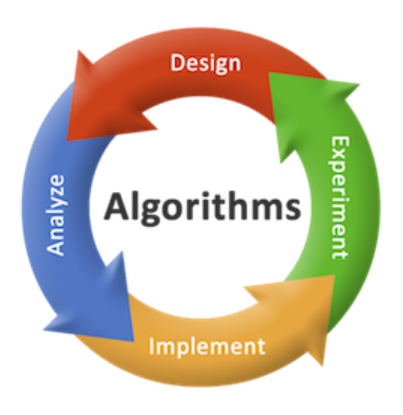
**Chapter 1: INTRODUCTION**

**Topic – 1: Summary**



* **Characteristics of Algorithm**
* **Analysis of Algorithm**
* **Performance Measurements of Algorithm**
* **Time & Space Trade-Offs**
* **Recursive Algorithm Analysis**

**Topic – 2: Computational Problem**

* **Computational problem:** Sets a relationship between input & output of a problem.
* For example, an input ***'n'*** gives an output saying it is ***even or not***.
* **Algorithm:** Step-by-step approach for solving a given problem.

**Topic – 3: Characteristics of an Algorithm**

* There maybe **no** input at all.
* **At least one** output is produced.
* The steps must be **clear** (***definiteness***).
* Output should **not** be uncertain etc.

**Topic – 4: Algorithm Analysis**

**Order Of Growth**

* **Number of inputs** are directly proportional to **execution speed**.
* **Order of growth:** Change in behaviour of an algorithm as the number of inputs change.

**Algorithm Categories**

* **Best case:** Also known as ***lower bound***.
* **Average case:** Also known as ***tight bound***.
* **Worst case:** Also known as ***upper bound***.

**Asymptotic Complexity**

* Asymptotic means **approaching a curve arbitrarily**.

**Topic – 5: Various Asymptotic Notations**

**Big O Problem**

* Its all about the **upper bound**, the **worst case**.
* **Big O** problem comes into play when the time complexity grows **proportional to the input size**.
* For example, a loop finding a particular element from an array having **'n'** elements.
* In this case, the time complexity increases as the number of elements increase.

**Ω Problem**

* Its all about the **lower bound**, the **best case**.
* This depends on the **minimum number of search or operations** to be performed.

**Θ Problem**

* Its all about the **path** between **worst case** & **best case** i.e. the **average case**.
* Constants **c1** and **c2** represent **intrinsic factors** of algorithm, affecting it.
* These intrinsic factors are the **type of operations** performed in the program.

**Topic – 6: Algorithm Performance Measurement**

**Introduction**

* Measurement of the amount of **time** & **space** required by the algorithm.

**Parameters Affecting Performance**

* Hardware being used.
* Abstract data type (any kind of data structure).
* Compiler’s efficiency.
* Programmer’s way of implementation.
* Algorithm’s complexity.
* **Size** of the input (like **'n'** elements in an array).

**Topic – 7: Calculating Time Complexity**

**Example (Recurrence)**

***int fib (int n)***

***{***

***if (n<=1)***

***return n;***

***else***

***return fib(n-1) + fib(n-2);***

***}***

**Solution**

**T(n) = c [for n<=1]**

**T(n) = T(n-1) + T(n-2) [for n>1]**

**Topic – 8: Solving Recurrence Method**

**Methods**

* Substitution method
* Iterative method
* Recurrence tree method
* Master’s method

**Topic – 9: Substitution Method**

**Let’s say, T(n) = T(n-k) + k [(n-k)=0, so n=k]**

**And, T(n) = 1 [for n=0]**

**T(n) = 1**

**T(n) = T(n-n) + n**

**T(n) = T(1) + n**

**T(n) = 1 + n**

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

**Topic – 10: Recurrence Tree Method**

**Introduction**

* Used for finding time complexity of **recurrent equations**.
* Uses **divide & conquer** approach.
* Root of tree is indicated by **second term** of the equation.

**Steps Involved**

* **Step 1:** Identifying **root** & **sub-problems**.
* **Step 2:** Confirm all level cost sums to be same.
* **Step 3:** Sum up all **pre-level costs**.
* **Step 4:** Find the **height** for the tree.
* **Step 5:** Find **upper bound** (max cost) as per height.

**Example (Root & Sub-Problems)**

**T(n) = 2T(n/2) + n2 [Root is n2]**

**Problems are divided into sub-problems,**

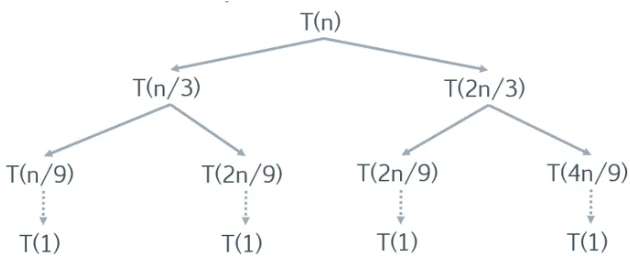
**2T(n/2) = T(n/2) + T(n/2)**

**Example (Full)**

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

**Root: n**

**Sub-problem: T(n/3) and T(2n/3)**

****

**Notice that summing cost at each level gives 'cn'.**

**The longest root to leaf path is:**

**n 🡪 2n/3 🡪 4n/9 🡪 8n/27 🡪 … 🡪 1 [until (2/3)kn]**

**= n 🡪 (2/3)n 🡪 (2/3)2n 🡪 (2/3)3n 🡪 … 🡪 1 [until (2/3)kn]**

**(As it decomposes to 1 very late than others)**

**Height of tree = k**

**(2/3)kn = 1**

**k = log(3/2n) = Height of tree**

**Total cost = Sum of [total sum of each level] = O(cn log(3/2n)) = O(n log(n))**

**Lower bound will be ω(n log(n)).**

**This is because is best case the constant 'c' is 1, making (cn log(cn)) as (n log(n)).**

**Verification with substitution method:**

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

**= d((n/3)log(n/3)) + d((2n/3)log(2n/3)) + cn**

**= dn log(n)**

**Topic – 11: Master’s Theorem**

**Introduction**

* Alternative way to solve **recurrence equations**.
* The equation must be in the following given form.

**T(n) = aT(n/b) + f(n) [a>=1, b>1]**

**Size of problem = (n/b) = ⌊n/b⌋ or ⌈n/b⌉ [⌊⌋ floor, ⌈⌉ is ceiling]**

**Possible Cases**

**Case 1:**

**For, f(n) = θ(nc) [c < logba]**

**Then, T(n) = θ(nlogba)**

**Case 2:**

**For, f(n) = θ(nc) [c = logba]**

**Then, T(n) = θ(nclog(n))**

**Case 3:**

**For, f(n) = θ(nc) [c > logba]**

**Then, T(n) = θ(f(n))**

**Note!**

**🡪 'a' and 'b' are being referred from the equation T(n) = aT(n/b) + f(n).**

**Example**

**Problem: Merge sort**

**Equation: T(n) = 2T(n/2) + θ(n)**

**Case: 2**

**Solution: θ(n log(n))**

**Example – II**

**Problem: Binary search**

**Equation: T(n) = T(n/2) + θ(1)**

**Case: 2**

**Solution: θ(log(n))**

**How Cases Affect Result?**

**Case 1:**

**Work done at leaves > Work done at nodes**

**Result = Work done at leaves**

**Case 2:**

**Work done at leaves = Work done at nodes**

**Result = (Height) \* (Work done at any level)**

**Case 3:**

**Work done at leaves < Work done at nodes**

**Result = Work done at root**

**"Work done" here means sum of all nodes at a particular level.**

**Example – III**

**Equation: T(n) = 9T(n/3) + n**

**Variables: a=9, b=3, f(n)=n**

**θ(nlogba) = θ(n2) [f(n) = n and c = 1]**

**Case: 1**

**Solution: T(n) = θ(n2)**

**Limitations**

* Some problems might fall **between** two cases.
* Such problems **can’t** be solved using **master’s theorem**.