3. Divide & Conquer (DC) Approach

3.1 What is Divide & Conquer?

Divide and Conquer is an algorithmic strategy that works by recursively breaking down a problem into two or more subproblems of the same type, until they become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem.

Divide & Conquer solves a large problem by doing the following operations:

- **Divide:** Break the problem into smaller subproblems
- **Conquer:** Solve each subproblem recursively
- **Combine:** Merge the sub-results into a final solution

General recurrence form: $T(n) = aT(n/b) + f(n)$

3.2 Example : Binary Search

Binary search divides a sorted list into halves.

Algorithm

```
binary_search(arr, target, low, high)
   if low > high then
      return -1 ;
   mid = (low + high) / 2 ;
   if arr[mid] == target then
      return mid ;
   else if target < arr[mid] then
      return binary_search(arr, target, low, mid - 1) ;
   else
      return binary_search(arr, mid + 1, high) ;
```

Time complexity : **O(log n)**.

3.3 Example : Merge Sort

Merge sort is the quintessential example of a DC sorting algorithm. It sorts a list by applying the three steps :

1. Split array into halves

2. Recursively sort the halves. The base case is when an array has 0 or 1 element (it's already sorted).
3. Merge sorted halves

Algorithm (simplified)

```
merge_sort(arr)
   if size(arr) <= 1 then
      return arr ;
   mid = size(arr)/2 ;
   left = merge_sort(arr[:mid]) ;
   right = merge_sort(arr[mid:]) ;
   return merge(left, right) ;
```

3.4 Example: Quick Sort

1. Choose pivot
2. Partition elements
3. Recursively sort partitions

Time complexity:

- Best/Average: **O(n log n)**
- Worst: **O(n²)**

4. Conclusion

Recursion solves problems by breaking them into smaller subproblems until reaching a base case. Divide & Conquer builds on recursion by dividing the problem, solving each part, and combining the results. Algorithms such as binary search, merge sort, and quicksort demonstrate the power of DC. The practical labs given with this chapter will deepen students understanding through hands-on coding, performance analysis, and visualization.

5. Application Exercises

Exercise 1: Understanding Recursion Depth

Explain what happens when the following function is called with n = 3:

```
int mystery(n) {
   if n == 0
      return 1 ;
   return n + mystery(n - 1)
```

Exercise 2: Using DC

Given a sorted list: [1, 3, 5, 6, 8, 10].

Trace the operations of binary search for target 8.

Exercise 3: Identify Components of DC

For merge sort, identify:

- Divide step
- Conquer step
- Combine step