbers is T(n).
→ If array contains one element, then merge sort takes constant time.
→ If array contains more than one element i.e. $n > 1$, then we break down the running time as :

Divide : The divide step just computes the middle of the subarray which takes constant time i.e. $D(n) = \theta(1)$

Conquer : We recursively solve two subproblems, each of size $n/2$, which contributes $2T(n/2)$ to the running time.

Combine : combining n-element subarray takes time $\theta(n)$

So, $C(n) = \theta(n)$

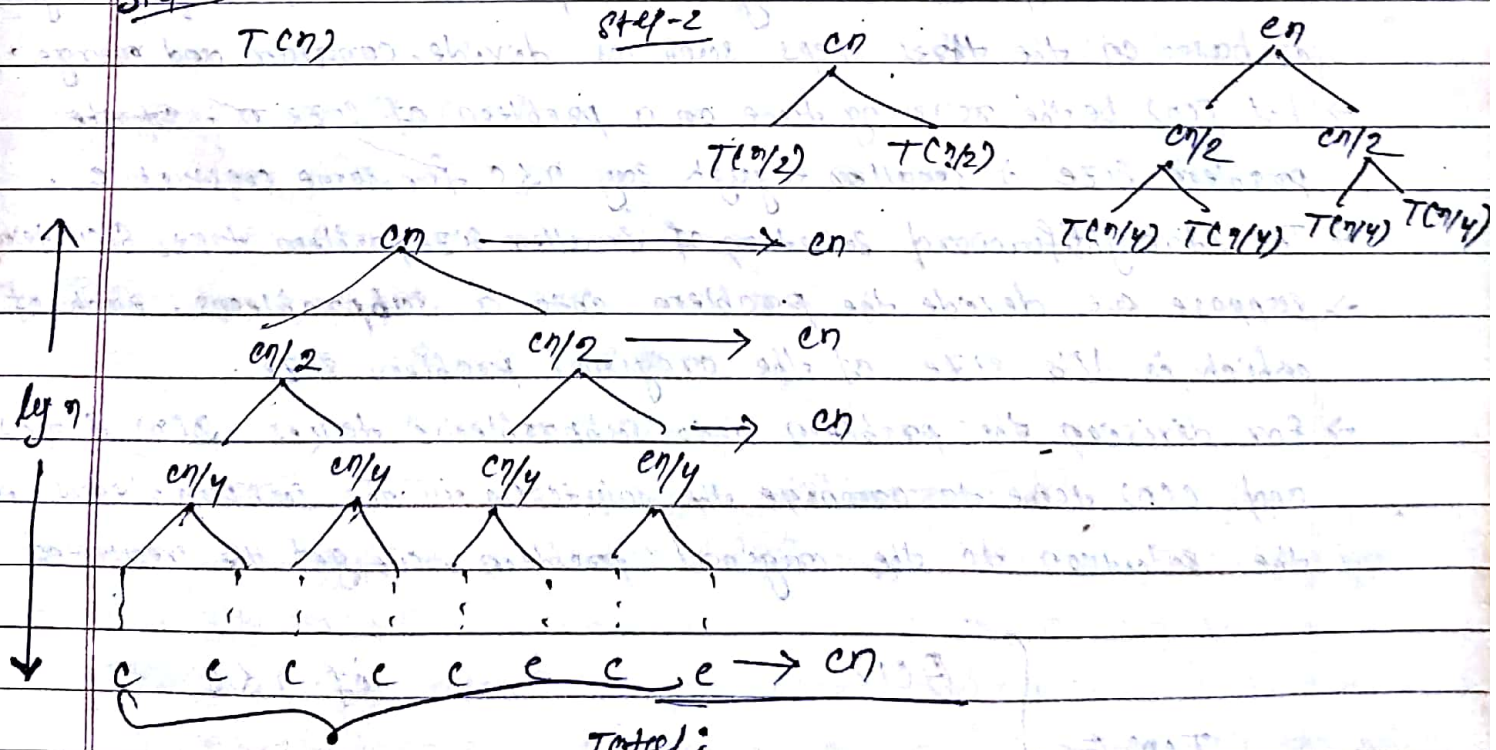
$$T(n) = \begin{cases} \theta(1) & \text{if } n = 1 \\ 2T(n/2) + \theta(n) & \text{if } n > 1 \end{cases}$$

We can rewrite the recurrence as

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

where constant c represents the time required to solve problems of size 1 as well as the time per array element of the divide and combine steps.

Step-1



The subproblem size for a node at depth $i = n/2^i$

$$\frac{n}{2^i} = 1$$

$$\Rightarrow \boxed{n = 2^i}$$

$$\Rightarrow \log n = \log 2^i$$

$$\Rightarrow \log_2 n = i \log_2 2$$

$$\Rightarrow \boxed{i = \log_2 n}$$

The depth of the tree is $\log n$.

The tree has $(\log n + 1)$ levels (i.e. $0, 1, 2, \ldots, \log n$)

each level costing cn

so, total cost is $cn(\log n + 1)$

$$= cn \log n + cn$$

$$= \Theta(n \log n)$$

Analysis Of ~~merge~~ ~~Sort~~ Devide-and-Conquer Approch:

→ The running time of an recursive algorithm can be described by a recurrence.

→ Recurrence describes the overall running time on a problem of size n in terms of the running time on small inputs.

→ we will then solve the recurrence by using mathematical tools and provide bounds on the performance of the algorithm.

→ A recurrence for the running time of a devide-and-conquer algorithm is based on the three steps such as devide, conquer and emerge.

→ Let $T(n)$ be the running time on a problem of size n. If the problem size is small enough say $n \leq c$ for some constant c.

→ The straightforward solution of smaller size problem takes $\Theta(1)$ time.

→ Suppose we devide the problem into a subproblems, each of which is $1/b$ size of the original problem size.

→ for division the problem into subproblems takes $D(n)$ times and $C(n)$ time to combine the solutions to the subproblems into the solution to the original problem, we get the recurrence as

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