**Chapter 2: DIVIDE & CONQUER ALGORITHM**

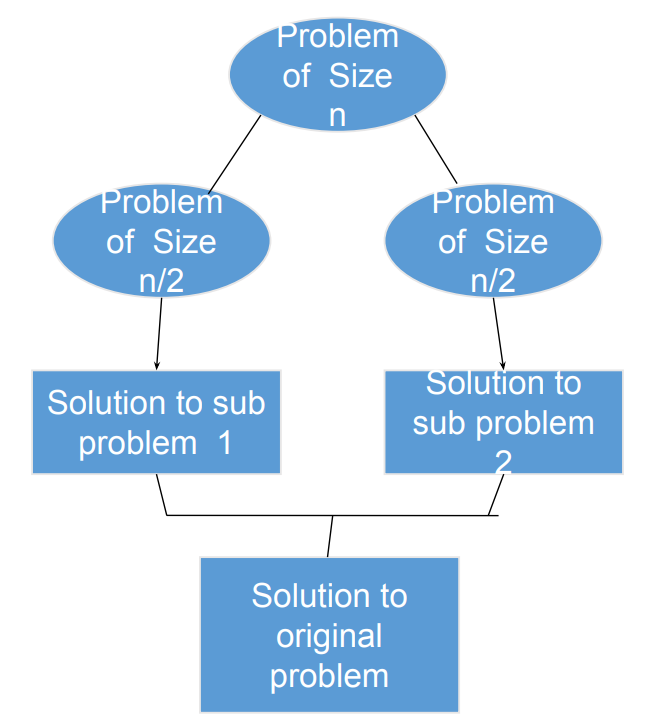
**Topic – 1: Overview**

* Introduction
* Recurrence relation
* Multiplying large integers problem
* Binary search
* Merge sort
* Quick sort
* Matrix multiplication
* Exponentiation

**Topic – 2: Introduction**

**Steps Involved**

* **Divide**
* **Recur:** Solve the sub-problems recursively.
* **Conquer** (**not** done in all algorithms)



**General Philosophy**

* General algorithm philosophy for **divide & conquer** is to keep **dividing** the problem until the **sub-problems** is optimal enough to be solve.

**Parts Of Solving**

* **Partitioning** of problem
* Reaching the **indivisible part** of the problem.
* When problems are **small enough** to be solved optimally.
* **Piecing** the solved sub-problems together.
* Height in these will be **log2(n)** because their branches **end together**.

**Topic – 2: Recurrence Relations**

* It can be used in solving **multiplication problems** by breaking them into smaller parts.
* Their analysis can be done using methods we discussed in **1st chapter**.

**Topic – 3: Multiplying Large Integers Problem**

**Introduction**

* It is solving multiplication problems through **expansion of terms**.
* Sub-problems get **halved** at each level.

**Formula – I (Example)**

**Problem: 1234 \* 981**

**0981 = (102 \* 9) + 81**

**1234 = (102 \* 12) + 34**

**1234 \* 981**

**= 0981 \* 1234**

**= ((102 \* 9) + 81) \* ((102 \* 12) + 34)**

**= 104 (9 + 12) + 102 (9 \* 34) + 102 (12 \* 81) + (81 \* 34)**

**= 104 (9 + 12) + 102 (9 \* 34) + 102 (12 \* 81) + (81 \* 34)**

**= 104 (9 + 12) + 102 ((9 \* 34) + (12 \* 81)) + (81 \* 34)**

**= 1210554**

**Formula – II (Same Example)**

**Problem: 1234 \* 981**

**w = 09 \* 102**

**x = 81 \* 102**

**y = 12 \* 102**

**z = 34 \* 102**

**Formula: (wz + xy) = (w + x) \* (y + z) – wy – xz**

**f(n) = Time taken for multiplication**

**g(n) = Time taken for additions & overhead operations**

**T(n) = f(n) + cg(n)**

**T(n) = n + 3T(n/2)**

**Structural Insights**

**Height = k = log2(n)**

**Total sub-problems = 3k**

**Total time spent**

**= 3k \* (n/2k)**

**= (3/2)k \* n**

**= (3/2)log2(n) \* n**

**= nlog2(3/2) \* n**

**= n1.59**

**= O(n1.59)**

**= O(nc)**

**Topic – 4: Time Complexities Of Algorithms**

**Binary Search**

**T(n) = T(n/2) + 1 = O(log(n))**

* **n/2** because we **discard** searching **one of the current halves** of array.
* **1** because it **checks the middle element** in the current half.

**Note!**

**🡪 There is no sub-problem in binary search as another half is not considered when other half is being searched.**

**Merge Sort**

**T(n) = 2T(n/2) = O(nlog(n))**

* **2** because it is divided into **two sub-problems**.

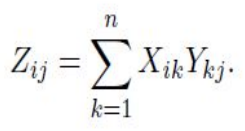
**Quick Sort**

**T(n) = 2T(n/2) + n = O(log(n))**

* This was the **best case**.
* In **worst case**, it goes up to **O(n2)**.

**Topic – 5: Matrix Multiplication**

**Introduction**



* **Z** is the **product matrix** of another two matrices **X** and **Y**.

**Types Of Matrix Multiplications**

* Iterative matrix multiplication
* Recursive matrix multiplication
* Strassen matrix multiplication

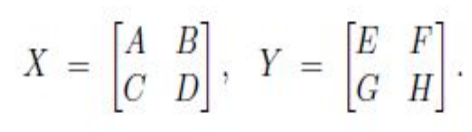
**Iterative Matrix Multiplication**

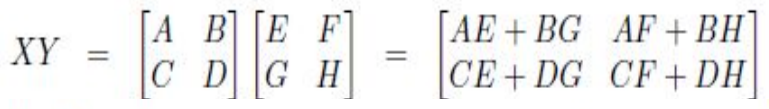
* Solved with **two nested loops**.

**Recursive Matrix Multiplication**

* Matrix is broken into **smaller parts** which will be multiplied in order to get various cells of **product matrix**.
* Now we can use **divide & conquer** algorithm here.
* **Worst case** complexity goes up to **O(n3)**.

**Strassen Matrix Multiplication**





* There are eight multiplications to perform which are **AE**, **BG**, **AF**, **BH**, **CE**, **DG**, **CF**, **DH**.
* But German mathematician **Volker Strassen** gave formula to compute it in **7 multiplications**!

**P = A11 \* (B12 – B22)**

**Q = (A11 + A12) \* B22**

**R = (A21 + A22) \* B11**

**S = A22 \* (B21 – B11)**

**T = (A11 + A22) \* (B11 + B22)**

**U = (A12 – A22) \* (B21 + B22)**

**V = (A11 – A21) \* (B11 + B12)**

**C11 = T + S – Q + U**

**C12 = P + Q**

**C21 = R + S**

**C22 = P + T – R – V**

**C11, C12, C21, C22 are cells of the product matrix.**

**T(n) = 8T(n/2) + O(n2) = O(n1.59)**

**Topic – 6: Exponentiation**

**Regular Method**

* Here, exponent of a given number is calculated using a **loop**.

**T(n) = O(n)**

**Divided Exponent Method**

* **Lessens** the number of steps to get the answer.
* If value of exponent is currently **even**, then it is written in form of **power to 2**.
* For **odd**, we make the exponent **even** (one less than current exponent) by expanding the term.
* Refer to example comparison below.

**Regular method:**

**a29**

**We will get the answer in 28 steps.**

**Division exponent method:**

**a29= a\*a28 = a\*(a14)2 = a\*(a7)2)2 = a\*(a\*a6)2)2 = a\*(a\*(a3)2)2)2 = a\*(a\*(a\*a2)2)2)2**

**We will get the answer in just 7 steps!**

**T(n) = T(n-1) + 1 = O(log(n))**

* **(n-1)** because the problem is **expanded** to smaller, calculative parts but in **irregular** manner.
* **1** because we check if the exponent is **odd or even**.

**Note!**

**🡪 Having a recursive function doesn’t always mean reduction in time complexity.**