Time Complexity

MODULE 2

Recurrence of linear search using substitution : $T(n) = O(n)$

Merge Sort:

Best, Average, Worst Case = $O(n\log_2 n)$

Quick Sort:

Best	Average	Worst Case :-
$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$

Min-Max Problem:-

$$T(n) = \begin{cases} 0 & \text{.... if } n=1 \\ 1 & \text{... if } n=2 \\ 2T(n/2)+2 & \text{.... if } n>2 \end{cases}$$

Large Integer Problem :- $T(n) = \begin{cases} 1 & \text{... if } n=1 \\ 3T(n/2) & \text{... if } n>1 \end{cases}$

Complexity :- $O(n^2)$

Strassen's Multiplication : $O(n^{2.81})$

$$T(n) = \begin{cases} 1 & \text{..... if } n=1 \\ 7T(n/2)+n^2 & \text{.... if } n>1 \end{cases}$$

By Aditya Potdar

MODULE 3 :- Greedy :-

Activity Selection Problem :-
If sorted array :- $O(n)$
else : $O(n\log n)$

Kruskal's Algorithm - $\log_2 |V|$ OR $O(|E| \log |E|)$

Prim's Algorithm :-
$T(n) = O(n^2)$
Improved to $O(|E| + |V| \log |V|)$ using Fibonacci heap, adjacency matrix

Djikstra's Algorithm :- $O(V^2)$

Knapsack :- $O(n\log_2 n)$

Job Sequencing :- Average $\rightarrow O(N^2)$
Use of structure and union $\rightarrow O(N)$

Optimal Merge Pattern :- $O(n\log n)$

Huffman Code :- $O(n\log n)$

Coin Change Problem :- $O(V)$ .. where V is given amount of money

MODULE 4 - DYNAMIC

Binomial Co-efficient :- $O(n^* \max(k, n-k))$

Making Coin-Change :- $O(n^* c)$
n = len (coins)
c = amount / min (coins)

~~Assembly-Line Scheduling :- $O(n*M)$~~
~~n = number of items~~

Assembly-Line-Scheduling :- $O(n)$

0/1 Knapsack Problem :- $O(n*M)$
n = number of item
M = capacity of knapsack

Johnson's Algorithm :- $O(n\log n)$ .. NS

Multistage Graphs :- $O(n + |E|)$

All-Pair Shortest Path (Floyd-Warshalls) :- $O(n^3)$

Longest Common Subsequence :- $O(mn)$

Matrix-Chain Multiplication :- $O(n^3)$

Travelling Salesman :- $O(n^2 2^n)$

MODULE 5 - BACKTRACKING

N-Queen Problem :- $O(N!)$

Sum-of-subset :- $O(2^n)$

Hamiltonian Cycle :- $O(2^n n^2)$ / $O(n^n)$ / $O(n!)$

15-Puzzle Problem :-

Knapsack : Worst → $O(2^n)$
Best → ~~O(2~~ $O(n)$

Travelling Salesman :- $O(n^2 2^n)$

Naive String-matching :- $O(m*(n-m))$
n = text string length
m = pattern length

Rabin-Karp Algorithm :- $O(n)$

Finite-Automata : $O(n)$ / $O(m^3 *$ no. of characters$)$
m = pattern length

Knuth-Morris-Pratt Algorithm :- $O(m+n)$

MODULE 6 :- Vertex Cover Problem - $O(V+E)$