🧩 Divide and Conquer - Complete Notes (Zero to Everything)

💡 1. Intuition / Idea

Think of **breaking the problem into smaller pieces**, solving each independently, and **combining** the result.

Just like Ramayan mein **Vanar Sena** alag-alag dikh gayi thi Lanka tak jaane ke liye, par goal ek hi tha! 😉

⚙️ 2. Steps / Strategy

> *Remember this 3-step:*

1. Divide

Break the main problem into **smaller subproblems**.

2. Conquer

Solve each subproblem **recursively**.

3. Combine

Merge or combine solutions of subproblems to get the **final answer**.

🛠️ 3. Time Complexity Format

Let's say problem size is `n`

You divide it into `k` subproblems, each of size `n/b`, and combining takes `O(n^d)`

Then:

```
T(n) = k * T(n/b) + O(n^d)
```

Use **Master Theorem** to solve this recurrence (explained below 👇).

🔢 4. Master Theorem (Time Analysis Shortcut)

For recurrence:

```
T(n) = aT(n/b) + O(n^d)
```

We compare **n^d vs n^log_b(a)**

Now 3 cases:

Case	Condition	Time Complexity
1	d < log_b(a)	O(n^log_b(a))
2	d == log_b(a)	O(n^d * log n)
3	d > log_b(a)	O(n^d)

🧠 *"Smaller dominates → log, Equal → log n, Larger dominates → polynomial"*

🧪 5. Famous Examples of Divide and Conquer

Problem	Divide Step	Combine Step	Time Complexity
🖌️ Merge Sort	Split array in half	Merge two sorted arrays	O(n log n)
🔍 Binary Search	Halve the array	No combine	O(log n)
💥 Quick Sort	Partition the array	Nothing	O(n log n) avg
🧠 Maximum Subarray (Kadane Alt)	Divide into left & right halves	Combine crossing subarray	O(n log n)
🟢 Matrix Multiplication	Split matrix into 4 submatrices	Add resulting products	O(n^3) or better
⚡ Strassen's Algorithm	7 multiplications instead of 8	Matrix operations	O(n^2.81)
📷 Closest Pair of Points	Divide point set	Find closest pair across mid	O(n log n)

Problem	Divide Step	Combine Step	Time Complexity
💾 Karatsuba Multiplication	Split numbers	Combine result with shifts	O(n^log23) ≈ O(n^1.58)

🧠 6. Recursion Tree Insight

Divide and conquer problems can be visualized as **recursion trees**, where:

- Each level has **more subproblems**,
- Combine step may take **linear time** per level.

⏳ Depth of tree = log(n) if divided by 2 every time.

⌛ Total work = work per level * number of levels.

🧑‍💻 7. Java Example: Merge Sort

```
public class MergeSort {
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;

            // Divide
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);

            // Conquer + Combine
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int[] leftArr = Arrays.copyOfRange(arr, left, mid + 1);
        int[] rightArr = Arrays.copyOfRange(arr, mid + 1, right + 1);

        int i = 0, j = 0, k = left;

```

```
        while (i < leftArr.length && j < rightArr.length) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k++] = leftArr[i++];
            } else {
                arr[k++] = rightArr[j++];
            }
        }

        while (i < leftArr.length) arr[k++] = leftArr[i++];
        while (j < rightArr.length) arr[k++] = rightArr[j++];
    }
}
```

🎨 8. Visualization Example – Merge Sort

Let's sort: `[38, 27, 43, 3, 9, 82, 10]`

```
                    [38 27 43 3 9 82 10]
                  /                       \
         [38 27 43]                     [3 9 82 10]
         /       \                         /         \
      [38]      [27 43]               [3 9]        [82 10]
                 /    \              /     \        /     \
               [27]  [43]          [3]    [9]     [82]   [10]

→ Merge upward till final sorted array is formed!
```

⚖️ 9. Pros & Cons

Pros ✅	**Cons ❌**
Breaks problem into manageable pieces	Recursion → Stack overhead
Often optimal in time complexity	Harder to debug sometimes
Ideal for parallelization 💻	May need extra space (e.g., Merge Sort)

🧭 10. When to Use Divide and Conquer

✅ When:

- Problem can naturally be **split into subproblems**
- Subproblems are **independent**
- You can **combine easily**

❌ Avoid When:

- Subproblems are tightly connected
- Combine step is very complex

✨ Bonus: Strassen's Matrix Multiplication

- Standard: $O(n^3)$
- Strassen's: Reduces multiplications from 8 → 7
- Time: **$O(n^{\log_2 7}) \approx O(n^{2.81})$**
