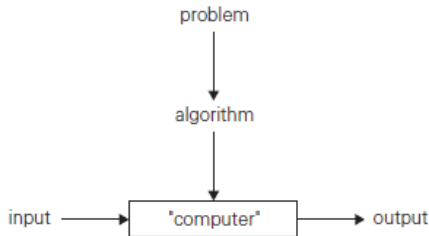


**INTRODUCTION**

Notion of an Algorithm - Fundamentals of Algorithmic Problem Solving - Important Problem Types - Fundamentals of the Analysis of Algorithmic Efficiency - Asymptotic Notations and their properties. Analysis Framework - Empirical analysis - Mathematical analysis for Recursive and Non-recursive algorithms - Visualization

**UNIT-I / PART-A**

1	<b>Define Algorithm. (June 06,07,May 13,17,Dec 18)</b> An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.
2	<b>Define order of an algorithm. (June 07)</b> The order of an algorithm is a standard notation of an algorithm that has been developed to represent function that bound the computing time for algorithms. The order of an algorithm is a way of defining its efficiency. It is usually referred as Big O notation.
3	<b>Write about Notion of Algorithm.</b>  <pre> graph TD     problem --&gt; algorithm     algorithm --&gt; computer     input --&gt; computer     computer --&gt; output     style computer fill:#fff,stroke:#000,stroke-width:1px </pre>
4	<b>What are the characteristics of an algorithm?</b> <ul style="list-style-type: none"> <li>✓ Input</li> <li>✓ Output</li> <li>✓ Definiteness</li> <li>✓ Finiteness</li> <li>✓ Effectiveness.</li> </ul>
5	<b>What are the different criteria used to improve the effectiveness of algorithm? (June 06,07)</b> <ul style="list-style-type: none"> <li>✓ Input - Zero or more quantities are externally supplied.</li> <li>✓ Output - At least one quantity is produced.</li> <li>✓ Definiteness - Each instruction is clear and unambiguous.</li> <li>✓ Finiteness - if we trace out the instruction of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.</li> <li>✓ Effectiveness - Every instruction must be very basic.</li> </ul>
6	<b>What are the features of efficient algorithm?(Dec 07)</b> <ul style="list-style-type: none"> <li>✓ Free of ambiguity</li> <li>✓ Efficient in execution time</li> <li>✓ Concise and compact</li> <li>✓ Completeness</li> <li>✓ Definiteness</li> <li>✓ Finiteness</li> </ul>

7	<b>What are the steps involved in designing and analyzing an algorithm?</b> <ul style="list-style-type: none"> <li>✓ Understand the problem</li> <li>✓ Decide on <ul style="list-style-type: none"> <li>❖ Computational Device</li> <li>❖ Exact Vs Approximate Algorithms</li> <li>❖ Data Structures</li> <li>❖ Algorithm Design Techniques</li> </ul> </li> <li>✓ Design an algorithms</li> <li>✓ Prove Correctness</li> <li>✓ Analyze the Algorithm</li> <li>✓ Code the Algorithm</li> </ul>
8	<b>What is an algorithm design techniques? (Dec 06)</b> An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.
9	<b>Define Pseudo Code</b> Pseudo Code is a mixture of a natural language and programming language - like constructs such as functions, loops, and decision making statements. A pseudo is usually more precise than a natural language.
10	<b>Mention different algorithm design techniques.</b> <ul style="list-style-type: none"> <li>✓ Methods of specifying an algorithm</li> <li>✓ Proving an algorithms correctness</li> <li>✓ Analyzing an algorithm</li> <li>✓ Coding an algorithm</li> </ul>
11	<b>Mention the technique for proving correctness of an algorithm.</b> A common technique for proving correctness is to use mathematical induction because algorithms iterations provide a natural sequence of steps needed for such proof.
12	<b>What are the kinds of algorithm efficiency?</b> <ul style="list-style-type: none"> <li>✓ Time efficiency</li> <li>✓ Space efficiency</li> </ul>
13	<b>Define time efficiency.</b> Time efficiency indicates how fast the algorithm runs.
14	<b>What is space efficiency?</b> Space efficiency indicates how much extra memory the algorithm needs.
15	<b>Mention the desirable characteristics of an algorithm.(Dec18)</b> <ul style="list-style-type: none"> <li>✓ Simplicity</li> <li>✓ Generality</li> <li>✓ Optimality</li> </ul>

16	<b>Mention the most important problem types. (Dec 07, Apr 08)</b> <ul style="list-style-type: none"> <li>✓ Sorting</li> <li>✓ Searching</li> <li>✓ String processing</li> <li>✓ Graph problems</li> <li>✓ Combinatorial problems</li> <li>✓ Geometric problems</li> <li>✓ Numerical problems</li> </ul>
17	<b>Mention the sorting problems.</b> The sorting problems ask us to rearrange the items of a given list in ascending order.
18	<b>Mention the two properties of sorting algorithms.</b> <ul style="list-style-type: none"> <li>✓ A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input.</li> <li>✓ An algorithm is said to be in place if it does not require extra memory.</li> </ul>
19	<b>Mention the searching problems.</b> The searching problem deals with finding a given value, called a search key, in a given set.
20	<b>State travelling salesman problem (TSP).</b> The travelling salesman problem is the problem of finding the shortest tour through n cities that visits every exactly once.
21	<b>State graph-coloring problem.</b> The graph-coloring problem asks us to assign the smallest number of colors to vertices of a graph so that no two adjacent vertices are the same color.
22	<b>Give examples of combinatorial problems.</b> <ul style="list-style-type: none"> <li>✓ Travelling salesman problem</li> <li>✓ Graph-coloring problem</li> </ul>
23	<b>State combinatorial problems.</b> These are the problems that ask (explicitly or implicitly) to find a combinatorial object—such as a permutation, a combination, or a subset—that satisfies certain constraints and has some desired property (maximizes a value or minimizes a cost)
24	<b>State geometric algorithms.</b> Geometric algorithms deal with geometric objects such as points, lines, and polygons.
25	<b>Give examples of computational geometry.</b> <ul style="list-style-type: none"> <li>✓ Closet -pair problem</li> <li>✓ Convex-hull problem</li> </ul>
26	<b>State Closet -pair problem.</b> Given n points in the plane, find the closet pair among them.
27	<b>State Convex-hull problem</b> To find the smallest convex polygon that would include all the points of a given set.

28	<b>Define basic operation in algorithm.(Dec 18)</b> To identify the most important operation of the algorithm called the basic operation, the operation contributing the most to the total running time, and compute the number of times the basic operation is executed.
29	<b>What is the purpose of Instruction Space?</b> Instruction space is the space needed to store the compiled version of the program instructions.
30	<b>What is the purpose of Data Space?</b> Data space is the space needed to store all the constant and variable values.
31	<b>What is the component of Data Space?</b> <ul style="list-style-type: none"> <li>✓ Space needed by constants and simple variables.</li> <li>✓ Space needed by component variables such as array. This category includes space needed by structures, and dynamically allocated memory.</li> </ul>
32	<b>What is the purpose of Environment Stack Space?</b> The Environment stack space is used to save the information needed to resume execution of partially Completed methods. Examples: If F1 invokes F2, then we must save a pointer to the Instruction of F1 to be executed when F2 terminates.
33	<b>What are the factors that determine the amount of Instruction Space Needed?</b> <ul style="list-style-type: none"> <li>✓ The compiler used to compile the program into machine code.</li> <li>✓ The compiler options in effect at the time of compilation.</li> <li>✓ The target computer.</li> </ul>
34	<b>Write down formula for Space Complexity Calculation? (Dec 13)</b> $S(P) = c + S_p$ (instance characteristics). Where c is a constant that denotes the fixed part of the space requirements and $S_p$ denotes the variable component.
35	<b>Write down formula for Time Complexity Calculation?</b> The Time T (P) taken by a program P is the sum of the compile time and the run time. $T(p) = \text{Compile time} + \text{Run time}$
36	<b>Write the methods used for Estimating Run Time? or How to measure an algorithm's running time (Dec 17)</b> <ul style="list-style-type: none"> <li>✓ Identify one or more key operations and determine the number of times these are performed .It is referred as operation count</li> <li>✓ Determine the total number of steps executed by the program. It is referred as step count</li> </ul>
37	<b>Define Program Step?</b> A program step is loosely defined to be a syntactically or semantically meaningful segment of a program for which the execution time is independent of the instance characteristics.
38	<b>Define Operation Count?</b> It is the method to estimate the time complexity of a program by selecting one or more operations, such as add, multiply and compare and to determine how many of each is done. The success of this method depends on our ability to identify the operations that contribute most to the time complexity.

39	<b>Define Step Count?</b> Step count determines the total no of steps executed by the program.
40	<b>Write the reason for determining operation count and step count.</b> Two important reasons to determine operation and step counts are <ul style="list-style-type: none"> <li>✓ To compare the time complexities of two programs that compute the same function</li> <li>✓ To predict the growth in run time as the instance characteristic change.</li> </ul>
41	<b>Define Asymptotic Notations.</b> The notation that will enable us to make meaningful statements about the time and space complexities of a program. This notation is called asymptotic notation.
42	<b>Mention the various Asymptotic Notations.(Dec 06)</b> <ul style="list-style-type: none"> <li>✓ Big Oh notation,</li> <li>✓ Theta notation</li> <li>✓ Omega notation</li> <li>✓ Little Oh notation.</li> </ul>
43	<b>Define Big 'O' Notation. (June 06, May 13)</b> $f(n) = O(g(n))$ iff positive constants $c$ and $n_0$ exist such that $f(n) \leq cg(n)$ for all $n, n \geq n_0$ . The definition states that the function $f$ is at most $c$ times the function $g$ except possibly when $n$ is smaller than $n_0$ .
44	<b>When is the function <math>f(n)</math> said to be <math>O(g(n))</math>? (Dec 18)</b> $f(n) = O(g(n))$ iff positive constants $c$ and $n_0$ exist such that $f(n) \leq cg(n)$ for all $n, n \geq n_0$ .
45	<b>Define big'O' ratio theorem?</b> Let $f(n)$ and $g(n)$ be such that $\lim_{n \rightarrow \infty} f(n)/g(n)$ exists . $f(n) = O(g(n))$ iff $\lim_{n \rightarrow \infty} f(n)/g(n) \leq c$ for some finite constant $c$ .
46	<b>Define Omega Notation. (June 07)</b> $f(n) = \Omega(g(n))$ iff positive constants $c$ and $n_0$ exist such that $f(n) \geq cg(n)$ for all $n, n \geq n_0$ . The definition states that the function $f$ is at least $c$ times the function $g$ except possibly when $n$ is smaller than $n_0$ . Here $c$ is some positive constant. Thus $g$ is a lower bound on the value of $f$ for all suitably large $n$ .
47	<b>Define Omega Ratio Theorem.</b> Let $f(n)$ and $g(n)$ be such that $\lim_{n \rightarrow \infty} g(n)/f(n)$ exists . $f(n) = \Omega(g(n))$ iff $\lim_{n \rightarrow \infty} g(n)/f(n) \leq c$ for some finite constant $c$ .
48	<b>When is the function <math>f(n)</math> said to be <math>\Omega(g(n))</math>?</b> $f(n) = \Omega(g(n))$ iff positive constants $c$ and $n_0$ exist such that $f(n) \geq cg(n)$ for all $n, n \geq n_0$ .

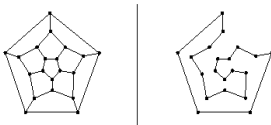
49	<p><b>Define Theta Notation. (Dec 14)</b></p> <p><math>f(n) = \Theta(g(n))</math> iff positive constants <math>c_1</math> and <math>c_2</math> and an <math>n_0</math> exist such that <math>c_1g(n) \leq f(n) \leq c_2g(n)</math> for all <math>n, n \geq n_0</math>.</p> <p>The definition states that the function <math>f</math> lies between <math>c_1</math> times the function <math>g</math> and <math>c_2</math> times the function <math>g</math> except possibly when <math>n</math> is smaller than <math>n_0</math>. Here <math>c_1</math> and <math>c_2</math> are positive constants. Thus <math>g</math> is both a lower and upper bound on the value of <math>f</math> for all suitably large <math>n</math>.</p>																											
50	<p><b>When is the function <math>f(n)</math> said to be <math>\Theta(g(n))</math>?</b></p> <p><math>f(n) = \Theta(g(n))</math> iff positive constants <math>c_1</math> and <math>c_2</math> and an <math>n_0</math> exist such that <math>c_1g(n) \leq f(n) \leq c_2g(n)</math> for all <math>n, n \geq n_0</math>.</p>																											
51	<p><b>Define Theta Ratio Theorem.</b></p> <p>Let <math>f(n)</math> and <math>g(n)</math> be such that <math>\lim_{n \rightarrow \infty} f(n)/g(n)</math> and <math>\lim_{n \rightarrow \infty} g(n)/f(n)</math> exists.</p> <p><math>f(n) = \Theta(g(n))</math> iff <math>\lim_{n \rightarrow \infty} f(n)/g(n) \leq c</math> and <math>\lim_{n \rightarrow \infty} g(n)/f(n) \leq c</math> for some finite constant <math>c</math>.</p>																											
52	<p><b>List the properties of asymptotic notations. (June 06, May 15)</b></p> <ul style="list-style-type: none"><li>✓ Sum rule : <math>O(f(n)) + O(g(n)) = O(\max\{f(n), g(n)\})</math></li><li>✓ Transitivity rule : <math>f(n) = O(g(n))</math> and <math>g(n) = O(h(n))</math> then <math>f(n) = O(h(n))</math></li><li>✓ Reflexivity: <math>f(n) = O(f(n))</math></li><li>✓ Any constant value is equivalent to <math>O(1)</math>.</li><li>✓ Polynomial Rule: Let <math>f(n)</math> be any polynomial in <math>n</math>, with <math>k</math> being the highest exponent. <math>f(n) = O(n^k)</math></li></ul>																											
53	<p><b>What is worst-case efficiency? (Dec18)</b></p> <p>The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size <math>n</math>, which is an input or inputs of size <math>n</math> for which the algorithm runs the longest among all possible inputs of that size.</p>																											
54	<p><b>Write the basic asymptotic efficiency classes.</b></p> <table><thead><tr><th>Class</th><th>Name</th><th>Example</th></tr></thead><tbody><tr><td>1</td><td>constant</td><td>access array element</td></tr><tr><td><math>\log n</math></td><td>logarithmic</td><td>binary search</td></tr><tr><td><math>n</math></td><td>linear</td><td>find median</td></tr><tr><td><math>n \log n</math></td><td>"n-log-n"</td><td>mergesort</td></tr><tr><td><math>n^2</math></td><td>quadratic</td><td>insertion sort</td></tr><tr><td><math>n^3</math></td><td>cubic</td><td>matrix multiplication</td></tr><tr><td><math>a^n</math></td><td>exponential</td><td>generating all subsets</td></tr><tr><td><math>n!</math></td><td>factorial</td><td>generating all permutations</td></tr></tbody></table>	Class	Name	Example	1	constant	access array element	$\log n$	logarithmic	binary search	$n$	linear	find median	$n \log n$	"n-log-n"	mergesort	$n^2$	quadratic	insertion sort	$n^3$	cubic	matrix multiplication	$a^n$	exponential	generating all subsets	$n!$	factorial	generating all permutations
Class	Name	Example																										
1	constant	access array element																										
$\log n$	logarithmic	binary search																										
$n$	linear	find median																										
$n \log n$	"n-log-n"	mergesort																										
$n^2$	quadratic	insertion sort																										
$n^3$	cubic	matrix multiplication																										
$a^n$	exponential	generating all subsets																										
$n!$	factorial	generating all permutations																										
55	<p><b>What is best-case efficiency? (Dec18)</b></p> <p>The best-case efficiency of an algorithm is its efficiency for the best-case input of size <math>n</math>, which is an input or inputs for which the algorithm runs the fastest among all possible inputs of that size.</p>																											

56	<p><b>What is average case efficiency? (Dec18)</b></p> <p>The average case efficiency of an algorithm is its efficiency for an average case input of size <math>n</math>. It provides information about an algorithm behavior on a “typical” or “random” input.</p>
57	<p><b>What are the classifications of algorithm?</b></p> <ul style="list-style-type: none"> <li>✓ Recursive algorithm</li> <li>✓ Non-recursive algorithm</li> </ul>
58	<p><b>What is recursive algorithm?</b></p> <p>An algorithm is said to be recursive if the same algorithm is invoked in the body. An algorithm that calls itself is direct recursive. Algorithm A is said to be indeed recursive if it calls another algorithm, which in turn calls A.</p>
59	<p><b>Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer. (May 15)</b></p> <p><b>ALGORITHM</b> <i>Binary</i>(<math>n</math>)</p> <p>//Input: A positive decimal integer <math>n</math></p> <p>//Output: The number of binary digits in <math>n</math>'s binary representation</p> <p><math>count \leftarrow 1</math></p> <p><b>while</b> <math>n &gt; 1</math> <b>do</b></p> <p><math>count \leftarrow count + 1</math></p> <p><math>n \leftarrow \lfloor n/2 \rfloor</math></p> <p><b>return</b> <math>count</math></p>
60	<p><b>Give the Euclid's algorithm for computing gcd(<math>m, n</math>) or Write an algorithm to compute the greatest common divisor of two numbers (May 16,17,Dec18)</b></p> <p><b>ALGORITHM</b> <i>Euclid</i>(<math>m, n</math>)</p> <p>//Computes gcd(<math>m, n</math>) by Euclid's algorithm</p> <p>//Input: Two nonnegative, not-both-zero integers <math>m</math> and <math>n</math></p> <p>//Output: Greatest common divisor of <math>m</math> and <math>n</math></p> <p><b>while</b> <math>n \neq 0</math> <b>do</b></p> <p><math>r \leftarrow m \bmod n</math></p> <p><math>m \leftarrow n</math></p> <p><math>n \leftarrow r</math></p> <p><b>return</b> <math>m</math></p>
61	<p><b>Compare the orders of growth of <math>n(n-1)/2</math> and <math>n^2</math>. (May 16)</b></p> $\lim_{n \rightarrow \infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{n^2} = \frac{1}{2} \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right) = \frac{1}{2}.$ <p>Since the limit is equal to a positive constant, the functions have the same order of growth or, symbolically, <math>\frac{1}{2}n(n-1) \in \Theta(n^2)</math>.</p>

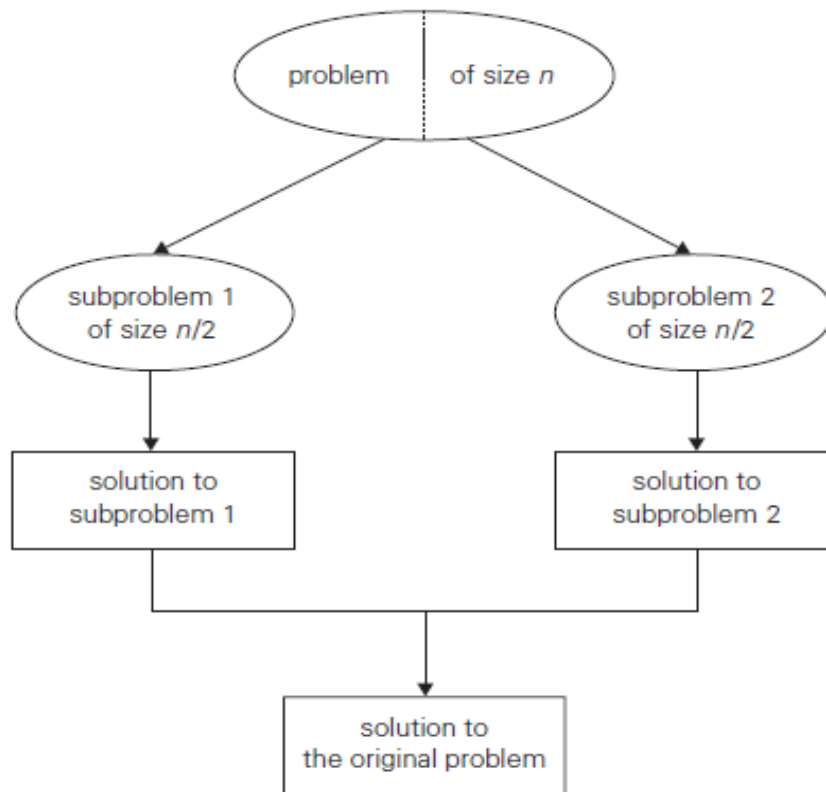
62	<p><b>Define Visualization.</b></p> <p>Algorithm visualization and can be defined as the use of images to convey some useful information about algorithms. That information can be a visual illustration of an algorithm's operation, of its performance on different kinds of inputs, or of its execution speed versus that of other algorithms for the same problem. To accomplish this goal, algorithm visualization uses graphic elements points, line segments, two- or three-dimensional bars, and so onto represent some "interesting events" in the algorithm's operation.</p>
<b>UNIT-I / PART-B</b>	
1	Describe briefly the notions of complexity of an algorithm. <i>(June 07)</i>
2	Explain the fundamentals of algorithmic problem solving. (or) Discuss briefly the sequence of steps in designing and analyzing an algorithm. <i>(Dec 06)</i>
3	Explain the various asymptotic notations of an algorithm in detail (or) Explain briefly Big oh Notation, Omega Notation and Theta Notations. Give examples. Give an account of basic efficiency classes <i>(Dec 07, May 17, May 18, Nov 18)</i>
4	Elaborate on how space and time complexities are calculated. Give Examples. <i>(May 13)</i>
5	Explain the general framework for analyzing the efficiency of algorithm. (or) Discuss the fundamentals of analysis framework. <i>(Dec 06, 07)</i>
6	Explain some of the important problem types used in the design of algorithm with an example. <i>(Dec 06, 07)</i>
7	a) List out the properties of asymptotic notations or big O. <i>(Dec 14)</i> b) Give a short account of basic efficiency classes <i>(Dec 06)</i>
8	Explain the following terms <i>(June 07)</i> a) Searching , b) Asymptotic notations, c) Analysis framework, d) Algorithm efficiency
9	Briefly explain the mathematical analysis of recursive and non-recursive algorithm. <i>(June 07, May 17)</i>
10	a) Discuss the general plan for analyzing the efficiency of non-recursive algorithm. <i>(Dec 07)</i> b) Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer and analyze its efficiency. <i>(Dec 07, May 16)</i>
11	Design a non-recursive algorithm for computing the product of two $n \times n$ matrices and also find time efficiency of algorithm. <i>(Dec 06)</i>
12	Design a recursive algorithm to compute the factorial function $F(n) = n!$ and derive the recurrence relation. <i>(Dec 06, 17)</i>
13	Design a recursive algorithm to find the number of moves in tower of Hanoi problem and find the time of complexity. <i>(Dec 13, May 18)</i>
14	a) Solve the recurrence equation of the Fibonacci Series. $T(n) = T(n-1) + T(n-2)$ subject to $T(0)=0, T(1) = 1$ . <i>(June 07, May 08)</i> b) Write the general plan for analyzing time efficiency of non-recursive algorithms and find the time complexity of element uniqueness problem. <i>(Dec 07, May 16)</i>



15	Derive a loose bound on the following equation: $F(x)=35x^8-22x^7+14x^5-2x^4-4x^2+x-15$ . (May 15)
16	If you have to solve the searching problem for a list of $n$ numbers, how can you take advantage of the fact that the list is known to be sorted? Give separate answers for (i) Lists represented as arrays. (ii) Lists represented as linked lists. Compare the time complexities involved in the analysis of both the algorithms. (May 15)
17	Give the definition and Graphical Representation of O-Notation. (May 16)
18	Explain the algorithm visualization and its applications in detail.
<p style="text-align: center;"><b>BRUTE FORCE AND DIVIDE-AND-CONQUER</b></p> <p>Brute Force – Computing an – String Matching – Closest-Pair and Convex-Hull Problems – Exhaustive Search – Travelling Salesman Problem – Knapsack Problem – Assignment problem. Divide and Conquer Methodology – Binary Search – Merge sort – Quick sort – Heap Sort – Multiplication of Large Integers – Closest-Pair and Convex – Hull Problems.</p>	
<b>UNIT-II/ PART-A</b>	
1	<p><b>What is Brute Force?</b></p> <p>Brute Force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.</p>
2	<p><b>What would be the most straightforward method for solving the sorting problem?</b></p> <p>Selection sort and bubble sort are the two better algorithms and it implements brute-force approach more clearly.</p>
3	<p><b>What are the advantages and disadvantages of brute force approach?</b></p> <p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>✓ applicable to a wide variety of problems</li> <li>✓ for important problems it yields reasonable algorithms of at least some practical value with no limitation on instance size</li> <li>✓ It can solve few instances of a problem with acceptable speed</li> <li>✓ simple to design and may be useful to solve small problem instances</li> </ul> <p><b>Disadvantage:</b></p> <ul style="list-style-type: none"> <li>✓ inefficient in general case</li> </ul>
4	<p><b>When is sorting method said to be stable?</b></p> <p>A sorting method is said to be stable when it has minimum number of swaps (less than 'n' number of comparisons) i.e. if the two data items of matching value are guaranteed not to be rearranged with respect to each other when the algorithm progresses.</p>
5	<p><b>Define unstable sort.</b></p> <p>A sorting method is said to be Unstable when it has maximum number of swaps (greater than 'n' number of comparisons) i.e. if the two data items of matching value are guaranteed not to be rearranged with respect to each other when the algorithm progresses.</p>

6	<p><b>List out some of the stable and unstable sorting techniques.</b></p> <p><b>Stable sorting techniques includes</b></p> <ul style="list-style-type: none"> <li>✓ Bubble sort</li> <li>✓ Insertion sort</li> <li>✓ Selection sort</li> <li>✓ Merge sort</li> </ul> <p><b>Unstable sorting techniques includes</b></p> <ul style="list-style-type: none"> <li>✓ Shell sort</li> <li>✓ Quick sort</li> <li>✓ Radix sort</li> <li>✓ Heap sort</li> </ul>
7	<p><b>Define Closet -pair problem. (May 16,17)</b></p> <p>Given n points in the plane, find the closet pair among them. Brute force approach</p>
8	<p><b>Define Convex.</b></p> <p>A set of points (finite or infinite) on the plane is called convex if for any two points P and Q in the set, the entire line segment with the end points at P and Q belongs to the set.</p>
9	<p><b>Define Convex-Hull.</b></p> <p>The Convex-Hull of a set S of points is the smallest convex set containing S. The smallest requirements means that the convex-hull of S must be a sub set of any convex set containing S.</p>
10	<p><b>Define Convex-Hull Problem.</b></p> <p>The problem of constructing the convex-hull for a given set S of n points.</p>
11	<p><b>Define Extreme points.</b></p> <p>The extreme point of a convex set is a point of set S that is not a middle point of any line segment with end points in the set.</p>
12	<p><b>Define Exhaustive search. (May 18)</b></p> <p>Exhaustive search is simply a brute-force approach to combinatorial problems. It suggests generating each and every element of the problem's domain, selecting those of them that satisfy all the constraints, and then finding a desired element.</p>
13	<p><b>Define travelling salesman problem.</b></p> <p>Travelling salesman problem finds the shortest tour through a given set of n cities that visits each city exactly once before returning to the city where it is started.</p>
14	<p><b>Define Hamiltonian circuit. (Dec 18)</b></p> <p>It is defined as a cycle that passes through all the vertices of a graph exactly once.</p> <div style="text-align: center;">  </div> <div style="display: flex; justify-content: space-around; margin-top: 5px;"> <span>INPUT</span> <span>OUTPUT</span> </div>

15	<p><b>Define Knapsack problem.(Dec 14)</b></p> <p>Given <math>n</math> items of known weights <math>w_1 \dots w_n</math> and values <math>v_1 \dots v_n</math> and knapsack of capacity <math>W</math>. The aim is to find the most valuable subset of the items that fit into the knapsack. The exhaustive search approach to knapsack problem leads to generating all the subsets of the set of <math>n</math> items given, computing the total weight of each subset to identify feasible subsets and finding a subset of the largest value among them.</p>
16	<p><b>Define assignment problem. (May 16)</b></p> <p>There are <math>n</math> people who need to be assigned to execute <math>n</math> jobs, one person per job. The cost that would accrue if the <math>i^{\text{th}}</math> person is assigned to the <math>j^{\text{th}}</math> job is a known quantity <math>c[i,j]</math> for each pair <math>i, j=1, \dots, n</math>. The problem is to find an assignment with the smallest total cost.</p>
17	<p><b>Define merge sort.</b></p> <p>The merge sort algorithm divides a given array <math>A[0..n-1]</math> by dividing it into two halves <math>A[0.. \lfloor n/2 \rfloor - 1]</math> and <math>A[\lfloor n/2 \rfloor .. n-1]</math>, sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one.</p> <p>If the list has even length, split the list into two equal sub lists. If the list has odd length, divide the list in two by making the first sub list one entry greater than the second sub list. Then split both the sub lists in two and go on until each of the sub lists are of size one. Finally start merging the individual sub lists to obtain a sorted list.</p>
18	<p><b>Define quick sort.</b></p> <p>Quick sort employs a divide-and-conquer strategy. It starts by picking an element from the list to be the "pivot." It then reorders the list so that all elements with values less than the pivot come before the pivot, and all elements with values greater than the pivot come after it (a process often called "partitioning"). It then sorts the sub-lists to the left and the right of the pivot using the same strategy, continuing this process recursively until the whole list is sorted.</p>
19	<p><b>What is a pivot element?</b></p> <p>The pivot element is the chosen number which is used to divide the unsorted data into two halves. The lower half contains less than value of the chosen number i.e. pivot element. The upper half contains greater than value of the chosen number i.e. pivot element. So the chosen number is now sorted.</p>
20	<p><b>Define divide and conquer design technique. (May 13,16,Dec17)</b></p> <ul style="list-style-type: none"> <li>✓ A problem's instance is divided into several smaller instances of the same problem, ideally of about the same size.</li> <li>✓ The smaller instances are solved</li> <li>✓ If necessary, the solutions obtained for the smaller instances are combined to get a solution to the original instance.</li> </ul>



21	<b>What is median-of-three -portioning method?</b> The median-of-three-portioning method is employed in quick sort to find the pivot value. This is done by randomly choosing three elements in the list and finding the median of the elements gives the pivot value.
22	<b>Define Binary Search. (Dec 07)</b> Binary search is a efficient algorithm for searching in a sorted array. It works by comparing a search key $K$ with the array's middle element $A[m]$ . If they match, the algorithm stops; otherwise, the same operation is repeated recursively for the first half of the array if $K < A[m]$ and for the second half if $K > A[m]$ .
23	<b>Define Strassen's matrix multiplication. (Dec 07)</b> Strassen showed that $2 \times 2$ matrix multiplications can be accomplished in 7 multiplication and 18 additions or subtractions. ( $2 \log_2 7 = 22.807$ ). This reduce can be done by Divide and Conquer Approach.
24	<b>How many multiplications are performed in two n-digit multiplication?</b> There is a divide-and-conquer algorithm for multiplying two $n$ -digit integers that requires about $n^{1.585}$ one-digit multiplications.
25	<b>What are the advantages of insertion sort?(Nov 2017)</b> The main advantage of the insertion sort is its simplicity. <ul style="list-style-type: none"> <li>✓ It also exhibits a good performance when dealing with a small list.</li> <li>✓ The insertion sort is an in-place sorting algorithm so the space requirement is minimal.</li> </ul>

## 26 Design a brute-force algorithm for computing the value of a polynomial

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

at a given point  $x^0$  and determine its worst-case efficiency class. (May 15)

**Algorithm** *BruteForcePolynomialEvaluation*( $P[0..n], x$ )  
 //The algorithm computes the value of polynomial  $P$  at a given point  $x$   
 //by the “highest-to-lowest term” brute-force algorithm  
 //Input: Array  $P[0..n]$  of the coefficients of a polynomial of degree  $n$ ,  
 // stored from the lowest to the highest and a number  $x$   
 //Output: The value of the polynomial at the point  $x$   
 $p \leftarrow 0.0$   
**for**  $i \leftarrow n$  **downto** 0 **do**  
      $power \leftarrow 1$   
     **for**  $j \leftarrow 1$  **to**  $i$  **do**  
          $power \leftarrow power * x$   
      $p \leftarrow p + P[i] * power$   
**return**  $p$

We will measure the input's size by the polynomial's degree  $n$ . The basic operation of this algorithm is a multiplication of two numbers; the number of multiplications  $M(n)$  depends on the polynomial's degree only. Although it is not difficult to find the total number of multiplications in this algorithm, we can count just the number of multiplications in the algorithm's inner-most loop to find the algorithm's efficiency class:

$$M(n) = \sum_{i=0}^n \sum_{j=1}^i 1 = \sum_{i=0}^n i = \frac{n(n+1)}{2} \in \Theta(n^2).$$

## 27 Derive the complexity of binary search algorithm. (May 15)

Let us find the number of key comparisons in the worst case  $C_{worst}(n)$ . The worst-case inputs include all arrays that do not contain a given search key, as well as some successful searches. Since after one comparison the algorithm faces the same situation but for an array half the size, we get the following recurrence relation for  $C_{worst}(n)$ :

$$C_{worst}(n) = C_{worst}(\lfloor n/2 \rfloor) + 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 1.$$

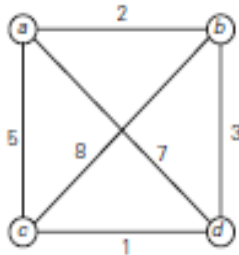
For the initial condition  $C_{worst}(1) = 1$ , we obtain

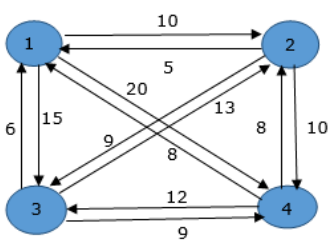
$$C_{worst}(2^k) = k + 1 = \log_2 n + 1.$$

For  $n = 2^k$  can be tweaked to get a solution valid for an arbitrary positive integer  $n$ :

$$C_{worst}(n) = \lfloor \log_2 n \rfloor + 1 = \lceil \log_2(n+1) \rceil.$$

28	<p><b>Devise an algorithm to make a change for 1655 using the Greedy strategy. The coins available are {1000, 500, 100, 50, 20, 10, 5}. (May 17)</b></p> <p>Algorithm:</p> <pre> While(there are more coins and the instance is not solved) { grab the largest remaining coin; // selection procedure if(adding the coin makes the change exceed the amount owed ) reject the coin; // feasibility check else add the coin to the change; if( the total value of the change equals the amount owed ) // solution check the instance is solved; } Solution for the given instance 1655 = 1000 + 500 +100 + 50 + 5. </pre>
29	<p><b>Define Heap sort.</b></p> <p>A heap can be defined as a binary tree with keys assigned to its nodes, one key per node, provided the following two conditions are met:</p> <ul style="list-style-type: none"> <li>✓ The shape property – the binary tree is essentially complete (or simply complete), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.</li> <li>✓ The parental dominance or heap property – the key in each node is greater than or equal to the keys in its children.</li> </ul>
30	<p><b>Explain String matching problem.</b></p> <p>String-matching problem: Given a string of <math>n</math> characters called the text and a string of <math>m</math> characters (<math>m \leq n</math>) called the pattern; find a substring of the text that matches the pattern. To put it more precisely, we want to find <math>i</math> – the index of the leftmost character of the first matching substring in the text – such that <math>t_i = p_0, \dots, t_{i+j} = p_j, \dots, t_{i+m-1} = p_{m-1}</math>:</p> <div style="text-align: center;"> <math display="block">  \begin{array}{ccccccccccc}  t_0 &amp; \dots &amp; t_i &amp; \dots &amp; t_{i+j} &amp; \dots &amp; t_{i+m-1} &amp; \dots &amp; t_{n-1} &amp; \text{text } T \\  &amp; &amp; \updownarrow &amp; &amp; \updownarrow &amp; &amp; \updownarrow &amp; &amp; &amp; \\  &amp; &amp; p_0 &amp; \dots &amp; p_j &amp; \dots &amp; p_{m-1} &amp; &amp; &amp; \text{pattern } P  \end{array}  </math> </div>
<b>UNIT-II/ PART-B</b>	
1	Explain how brute force approach is applied to solve closest-Pair and convex-Hull problem. (Dec 17)
2	Explain the concept of travelling salesman problem, knapsack problem and assignment problem using exhaustive search.
3	Explain the concept of knapsack problem using exhaustive search.
5	Explain in detail about merge sort. Illustrate the algorithm with a numeric example. (Dec 06,13,14,May 16,18)
6	Apply merge sort to sort the list E, X, A, M, P, L, E in alphabetical order.
7	Write an algorithm for merge sorting. Show the intermediate steps when the numbers 310, 285, 179, 652, 351, 423, 861, 254, 450, 520 are sorted using Merge Sort. (Dec 14)
8	Devise an algorithm to sort the following elements using Merge sort technique 286, 45, 278, 368, 475, 389, 656, 788, 503, 126. (Dec 13)

9	Distinguish between quick sort and merge sort, and arrange the following numbers in increasing order using merge sort.(18,29,68,32,43,37,87,24,47,50). (May 13)
10	Explain quick sort using divide and conquer method.
11	Sort the following set of elements using quick sort. 12,24,8,71,4,23,6 (June 06)
12	Write an algorithm for binary search using divide and conquer and analyze the time complexity. (June 06, Dec 14) (or) What is divide and conquer strategy and explain the binary search with suitable example. (May 17)
13	Apply binary search algorithm to find the key value 32 in the following list of elements. 1,7,12, 16, 18, 21,24,32,34.
14	Explain binary search algorithm in detail. Using Binary Search, search for the elements 151, -14, and 9 in the given set of sorted elements 15, 6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151.
15	Write Strassen's matrix multiplication algorithm. Is there any time efficiency improvement compared to ordinary matrix multiplication?
16	Explain how divide and conquer method is applied in multiplication of large integers? (May 16)
17	Compute $2101 \times 1130$ by applying the divide and conquer algorithm.
18	Explain how divide and conquer method is applied to solve closest-Pair and convex-Hull problem. (May 15)
19	Derive the worst case analysis of Merge Sort using suitable illustrations.(May 15)
20	Solve the following using Brute-Force algorithm: Find whether the given string follows the specified pattern and return 0 or 1 accordingly.(May 15,17) Examples: (1) Pattern: "abba", input:"redblueredblue" should return 1 (2) Pattern: "aaaa", input:"asdadasdasd" should return 1 (3) Pattern: "aabb", input:"xyzabcxzyabc" should return 0
21	Explain the convex hull problem and the solution involved behind it. (May 15)
22	A pair contains two numbers, and its second number is on the right side of the first one in an array. The difference of a pair is the minus result while subtracting the second number from the first one. Implement a function which gets the maximal difference of all pairs in an array (using Divide and Conquer method). (May 15)
23	Find all the solution to the travelling salesman problem (cities and distances shown below) by exhaustive search. Give the optimal solution. (May 16) 

24	Write an algorithm for quick sort and write its time complexity with example list are 5, 3,1,9,8,2,4,7 (May 17)
	How the solution can be obtained using branch and bound method.(Nov 2018) 
25	Explain in detail about Heap sort with an example.
26	Explain how brute force approach is applied for computing an and string matching problem.
27	Solve travelling salesman problem using brute force approach for the given example.
<p align="center"><b>DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE</b></p> <p>Dynamic programming – Principle of optimality - Coin changing problem, Computing a Binomial Coefficient – Floyd’s algorithm – Multi stage graph - Optimal Binary Search Trees – Knapsack Problem and Memory functions.</p> <p>Greedy Technique – Container loading problem - Prim’s algorithm and Kruskal's Algorithm – 0/1 Knapsack problem, Optimal Merge pattern - Huffman Trees.</p>	
<b>UNIT-III/ PART-A</b>	
1	<p><b>What do you mean by dynamic programming? (May 17)</b></p> <p>Dynamic programming is a technique for solving problems with overlapping sub problems. Rather than solving overlapping sub problems again and again, dynamic programming suggests solving each of the smaller sub problems only once and recording the results in a table from which a solution to the original problem can then be obtained.</p>
2	<p><b>What does dynamic programming have in common with divide and conquer?</b></p> <p>Both the techniques are based on dividing a problems instance into smaller instances of the same problem.</p>
3	<p><b>What is the difference between dynamic programming with divide and conquer method?</b></p> <p>Divide and conquer divides an instance into smaller instances with no intersections whereas dynamic programming deals with problems in which smaller instances overlap. Consequently divide and conquer algorithm do not explicitly store solutions to smaller instances and dynamic programming algorithms do.</p>
4	<p><b>Define binomial coefficient.</b></p> <p>A binomial coefficient <math>C(n, k)</math> can be defined as the coefficient of <math>X^k</math> in the expansion of <math>(1 + X)^n</math>. A binomial coefficient <math>C(n, k)</math> also gives the number of ways, disregarding order that <math>k</math> objects can be chosen from among <math>n</math> objects; more formally, the number of <math>k</math>-element subsets or <math>k</math>-combinations of an <math>n</math>-element set.</p>



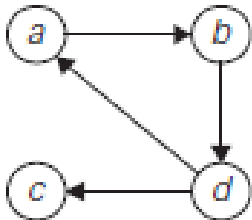
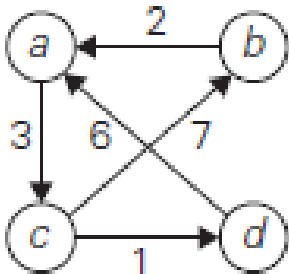
5	<p><b>Write the equation for calculating binomial coefficient.</b></p> <p>The value of <math>C(n, k)</math> can recursively calculated using following standard formula for Binomial coefficients that <math>k</math> objects can be chosen from among <math>n</math> objects.</p> $C(n, k) = C(n-1, k-1) + C(n-1, k)$ $C(n, 0) = C(n, n) = 1$
6	<p><b>Which algorithm is used to find out the transitive closure of a matrix?</b></p> <p>Warshall algorithm is used to find the transitive closure of the given digraph</p>
7	<p><b>What is the Purpose of the Floyd algorithm?</b></p> <p>The Floyd's algorithm is used to find the shortest distance between every pair of vertices in a graph.</p>
8	<p><b>What are the conditions involved in the Floyd's algorithm?</b></p> <ul style="list-style-type: none"> <li>✓ Construct the adjacency matrix.</li> <li>✓ Set the diagonal elements to zero</li> <li>✓ <math>A_k[i,j] = \min \{A_{k-1}[i,j], A_{k-1}[i,k] \text{ and } A_{k-1}[k,j]\}</math></li> </ul>
9	<p><b>Mention the methods for generating transitive closure of digraph.</b></p> <ul style="list-style-type: none"> <li>✓ Depth First Search (DFS)</li> <li>✓ Breadth First Search (BFS)</li> </ul>
10	<p><b>Define principle of optimality. (Dec 14)</b></p> <p>It states that an optimal sequence of decisions has the property that whenever the initial stage or decisions must constitute an optimal sequence with regard to stage resulting from the first decision</p>
11	<p><b>Define Optimal Binary Search Tree (OBST). (June 06)</b></p> <p>Dynamic programming can be used for constructing an optimal binary search tree for a given set of keys and known probabilities of searching for them. If probabilities of searching for an element of a set are known, the average number of comparisons in a search will have smallest possible value in an OBST.</p>
12	<p><b>Define knapsack problem using dynamic programming.</b></p> <p>Designing a dynamic programming algorithm for the knapsack problem: given <math>n</math> items of known weights <math>w_1, \dots, w_n</math> and values <math>v_1, \dots, v_n</math> and a knapsack of capacity <math>W</math>, find the most valuable subset of the items that fit into the knapsack. We assume here that all the weights and the knapsack capacity are positive integers; the item values do not have to be integers.</p>
13	<p><b>Define feasible solution. (Dec 13)</b></p> <p>Any subset that satisfies problem's constraints called feasible solution.</p>
14	<p><b>Define Memory function techniques. (Dec 06)</b></p> <p>The memory function technique seeks to combine strengths of the top-down and bottom-up approaches to solving problems with overlapping sub problems. It does this by solving, in the top-down fashion but only once, just necessary sub problems of a given problem and recording their solutions in a table.</p>

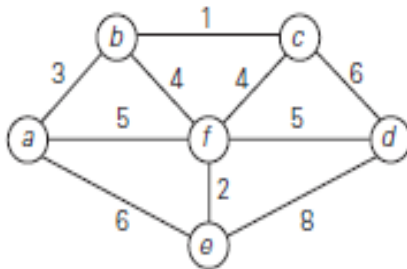
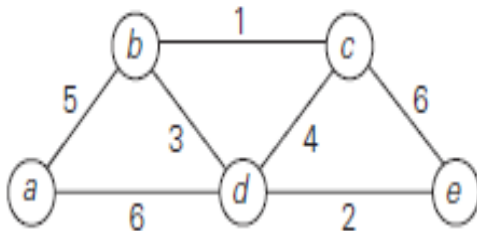
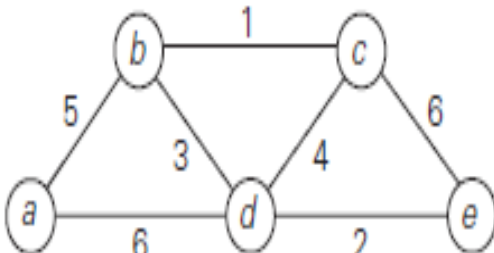
15	<p><b>State the general principle of greedy algorithm. (May 17)</b></p> <p>The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached.</p>
16	<p><b>Define objective function.</b></p> <p>To find a feasible solution that either maximizes or minimizes a given objective function.</p>
17	<p><b>Define optimal solution. (June 07, Dec 13)</b></p> <p>It has to be the best choice among all feasible solution available on that step.</p>
18	<p><b>Mention the two classic algorithms for the minimum spanning tree problem.</b></p> <ul style="list-style-type: none"> <li>✓ Prim's algorithm</li> <li>✓ Kruskal's algorithm</li> </ul>
19	<p><b>Define spanning tree.</b></p> <p>A spanning tree of a connected graph is its connected acyclic sub graph that contains all the vertices of the graph</p>
20	<p><b>Define minimum spanning tree. (June 06, 07, May 18)</b></p> <p>A minimum spanning tree of a weighted connected graph is its spanning tree of the smallest weight.</p>
21	<p><b>Define weight of a tree.</b></p> <p>Weight of a tree is defined as the sum of the weights on all its edges.</p>
22	<p><b>State two obstacles for constructing minimum spanning tree using exhaustive-search approach.</b></p> <ul style="list-style-type: none"> <li>✓ The number spanning tree grows exponentially with the graph size</li> <li>✓ Generating all spanning trees for a given graph is not easy; in fact, it is more difficult than finding a minimum spanning tree for a weighted graph by using one of several efficient algorithms available for this problem.</li> </ul>
23	<p><b>Does prim's algorithm always yield a minimum spanning tree?</b></p> <p>Yes. prim's algorithm always yield a minimum spanning tree</p>
24	<p><b>Define minimum spanning tree problem.</b></p> <p>A minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.</p>
25	<p><b>Write the concept of Prim's spanning tree.</b></p> <p>Prim's algorithm constructs a minimum spanning tree through a sequence of expanding sub trees. The initial sub tree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices. On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. The algorithm stops after all the graph's vertices have been included in the tree being constructed</p>
26	<p><b>How efficient is prim's algorithm?</b></p> <p>It depends on the data structures chosen for the graph itself and for the priority queue of the set V-VT whose vertex priorities are the distances to the nearest tree vertices.</p>

27	<b>How priority queues are implemented?</b> We can also implement the priority queues as a min-heap.
28	<b>Define min-heap.</b> A min-heap is a complete binary tree in which every element is less than or equal to its children.
29	<b>What is the purpose of Dijkstra's Algorithm?</b> Dijkstra's algorithm is used to find the shortest path between sources to every vertex. This algorithm is applicable to undirected and directed graphs with nonnegative weights only.
30	<b>Define code word.</b> Encode a text that comprises symbols from some n-symbol alphabet by assigning to each of the text's symbols some sequence of bits called the code word.
31	<b>Define the single source shortest paths problem. (May 16)</b> Dijkstra's algorithm solves the single-source shortest-path problem of finding shortest paths from a given vertex (the source) to all the other vertices of a weighted graph or digraph. It works as Prim's algorithm but compares path lengths rather than edge lengths. Dijkstra's algorithm always yields a correct solution for a graph with nonnegative weights.
32	<b>Write the concept of kruskal's algorithm.</b> Kruskal's algorithm looks at a minimum spanning tree for a weighted connected graph $G=(V,E)$ as an acyclic sub graph with $ V -1$ edges for which the sum of the edge weights is the smallest. Consequently, the algorithm constructs a minimum spanning tree as an expanding sequence of sub graphs, which are always acyclic but are not necessarily connected on the intermediate stages of the algorithm. The algorithm begins by sorting the graph's edges in non decreasing order of their weights. Then, starting with the empty sub graph, it scans this sorted list, adding the next edge on the list to the current sub graph if such an inclusion does not create a cycle and simply skipping the edge otherwise.
33	<b>Define fixed-length encoding.</b> Assigning each character a bit string of the same length $m$ called pixel length encoding ( $m \geq \log_2 n$ ).
34	<b>State Huffman's algorithm.</b> <ul style="list-style-type: none"> <li>✓ S Initialize <math>n</math> one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's weight. (More generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves.</li> <li>✓ Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight make them the left and right sub tree of a new tree and record the sum of their weights in the root of the new tree as its weight.</li> </ul>

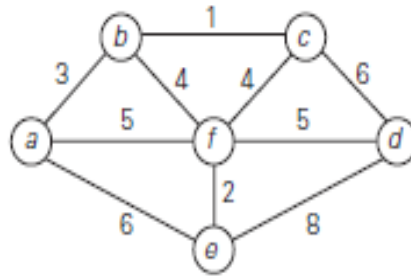
35	<b>Define Variable – length encoding.</b> Assigning code words of different lengths to different characters called Variable-length encoding.
36	<b>Define Huffman tree.</b> A tree constructed using Huffman algorithm is called a Huffman tree. A Huffman tree is a binary tree that minimizes the weighted path length from the root to the leaves of predefined weights
37	<b>Define Huffman code. (Dec 06)</b> A Huffman code is an optimal prefix-free variable-length encoding scheme that assigns bit strings to symbols based on their frequencies in a given text. This is accomplished by a greedy construction of a binary tree whose leaves represent the alphabet symbols and whose edges are labeled with 0's and 1's.
38	<b>Write down the optimization technique used for Warshall's algorithm. State the rules and assumptions which are implied behind that. (May 15)</b> Optimization technique used in Warshall's algorithm is Dynamic programming. Dynamic programming is a technique for solving problems with overlapping sub problems. Typically, these sub problems arise from a recurrence relating a solution to a given problem with solutions to its smaller sub problems of the same type. Dynamic programming suggests solving each smaller sub problem once and recording the results in a table from which a solution to the original problem can be then obtained.
39	<b>List out the memory functions used under Dynamic programming. (May 15)</b> Memory functions used under dynamic programming technique is called memoization. It is typical of an algorithm based on the classic bottom-up dynamic programming approach, however, to solve all smaller sub problems of a given problem. One variation of the dynamic programming approach seeks to avoid solving unnecessary sub problems.
40	<b>What is Multistage graph?</b> A Multistage graph is a directed graph in which the nodes can be divided into a set of stages such that all edges are from a stage to next stage only (In other words there is no edge between vertices of same stage and from a vertex of current stage to previous stage) i.e it is used to find a minimum cost path from source 's' to sink 't'
41	<b>Write down the formula for finding minimum cost path in Multistage graph</b> Let path(i,j) be some specification of the minimal path from vertex j in set i to vertex t; C(i,j) is the cost of this path; c(j,t) is the weight of the edge from j to t. $C(i,j) = \min_{l \in V_{i+1} \text{ \& } (j,l) \in E} \{ c(j,l) + C(i+1,l) \}$

## UNIT-III/ PART-B

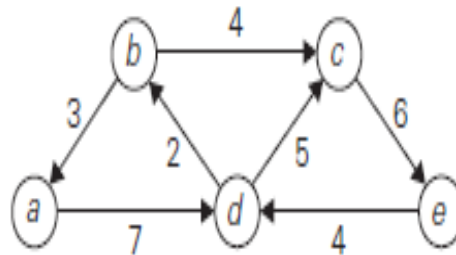
1	Explain how dynamic programming technique is applied to compute binomial coefficient.																		
2	Explain Warshall's algorithm to find the transitive closure with an example. Prove that the time efficiency of warshall's algorithm is cubic.(May 18)																		
3	Write Floyd's algorithm for the all-pairs shortest path problem and explain with an example.																		
4	Explain an algorithm to find optimal binary search tree with example or write an algorithm to construct the OBST tree for the given the roots $r(i, j)$ , $0 \leq i \leq j \leq n$ . Also prove that this could be performed in time $O(n)$ .(Dec 07,14,May 15)																		
5	Explain the pseudo code for Warshall's algorithm and apply for the following diagram. <div></div>																		
6	Write and explain Floyd's algorithm for the all-pairs shortest path problem. Using this find the length of the shortest path between all pairs of vertices of the following graph. (Dec 06,07) <div></div>																		
7	Develop the optimal binary search tree for the following instance and explain the algorithm in detail. <div><table><tr><td>Key</td><td>A</td><td>B</td><td>C</td><td>□</td></tr><tr><td>Probability</td><td>0.1</td><td>0.2</td><td>0.4</td><td>0.3</td></tr></table></div>	Key	A	B	C	□	Probability	0.1	0.2	0.4	0.3								
Key	A	B	C	□															
Probability	0.1	0.2	0.4	0.3															
8	Explain knapsack problem using dynamic programming with an example.																		
9	Apply the dynamic programming algorithm to solve the following instance of the knapsack problem and explain in detail. <div><table><tr><td>item</td><td>weight</td><td>value</td></tr><tr><td>1</td><td>3</td><td>\$25</td></tr><tr><td>2</td><td>2</td><td>\$20</td></tr><tr><td>3</td><td>1</td><td>\$15</td></tr><tr><td>4</td><td>4</td><td>\$40</td></tr><tr><td>5</td><td>5</td><td>\$50</td></tr></table><p>Knapsack capacity <math>W=6</math></p></div>	item	weight	value	1	3	\$25	2	2	\$20	3	1	\$15	4	4	\$40	5	5	\$50
item	weight	value																	
1	3	\$25																	
2	2	\$20																	
3	1	\$15																	
4	4	\$40																	
5	5	\$50																	

10	<p>Write an memory function algorithm to solve the following knapsack problem. (Dec 07)</p> <table><thead><tr><th>item</th><th>weight</th><th>value</th></tr></thead><tbody><tr><td>1</td><td>2</td><td>\$12</td></tr><tr><td>2</td><td>1</td><td>\$10</td></tr><tr><td>3</td><td>3</td><td>\$20</td></tr><tr><td>4</td><td>2</td><td>\$15</td></tr></tbody></table> <p>Knapsack capacity <math>W= 5</math></p>	item	weight	value	1	2	\$12	2	1	\$10	3	3	\$20	4	2	\$15
item	weight	value														
1	2	\$12														
2	1	\$10														
3	3	\$20														
4	2	\$15														
11	<p>Explain the method for finding the minimum spanning tree for a connected graph using Prim's algorithm with an example. (June 06,Dec 14,18)</p>															
12	<p>Construct a minimum spanning tree using Kruskal's algorithm with your own example.(Dec 06,07,May 15)</p>															
13	<p>How will find the shortest path between two given vertices using Dijkstra's algorithm? Explain the pseudo code with an example. (June 06,07,Dec 18)</p>															
14	<p>Explain the pseudo code for prim's algorithm and apply the same to minimum spanning tree for the following graph. (Dec 07,May 18)</p> 															
15	<p>Explain the pseudo code for prim's algorithm and apply the same to minimum spanning tree for the following graph.</p> 															
16	<p>Apply Kruskal's algorithm to find a minimum spanning tree of the following graph. (Dec 07)</p> 															

- 17 Apply Kruskal's algorithm to find a minimum spanning tree of the following graph.



- 18 Solve the following instance of the single source shortest path problem with vertex a as the source.



- 19 Explain the construction of Huffman coding tree with an example.

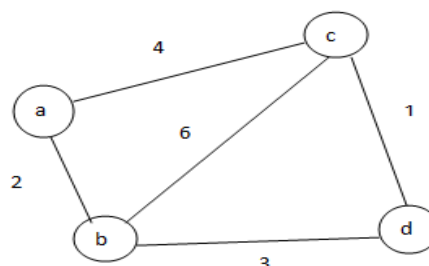
- 20 Consider the five character alphabet {A,B,C,D,\_} with the following occurrence probabilities and construct Huffman tree.

Character	A	B	C	D	_
Probability	0.35	0.1	0.2	0.15	0.15

- 21 Given the mobile numeric keypad. You can only press buttons that are up, left, right or down to the first numbered pressed to obtain the subsequent numbers. You are not allowed to press bottom row corner buttons (i.e. \* and #). Given a number N, how many key strokes will be involved to press the given number. What is the length of it? Which dynamic programming technique could be used to find solution for this? Explain each step with the help of a pseudo code and derive its time complexity. (May 15)

- 22 Let  $A = \{l/119, m/96, c/247, g/283, h/72, f/77, k/92, j/19\}$  be the letters and its frequency of distribution in a text file. Compute a suitable Huffman coding to compress the data effectively. (May 15)

- 23 Discuss about the algorithm and pseudocode to find minimum spanning tree using Prim's algorithm. Find the Minimum Spanning tree for the graph shown below.



And Discuss about the efficiency of the Algorithm. (May 16)

- 24 Solve the following instance of the 0/1 knapsack problem given the knapsack capacity in  $W=5$  using dynamic programming and explain it. (May 17)

Items	Weight	Value
1	4	10
2	3	20
3	2	15
4	5	25

- 25 Write the Huffman's Algorithm. Construct the Huffman's tree for the following data and obtain its Huffman's code. (May 17)

Character	A	B	C	D	E	—
Probability	0.5	0.35	0.5	0.1	0.4	0.2

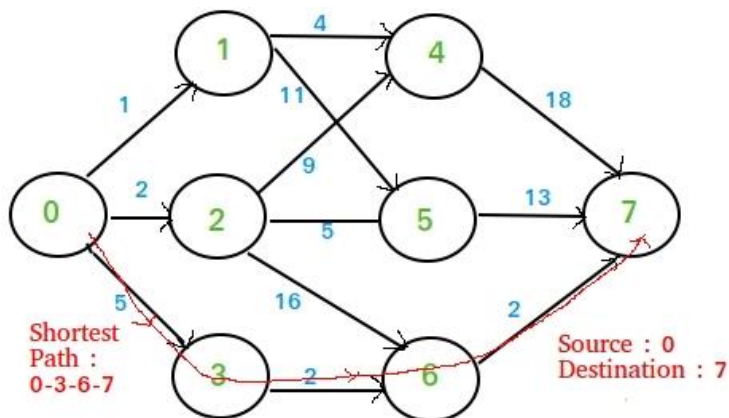
- 26 Explain Multistage graph with an example

- 27 Explain Container loading problem with an example

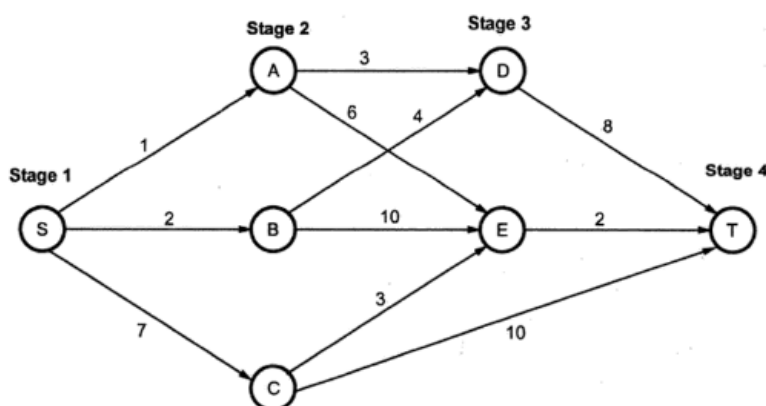
- 28 Explain Greedy 0/1 Knapsack problem

- 29 Explain Coin changing problem with an example

- 30 Find a minimum cost path from S to t for the following multistage graph



- 31 Find a minimum cost path from S to t for the following multistage graph. Use Forward and Backward approach also for solving this problem





32	<p>Find a minimum cost path from S to t for the following multistage graph. Use Forward and Backward approach also for solving this problem</p> <p style="text-align: center;"><b>MULTI STAGE GRAPH</b></p>
33	<p>Find a minimum cost path from S to t for the following multistage graph. Use Forward and Backward approach also for solving this problem</p>
<b>ITERATIVE IMPROVEMENT</b> The Simplex Method - The Maximum-Flow Problem - Maximum Matching in Bipartite Graphs, Stable marriage Problem.	
<b>UNIT-IV / PART-A</b>	
1	<p><b>What is the purpose of iterative improvement technique? (Dec 18)</b></p> <p>The iterative-improvement technique involves finding a solution to an optimization problem by generating a sequence of feasible solutions with improving values of the problem's objective function. Each subsequent solution in such a sequence typically involves a small, localized change in the previous feasible solution. When no such change improves the value of the objective function, the algorithm returns the last feasible solution as optimal and stops.</p>
2	<p><b>List out the problems that can be solved by iterative improvement algorithms.</b></p> <ul style="list-style-type: none"> <li>✓ Linear programming</li> <li>✓ Maximizing the flow in a network,</li> <li>✓ Matching the maximum possible number of vertices in a graph.</li> </ul>
3	<p><b>Define simplex method.</b></p> <p>The simple method is the classic method for solving the general linear programming problem. It works by generating a sequence of adjacent extreme points of the problem's feasible region with improving values of the objective function.</p>

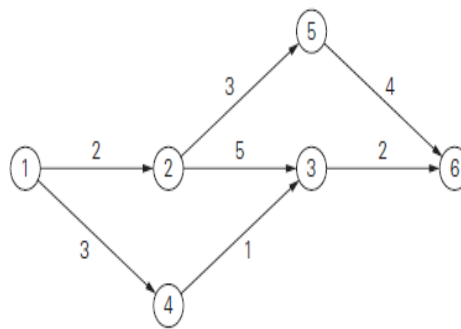
4	<b>Define feasible solution.</b> Any point $(x, y)$ that satisfies all the constraints of the problem
5	<b>Define feasible region.</b> Feasible region is the set of all its feasible points.
6	<b>Define extreme point theorem. (May 16, Dec 17)</b> Any linear programming problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an extreme point of the problem's feasible region.
7	<b>Define basic feasible solution.</b> If all the coordinates of a basic solution are nonnegative, the basic solution is called a basic feasible solution.
8	<b>Define source vertex.</b> It contains exactly one vertex with no entering edges; this vertex is called the source and assumed to be numbered 1.
9	<b>Define simplex tableau.</b> Simplex method progresses through a series of adjacent extreme points (basic feasible solutions) with increasing values of the objective function. Each such point can be represented by a simplex tableau, a table storing the information about the basic feasible solution corresponding to the extreme point.
10	<b>What is the need of maximum-flow problem?</b> The maximum-flow problem asks to find the maximum flow possible in a network, a weighted directed graph with a source and a sink.
11	<b>Define sink vertex.</b> It contains exactly one vertex with no leaving edges; this vertex is called the sink and assumed to be numbered $n$ .
12	<b>Define edge capacity.</b> The weight $u_{ij}$ of each directed edge $(i, j)$ is a positive integer, called the edge capacity. E.x; This number represents the upper bound on the amount of the material that can be sent from $i$ to $j$ through a link represented by this edge.
13	<b>Define value of the flow.</b> The quantity, the total outflow from the source or equivalently, the total inflow into the sink-is called the value of the flow. It is denoted it by $v$ . It is this quantity that we will want to maximize over all possible flows in a network
14	<b>Define flow.</b> A (feasible) flow is an assignment of real numbers $x_{ij}$ to edges $(i, j)$ of a given network that satisfy flow-conservation constraints and the capacity constraints $0 \leq x_{ij} \leq u_{ij}$ for every edge $(i, j) \in E$ .
15	<b>Define Ford-Fulkerson method.</b> The Ford-Fulkerson method is a classic template for solving the maximum flow problem by the iterative-improvement approach. The shortest augmenting-path method implements this idea by labeling network vertices in the breadth-first search manner.

16	<p><b>Write the optimization function for maximum-flow problem.</b></p> <p>The maximum-flow problem can be stated formally as the following optimization problem:</p> $\begin{aligned} &\text{maximize } v = \sum_{j: (1,j) \in E} x_{1j} \\ &\text{subject to } \sum_{j: (j,i) \in E} x_{ji} - \sum_{j: (i,j) \in E} x_{ij} = 0 \quad \text{for } i = 2, 3, \dots, n-1 \\ &\quad 0 \leq x_{ij} \leq u_{ij} \quad \text{for every edge } (i, j) \in E. \end{aligned}$
17	<p><b>Define forward edge.</b></p> <p>An undirected graph in which any two consecutive vertices <math>i, j</math> are either connected by a directed edge from <math>i</math> to <math>j</math> with some positive unused capacity <math>r_{ij} = u_{ij} - x_{ij}</math>. Edges of this kind are called forward edges because their tail is listed before their head in the vertex list <math>1 \rightarrow \dots i \rightarrow j \dots \rightarrow n</math> defining the path.</p>
18	<p><b>Define backward edge.</b></p> <p>A undirected graph in which any two consecutive vertices <math>i, j</math> are either connected by a directed edge from <math>j</math> to <math>i</math> with some positive flow <math>x_{ji}</math>. Edges of this kind are called backward edges because their tail is listed after their head in the path list <math>1 \rightarrow \dots i \leftarrow j \dots \rightarrow n</math>.</p>
19	<p><b>Define Max-Flow Min-Cut theorem.</b></p> <p>Max-Flow Min-Cut theorem is defined as the value of a maximum flow in a network is equal to the capacity of its minimum cut.</p>
20	<p><b>What you mean by matching in a graph.</b></p> <p>A matching in a graph is a subset of its edges with the property that no two edges share a vertex.</p>
21	<p><b>Define maximum-matching problem. (May 18)</b></p> <p>The maximum-matching problem is the problem of finding a maximum matching (matching with the largest number of edges) in a given graph.</p>
22	<p><b>What is mean by bipartite graph? (or) Define 2-colorable in bipartite graph. (Dec 17)</b></p> <p>In a bipartite graph, all the vertices can be partitioned into two disjoint sets <math>V</math> and <math>U</math>, not necessarily of the same size, so that every edge connects a vertex in one of these sets to a vertex in the other set. In other words, a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also said to be 2-colorable.</p>
23	<p><b>What do you mean by maximum weight matching?</b></p> <p>The problem of maximizing the sum of the weights on edges connecting matched pairs of vertices. This problem is called maximum-weight matching.</p>
24	<p><b>When marriage matching problem is said to stable?</b></p> <p>A marriage matching <math>M</math> is called stable if there is no blocking pair for it.</p>

25	<p><b>What is mean by stable marriage problem?</b></p> <p>Consider a set <math>Y = \{m_1, m_2, \dots, m_n\}</math> of <math>n</math> men and a set <math>X = \{w_1, w_2, \dots, w_n\}</math> of <math>n</math> women. Each man has a preference list ordering the women as potential marriage partners with no ties allowed. Similarly, each woman has a preference list of the men, also with no ties.</p>
26	<p><b>Define marriage matching.</b></p> <p>A marriage matching <math>M</math> is a set of <math>n</math> <math>(m, w)</math> pairs whose members are selected from disjoint <math>n</math>-element sets <math>Y</math> and <math>X</math> in a one-one fashion, i.e., each man <math>m</math> from <math>Y</math> is paired with exactly one woman <math>w</math> from <math>X</math> and vice versa.</p>
27	<p><b>Define blocking pair.</b></p> <p>A pair <math>(m, w)</math>, where <math>m \in Y</math>, <math>w \in X</math>, is said to be a blocking pair for a marriage matching <math>M</math> if man <math>m</math> and woman <math>w</math> are not matched in <math>M</math> but they prefer each other to their mates in <math>M</math>.</p>
28	<p><b>When marriage matching problem is said to unstable?</b></p> <p>A marriage matching <math>M</math> is called stable if there is a blocking pair for it.</p>
29	<p><b>Write the algorithm for stable marriage problem.</b></p> <p><b>Input:</b> A set of <math>n</math> men and a set of <math>n</math> women along with rankings of the women by each man and rankings of the men by each woman with no ties allowed in the rankings</p> <p><b>Output:</b> A stable marriage matching</p> <ul style="list-style-type: none"> <li>✓ Start with all the men and women being free.</li> <li>✓ While there are free men, arbitrarily select one of them and do the following:</li> </ul> <p><b>Proposal:</b> The selected free man <math>m</math> proposes to <math>w</math>, the next woman on his preference list (who is the highest-ranked woman who has not rejected him before).</p> <p><b>Response:</b> If <math>w</math> is free, she accepts the proposal to be matched with <math>m</math>. If she is not free, she compares <math>m</math> with her current mate. If she prefers <math>m</math> to him, she accepts <math>m</math>'s proposal, making her former mate free; otherwise, she simply rejects <math>m</math>'s proposal, leaving <math>m</math> free.</p> <ul style="list-style-type: none"> <li>✓ Return the set of <math>n</math> matched pairs.</li> </ul>
30	<p><b>Prove that the stable marriage algorithm terminates after no more than <math>n^2</math> iterations with a stable marriage output.</b></p> <p>The algorithm starts with <math>n</math> men having the total of <math>n^2</math> women on their ranking lists. On each iteration, one man makes a proposal to a woman. This reduces the total number of women to whom the men can still propose in the future because no man proposes to the same woman more than once. Hence, the algorithm must stop after no more than <math>n^2</math> iterations.</p>

31	<p><b>Prove that the stable marriage algorithm always yields a gender-optimal stable matching.</b></p> <p>To prove that a man (woman)-optimal matching is unique for a given set of participant preferences. Therefore the algorithm's output does not depend on the order in which the free men (women) make their proposals.</p>
32	<p><b>What do you mean by 'Perfect Matching' in bipartite graphs? (May 15,17)</b></p> <p>A perfect matching is a matching in which each node has exactly one edge incident on it. A Bipartite Graph <math>G=(V,E)</math> is a graph in which the vertex set <math>V</math> can be divided into two disjoint subsets <math>X</math> and <math>Y</math> such that every edge <math>e \in E</math> has one end point in <math>X</math> and the other end point in <math>Y</math>. A matching <math>M</math> is a subset of edges such that each node in <math>V</math> appears in at most one edge in <math>M</math>.</p>
33	<p><b>Define flow 'cut'. (May 15)</b></p> <p>If all the edges of a cut were deleted from the network, there would be no directed path from source to sink.</p>
34	<p><b>State: Planar coloring graph problem. (May 17)</b></p> <p>A graph is planar if it can be drawn in a plane without edge-crossings. Any planar graph can have its nodes colored such that no two adjacent nodes have the same color.</p>
35	<p><b>What is solution space? Give an example. (Dec 2018)</b></p> <p>search space, or solution space is the set of all possible points (sets of values of the choice variables) of an optimization problem that satisfy the problem's constraints, potentially including inequalities, equalities, and integer constraints.</p>
<b>UNIT-IV / PART-B</b>	
1	Explain geomtric interpretation of linear programming with an example.
2	Explain the steps to solve simplex method using linear programming with an example.(or) Describe in detail the simplex algorithm methods.(May 16,17,18,Dec 17,18)
3	<p>Solve the following problem using simplex method:</p> $\begin{aligned} &\text{maximize } Z = 3x + 5y \\ &\text{subject to } x + y \leq 4 \\ &\quad \quad \quad x + 3y \leq 6; x \geq 0, y \geq 0. \end{aligned}$
4	<p>Use simplex method to solve the LPP</p> $\begin{aligned} &\text{maximize } Z = 4x + 10y \\ &\text{subject to } 2x + y \leq 50 \\ &\quad \quad \quad 2x + 5y \leq 100 \\ &\quad \quad \quad 2x + 3y \leq 90; x \geq 0, y \geq 0. \end{aligned}$
5	Explain the maximum flow problem with an example.

- 6 Write the pseudo code for Shortest Augmenting Path method and explain in detail. (May 15,16)



- 7 Explain maximum matching problem in bipartite graphs
- 8 Write the pseudo code for maximum bipartite matching.
- 9 Explain stable marriage algorithm with suitable example. (May 15,18,Dec17,18)

- 10 Explain the following  
a) Blocking pair b) Stable marriage problem c) Man-optimal d) Woman-optimal

- 11 Solve the instance of the stable marriage problem given by the ranking matrix and find the stable and unstable matching.

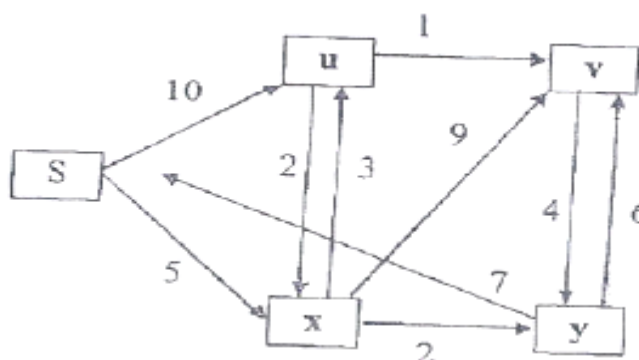
Ranking matrix

	Ann	Lea	Sue
Bob	2,3	1,2	3,3
Jim	3,1	1,3	2,1
Tom	3,2	2,1	1,2

- 12 Maximize  $p = 2x + 3y + z$  subject to  
 $x + y + z \leq 40$   
 $2x + y - z \geq 10$   
 $-y + z \geq 10$   
 $x \geq 0, y \geq 0, z \geq 0$  (May 15)

- 13 Write down the optimality condition and algorithmic implementation for finding M-augmenting paths in bipartite graphs. (May 15)

- 14 How do you compute maximum flow for the following graph using Ford-Fulkerson method? (May 15)



- 15 State and prove Max-Flow Min-Cut Theorem. (May 16)

**COPING WITH THE LIMITATIONS OF ALGORITHM POWER**

Lower - Bound Arguments - P, NP NP- Complete and NP Hard Problems. Backtracking - n-Queen problem - Hamiltonian Circuit Problem - Subset Sum Problem. Branch and Bound - LIFO Search and FIFO search - Assignment problem - Knapsack Problem - Travelling Salesman Problem - Approximation Algorithms for NP-Hard Problems - Travelling Salesman problem - Knapsack problem.

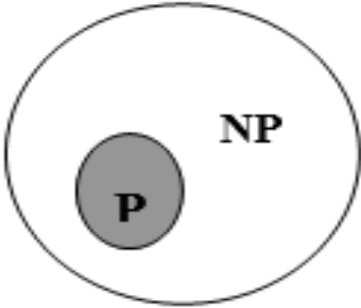
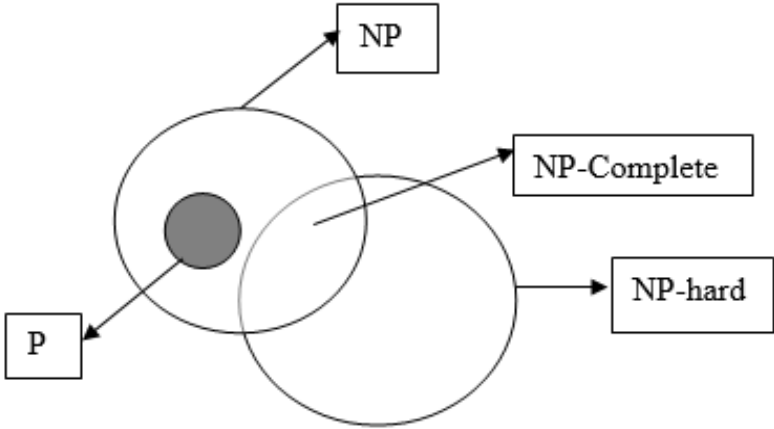
**UNIT-V / PART-A**

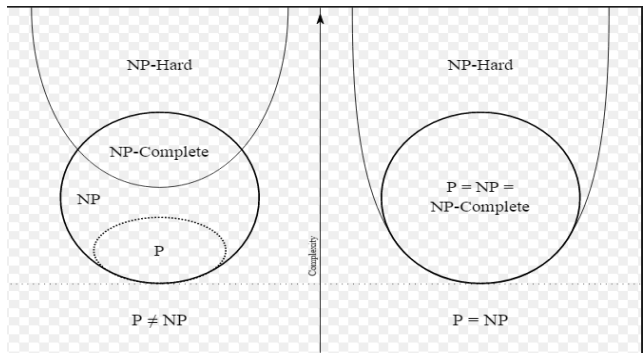
1	<p><b>Define trivial lower bound of a class. (May 18)</b></p> <p>The simplest method of obtaining a lower-bound class is based on counting the number of items in the problem's input that must be processed and the number of output items that need to be produced. Since any algorithm must at least "read" all the items it needs to process and "write" all its outputs, such a count yields a trivial lower bound.</p>
2	<p><b>Define tournament tree.</b></p> <p>A tournament tree is a complete binary tree reflecting results of a "knockout tournament": its leaves represent n players entering the tournament, and each internal node represents a winner of a match played by the players represented by the node's children. Hence, the winner of the tournament is represented by the root of the tree.</p>
3	<p><b>Define tractable problems.</b></p> <p>Problems that can be solved in polynomial time are called tractable.</p>
4	<p><b>Define intractable problems.</b></p> <p>Problems that cannot be solved in polynomial time are called intractable.</p>
5	<p><b>Define decision tree.</b></p> <p>A decision tree is a flowchart-like structure in which internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.</p> <p>In decision analysis a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.</p> <p>A decision tree consists of 3 types of nodes:</p> <ul style="list-style-type: none"> <li>✓ Decision nodes - commonly represented by squares</li> <li>✓ Chance nodes - represented by circles</li> <li>✓ End nodes - represented by triangles</li> </ul>

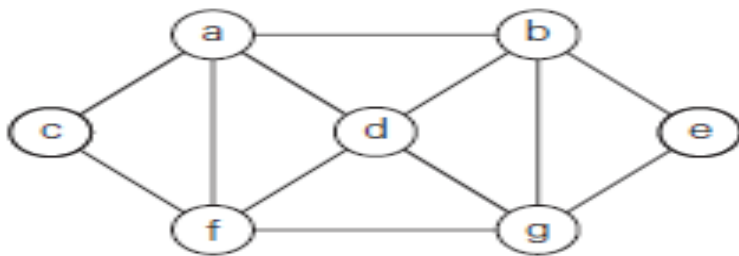
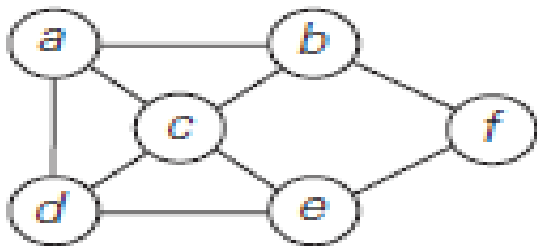
6	<b>On what basis problems are classified?</b> Problems are classified into two types based on time complexity. They are <ul style="list-style-type: none"> <li>✓ Polynomial (P) Problem.</li> <li>✓ Non-Polynomial (NP) Problem</li> </ul>
7	<b>Define Polynomial (P) problem. (May 17)</b> Class P is a class of decision problems that can be solved in polynomial time by (deterministic) algorithms. This class of problems is called polynomial.
8	<b>Define Non Polynomial (NP) problem. (Dec 06, May 17)</b> Class NP is the class of decision problems that can be solved by nondeterministic polynomial algorithms. This class of problems is called nondeterministic polynomial
9	<b>Give the classification of Non Polynomial problem.</b> Non-Polynomial (NP) problems are classified into two types. They are: <ul style="list-style-type: none"> <li>✓ NP-Hard</li> <li>✓ NP-Complete</li> </ul>
10	<b>Give some examples of Polynomial problem.</b> <ul style="list-style-type: none"> <li>✓ Selection sort</li> <li>✓ Bubble Sort</li> <li>✓ String Editing</li> <li>✓ Factorial, etc.</li> </ul>
11	<b>Give some examples of Non-Polynomial problem. (Dec 06)</b> <ul style="list-style-type: none"> <li>✓ Travelling Salesman Problem</li> <li>✓ Knapsack Problem.</li> </ul>
12	<b>Define Deterministic algorithm</b> It has a property that the result of every operation is uniquely defined, and then this type of algorithm is referred to as Deterministic algorithm. Such an algorithm agrees with the way programs are executed on a computer. The machine executing such algorithm is referred as Deterministic machine.
13	<b>Define Non-Deterministic algorithm.</b> It has a property that the result of every operation is not uniquely defined, but subject to set of possibilities, then this type of algorithms is referred to as Non-deterministic algorithm. The possible functions are <ul style="list-style-type: none"> <li>✓ Choice (S)-It arbitrarily chooses one of the value of set S</li> <li>✓ Failure ()-signals an Unsuccessful completion</li> <li>✓ Success ()-signals an Successful completion</li> </ul>

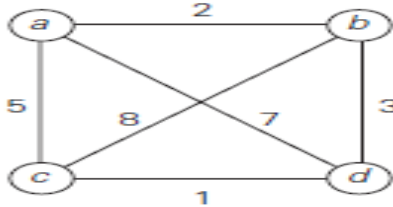
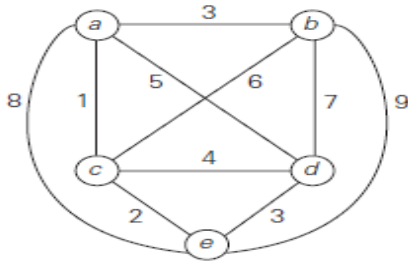


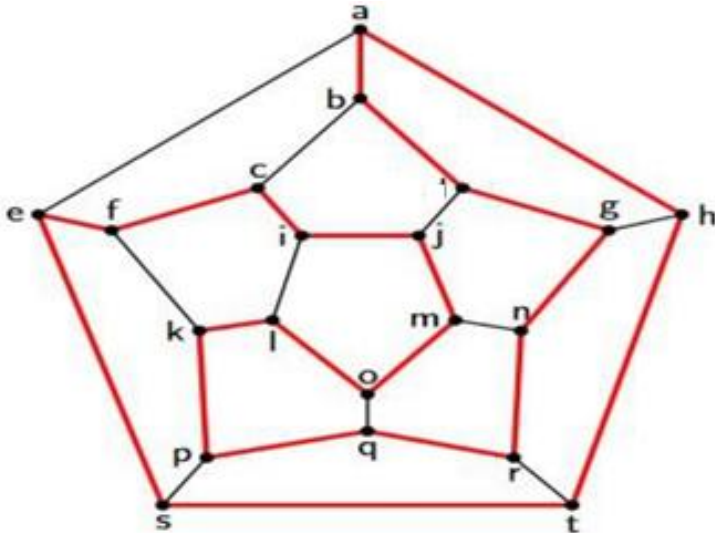
14	<b>Give an example for Class P problem.</b> ✓ Graph Coloring.
15	<b>Give an example for Class NP problem.</b> ✓ Traveling Salesman Problem. ✓ Knapsack problem
16	<b>Define decision problem.</b> Any problem for which the answer is either zero or one is called a decision problem. An algorithm for a decision problem is termed a Decision algorithm.
17	<b>Define optimization Problem.</b> Any problem that involves the identification of an optimal (maximum or minimum) value of a given cost function is known as an optimization problem. An Optimization algorithm is used to solve an optimization problem.
18	<b>When a decision problem is said to be polynomially reducible?</b> A decision problem D1 is said to be polynomially reducible to a decision problem D2, if there exists a function $t$ that transforms instances of D1 to instances of D2 such that: ✓ $t$ maps all yes instances of D1 to yes instances of D2 and all no instances of D1 to no instances of D2 ✓ It is computable by a polynomial time algorithm
19	<b>When decision problem D is said to be NP-complete?</b> A decision problem D is said to be NP-complete if: ✓ it belongs to class NP ✓ every problem in NP is polynomially reducible to D
20	<b>Define n-queens problem.</b> The problem is to place $n$ queens on an $n \times n$ chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.
21	<b>Define backtracking. (June 06)</b> The principal idea is to construct solutions one component at a time and evaluate such partially constructed candidates as follows. If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component. If there is no legitimate option for the next component, no alternatives for any remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

22	<p><b>Define state space tree. (Dec 06, May 16)</b></p> <p>It is convenient to implement this kind of processing by constructing a tree of choices being made, called the state-space tree. Its root represents an initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of a solution; the nodes of the second level represent the choices for the second component, and so on.</p>
23	<p><b>When a node in a state space tree is said to promising and non-promising? (Dec 17)</b></p> <p>A node in a state-space tree is said to be promising if it corresponds to a partially constructed solution that may still lead to a complete solution; otherwise, it is called non promising. Leaves represent either non promising dead ends or complete solutions found by the algorithm.</p>
24	<p><b>Define Hamiltonian circuit problem. (June 06, Dec 14, May 15)</b></p> <p>Let <math>G = (V, E)</math> be a connected graph, with <math>n</math> vertices. A Hamiltonian cycle is a round trip path along <math>n</math> edges of <math>G</math> that visits every vertex once and returns to its starting position.</p>
25	<p><b>Mention the relation between P and NP.</b></p> 
26	<p><b>Mention the relation between P, NP, NP-Hard and NP Complete Problem.</b></p> 
27	<p><b>What is meant by sum of subset problem?</b></p> <p>The sum of subset problem is to find a subset of a given set <math>A = \{a_1, \dots, a_n\}</math> of <math>n</math> positive integers whose sum is equal to a given positive integer <math>d</math>.</p>
28	<p><b>Define assignment problem.</b></p> <p>Assignment problem is the problem of assigning <math>n</math> people to <math>n</math> jobs so that the total cost of the assignment is as small as possible.</p>

29	<b>Define branch and bound method.</b> <ul style="list-style-type: none"> <li>✓ Branch and bound is an algorithm that enhances the idea of generating a state space tree with idea of estimating the best value obtainable from a current node of the decision tree</li> <li>✓ If such an estimate is not superior to the best solution seen up to that point in the processing, the node is eliminated from further consideration.</li> </ul>
30	<b>Define NP hard problems.</b> If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.
31	<b>Define NP complete problems.(Dec 06)</b> A problem that is NP-complete has the property that it can be solved in polynomial time iff if all other NP-complete problems can also be solved in polynomial time.
32	<b>Define cost of tour. (Dec 14)</b> Cost of a tour is defined as the sum of the cost of the edges on the tour.
33	<b>Define Live and Dead nodes. (Dec 14)</b> <ul style="list-style-type: none"> <li>✓ Live node → a node which has been generated and all of whose children have not yet been generated.</li> <li>✓ E-node → the live node whose children are currently being generated.</li> <li>✓ Dead node → a generated node which is not to be expanded further or all of whose children have been generated.</li> </ul>
34	<b>How NP-hard problems are different from NP-Complete? (May 15)</b> <b>NP-hard :</b> If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time. <b>NP-Complete:</b> A problem that is NP-complete has the property that it can be solved in polynomial time iff if all other NP-complete problems can also be solved in polynomial time. 
35	<b>Give the purpose of lower bound. (May 16)</b> Given a class of algorithms for solving a particular problem, a lower bound indicates the best possible efficiency any algorithm from this class can have.
36	<b>What is Euclidean minimum spanning tree problem? (May 16)</b> The Euclidean minimum spanning tree or EMST is a minimum spanning tree of a set of $n$ points in the plane, where the weight of the edge between each pair of points is the Euclidean distance between those two points.

37	<p><b>What is an articulation point in a graph? (May 17)</b></p> <p>A vertex of a connected graph is said to be its articulation point if its removal with all edges incident to it breaks the graph into disjoint pieces.</p>																														
<b>UNIT-V / PART-B</b>																															
1	Explain decision tree with an example.																														
2	What is backtracking? Write the template of general backtracking algorithm and explain in detail with suitable example. (June 07)																														
3	Explain n-queen's problem. Draw a portion of the state space tree and perform backtracking search for a solution to 4-queens problem. (June 06,Dec 07,14)																														
4	Explain how backtracking method is applied to solve subset sum problem with suitable example.(June 0,Dec 07)																														
5	Write a pseudo code for backtracking algorithm and apply backtracking to solve the following instances of the subset sum problem: S= {1, 3, 4,5} d=11 and d=8. (Dec 06)																														
6	Explain in detail about Hamiltonian circuit problem with suitable example.																														
7	<p>Explain Hamiltonian circuit in a graph. Use backtracking to get a Hamiltonian circuit of following the graph.</p> 																														
8	<p>Explain Hamiltonian circuit in a graph. Use backtracking to get a Hamiltonian circuit of following the graph.(Dec 07)</p> 																														
9	Explain assignment problem using branch and bound method or Explain how job assignment problem could be solved, given $n$ tasks and $n$ agents where each agent has a cost to complete each task, using Branch and bound technique. (May 15)																														
10	<p>Solve the following assignment problem using branch and bound technique. Explain in detail how branch and bound technique is useful for solving assignment problems.</p> <table><tr><td></td><td>job 1</td><td>job 2</td><td>job 3</td><td>job 4</td><td></td></tr><tr><td><math>C =</math></td><td>9</td><td>2</td><td>7</td><td>8</td><td>person a</td></tr><tr><td></td><td>6</td><td>4</td><td>3</td><td>7</td><td>person b</td></tr><tr><td></td><td>5</td><td>8</td><td>1</td><td>8</td><td>person c</td></tr><tr><td></td><td>7</td><td>6</td><td>9</td><td>4</td><td>person d</td></tr></table>		job 1	job 2	job 3	job 4		$C =$	9	2	7	8	person a		6	4	3	7	person b		5	8	1	8	person c		7	6	9	4	person d
	job 1	job 2	job 3	job 4																											
$C =$	9	2	7	8	person a																										
	6	4	3	7	person b																										
	5	8	1	8	person c																										
	7	6	9	4	person d																										

11	Apply Branch and Bound algorithm to solve the Travelling salesman problem for <i>(May 17)</i> <div></div>															
12	Solve the following instance of the travelling salesman problem by branch and bound method and explain in detail. <i>(Dec 07)</i> <div></div>															
13	Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail. <table><tr><th>item</th><th>weight</th><th>value</th></tr><tr><td>1</td><td>10</td><td>\$100</td></tr><tr><td>2</td><td>7</td><td>\$63</td></tr><tr><td>3</td><td>8</td><td>\$56</td></tr><tr><td>4</td><td>4</td><td>\$12</td></tr></table> <p>The knapsack capacity W is 16.</p>	item	weight	value	1	10	\$100	2	7	\$63	3	8	\$56	4	4	\$12
item	weight	value														
1	10	\$100														
2	7	\$63														
3	8	\$56														
4	4	\$12														
14	Explain knapsack problem using branch and bound technique. <i>(Dec 07)</i>															
15	Explain travelling salesman problem using branch and bound technique.															
16	Explain the concept of P, NP, NP hard and NP complete.															
17	Explain in detail about approximation algorithms for NP hard problems. How all you quantify the accuracy of an approximation algorithm? <i>(May 17)</i>															
18	Explain approximation algorithm for travelling salesman problem with an example. (or) Suggest an approximation algorithm for traveling salesperson problem. Assume that the cost function satisfies the triangle inequality. Elaborate on the nearest neighbor algorithm and multi fragment heuristic algorithm for TSP problem. <i>(May 15,18)</i>															
19	Explain approximation algorithm for knapsack problem.															
20	Explain in detail about branch and bound technique with an example. <i>(June 07)</i>															
21	Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail. <i>(Dec 06)</i> <table><tr><th>Item</th><th>Weight</th><th>value</th></tr><tr><td>1</td><td>4</td><td>\$40</td></tr><tr><td>2</td><td>7</td><td>\$42</td></tr><tr><td>3</td><td>5</td><td>\$25</td></tr><tr><td>4</td><td>3</td><td>\$12</td></tr></table> <p>The knapsack capacity W is 10.</p>	Item	Weight	value	1	4	\$40	2	7	\$42	3	5	\$25	4	3	\$12
Item	Weight	value														
1	4	\$40														
2	7	\$42														
3	5	\$25														
4	3	\$12														

22	<p>Apply the branch and bound algorithm to solve the following knapsack problem and explain in detail. <i>(Dec 06)</i></p> <table><tr><th>Item</th><th>Weight</th><th>value</th></tr><tr><td>1</td><td>2</td><td>1</td></tr><tr><td>2</td><td>3</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr></table> <p>The knapsack capacity W is 6.</p>	Item	Weight	value	1	2	1	2	3	2	3	4	5
Item	Weight	value											
1	2	1											
2	3	2											
3	4	5											
23	<p>The Knight is placed on the first block of an empty board and, moving accordingly to the rules of chess, must visit each square exactly once. Solve the above problem using backtracking procedure. <i>(May 15) Out of Syllabus</i></p>												
24	<p>Implement an algorithm for Knapsack problem using NP-Hard approach.<i>(May 15)</i></p>												
25	<p>Give any five undecidable problems and explain the famous halting problem. <i>(May 16)</i></p>												
26	<p>State the subset-sum problem and Complete state-space tree of the backtracking algorithm applied to the instance <math>A= \{3, 5, 6, 7\}</math> and <math>d= 15</math> of the subset-sum problem. <i>(May 16)</i></p>												
27	<p>Describe the backtracking solution to solve 8-Queens problem. <i>(May 17)</i></p>												
28	<p>Find a Hamiltonian circuit or disprove its existence in the graph given below. <i>(Dec 17)</i></p> 												
29	<p>Give the methods for establishing lower bound. <i>(Dec 17)</i></p>												
30	<p>Elaborate on the nearest neighbor algorithm and multifragment heuristic algorithm for TSP problem. <i>(May 17)</i></p>												