Assignment - 1

1. Define Algorithm, Time Complexity and Space Complexity.

→ (i) Algorithm:

→ An algorithm is any well defined computational procedure that takes some values or set of values as input and produce some values or set of values as output.

→ An algorithm is thus a sequence of computation steps that transform the input into output.

→ The output should be generated in finite time.

(ii) Time Complexity:

→ The amount of time taken by any algorithm to run is called time complexity.

(iii) Space Complexity:

→ The amount of space taken by an algorithm is called space compleity.

2. Explain why analysis of an algorithm is important? Explain: Worst Case, Best Case and Average Case complexity with suitable example.

→ Analyzing an algorithm has come to mean predicting the resources that the algorithm requires.

→ Resources such as memory, communication bandwidth or computer hardware are of

primary concern, but most often it is computational time that we want to measure.

→ By analyzing several candidate algorithms for a problem, we can identify a most efficient one. Importance of Analysis:

→ To predict the behaviour of an algorithm without implementing it on specific computer

→ It is impossible to predict the exact behaviour of an algorithm. There are too many influencing factors.

→ The analysis is thus only an approximation, it is not perfect.

→ By analyzing different algorithms, we can compare them to determine the best one for our purpose.

→ There are three types of algorithm Analysis

(i) Best case:

→ Define the input for which algorithm takes less time or minimum time. In the best case calculate the lower bound of an algorithm.

→ Example: In the linear search when search data is present at the first location of large data then the best case occurs.

(ii) Worst Case:

→ Define the input for which algorithm takes a long time or max^m time.

→ In the worst case, calculate the upper bound of an algorithm.

→ Example: In linear search when search data is not present at all then worst case occurs

(iii) Average Case:

→ In the average case take all random inputs and calculate the computation time for all inputs and divide by total no. of inputs.

→ The average case gives value betⁿ upper bound and lower bound.

∴ ~~Average case~~ = Avg

Example: In linear search when the element is something other than first and last present in array is called average case.

5. Solve the recurrence:

1. $T(n) = 7T\left(\frac{n}{2}\right) + n^3$

→ Here comparing the eqⁿ with standard eqⁿ of Master theorem

i.e $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$a = 7;\ b = 2;\ f(n) = n^3 \therefore d = 3$

$\therefore a < b^d$

i.e $7 < 2^3$

Hence

$T(n) = \theta(n^d)$

$T(n) = \theta(n^3)$

2. $T(n) = 2T(\frac{n}{2}) + n$

→ Here, comparing given eqⁿ with standard eqⁿ of Master's Theorem i.e

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$\therefore a = 2 ; b = 2 ; f(n) = n$
$\therefore d = 1$

Now, $a = b^d$
i.e $2 = 2^1$

Thus, by Master's Theorem

$T(n) = \theta(n^d \log n)$
$T(n) = \theta(n \log n)$

3. $T(n) = T(\frac{n}{2}) + n$

→ Here, comparing given eqⁿ with standard eqⁿ of Master's Theorem i.e

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$\therefore a = 1 ; b = 2 ; f(n) = n$
$\therefore d = 1.$

Now, $a < b^d$
i.e $1 < 2^1$

Thus, from Master's Theorem

$T(n) = \theta(n^d)$

$T(n) = \theta(n)$

4. $T(n) = 5T\left(\frac{n}{2}\right) + n^2$

→ Here, comparing the given eqn with standard eqn of Master Theorem

i.e $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$\therefore a = 5 ; b = 2 ; f(n) = n^2$

$\therefore d = 2$

Now, ~~$a < b^d$~~

i.e $5 > 2^2$

Thus, from Master's Theorem

$T(n) = \theta(n^{\log_b a})$

$T(n) = \theta(n^{\log_2 5})$

5. $T(n) = 4T\left(\frac{n}{4}\right) + n$

→ Here, comparing the given eqn with standard eqn of Master Theorem

$$\text{i.e } T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$\therefore a=4 ; b=4 ; f(n)=n$

$\therefore d=1$

Now, $a = b^d$

i.e $4 = 4^1$

Thus, By Master's Theorem

$$T(n) = \Theta(n^d \log n)$$

$$T(n) = \underline{\Theta(n \log n)}$$

4. Explain asymptotic notation in detail.

→ The efficiency of an algorithm can be measured by the asymptotic notation.

→ There are three types of asymptotic notation

1. Big Oh (O): It ~~divides~~ defines the upper bound of an algorithm.

→ Let $f(n)$ and $g(n)$ be two non negative fun^n. Let n_0 and constant C are two integers such that n_0 denotes some values of input and $n \geq n_0$. Similarly C is some constant such that $C > 0$.

→ We can write,

$O(g(n)) = \{ f(n) : \text{there exists positive constant } c \text{ an } n_0 \text{ such that } 0 \leq f(n) \leq g(n) \ \forall \ n \geq n_0 \}$

Eg: $f(n) = 2n+2$; $g(n) = n^2$; $c = 1$

Let $n=1 \Rightarrow f(1) = 4 \quad g(n) = 1$

$n=2 \quad f(2) = 6 \quad g(n) = 4$

$n=3 \quad f(3) = 8 \quad g(n) = 9$

$\therefore n_0 = 3$

2. Omega (Ω): It defines the lower bound of an algorithm

→ A fun^n $f(n)$ is said to be in $\Omega(g(n))$ if $f(n)$ is bounded below by some +ve constant multiple of $g(n)$ such that.

$\Omega(g(n)) = \{ f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such } 0 \leq c \cdot g(n) \leq f(n) \}$

Eg: $f(n) = 2n^2+5$; $g(n) = 7n$; $c = 1$

Let

$n=1 \quad f(1) = 7$; $g(n) = 7$

$n=2 \quad f(2) = 13$; $g(n) = 14$

$n=3 \quad f(3) = 23$; $g(n) = 21$

$n=4 \quad f(4) = 37$; $g(n) = 28$

$\therefore n_0 = 3$

vision

4. Theta (Θ): It bounds the fun^n from above and below.

→ Let f(n) and g(n) be two non negative fun^n. There are two +ve constants c_1 and c_2 such that

$\Theta(g(n)) = \{ f(n)$: there exists +ve constants c_1, c_2 & n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \}$

5. Explain the characteristics of an algorithm.

→ The various characteristics of algorithm are:

1. Input: The algorithm should externally supply zero or more quantities i.e (atleast one input)
2. Output: It should generate at least one output
3. finiteness: The algorithm should terminate after some steps.
4. Effectiveness: Which is in simple language with less time and space complexity
5. Unambiguity: Instruction given by algo should be clear and straight forward