



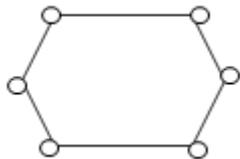
MATURE

PDF

UNIT 4 – Backtracking and Branch & Bound MCQs

1. What will be the chromatic number of the following graph?

- a) 1 b) 2 c) 3 d) 4



2. What is the condition for proper coloring of a graph?

a) two vertices having a common edge should not have same color

b) two vertices having a common edge should always have same color

c) all vertices should have a different color

d) all vertices should have same color

3. What is a chromatic number?

a) The maximum number of colors required for proper edge coloring of graph

b) The maximum number of colors required for proper vertex coloring of graph

c) The minimum number of colors required for proper vertex coloring of graph

d) The minimum number of colors required for proper edge coloring of graph

4. The Data structure used in standard implementation of Breadth First Search is?

a) Stack

b) Queue

c) Linked List

d) Tree

5. The Data structure used in standard implementation of Depth First Search is?

a) Stack

b) Queue

c) Linked List

d) Tree

6. Backtracking algorithm is implemented by constructing a tree of choices called as?

a) State-space tree b) State-chart tree c) Node tree d) Backtracking tree

7. A node is said to be _____ if it has a possibility of reaching a complete solution.

a) Non-promising b) **Promising** c) Succeeding d) Preceding

8. In what manner is a state-space tree for a backtracking algorithm constructed?

a) Depth-first search b) Breadth-first search c) Twice around the tree d) Nearest neighbour first

9. _____ enumerates a list of promising nodes that could be computed to give the possible solutions of a given problem.

a) Exhaustive search b) Brute force c) **Backtracking** d) Divide and conquer

10. A _____ is a round trip path along n edges of G that visits every vertex once and return to its starting position

a) MST b) TSP c) Multistage Graph d) **Hamiltonian Cycle**

11. In general, backtracking can be used to solve?

a) Numerical problems b) Exhaustive search c) **Combinatorial problems** d) Graph coloring problems

12. Which of the following is not a branch and bound strategy to generate branches?

a) LIFO branch and bound b) FIFO branch and bound
c) Lowest cost branch and bound d) **Highest cost branch and bound**

13. Which of the following can traverse the state space tree only in DFS manner?

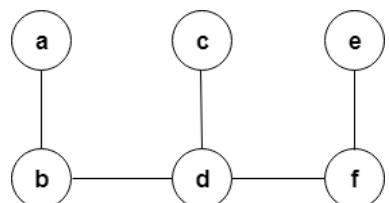
a) branch and bound b) dynamic programming c) greedy algorithm d) **backtracking**

14. which of the following problems is similar to that of a Hamiltonian path problem?

a) knapsack problem b) closest pair problem
c) **travelling salesman problem** d) assignment problem

15. How many Hamiltonian paths does the following graph have?

a) 1 b) 2 c) 3 d) 4



AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING
DEPARTMENT OF INFORMATION TECHNOLOGY
CS8451- DESIGN AND ANALYSIS OF ALGORITHMS
UNIT I INTRODUCTION
PART-A

- 1. State the transpose symmetry property of O and Ω . [Nov/Dec 2019]**
 $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

- 2. Define recursion. [Nov/Dec 2019]**

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily

- 3. How do you measure the efficiency of an Algorithm? [Apr/May 2019]**

- Size of the Input
- Running Time

- 4. Prove that $f(n)=O(g(n))$ and $g(n)=O(f(g(n)))$ then $f(n)=\Theta(g(n))$ [Apr/May 2019]**

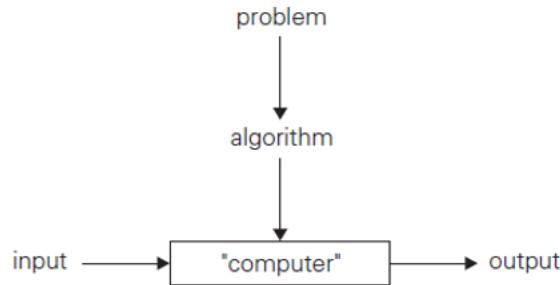
Prove by contradiction. Assume $f(n) = O(g(n))$, by the definition, there exist constants $c, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ or $0 \leq n \leq cn^1 + \sin n$ for all $n \geq n_0$. It implies $(0.6) 0 \leq 1 \leq c \sin n$ for all $n \geq n_0$. Can it be true? To show that the answer is No, it suffices to show: For any $n_0 > 0$, we can always pick an $n \geq n_0$ such that $c \sin n < 1$.

- 5. What is basic operation ?[Apr/May 2018]**

The operation that contributes most towards the running time of the algorithm. The running time of an algorithm is the function defined by the number of steps (or amount of memory) required to solve input instances of size n .

- 6. What is an Algorithm? [Apr/May 2017]**

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



- 7. How to measure algorithm's running time? [Nov/Dec 2017]**

Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermostloop.

8. Compare the order of growth $n(n-1)/2$ and n^2 . [May/June2016]

n	$n(n-1)/2$	n^2
Polynomial	Quadratic	Quadratic
1	0	1
2	1	4
4	6	16
8	28	64
10	45	10^2
10^2	4950	10^4
Complexity	Low	High
Growth	Low	high

$n(n-1)/2$ is lesser than the half of n^2

9. Define recurrence relation. [Nov/Dec2016]

A recurrence relation is an equation that defines a sequence based on a rule that gives the next term as a function of the previous term(s). The simplest form of a recurrence relation is the case where the next term depends only on the immediately previous term.

10. Write down the properties of asymptotic notations. [April/May2015]

Asymptotic notation is a notation, which is used to take meaningful statement about the efficiency of a program. To compare and rank such orders of growth, computer scientists use three notations, they are:

- O - Big ohnotation
- Ω - Big omeganotation
- Θ - Big theta notation

11. Give the Euclid's Algorithm for Computing gcd(m,n) [Apr/May '16, '18]

Algorithm Euclid_gcd(m, n)

//Computes gcd(m, n) by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

while $n \neq 0$ **do**

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m

Example: $\text{gcd}(60, 24) = \text{gcd}(24, 12) = \text{gcd}(12, 0) = 12$.

12. Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer. [Apr May 2015]

Algorithm Binary(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n 's binary representation

```

count ← 1
while  $n > 1$  do
    count ← count + 1
     $n \leftarrow \lfloor n/2 \rfloor$ 
return count

```

13. Define Little “oh”. [April/May2014]

The function $f(n) =$

$O(g(n))$ iff $\lim f(n)$

$=0$

$n \rightarrow \infty g(n)$

14. Define Little Omega. [April/May2014]

The function $f(n) =$

$\omega(g(n))$ iff $\lim f(n)$

$=0$

$n \rightarrow \infty g(n)$

15. Define Big Theta Notations [Nov/Dec2014]

A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constants c_1 and c_2 and some nonnegative integer n_0 such that c_1

$g(n) \leq t(n) \leq c_2 g(n)$ for all $n \geq n_0$

16. Define Big Omega Notations. [May/June2013]

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$t(n) \geq c g(n)$ for all for all $n \geq n_0$

17. What is Big ‘Oh’ notation? [May/June2012]

A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constant c and some nonnegative integers n_0 such that

$t(n) \leq c g(n)$ for all $n \geq n_0$

18. Give the two major phases of performance evaluation.

Performance evaluation can be loosely divided into two major phases:

- a prior estimates (performance analysis)
- a Posterior testing (performance measurement)

19. What are six steps processes in algorithmic problem solving? [Nov/Dec2009]

- Understanding theproblem
- Decision making on - Capabilities of computational devices, Choice of exact or approximate problem solving, Datastructures
- Algorithmicstrategies
- Specification ofalgorithm
- Algorithmicverification
- Analysis of algorithms

20. What are the basic asymptotic efficiencyclasses?

The various basic efficiency classes are

- Constant:1
- Logarithmic: $\log n$
- Linear: n
- $N \cdot \log n$: $n \log n$
- Quadratic: n^2
- Cubic: n^3
- Exponential: 2^n
- Factorial: $n!$

21. List the factors which affects the running time of thealgorithm.

- A. Computer
- B. Compiler
- C. Input to thealgorithm
 - i. The content of the input affects the runningtime
 - ii. Typically, the input size is the mainconsideration.

22. Give an non-recursive algorithm to find out the largest element in a list of nnumbers.

ALGORITHMMaxElement(A[0..n-1])

//Determines the value of the largest element in a given array Input:An array A[0..n-1] of real numbers

//Output: The value of the largest element in A
maxvalif \hat{Y} a[0] for I if \hat{Y} 1 to n-1 do

```
if A[I] >maxval
return maxvalif $\hat{Y}$ 
A[I] return maxval
```

23. Write a recursive algorithm for computing the nth fibonacci number.

ALGORITHMF(n)

```
// Computes the nth Fibonacci number recursively by using the definition
// Input A non-negative integer n
```

```
// Output The nth Fibonacci number
if n = 1 return n
else return F(n-1)+F(n-2)
```

24. What is algorithm visualization?

Algorithm visualization can be defined as the use of images to convey some useful information about algorithms. Two principal variations are Static algorithm visualization Dynamic Algorithm visualization(also called algorithm animation)

25. What is the order of growth?

The Order of growth is the scheme for analyzing an algorithm's efficiency for different input sizes which ignores the multiplicative constant used in calculating the algorithm's running time. Measuring the performance of an algorithm in relation with the input size n is called the order of growth.

PART-B & C

1. Explain about algorithm with suitable example (Notion of algorithm).

An algorithm is a sequence of unambiguous instructions for solving a computational problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

Algorithms – Computing the Greatest Common Divisor of Two Integers(gcd(m, n): the largest integer that divides both m and n .)

Euclid's algorithm: gcd(m, n) = gcd($n, m \bmod n$)

Step1: If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.

Step2: Divide m by n and assign the value of the remainder to r .

Step 3: Assign the value of n to m and the value of r to n . Go to Step 1.

Algorithm Euclid(m, n)

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

About This algorithm

Finiteness: how do we know that Euclid's algorithm actually comes to a stop

Definiteness: nonambiguity

Effectiveness: effectively computable.

Consecutive Integer Algorithm

Step1: Assign the value of $\min\{m, n\}$ to t .

Step2: Divide m by t . If the remainder of this division is 0, go to Step3;otherwise, go to Step 4.

Step3: Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop; otherwise, proceed to Step4.

Step4: Decrease the value of t by 1. Go to Step2.

About This algorithm

- Finiteness
- Definiteness
- Effectiveness

Middle-school procedure

Step1: Find the prime factors of m.

Step2: Find the prime factors of n.

Step3: Identify all the common factors in the two prime expansions found in Step1 and Step2. (If p is a common factor occurring P_m and P_n times in m and n, respectively, it should be repeated in $\min\{P_m, P_n\}$ times.)

Step4: Compute the product of all the common factors and return it as the gcd of the numbers given.

2. Write short note on Fundamentals of Algorithmic Problem Solving

Understanding the problem

- Asking questions, do a few examples by hand, think about special cases, etc.

Deciding on

- Exact vs. approximate problem solving
- Appropriate data structure

Design an algorithm

Proving correctness

Analyzing an algorithm

Time efficiency : how fast the algorithm runs

Space efficiency: how much extra memory the algorithm needs.

Coding an algorithm

3. Discuss important problem types that you face during Algorithm Analysis.

Sorting

Rearrange the items of a given list in ascending order.

Input: A sequence of n numbers $\langle a_1, a_2, \dots, a_n \rangle$

Output: A reordering $\langle a'_1, a'_2, \dots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

A specially chosen piece of information used to guide sorting. I.e., sort student records by names.

Examples of sorting algorithms

- Selection sort
- Bubble sort
- Insertion sort
- Merge sort
- Heap sort

Evaluate sorting algorithm complexity: the number of key comparisons.

Two properties

Stability: A sorting algorithm is called stable if it preserves the relative order of any two equal elements in its input.

In place: A sorting algorithm is in place if it does not require extra memory, except, possibly for a few memory units.

- ✓ searching
- Find a given value, called a search key, in a given set.
- Examples of searching algorithms
 - Sequential searching
 - Binary searching...
- ✓ string processing
 - A string is a sequence of characters from an alphabet.
 - Text strings: letters, numbers, and special characters.
 - String matching: searching for a given word/pattern in a text.
- ✓ graph problems
 - Informal definition
 - A graph is a collection of points called vertices, some of which are connected by line segments called edges.
 - Modeling real-life problems
 - Modeling WWW
 - communication networks
 - Project scheduling ...

Examples of graph algorithms

- Graph traversal algorithms
- Shortest-path algorithms
- Topological sorting
- ✓ combinatorial problems
- ✓ geometric problems
- ✓ Numerical problems

4. Discuss Fundamentals of the analysis of algorithm efficiency elaborately. [Nov/Dec 2019, Apr/May 2019]

Algorithm's efficiency

Three notations

- Analyze of efficiency of Mathematical Analysis of Recursive Algorithms
- Analyze of efficiency of Mathematical Analysis of non-Recursive Algorithms
- Analysis of algorithms means to investigate an algorithm's efficiency with respect to resources: running time and memory space.
- Time efficiency: how fast an algorithm runs.
- Space efficiency: the space an algorithm requires.
- Measuring an input's size
- Measuring running time
- Orders of growth (of the algorithm's efficiency function)
- Worst-case, best-case and average efficiency

Measuring Input Sizes

Efficiency is defined as a function of input size.

Input size depends on the problem.

Example 1, what is the input size of the problem of sorting n numbers?

Example 2, what is the input size of adding two n by n matrices?

Units for Measuring Running Time

Measure the running time using standard unit of time measurements, such as seconds, minutes?

Depends on the speed of the computer. count the number of times each of an algorithm's operations is executed.

Difficult and unnecessary count the number of times an algorithm's basic operation is executed.

Basic operation: the most important operation of the algorithm, the operation contributing the most to the total running time.

For example, the basic operation is usually the most time-consuming operation in the algorithm's innermost loop.

Orders of Growth

consider only the leading term of a formula

Ignore the constant coefficient.

Worst-Case, Best-Case, and Average-Case Efficiency

Algorithm efficiency depends on the input size n

For some algorithms efficiency depends on type of input.

Example: Sequential Search

Problem: Given a list of n elements and a search key K, find an element equal to K, if any.

Algorithm: Scan the list and compare its successive elements with K until either a matching element is found (successful search) or the list is exhausted (unsuccessful search)

Worst case Efficiency

Efficiency (# of times the basic operation will be executed) for the worst case input of size n.

The algorithm runs the longest among all possible inputs of size n.

Best case Efficiency (# of times the basic operation will be executed) for the best case input of size n.

The algorithm runs the fastest among all possible inputs of size n.

Average case: Efficiency (#of times the basic operation will be executed) for a typical/random input of size n.

NOT the average of worst and best case

5. Elaborate Asymptotic analysis of an algorithm with an example.

Three notations used to compare orders of growth of an algorithm's basic operation count
 $O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$

A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$, if $t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constant c and some nonnegative integer n_0 such that $t(n) \leq cg(n)$ for all $n \geq n_0$

$\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some

constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constant c and some nonnegative integer n_0 such that $t(n) \geq cg(n)$ for all $n \geq n_0$

$\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$

A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n , i.e., if there exist some positive constant c_1 and c_2 and some nonnegative integer n_0 such that $c_2 g(n) \leq t(n) \leq c_1 g(n)$ for all $n \geq n_0$

6. List out the Steps in Mathematical Analysis of non recursive Algorithms.

Steps in mathematical analysis of nonrecursive algorithms:

- Decide on parameter n indicating input size
- Identify algorithm's basic operation
- Check whether the number of times the basic operation is executed depends only on the input size n . If it also depends on the type of input, investigate worst, average, and best case efficiency separately.
- Set up summation for $C(n)$ reflecting the number of times the algorithm's basic operation is executed.

Example: Finding the largest element in a given array

Algorithm MaxElement ($A[0..n-1]$)

```
//Determines the value of the largest element in a given array
//Input: An array A[0..n-1] of real numbers
//Output: The value of the largest element in A maxval ← A[0]
for i ← 1 to n-1 do
if A[i] > maxval
maxval ← A[i]
return maxval
```

7. List out the Steps in Mathematical Analysis of Recursive Algorithms.

Decide on parameter n indicating input size

Identify algorithm's basic operation

Determine worst, average, and best case for input of size n

- ✓ Set up a recurrence relation and initial condition(s) for $C(n)$ -the number of times the basic operation will be executed for an input of size n (alternatively count recursive calls).
- ✓ Solve the recurrence or estimate the order of magnitude of

$$\begin{array}{ll} \text{the solution } F(n) = 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \dots 3 * 2 * 1 & \text{if } n > 0 \end{array}$$

- ✓ Recursive definition

$$\begin{array}{ll} F(n) = 1 & \text{if } n = 0 \\ n * F(n-1) & \text{if } n > 0 \end{array}$$

Algorithm $F(n)$

```
if n=0
else
    return 1          //base case

    return F (n -1) * n      //general case
```

Example Recursive evaluation of $n!$ (2)

✓ Two Recurrences

The one for the factorial function

value: $F(n) = F(n - 1) * n$ for every $n > 0$

$F(0) = 1$

The one for number of multiplications to compute $n!$, $M(n) = M(n - 1) + 1$ for every $n > 0$

$M(0) = 0$

$M(n) \in \Theta(n)$

8. Explain in detail about linear search.

Sequential Search searches for the key value in the given set of items sequentially and returns the position of the key value else returns -1.

ALGORITHM *SequentialSearch($A[0..n - 1]$, K)*

```
//Searches for a given value in a given array by sequential search
//Input: An array  $A[0..n - 1]$  and a search key  $K$ 
//Output: The index of the first element of  $A$  that matches  $K$ 
//          or -1 if there are no matching elements
i  $\leftarrow 0$ 
while  $i < n$  and  $A[i] \neq K$  do
     $i \leftarrow i + 1$ 
if  $i < n$  return  $i$ 
else return -1
```

Analysis:

For sequential search, best-case inputs are lists of size n with their first elements equal to a search key; accordingly,

$$C_{bw}(n) = 1.$$

Average Case Analysis:

(a) The standard assumptions are that the probability of a successful search is equal p and $1-p$ for failure.

(b) the probability of the first match occurring in the i th position of the list is the same for every i . Under these assumptions- the average number of key comparisons $C_{avg}(n)$ is found as follows.

In the case of a successful search, the probability of the first match occurring in the i th position of the list is p/n for every i , and the number of comparisons made by the algorithm in such a situation is obviously i . In the case of an unsuccessful search, the number of comparisons is n with the probability of such a search being $(1-p)$. Therefore

$$\begin{aligned} C_{avg}(n) &= [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \dots + i \cdot \frac{p}{n} + \dots + n \cdot \frac{p}{n}] + n \cdot (1-p) \\ &= \frac{p}{n}[1 + 2 + \dots + i + \dots + n] + n(1-p) \\ &= \frac{p}{n} \frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p). \end{aligned}$$

For example, if $p = 1$ (i.e., the search must be successful), the average number of key comparisons made by sequential search is $(n + 1) / 2$; i.e., the algorithm will inspect, on average, about half of the list's elements. If $p = 0$ (i.e., the search must be unsuccessful), the average number of key comparisons will be n because the algorithm will inspect all n elements on all such inputs.

8. Write an Algorithm using recursion that determines the GCD of two numbers. Determine the time and space complexity. [Nov/Dec 2019]

Extended Euclidean Algorithm:

Extended Euclidean algorithm also finds integer coefficients x and y such that:

$$ax + by = \gcd(a, b)$$

Examples:

Input: $a = 30, b = 20$

Output: $\gcd = 10$

$$x = 1, y = -1$$

(Note that $30*1 + 20*(-1) = 10$)

Input: $a = 35, b = 15$

Output: $\gcd = 5$

$$x = 1, y = -2$$

(Note that $35*1 + 15*(-2) = 5$)

The extended Euclidean algorithm updates results of $\gcd(a, b)$ using the results calculated by recursive call $\gcd(b \% a, a)$. Let values of x and y calculated by the recursive call be x_1 and y_1 . x and y are updated using the below expressions.

$$x = y_1 - \lfloor b/a \rfloor * x_1$$

$$y = x_1$$

Time complexity is $\log_2(\max(a,b))$ and in good case, if $a \mid b$ or $b \mid a$ then time complexity is $O(1)$

UNIT II BRUTE FORCE AND DIVIDE AND CONQUER

QUESTION BANK

PART - A

- ## 1. State the Convex Hull Problem. [Nov/Dec 2019]

The **convex hull** of a set of points is defined as the smallest **convex polygon**, that encloses all of the points in the set. **Convex** means that the **polygon** has no corner that is bent inwards.

2. Write the Brute force algorithm to string matching.[Apr/May 2019]

Algorithm NAÏVE(Text, Pattern)

n=length[Text]

n=length[Pattern]

for $s=1$ to $n-m$

if pattern[1...m]==Text[s+1...s+m]

Print "location of pattern is found with shift s"

3. What is the time and space complexity of Merge Sort? [Apr/May 2019]

Time Complexity	Space Complexity
Best Case: $\Theta(n \log n)$	Best Case: $\Theta(n \log n)$
Average Case: $\Theta(n \log n)$	Average Case: $n \log n$
Worst Case: $\Theta(n \log n)$	Worst Case: $n \log n$

- #### **4. What is exhaustive search? [Apr/May 2018]**

An Exhaustive Search, also known as generate and test, is a very general problem solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

- ## 5. State Master's Theorem. [Apr/May 2018]

Let $T(n)$ be a monotonically increasing function that satisfies

$$T(n) = aT(n/b)$$

$$+ f(n) T(1) = c$$

Where $a > 1$, $b < 0$

Where $a \geq 1, b \geq 2, c > 0$. If $f(n) \in \Theta(n^c)$ where $d \leq c, \dots$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

- ## 6. What is Closest Pair Problem? [May/June 2016, Apr/May 2017]

The closest-pair problem finds the two closest points in a set of n points. It is the simplest of a variety of problems in computational geometry that deals with proximity of points in the plane or higher-dimensional spaces. The distance between two Cartesian coordinates is calculated by Euclidean distance formula

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- ## 7. Give the General Plan for Divide and Conquer Algorithms [Nov/Dec 2017]

A **divide and conquer algorithm** works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (**divide**), until these become simple enough to be solved directly (**conquer**).

Divide-and-conquer algorithms work according to the following general plan:

- A problem is divided into several subproblems of the same type, ideally of about equal size.
- The subproblems are solved (typically recursively, though sometimes a different algorithm is employed, especially when subproblems become small enough).
- If necessary, the solutions to the subproblems are combined to get a solution to the original problem.

Example: Merge sort, Quick sort, Binary search, Multiplication of Large Integers and Strassen's Matrix Multiplication.

8. Write the advantages of insertion sort. [Nov/Dec 2017]

- Simple implementations
- Efficient for Small Data Sets
- Stable
- More efficient
- Online

9. Derive the Complexity of Binary Search [Apr/May 2015]

In conclusion we are now able completely describe the computing time of binary search by giving formulas that describe the best, average and worst cases.

Successful searches	Unsuccessful searches
Best case - $\Theta(1)$ Average case - $\Theta(\log_2 n)$ Worst case - $\Theta(\log_2 n)$	Best case, Average case, Worst case - $\Theta(\log_2 n)$

10. Write about traveling salespersonproblem.

Let $g = (V, E)$ be a directed. The tour of G is a directed simple cycle that includes every vertex in V . The cost of a tour is the sum of the cost of the edges on the tour. The traveling salesperson problem to find a tour of minimum cost.

11. What is binarysearch?

Binary search is a remarkably efficient algorithm for searching in a sorted array. It works by comparing a search key K with the arrays 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 the second half if $K > A[m]$.

$K > A[0] \dots A[m-1] A[m] A[m+1] \dots A[n-1]$
search here if $K \neq A[m]$

12. What is Knapsack problem? [Nov/Dec 2019, Nov/Dec2014]

A bag or sack is given capacity and n objects are given. Each object has weight w_i and profit p_i . Fraction of object is considered as x_i (i.e) $0 <= x_i <= 1$. If fraction is 1 then entire object is put into sack. When we place this fraction into the sack, we get $w_i x_i$ and $p_i x_i$.

13. What is convexhull?

Convex Hull is defined as: If S is a set of points then the Convex Hull of S is the smallest convex set containing

14. Write the algorithm for Iterative binarysearch.

```

Algorithm BinSearch(a,n,x)
    //Given an array a[1:n] of elements in nondecreasing
    // order, n>0, determine whether x is present
    {
        low := 1;
        high := n;
        while (low < high) do
        {
            mid := [(low+high)/2];
            if(x < a[mid]) then high:= mid-1;
            else if (x >a[mid]) then low:=mid + 1;
            else return mid;
        }
        return 0;
    }

```

15. Define internal path length and external pathlength.

The internal path length 'I' is the sum of the distances of all internal nodes from the root.
The external path length E, is defines analogously as sum of the distance of all external nodes from the root.

16. Write an algorithm for brute force closest-pair problem. [Nov/Dec2016]

```

Algorithm BruteForceClosestPair(P )
    //Finds distance between two closest points in the plane by brute force
    //Input: A list P of n ( $n \geq 2$ ) points  $p_1(x_1, y_1), \dots, p_n(x_n, y_n)$ 
    //Output: The distance between the closest pair of points
     $d \leftarrow \infty$ 
    for  $i \leftarrow 1$  to  $n - 1$  do
        for  $j \leftarrow i + 1$  to  $n$  do
             $d \leftarrow \min(d, \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2})$  //sqrt is square root
    return  $d$ 

```

17. 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.

```

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

```

18. Show the recurrence relation of divide-and-conquer?

The recurrence relation is

$$T(n) = g(n) \\ T(n_1) + T(n_2) + \dots + T(n_{BTL}) + f(n)$$

19. What is the Quick sort and Write the Analysis for the Quick sort?

In quick sort, the division into sub arrays is made so that the sorted sub arrays do not need to be merged later. In analyzing QUICKSORT, we can only make the number of element comparisons $c(n)$. It is easy to see that the frequency count of other operations is of the same order as $C(n)$.

20. List out the Advantages in Quick Sort

It is in-place since it uses only a small auxiliary stack

- It requires only $n \log(n)$ time to sort n items
- It has an extremely short inner loop

This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues

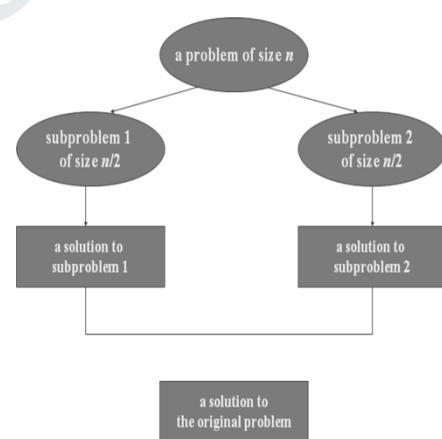
21. What is Divide and Conquer Algorithm? [MAY/JUNE 2016][NOV/DEC 2017]

It is a general algorithm design techniques that solved a problem's instance by dividing it into several smaller instances, solving each of them recursively, and then combining their solutions to the original instance of the problem.

PART B & C**1. Explain Divide and Conquer Method**

The most well-known algorithm design strategy is Divide and Conquer Method. It

1. Divide the problem into two or more smaller subproblems.
2. Conquer the subproblems by solving them recursively.
3. Combine the solutions to the subproblems into the solutions for the original problem.



- ✓ Divide and Conquer Examples
- Sorting: mergesort and quicksort
 - Tree traversals
 - Binary search

- Matrix multiplication-Strassen's algorithm

2. Explain Merge Sort with suitable example.

- ✓ Merge sort definition.

Mergesort sorts a given array A[0..n-1] by dividing it into two halves a[0..(n/2)-1] and A[(n/2)..n-1] sorting each of them recursively and then merging the two smaller sorted arrays into a single sorted one.

- ✓ Steps in Merge Sort

1. Divide Step

If given array A has zero or one element, return S; it is already sorted. Otherwise, divide A into two arrays, A1 and A2, each containing about half of the elements of A.

2. Recursion Step

Recursively sort array A1 and A2.

3. Conquer Step

Combine the elements back in A by merging the sorted arrays A1 and A2 into a sorted sequence

- ✓ Algorithm for merge sort.

ALGORITHM

Mergesort(A[0..n-1])

//Sorts an array A[0..n-1] by recursive mergesort

//Input: An array A[0..n-1] of orderable elements

//Output: Array A[0..n-1] sorted in

nondecreasing order if n > 1

copy A[0..(n/2)-1] to B[0..(n/2)-1]

copy A[(n/2)..n-1] to C[0..(n/2)-1]

Mergesort(B[0..(n/2)-1])

Mergesort(C[0

..(n/2)-1])

Merge(B,C,A)

- ✓ Algorithm to merge two sorted arrays into one.

ALGORITHM Merge (B [0..p-1], C[0..q-1],

A[0..p+q-1])

//Merges two sorted arrays into one sorted array

//Input: arrays B[0..p-1] and C[0..q-1] both sorted

//Output: sorted array A [0..p+q-1] of the

elements of B & C I 0; j 0; k 0

while I < p

and j < q do

if B[I] <=

C[j]

A[k] B [I]; I I+1

else

A[k]

C[j]

]; j j+1 k

k+

1

if i = p

copy C[j..q-1] to A

[k..p+q-1] else
copy B[i..p-1] to A [k..p+q-1]

3. Discuss Quick Sort Algorithm and Explain it with example. Derive Worst case and Average Case Complexity. [Apr/May 2019]

Quick Sort definition

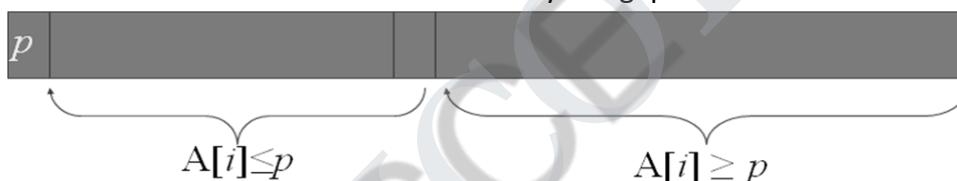
Quick sort is an algorithm of choice in many situations because it is not difficult to implement, it is a good "general purpose" sort and it consumes relatively fewer resources during execution.

Quick Sort and divide and conquer

- Divide: Partition array $A[l..r]$ into 2 subarrays, $A[l..s-1]$ and $A[s+1..r]$ such that each element of the first array is $\leq A[s]$ and each element of the second array is $\geq A[s]$. (Computing the index of s is part of partition.)
- Implication: $A[s]$ will be in its final position in the sorted array.
- Conquer: Sort the two subarrays $A[l..s-1]$ and $A[s+1..r]$ by recursive calls to quicksort
- Combine: No work is needed, because $A[s]$ is already in its correct place after the partition is done, and the two subarrays have been sorted.

✓ Steps in Quicksort

- Select a pivot w.r.t. whose value we are going to divide the sublist. (e.g., $p = A[l]$)
- Rearrange the list so that it starts with the pivot followed by a \leq sublist (a sublist whose elements are all smaller than or equal to the pivot) and a \geq sublist (a sublist whose elements are all greater than or equal to the pivot) Exchange the pivot with the last element in the first sublist(i.e., \leq sublist) – the pivot is now in its final position
- Sort the two sublists recursively using quicksort.



✓ The Quicksort Algorithm

ALGORITHM Quicksort($A[l..r]$)

//Sorts a subarray by quicksort

//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right indices l and r

//Output: The subarray $A[l..r]$ sorted in nondecreasing order if $l < r$

s \square Partition ($A[l..r]$) // s is a split position Quicksort($A[l..s-1]$)

Quicksort($A[s+1..r]$)

ALGORITHM Partition ($A[l ..r]$)

//Partitions a subarray by using its first element as a pivot

//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right indices l and r ($l < r$)

//Output: A partition of $A[l..r]$, with the split position returned as this function's value P $\square A[l]$

i $\square l$; j $\square r + 1$;

Repeat

repeat i $\square i + 1$ until $A[i] \geq p$ //left-

right scan repeat j $\square j - 1$ until $A[j]$

$\leq p$ //right-left scan

```

if (i < j)           //need to continue
    with the scan swap(A[i], a[j])
until i >= j        //no
need to scan swap(A[l], A[j])
return j

```

✓ Advantages in Quick Sort

- It is in-place since it uses only a small auxiliary stack.
- It requires only $n \log(n)$ time to sort n items.
- It has an extremely short inner loop

- This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues.

✓ Disadvantages in Quick Sort

- It is recursive. Especially if recursion is not available, the implementation is extremely complicated.
- It requires quadratic (i.e., n^2) time in the worst-case.
- It is fragile i.e., a simple mistake in the implementation can go unnoticed and cause it to perform badly.

✓ Efficiency of Quicksort

Based on whether the partitioning is balanced.

Best case: split in the middle — $\Theta(n \log n)$

$C(n) = 2C(n/2) + \Theta(n)$ //2 subproblems of size $n/2$ each

Worst case: sorted array! — $\Theta(n^2)$

$C(n) = C(n-1) + n+1$ //2 subproblems of size 0 and $n-1$ respectively

Average case: random arrays — $\Theta(n \log n)$

4. Explain in detail about Travelling Salesman Problem using exhaustive search. [Nov/Dec 2019]

Given n cities with known distances between each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city

Alternatively: Find shortest *Hamiltonian circuit* in a weighted connected graph

Efficiency: $\Theta((n-1)!)$

Refer Anany Levitin's "Introduction to Design and Analysis of Algorithms" for examples.

5. Explain in detail about Knapsack Problem. [Apr/May 2019]

Given n items:

weights: $w_1 \ w_2 \ \dots \ w_n$

values: $v_1 \ v_2 \ \dots \ v_n$ a

knapsack of capacity W

Find most valuable subset of the items that fit into the knapsack

Example: Knapsack capacity W=16

item	weight	value
1	2	\$20
2	5	\$30

3	10	\$50
4	5	\$10
Subset	Total weight	Total value
{1}	2	\$20
{2}	5	\$30
{3}	10	\$50
{4}	5	\$10
{1,2}	7	\$50
{1,3}	12	\$70
{1,4}	7	\$30
{2,3}	15	\$80
{2,4}	10	\$40
{3,4}	15	\$60
{1,2,3}	17	not feasible
{1,2,4}	12	\$60

6. Write algorithm to find closest pair of points using divide and conquer and explain it with example. Derive the worst case and average case time complexity. [Nov/Dec 2019]

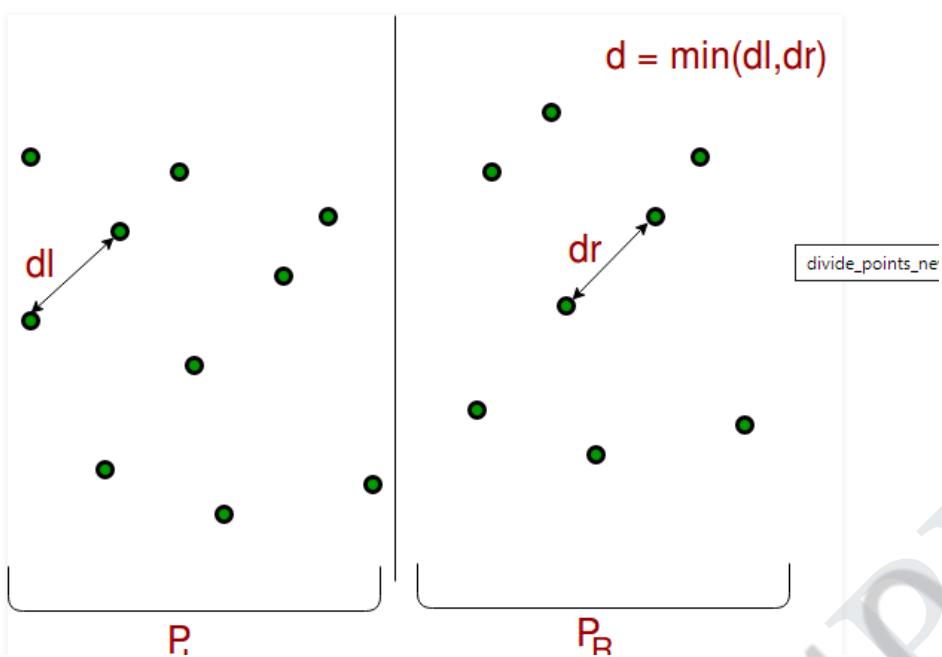
Following are the detailed steps of a $O(n (\log n)^2)$ algorithm.

Input: An array of n points $P[]$

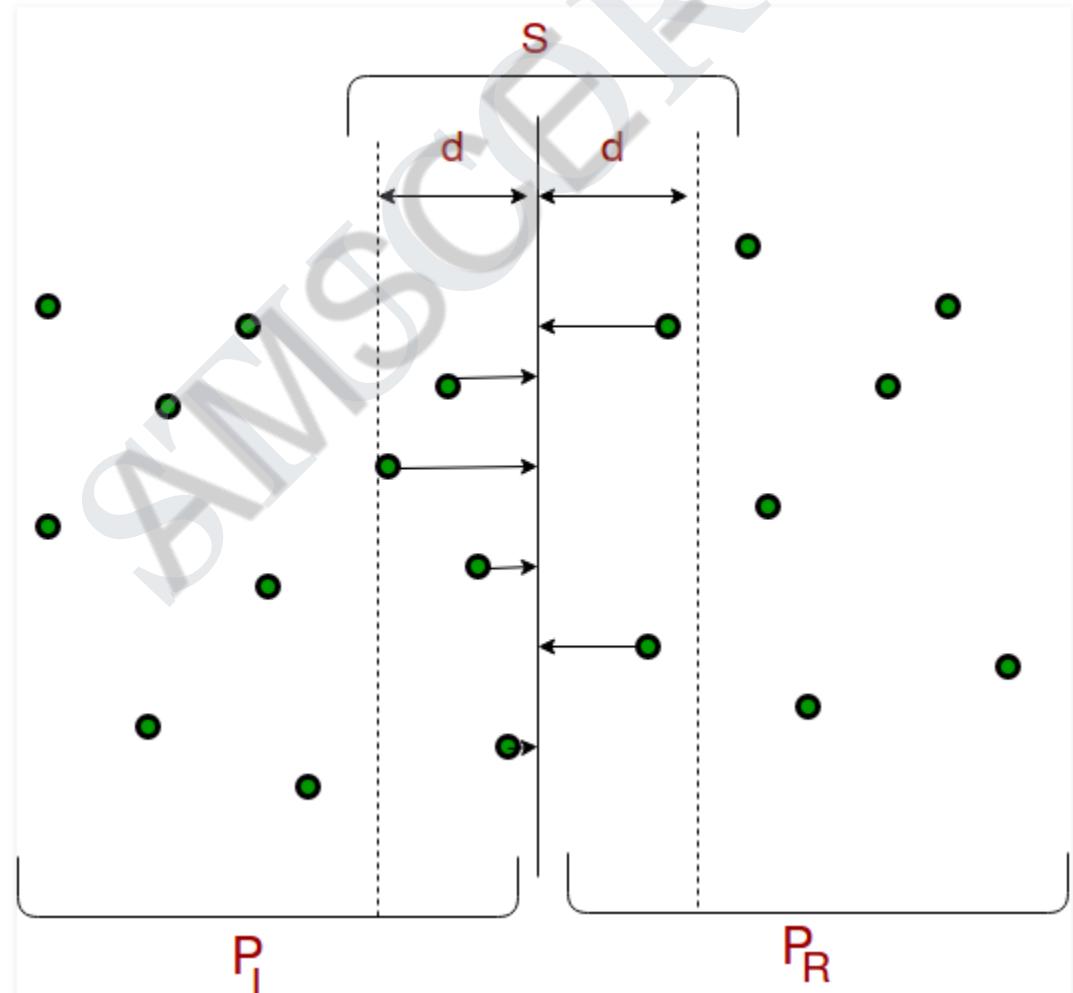
Output: The smallest distance between two points in the given array.

As a pre-processing step, the input array is sorted according to x coordinates.

- 1) Find the middle point in the sorted array, we can take $P[n/2]$ as middle point.
- 2) Divide the given array in two halves. The first subarray contains points from $P[0]$ to $P[n/2]$. The second subarray contains points from $P[n/2+1]$ to $P[n-1]$.
- 3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr . Find the minimum of dl and dr . Let the minimum be d .



From the above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from the left half and the other is from the right half. Consider the vertical line passing through $P[n/2]$ and find all points whose x coordinate is closer than d to the middle vertical line. Build an array $\text{strip}[]$ of all such points.



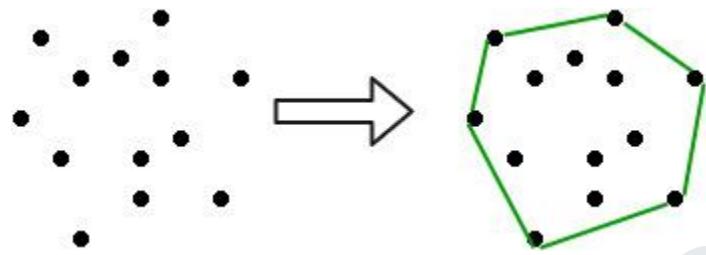
5) Sort the array strip[] according to y coordinates. This step is $O(n\log n)$. It can be optimized to $O(n)$ by recursively sorting and merging.

6) Find the smallest distance in strip[]. This is tricky. From the first look, it seems to be a $O(n^2)$ step, but it is actually $O(n)$. It can be proved geometrically that for every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate). See this for more analysis.

7) Finally return the minimum of d and distance calculated in the above step (step 6)

7. What is Convex hull problem? Explain the brute force approach to solve convex-hull with an example. Derive time complexity. [Apr/May 2019]

A convex hull is the smallest convex polygon containing all the given points.



Input is an array of points specified by their x and y coordinates. The output is the convex hull of this set of points.

- The brute-force method expresses the fundamental solution, which gives you the basic building blocks and understanding to approach more complex solutions
- It's faster to implement
- It's still a viable solution when n is small, and n is usually small.

Brute-force construction

Iterate over every pair of points (p,q)

If all the other points are to the right (or left, depending on implementation) of the line formed by (p,q), the segment (p,q) is part of our result set (i.e. it's part of the convex hull)

Here's the top-level code that handles the iteration and construction of resulting line segments:

```
/**
 * Compute convex hull
 */
var computeConvexHull = function() {
    console.log("--- ");

    for(var i=0; i<points.length; i++) {
        for(var j=0; j<points.length; j++) {
            if(i === j) {

```

```
        continue;  
    }  
  
    var ptI = points[i];  
    var ptJ = points[j];  
  
    // Do all other points lie within the half-plane to the right  
    var allPointsOnTheRight = true;  
    for(var k=0; k<points.length; k++) {  
        if(k === i || k === j) {  
            continue;  
        }  
  
        var d = whichSideOfLine(ptI, ptJ, points[k]);  
        if(d < 0) {  
            allPointsOnTheRight = false;  
            break;  
        }  
    }  
  
    if(allPointsOnTheRight) {  
        console.log("segment " + i + " to " + j);  
        var pointAScreen = cartToScreen(ptI, getDocumentWidth(), getDocumentHeight());  
        var pointBScreen = cartToScreen(ptJ, getDocumentWidth(), getDocumentHeight());  
        drawLineSegment(pointAScreen, pointBScreen);  
    }  
};
```

UNIT-III DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE**QUESTION BANK****PART - A****1. State the Principle of Optimality [Apr/May 2019, Nov/Dec 2017, Nov/Dec 2016]**

The principle of optimality is the basic principle of dynamic programming. It states that an optimal sequence of decisions or choices, each subsequence must also be optimal.

2. What is the Constraint for binary search tree for insertion? [Apr/May 2019]

When inserting or searching for an element in a binary search tree, the key of each visited node has to be compared with the key of the element to be inserted or found. The shape of the binary search tree depends entirely on the order of insertions and deletions, and can become degenerate.

3. Define multistage graph. Give Example. [Nov/Dec 2018]

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).

4. How Dynamic Programming is used to solve Knapsack Problem? [Nov/Dec 2018]

An example of dynamic programming is Knapsack problem. The solution to the Knapsack problem can be viewed as a result of sequence of decisions. We have to decide the value of x_i for $1 \leq i \leq n$. First we make a decision on x_1 and then on x_2 and so on. An optimal sequence of decisions maximizes the object function $\sum p_i x_i$.

5. Define transitive closure of directive graph. [Apr/May 2018]

The transitive closure of a directed graph with 'n' vertices is defined as the n-by-n Boolean matrix $T = \{t_{ij}\}$, in which the elements in the i th row ($1 \leq i \leq n$) and the j th column ($1 \leq j \leq n$) is 1 if there exists a non trivial directed path from the i th vertex to the j th vertex otherwise, t_{ij} is 0 .

6. Define the Minimum Spanning tree problem. [Apr/May 2018]

A **minimum spanning tree (MST)** or **minimum weight spanning tree** is a subset of the edges of a **connected**, edge-weighted (un)directed graph that connects all the **vertices** together, without any cycles and with the minimum possible total edgeweight.

7. What does Floyd's Algorithm do? [Nov/Dec 2017]

Floyd's algorithm is an application, which is used to find the entire pairs shortest paths problem. Floyd's algorithm is applicable to both directed and undirected weighted graph, but they do not contain a cycle of a negative length.

8. State the Assignment Problem [May/June 2016]

There are n people who need to be assigned to execute n jobs as one person per job. Each person is assigned to exactly one job and each job is assigned to exactly one person.

9. Define the Single Source Shortest Path Problem. [May/June 2016]

Single source shortest path problem can be used to find the shortest path from single source to all other vertices.

Example:Dijkstras algorithm

10. List out the memory function under dynamic programming. [Apr/May 2015]

- Top-Down Approach
- Bottom –Up Approach

11. What is Huffman trees?

A Huffman tree is binary tree that minimizes the weighted path length from the root to the leaves containing a set of predefined weights. The most important application of Huffman trees are Huffman code.

12. List the advantage of Huffman's encoding?

- Huffman's encoding is one of the most important file compression methods.
- It is simple
- It is versatility
- It provides optimal and minimum length encoding

13. What do you mean by Huffman code?

A Huffman code is a optimal prefix tree variable length encoding scheme that assigns bit strings to characters based on their frequencies in a given text.

14. What is greedy method?

The greedy method is the most straight forward design, which is applied for change making problem.

The greedy technique suggests constructing a solution to an optimization problem through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. On each step, the choice made must be feasible, locally optimal and irrevocable.

15. What do you mean by row major and column major?

In a given matrix, the maximum elements in a particular row is called row major.

In a given matrix, the maximum elements in a particular column is called column major.

16. Compare Greedy method and Dynamic Programming

Greedy method	Dynamic programming
1.Only one sequence of decisionis generated.	1.Many number of decisions are generated.
2.It does not guarantee to give an optimal solution always.	2.It definitely gives an optimal solution always.

17. Show the general procedure of dynamic programming. [APR/MAY 2017]

The development of dynamic programming algorithm can be broken into a sequence of 4 steps.

- Characterize the structure of an optimalsolution.
- Recursively define the value of the optimalsolution.
- Compute the value of an optimal solution in the bottom-up fashion.
- Construct an optimal solution from the computed information

18.Define Kruskal Algorithm.

Kruskal's algorithm is another greedy algorithm for the minimum spanning tree problem.

Kruskal's algorithm constructs a minimum spanning tree by selecting edges in increasing order of their weights provided that the inclusion does not create a cycle. Kruskal's algorithm provides

a optimal solution.

19. List the features of dynamic programming?

Optimal solutions to sub problems are retained so as to avoid recomputing their values. Decision sequences containing subsequences that are sub optimal are not considered. It definitely gives the optimal solution always.

20. Write the method for Computing a Binomial Coefficient

Computing binomial coefficients is non optimization problem but can be solved using dynamic programming.

Binomial coefficients are represented by $C(n, k)$ or (nk) and can be used to represent the coefficients of a binomial:

$(a + b)^n = C(n, 0)a^n + \dots + C(n, k)a^{n-k}b^k + \dots + C(n, n)b^n$ The recursive relation is defined by the prior power

$$C(n, k) = C(n-1, k-1) + C(n-1, k) \text{ for } n > k > 0$$

$$\text{IC } C(n, 0) = C(n, n) = 1$$

PART B & C

1. Explain Kruskal's Algorithm

Greedy Algorithm for MST: Kruskal

Edges are initially sorted by increasing weight

Start with an empty forest

-grow MST one edge at a time

intermediate stages usually have forest of trees (not connected)

at each stage add minimum weight edge among those not yet used that does not create a cycle

at each stage the edge may:

expand an existing tree

combine two existing trees into a single tree

create a new tree

need efficient way of detecting/avoiding cycles

algorithm stops when all vertices are included

ALGORITHM Krusal(G)

//Input: A weighted connected graph $G = \langle V, E \rangle$

//Output: E_T , the set of edges composing a minimum spanning tree of G.

Sort E in nondecreasing order of the edge weights

$$w(e_{i1}) \leq \dots \leq w(e_{i|E|})$$

$E_T \leftarrow \emptyset$; $e_{\text{counter}} \leftarrow 0$ //initialize the set of tree edges and its size

$$k \leftarrow 0$$

while $e_{\text{counter}} < |V| - 1$ do

$$k \leftarrow k + 1$$

if $E_T \cup \{e_{ik}\}$ is acyclic

$$E_T \leftarrow E_T \cup \{e_{ik}\}; e_{\text{counter}} \leftarrow e_{\text{counter}} + 1$$

return E

2. Discuss Prim's Algorithm in detail.

- ✓ Minimum Spanning Tree (MST)
 - Spanning tree of a connected graph G : a connected acyclic subgraph (tree) of G that includes all of G 's vertices.
 - Minimum Spanning Tree of a weighted, connected graph G : a spanning tree of G of minimum total weight.
- ✓ Prim's MST algorithm
- ✓ Start with a tree, T_0 , consisting of one vertex
- ✓ -Grow|| tree one vertex/edge at a time
 - Construct a series of expanding subtrees T_1, T_2, \dots, T_{n-1} . At each stage construct T_{i+1} from T_i by adding the minimum weight edge connecting a vertex in tree (T_i) to one not yet in tree
 - choose from -fringe|| edges
- ✓ (this is the -greedy|| step!) Or (another way to understand it)
- ✓ expanding each tree (T_i) in a greedy manner by attaching to it the nearest vertex not in that tree. (a vertex not in the tree connected to a vertex in the tree by an edge of the smallest weight)
- ✓ Algorithm stops when all vertices are included

Algorithm: ALGORITHM Prim(G)

//Prim's algorithm for constructing a minimum spanning tree

//Input A weighted connected graph $G = V, E$

//Output E_T , the set of edges composing a minimum spanning tree of G $V_T \{v_0\}$

\leftarrow
 $E_T \leftarrow F$

for $i \leftarrow 1$ to $|V|-1$ do

Find the minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u) such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$
 \leftarrow
 $E_T \quad E_T \cup \{e^*\}$

3. Write short note on Greedy Method. [Nov/Dec 2019]

- ✓ A greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
 - The choice made at each step must be:
 - Satisfy the problem's constraints
 - locally optimal
 - Be the best local choice among all feasible choices
 - Irrevocable
 - Once made, the choice can't be changed on subsequent steps.
- ✓ Applications of the Greedy Strategy
 - Optimal solutions:
 - change making

- Minimum Spanning Tree (MST)
- Single-source shortest paths
- Huffman codes
- Approximations:
 - Traveling Salesman Problem (TSP)
 - Knapsack problem
 - other optimization problems

4. What does dynamic programming have in common with divide-and-Conquer?

- ✓ **Dynamic Programming**
- Dynamic Programming is a general algorithm design technique. -Programming here means -planning.
- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems
- Main idea:
 - solve several smaller (overlapping) subproblems
 - record solutions in a table so that each subproblem is only solved once
 - final state of the table will be (or contain) solution

Dynamic programming vs. divide-and-conquer

- ✓ partition a problem into overlapping subproblems and independent ones
- ✓ store and not store solutions to subproblems

Example: Fibonacci numbers

Recall definition of Fibonacci numbers:

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n-1) + f(n-2)$$

5. Explain how to Floyd's Algorithm works. [Apr/May 2019]

- All pairs shortest paths problem: In a weighted graph, find shortest paths between every pair of vertices.
- Applicable to: undirected and directed weighted graphs; no negative weight.

Same idea as the Warshall's algorithm : construct solution through series of matrices $D(0), D(1), \dots, D(n)$

Refer Examples from Anany Levitin. "Introduction to Design and Analysis of Algorithms".

6. Construct optimal binary search tree for the following 5 keys with probabilities as indicated.

i	0	1	2	3	4	5
p		.15	.10	.05	.10	.20
q	0.05	.10	.05	.05	.05	.10

[Nov/Dec 2019]

Definition about Optimal Binary search trees.

Refer Examples from Anany Levitin. "Introduction to Design and Analysis of Algorithms".

7. Write Huffman code algorithm and derive its complexity. [Apr/May 2019]

Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be “cccd” or “ccb” or “acd” or “ab”.

There are mainly two major parts in Huffman Coding

- 1) Build a Huffman Tree from input characters.
- 2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

Refer Examples from Anany Levitin. “Introduction to Design and Analysis of Algorithms”.

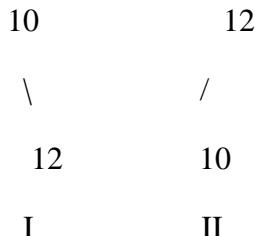
10. Outline the Dynamic Programming approach to solve the Optimal Binary Search Tree problem and analyze it time complexity. [Nov/Dec 2019]

Given a sorted array $\text{keys}[0.. n-1]$ of search keys and an array $\text{freq}[0.. n-1]$ of frequency counts, where $\text{freq}[i]$ is the number of searches to $\text{keys}[i]$. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

Input: $\text{keys}[] = \{10, 12\}$, $\text{freq}[] = \{34, 50\}$

There can be following two possible BSTs



We need to calculate $\text{optCost}(0, n-1)$ to find the result.

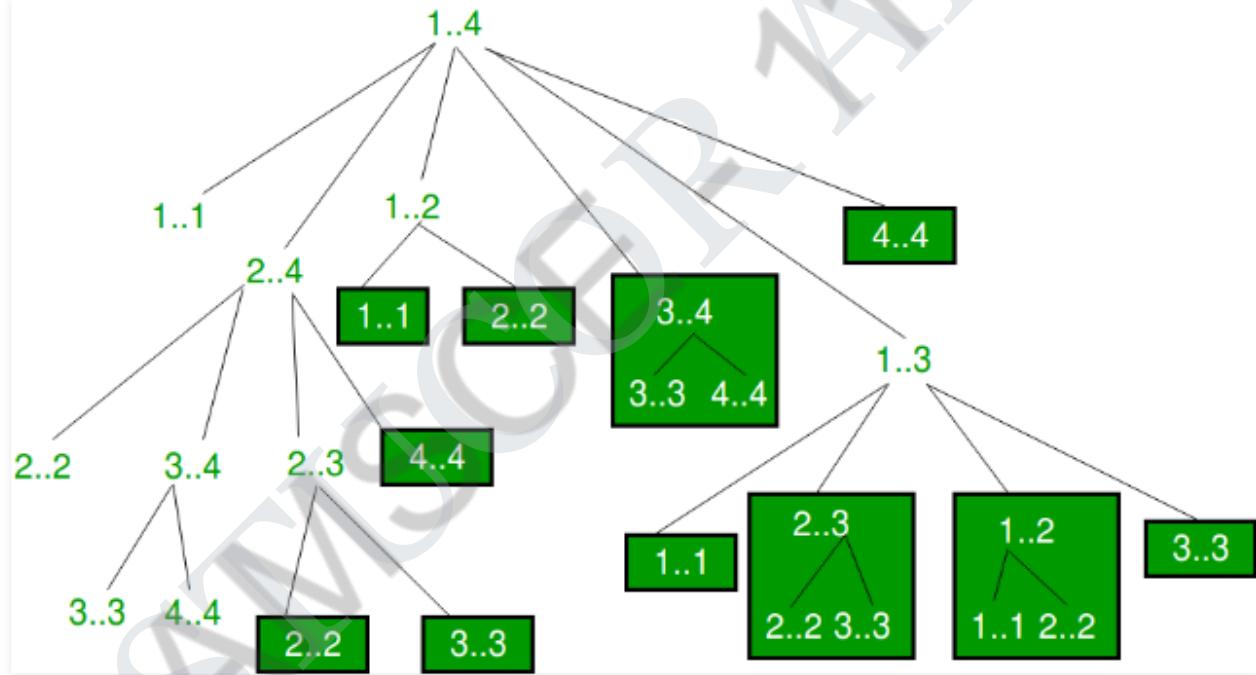
The idea of above formula is simple, we one by one try all nodes as root (r varies from i to j in second term). When we make rth node as root, we recursively calculate optimal cost from i to r-1 and r+1 to j.

We add sum of frequencies from i to j (see first term in the above formula), this is added because every search will go through root and one comparison will be done for every search.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. We can see many subproblems being repeated in the following recursion tree for $\text{freq}[1..4]$.



UNIT- IV ITERATIVE IMPROVEMENT

PART A

1. State the Principle of Duality. [Apr/May 2019]

The **principle of duality** in Boolean algebra **states** that if you have a true Boolean statement (equation) then the **dual** of this statement (equation) is true. The **dual** of a boolean statement is found by replacing the statement's symbols with their counterparts.

2. Define the Capacity Constraint in the context of Maximum flow problem. [Apr/May 2019]

It represents the **maximum** amount of **flow** that can pass through an edge. for each (**capacity constraint**: the **flow** of an edge cannot exceed its **capacity**) for each (**conservation of flows**: the sum of the **flows** entering a node must equal the sum of the **flows** exiting a node, except for the source and the sink nodes)

3. Define iterative improvement technique [Nov/Dec 2018]

This is a computational technique in which with the help of initial feasible solution the optimal solution is obtained iteratively until no improvement is found.

4. What is solution space? Give an example [Nov/Dec 2018]

In mathematical optimization, a feasible region, feasible set, 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. In linear programming problems, the feasible set is a convex polytope.

5. What is articulation point in graph? [Apr/May 2017]

A vertex in an undirected connected **graph** is an **articulation point** (or cut vertex) iff removing it (and edges through it) disconnects the **graph**. It can be thought of as a single **point** of failure.

6. What is state space graph? [May/June 2016]

Graph organization of the solution space is state space tree.

7. What do you mean by ‘perfect matching’ in bipartite graph? [Apr/May 2015]

A perfect matching of a graph is a matching (i.e., an independent edge set) in which every vertex of the graph is incident to exactly one edge of the matching. A perfect matching is therefore a matching containing $n/2$ edges (the largest possible), meaning perfect matching are only possible on graphs with an even number of vertices.

8. Define Flow ‘cut’. [Apr/May 2015]

It contains exactly one vertex with no entering edges; this vertex is called the **source** and assumed to be numbered 1. It contains exactly one vertex with no leaving edges; this vertex is called the **sink** and assumed to be numbered n . The weight uij of each directed edge (i, j) is a positive integer, called the edge **capacity**. (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.) A digraph satisfying these properties is called a **flownetwork** or simply a **network**.

Cut is a collection of arcs such that if they are removed there is no path from source to sink

9. What is maximum cardinality matching? [Nov/Dec 2016]

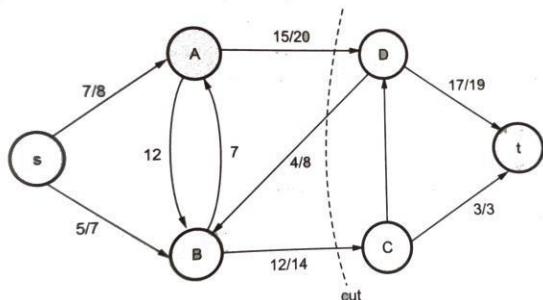
A **maximum matching** (also known as **maximum-cardinality matching**) is a **matching** that contains the largest possible number of edges. There may be many **maximum matchings**. The

matching number of a graph is the size of a **maximum matching**.

10. Define Network Flow and Cut[Apr/May 2015, Nov/Dec 2015]

A network flow graph $G = (V, E)$ is a directed graph with two special vertices: the source vertex s , and the sink (destination) vertex t . A flow network (also known as a transportation network) is a directed graph where each edge has a capacity and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge.

A cut is a collection of arcs such that if they are removed there is no path from s to t



A cut is said to be minimum in a network whose capacity is minimum over all cuts of the network.

11. What is meant by Bipartite Graph?[Nov/Dec 2017]

A Bipartite Graph $G = (V, E)$ is a graph in which the vertex set V can be divided into two disjoint subsets X and Y such that every edge $e \in E$ has one end point in X and the other end point in Y . A matching M is a subset of edges such that each node in V appears in at most one edge in M .

12. Give the Floyd's algorithm

```

ALGORITHM Floyd(
    W[1..n,1..n])
    //Implements Floyd's algorithm for the all-pair shortest-path problem
    //Input The weight matrix W of a graph
    //Output The distance matrix of the shortest paths' lengths
    D  $\downarrow$  W
    for k  $\downarrow$  1 to n do
        for i  $\downarrow$  1 to n do
            for j  $\downarrow$  1 to n do
                D[i,j]  $\downarrow$  min{D[i,j], D[i,k] + D[k,j]}
    return D
  
```

13. Define Ford – Fulkerson Method.

1. Start with the zero flow ($x_{ij} = 0$ for every edge)
2. On each iteration, try to find a *flow-augmenting path* from source to sink, which a path along which some additional flow can be sent
3. If a flow-augmenting path is found, adjust the flow along the edges of this path to get a flow of increased value and try again
4. If no flow-augmenting path is found, the current flow is maximum
5. How to find flow augmenting path in Network flow problem

14. Define Stable Marriage Problem.

SMP (Stable Marriage Problem) is the problem of finding a stable matching between two sets of elements given a set of preferences for each element.

15. State the extremepoints

Any LP Problem with a nonempty bounded feasible region has an optimal solution; moreover an optimal solution can always be founded at an extreme point of the problem's feasible region.

This theorem implies that to solve a linear programming problem. at least in the case of a bounded feasible region. We can ignore all but a finite number of points in its feasible region.

16. Write the three important things in Ford-Fulkerson method.

- 1.Residual network
- 2.Augmenting path
- 3.Cuts

17.How is a transportation network represented? [APR/MAY 2018]

Transportation networks generally refer to a set of links, nodes, and lines that represent the infrastructure or supply side of the transportation. The links have characteristics such as speed and capacity for roadways; frequency and travel time data are defined on transit links or lines for the transit system.

18. When a linear programming is said to be unbounded? [Nov/Dec 2019]

An unbounded solution of a linear programming problem is a situation where objective function is infinite. A linear programming problem is said to have unbounded solution if its solution can be made infinitely large without violating any of its constraints in the problem. Since there is no real applied problem which has infinite return, hence an unbounded solution always represents a problem that has been incorrectly formulated.

19. What is residual network in the context of flow networks? [Nov/Dec 2019]

Residual Graph of a flow network is a graph which indicates additional possible flow. If there is a path from source to sink in residual graph, then it is possible to add flow. Every edge of a residual graph has a value called residual capacity which is equal to original capacity of the edge minus current flow.

PART B & C**1. Explain in detail about Simplex Method.**

Every LP Problem can be represented in such form

Maximize $3x+5y$

Subject to $x+y \leq 4$

$x \geq 0, y \geq 0$

Maximize $3x+5y+0u+0v$

Subject to $x+y+u=4$

$x+3y+v=6$

There is a 1-1 correspondence between extreme points of LP's feasible region and its basic feasible solutions.

$$\begin{aligned} \text{maximize } & z = 3x + 5y + 0u + 0v \\ \text{subject to } & x + y + u = 4 \\ & x + 3y + v = 6 \\ & x \geq 0, y \geq 0, u \geq 0, v \geq 0 \text{ basic feasible solution} \\ & (0, 0, 4, 6) \end{aligned}$$

Simplex method:

Step 0 [Initialization] Present a given LP problem in standard form and set up initial tableau.

Step 1 [Optimality test] If all entries in the objective row are nonnegative — stop: the tableau represents an optimal solution.

Step 2 [Find entering variable] Select (the most) negative entry in the objective row. Mark its column to indicate the entering variable and the pivot column.

Step 3 [Find departing variable] For each positive entry in the pivot column, calculate the θ -ratio by dividing that row's entry in the rightmost column by its entry in the pivot column. (If there are no positive entries in the pivot column — stop: the problem is unbounded.) Find the row with the smallest θ -ratio, mark this row to indicate the departing variable and the pivot row.

Step 4 [Form the next tableau] Divide all the entries in the pivot row by its entry in the pivot column. Subtract from each of the other rows, including the objective row, the new pivot row multiplied by the entry in the pivot column of the row in question. Replace the label of the pivot row by the variable's name of the pivot column and go back to Step 1.

$$\begin{aligned} \text{maximize } & z = 3x + 5y + 0u + 0v \\ \text{subject to } & x + y + u = 4 \\ & x + 3y + v = 6 \\ & x \geq 0, y \geq 0, u \geq 0, v \geq 0 \end{aligned}$$

Refer, Anany Levitin's "Introduction to Design and Analysis of Algorithms" for problem solution.

2. Explain in detail about Maximum Flow Problem [Apr/May 2019]

Problem of maximizing the flow of a material through a transportation network (e.g., pipeline system, communications or transportation networks)

Formally represented by a connected weighted digraph with n vertices numbered from 1 to n with the following properties:

- ✓ contains exactly one vertex with no entering edges, called the *source* (numbered 1)
- ✓ contains exactly one vertex with no leaving edges, called the *sink* (numbered n)
- ✓ has positive integer weight u_{ij} on each directed edge (i,j) , called the *edge capacity*, indicating the upper bound on the amount of the material that can be sent from i to j through this edge

Definition of flow:

A *flow* is an assignment of real numbers x_{ij} to edges (i,j) of a given network that satisfy the following:

- **flow-conservation requirements:** The total amount of material entering an intermediate vertex must be equal to the total amount of the material leaving the vertex
- **capacity constraints**

$$0 \leq x_{ij} \leq u_{ij} \text{ for every edge } (i,j) \in E$$

Flow value and Maximum Flow Problem

Since no material can be lost or added to by going through intermediate vertices of the network, the total amount of the material leaving the source must end up at the sink:

$$\sum_{j: (1,j) \in E} x_{1j} = \sum_{j: (j,n) \in E} x_{jn}$$

The *value* of the flow is defined as the total outflow from the source (= the total inflow into the sink).

Maximum-Flow Problem as LP problem

$$\text{Maximize } v = \sum_{j: (1,j) \in E} x_{1j}$$

$$j: (1,j) \in E$$

Augmenting Path (Ford-Fulkerson) Method

- Start with the zero flow ($x_{ij} = 0$ for every edge)
- On each iteration, try to find a *flow-augmenting path* from source to sink, which a path along which some additional flow can be sent
- If a flow-augmenting path is found, adjust the flow along the edges of this path to get a flow of increased value and try again
- If no flow-augmenting path is found, the current flow is maximum

Finding a Flow augmenting Path

To find a flow-augmenting path for a flow x , consider paths from source to sink in the underlying undirected graph in which any two consecutive vertices i,j are either:

- connected by a directed edge (i to j) with some positive unused capacity $r_{ij} = u_{ij} - x_{ij}$ known as *forward edge* (\rightarrow)
- connected by a directed edge (j to i) with positive flow x_{ji} known as *backward edge* (\leftarrow)

If a flow-augmenting path is found, the current flow can be increased by r units by increasing x_{ij} by r on each forward edge and decreasing x_{ji} by r on each backward edge where $r = \min \{r_{ij} \text{ on all forward edges, } x_{ji} \text{ on all backward edges}\}$

- Assuming the edge capacities are integers, r is a positive integer
- On each iteration, the flow value increases by at least 1
- Maximum value is bounded by the sum of the capacities of the edges leaving the source; hence the augmenting-path method has to stop after a finite number of iterations

- The final flow is always maximum, its value doesn't depend on a sequence of augmenting paths used
- The augmenting-path method doesn't prescribe a specific way for generating flow-augmenting paths
- Selecting a bad sequence of augmenting paths could impact the method's efficiency.

Definition of a Cut:

Let X be a set of vertices in a network that includes its source but does not include its sink, and let X^c , the complement of X , be the rest of the vertices including the sink. The *cut* induced by this partition of the vertices is the set of all the edges with a tail in X and a head in X^c .

Capacity of a cut is defined as the sum of capacities of the edges that compose the cut.

- We'll denote a cut and its capacity by $C(X, X^c)$ and $c(X, X^c)$
- Note that if all the edges of a cut were deleted from the network, there would be no directed path from source to sink
- *Minimum cut* is a cut of the smallest capacity in a given network

Max-Flow Min-Cut Theorem

- The value of maximum flow in a network is equal to the capacity of its minimum cut
- The shortest augmenting path algorithm yields both a maximum flow and a minimum cut:
 - maximum flow is the final flow produced by the algorithm
 - minimum cut is formed by all the edges from the labeled vertices to unlabeled vertices on the last iteration of the algorithm
 - all the edges from the labeled to unlabeled vertices are full, i.e., their flow amounts are equal to the edge capacities, while all the edges from the unlabeled to labeled vertices, if any, have zero flow amounts on them.

Refer, Anany Levitin's "Introduction to Design and Analysis of Algorithms" for problem solution.

3. Outline the stable marriage problem with example. [Nov/Dec 2019]

The Stable Marriage Problem states that given N men and N women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are "stable".

Consider the following example.

Let there be two men m_1 and m_2 and two women w_1 and w_2 .

Let m_1 's list of preferences be $\{w_1, w_2\}$

Let m_2 's list of preferences be $\{w_1, w_2\}$

Let w_1 's list of preferences be $\{m_1, m_2\}$

Let w_2 's list of preferences be $\{m_1, m_2\}$

The matching $\{ \{m_1, w_2\}, \{w_1, m_2\} \}$ is not stable because m_1 and w_1 would prefer each other over their assigned partners. The matching $\{m_1, w_1\}$ and $\{m_2, w_2\}$ is stable because there are no two people of opposite sex that would prefer each other over their assigned partners.

It is always possible to form stable marriages from lists of preferences (See references for proof). Following is Gale–Shapley algorithm to find a stable matching:

The idea is to iterate through all free men while there is any free man available. Every free man goes to all women in his preference list according to the order. For every woman he goes to, he checks if the woman is free, if yes, they both become engaged. If the woman is not free, then the woman chooses either says no to him or dumps her current engagement according to her preference list. So an engagement done once can be broken if a woman gets better option. Time Complexity of Gale-Shapley Algorithm is $O(n^2)$.

```

Initialize all men and women to free
while there exist a free man m who still has a woman w to propose to
{
    w = m's highest ranked such woman to whom he has not yet proposed
    if w is free
        (m, w) become engaged
    else some pair (m', w) already exists
        if w prefers m to m'
            (m, w) become engaged
            m' becomes free
        else
            (m', w) remain engaged
}

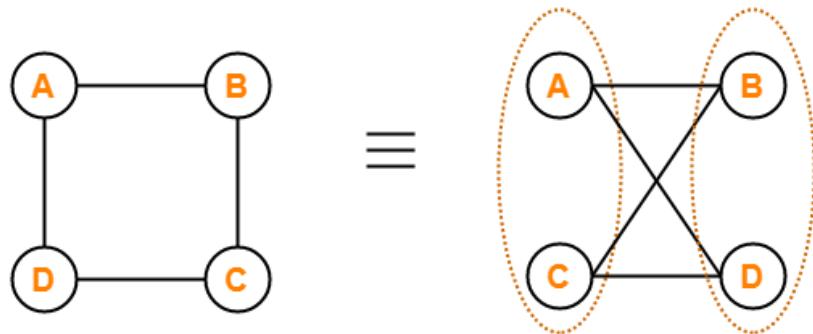
```

Input is a 2D matrix of size $(2*N) * N$ where N is number of women or men. Rows from 0 to $N-1$ represent preference lists of men and rows from N to $2*N - 1$ represent preference lists of women. So men are numbered from 0 to $N-1$ and women are numbered from N to $2*N - 1$. The output is list of married pairs.

4. What is bipartite graph? Is the subset of bipartite graph is bipartite? Outline with example. [Nov/Dec 2019]

A bipartite graph is a special kind of graph with the following properties-

- It consists of two sets of vertices X and Y.
- The vertices of set X join only with the vertices of set Y.
- The vertices within the same set do not join.

**Example of Bipartite Graph**

Here,

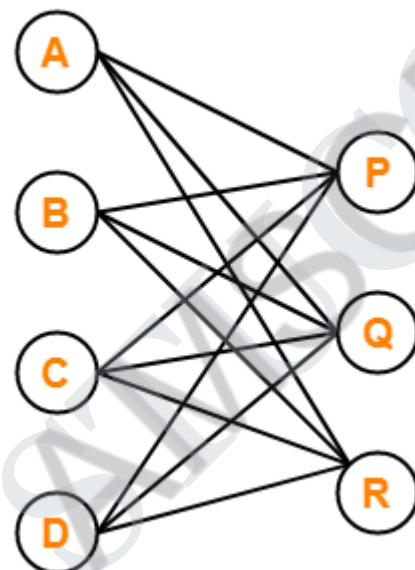
The vertices of the graph can be decomposed into two sets.

The two sets are $X = \{A, C\}$ and $Y = \{B, D\}$.

The vertices of set X join only with the vertices of set Y and vice-versa.

The vertices within the same set do not join.

Therefore, it is a bipartite graph.



Complete Bi-partite graph.

Here,

This graph is a bipartite graph as well as a complete graph.

Therefore, it is a complete bipartite graph.

This graph is called as $K4,3$.

Refer Anany Levitin's "Introduction to Design and Analysis of Algorithms" for example.

7. Solve the following equation using Simplex Method. [Apr/May 2019]

Maximize: $18x_1 + 12.5x_2$

Subject to $x_1 + x_2 \leq 20$

$x_1 \leq 12$

$x_2 \leq 16$

$x_1, x_2 \geq 0$

Refer Anany Levitin's "Introduction to Design and Analysis of Algorithms" for example and notes for solution.

UNIT V COPING WITH LIMITATIONS OF ALGORITHMIC POWER

PART A

1. Define NP Completeness and NP Hard. [Apr/May 2019]

The problems whose solutions have computing times are bounded by polynomials of small degree.

2. State Hamiltonian Circuit Problem [Nov/Dec 2019, Apr/May 2019]

Hamiltonian circuit problem is a problem of finding a Hamiltonian circuit. Hamiltonian circuit is a circuit that visits every vertex exactly once and return to the starting vertex.

3. Define P and NP Problems. [Nov/Dec 2018]

In computational complexity theory, P, also known as PTIME or DTIME(n), is a fundamental complexity class. It contains all decision problems that can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time.

NP: the class of decision problems that are solvable in polynomial time on a nondeterministic machine (or with a nondeterministic algorithm). (A deterministic computer is what we know). A nondeterministic computer is one that can “guess” the right answer or solution think of a nondeterministic computer as a parallel machine that can freely spawn an infinite number of processes

4. Define sum of subset Problem. [Nov/Dec 2018]

Subset sum problem is a problem, which is used to find a subset of a given set $S=\{S_1, S_2, S_3, \dots, S_n\}$ of n positive integers whose sum is equal to given positive integer d .

5. Differentiate Feasible and Optimal Solution. [Nov/Dec 2017]

Feasible solution means set which contains all the possible solution which follow all the constraints.

An **optimal solution** is a feasible **solution** where the objective function reaches its maximum (or minimum) value – for example, the most profit or the least cost. A globally **optimal solution** is one where there are no other feasible **solutions** with better objective function values

6. Explain Promising and Non-Promising Node [Nov/Dec 2017]

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.

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.

7. State the reason for terminating search path at the current node in branch and bound algorithm.[Nov/Dec 2016]

In general, we terminate a search path at the current node in a state-space tree of a branch-and-bound algorithm for any one of the following three reasons:

- The value of the node's bound is not better than the value of the best solution seen so far.
- The node represents no feasible solutions because the constraints of the problem are already violated.
- The subset of feasible solutions represented by the node consists of a single point (and hence no further choices can be made)—in this case we compare the value of the objective function for this feasible solution with that of the best solution seen so far and update the latter with the former if the new solution is better.

8. How is lower bound found by problem reduction? [Apr/May 2018]

If problem P is at least as hard as problem Q ,then a lower bound for Q is also a lower bound for P. Hence ,find problem Q with a known lower bound that can be reduced to problem P in question. then any algorithm that solves P will also solve Q.

9. What are tractable and non-tractable problems? [Apr/May 2018]

Problems that are solvable by polynomial time algorithms as being **tractable**, and **problems** that require super polynomial time as being **intractable**.

- Sometimes the line between what is an 'easy' **problem** and what is a 'hard' **problem** is a fine one

10. Compare backtracking and branch bound techniques.

Backtracking is applicable only to non optimization problems.

Backtracking generates state space tree in depth first manner.

Branch and bound is applicable only to optimization problem.

Branch and bound generated a node of state space tree using best first rule.

11.What are the searching techniques that are commonly used in Branch-and-Bound method.

The searching techniques that are commonly used in Branch-and-Bound method are:

- i. FIFO
- ii. LIFO
- iii. LC
- iv. Heuristic search

12.Illustrate 8 – Queens problem.

The problem is to place eight queens on a 8 x 8 chessboard so that no two queen “attack” that is, so that no two of them are on the same row, column or on the diagonal.

13. Show the purpose of lower bound. [MAY/JUNE 2016]

Lower bound of a problem is an estimate on a minimum amount of work needed to solve a given problem

14.Define chromatic number of the graph.

The m – color ability optimization problem asks for the smallest integer m for which the graph G can be colored. This integer is referred to as the chromatic number of the graph.

15.What is Absolute approximation?

A is an absolute approximation algorithm if there exists a constant k such that, for every instance I of P, $|A^*(I) - A(I)| \leq k$. For example, Planar graph coloring.

16.What is Relative approximation?

A is an relative approximation algorithm if there exists a constant k such that, for every instance I of P, $\max\{A^*(I), A(I)\} / A^*(I) \leq k$. For example, Vertex cover.

17.Show the application for Knapsack problem

The Knapsack problem is problem in combinatorial optimization. It derives its name from the maximum problem of choosing possible essential that can fit into one bag to be carried on a trip. A similar problem very often appears in business, combinatory, complexitytheory, cryptography and applied mathematics.

18. Define Branch-and-Bound method.

The term Branch-and-Bound refers to all the state space methods in which all children of the E-node are generated before any other live node can become the E-node.

19. Define backtracking.

Depth first node generation with bounding function is called backtracking. The backtracking algorithm has its virtue the ability to yield the answer with far fewer than m trials.

20.List the factors that influence the efficiency of the backtracking algorithm?

The efficiency of the backtracking algorithm depends on the following four factors. They are:

- The time needed to generate the next x_{BTL}
- The number of x_k satisfying the explicit constraints.
- The time for the bounding functions B_k
- The number of x_k satisfying the B_k

21. When is a problem said to be NP Hard?

A problem is said to be NP-hard if everything in NP can be transformed in polynomial time into it, and a problem is NP-complete if it is both in NP and NP-hard.

PART-B & C**1. Elaborate how backtracking technique can be used to solve the n-queens problem. Explain with an example. [Nov/Dec 2019]**

The problem is to place eight queens on a 8×8 chessboard so that no two queen -attack|| that is, so that no two of them are on the same row, column or on the diagonal.

			Q				
					Q		
							Q
		Q					
							Q
						Q	
Q							
			Q				

Algorithm

isValid(board, row, col)

Input: The chess board, row and the column of the board.

Output: True when placing a queen in row and place position is a valid or not.

Begin

```
if there is a queen at the left of current col, then
    return false
if there is a queen at the left upper diagonal, then
    return false
if there is a queen at the left lower diagonal, then
    return false;
return true //otherwise it is valid place
```

End

solveNQueen(board, col)

Input: The chess board, the col where the queen is trying to be placed.

Output: The position matrix where queens are placed.

Begin

```
if all columns are filled, then
    return true
for each row of the board, do
    if isValid(board, i, col), then
        set queen at place (i, col) in the board
        if solveNQueen(board, col+1) = true, then
            return true
        otherwise remove queen from place (i, col) from board.
```

done

return false

End

2. Explain Backtracking technique.

Backtracking technique is a refinement of this approach. Backtracking is a surprisingly simple approach and can be used even for solving the hardest Sudoku puzzle.

Problems that need to find an element in a domain that grows exponentially with the size of the input, like the Hamiltonian circuit and the Knapsack problem, are not solvable in polynomial time. Such problems can be solved by the exhaustive search technique, which requires identifying the correct solution from many candidate solutions.

Steps to achieve Goal:

- Backtracking possible by constructing the state-space tree, this is a tree of choices.
- The root of the state-space tree indicates the initial state, before the search for the solution begins.
- The nodes of each level of this tree signify the candidate solutions for the corresponding component.
- A node of this tree is considered to be promising if it represents a partially constructed solution that can lead to a complete solution, else they are considered to be non-promising.
- The leaves of the tree signify either the non-promising dead-ends or the complete solutions.
- Depth-First-search method usually for constructing these state-space-trees.

- If a node is promising, then a child-node is generated by adding the first legitimate choice of the next component and the processing continues for the child node.

Example

- This technique is illustrated by the following figure.
- Here the algorithm goes from the start node to node 1 and then to node 2.
- When no solution is found it backtracks to node1 and goes to the next possible solution node
 - i. But node 3 is also a dead-end.

3.

4. Give solution to Hamiltonian circuit using Backtracking technique

- The graph of the Hamiltonian circuit is shown below.
- In the below Hamiltonian circuit, circuit starts at vertex i, which is the root of the state-space tree.
- From i, we can move to any of its adjoining vertices which are j, k, and l. We first select j, and then move to k, then l, then to m and thereon to n. But this proves to be a dead-end.
- So, we backtrack from n to m, then to l, then to k which is the next alternative solution. But moving from k to m also leads to a dead-end.
- So, we backtrack from m to k, then to j. From there we move to the vertices n, m, k, l and correctly return to i. Thus the circuit traversed is i->j->n->m->k->l->i.

5. Give solution to Subset sum problem using Backtracking technique [Apr/May 2019]

- In the Subset-Sum problem, we have to find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a positive integer t .
- Let us assume that the set S is arranged in ascending order. For example, if $S = \{2, 3, 5, 8\}$ and if $t = 10$, then the possible solutions are $\{2, 3, 5\}$ and $\{2, 8\}$.
- The root of the tree is the starting point and its left and right children represent the inclusion and exclusion of 2.
- Similarly, the left node of the first level represents the inclusion of 3 and the right node the exclusion of 3.
- Thus the path from the root to the node at the i th level shows the first i numbers that have been included in the subsets that the node represents.
- Thus, each node from level 1 records the sum of the numbers $Ssum$ along the path upto that particular node.
- If $Ssum$ equals t , then that node is the solution. If more solutions have to be found, then we can backtrack to that node's parent and repeat the process. The process is terminated for any non-promising node that meets any of the following two conditions

6. Explain P, NP and NP complete problems.

Definition:

An algorithm solves a problem in polynomial time if its worst-case time efficiency belongs to $O(P(n))$.

$P(n)$ is a polynomial of the problem 's input size n .

Tractable:

Problems that can be solved in polynomial time are called tractable.

Intractable:

Problems that cannot be solved in polynomial time are called intractable. We cannot solve arbitrary instances in reasonable amount of time.

Huge difference between running time. Sum and composition of 2 polynomial results in polynomial .Development of extensive theory called computational complexity.

Definition: II

P and NP Problems:

Class P:

It is a class of decision problems that can be solved in polynomial time by deterministic algorithm, where as deterministic algorithm is every operation uniquely defined.

Decision problems:

A problem with yes/no answers called decision problems.

Restriction of P to decision problems. Sensible to execute problems not solvable in polynomial time because of their large output.

Eg: Subset of the given set

Many important problems that are decision problem can be reduced to a series of decision problems that are easier to study.

Undecidable problem:

Some decision problems cannot be solved at all by any algorithm.

Halting Problem:

Given a computer program and an input to it, determine whether the program halt at input or continue work indefinitely on it.

Definition: III (A non-deterministic algorithm)

Here two stage procedure:

Non-deterministic (Guessing stage)

Deterministic (Verification Stage)

A Non-deterministic polynomial means, time efficiency of its verification stage is polynomial.

Definition: IV

Class NP is the class of decision problem that can be solved by non-deterministic polynomial algorithm. This class of problems is called non-deterministic polynomial.

$$P \subseteq NP$$

$P=NP$ imply many hundreds of difficult combinational decision problem can be solved by polynomial time

NP-Complete:

Many well-known decision problems known to be NP-Complete where $P=NP$ is more doubt. Any problem can be reduced to polynomial time.

Definition: V

A decision problem D1 is polynomially reducible to a decision problem D2. If there exists a function t, t transforms instances D1 to D2

Definition: VI

A decision problem D is said to be NP Complete if it belongs to class NP. Every problem in NP is polynomially reducible to p.

7. Explain the approximation algorithm for the travelling salesman problem (TSP) [Apr/May 2019]

- a. Nearest neighbor algorithm
- b. Twice –around the tree algorithm

Nearest Neighbor algorithm:

- i. Simply greedy method
- ii. Based on the nearest neighbor heuristic
- iii. Always go for nearest unvisited city

Algorithm:

Step1: Choose the arbitrary city as the start

Step2: Repeat all cities (unvisited)

Step3: Return to the starting city

It is difficult to solve the travelling salesman problem approximately.

However, there is a very important subset of instances called Euclidean, for which we can make a non-trivial assertion about the accuracy of the algorithm.

➤ Triangle inequality:

$$d[i,j] \leq d[i,k] + d[k,j] \text{ for any triple of cities}$$

➤ Symmetry:

$$d[i,j] = d[j,i] \text{ for any pair of}$$

cities I and j The accuracy ratio for any such instance with $n \geq 2$ cities.

$$f(S_a)/f(S^*) \leq 1/2 \lceil \log_2 n \rceil + 1$$

where $f(S_a)$ and $f(S^*)$ are the length of the nearest neighbour and shortest tour respectively.

Twice –around the tree algorithm:

Twice around the tree algorithm is an approximation algorithm for the TSP with Euclidean distances. Hamiltonian circuit & spanning tree.

✓ Polynomial time:

Within polynomial time twice around the tree can be solved by prim's or kruskal's algorithm.

- length of the tour S_a
- Optimal tour S^* , i.e. S_a twice of S^*
- $f(S_a) \leq 2 f(S^*)$ (Hamiltonian circuit)

- Removing a edge from S^* yields spanning tree T of weight $w(T)$ $w(T) > w(S^*)$
- $f(S^*) > w(T)/Tree \geq w(T^*)/Tree \geq f(S^*)/2 > w(T^*)$
- The Length of the walk \geq Length of the tour (Sa) $2f(S^*) > f(Sa)$

7. Outline the steps to find approximate solution to NP-Hard optimization problems using approximation algorithms with an example. [Nov/Dec 2019]

Given an optimization problem P , an algorithm A is said to be an approximation algorithm for P , if for any given instance I , it returns an approximate solution, that is a feasible solution.

P An optimization problem

A An approximation algorithm

I An instance of P

A^*

(I) Optimal value for the instance I

$A(I)$ Value for the instance I generated by A

1. Absolute approximation

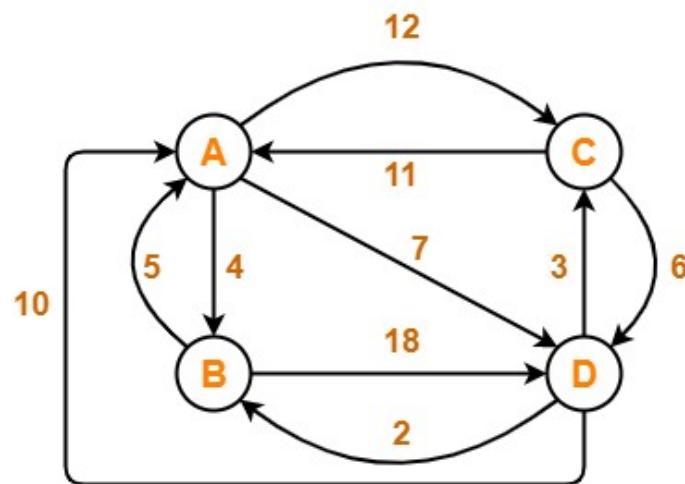
- A is an absolute approximation algorithm if there exists a constant k such that, for every instance I of P , $|A^*(I) - A(I)| \leq k$.
- For example, Planar graph coloring.

2. Relative approximation

- A is an relative approximation algorithm if there exists a constant k such that, for every instance I of P , $\max \{A^*(I)/A(I), A(I) / A^*(I)\} \leq k$.
- Vertex cover.

ASSIGNMENT QUESTIONS

- 1.** Solve Travelling Salesman Problem using Branch and Bound Algorithm in the following graph-



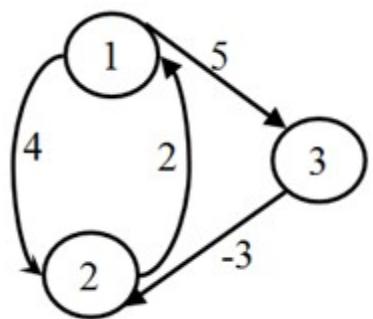
- 2.** Solve the following 0/1 Knapsack Problem using Branch & Bound Algorithm (both LC BB & FIFO BB)

[Hint: LC BB Method is explained in the Slide Uploaded. For FIFO Method the child nodes are explored in First in First out manner.] **Capacity =10**

ITEMS	Weights	Profits
A	2	\$40
B	3.14	\$50
C	1.98	\$100
D	5	\$95
E	3	\$30

- 3.** Solve the following sum of subset problem using backtracking: $w= \{1, 3, 4, 5\}$, $m=8$. Find the possible subsets of 'w' that sum to 'm'. [Draw state space tree to find the solution]

4. Find the **shortest paths** between all pairs of vertices in a graph given below



1. Optimal path is: A → C → D → B → A

Cost of Optimal path = 25 units

2. 1, 0, 1, 1, 0

3. 2 subsets

4. <https://sandynguyen.wordpress.com/2012/11/20/floyds-algorithm-all-pairs-shortest-path/>

CS8451 DESIGN AND ANALYSIS OF ALGORITHMS

CSE - SEMESTER 4
REG. 2017

UNIT I INTRODUCTION

1. Recursion is a method in which the solution of a problem depends on

- a) Larger instances of different problems
- b) Larger instances of the same problem
- c) Smaller instances of the same problem
- d) Smaller instances of different problems

Answer: c

Explanation: In recursion, the solution of a problem depends on the solution of smaller instances of the same problem.

2. Which of the following problems can't be solved using recursion?

- a) Factorial of a number
- b) Nth fibonacci number
- c) Length of a string
- d) Problems without base case

Answer: d

Explanation: Problems without base case leads to infinite recursion call. In general, we will assume a base case to avoid infinite recursion call. Problems like finding Factorial of a number, Nth Fibonacci number and

Length of a string can be solved using recursion.

3. Recursion is similar to which of the following?

- a) Switch Case
- b) Loop
- c) If-else
- d) if elif else

Answer: b

Explanation: Recursion is similar to a loop.

4. In recursion, the condition for which the function will stop calling itself is

- a) Best case
- b) Worst case
- c) Base case
- d) There is no such condition

Answer: c

Explanation: For recursion to end at some point, there always has to be a condition for which the function will not call itself. This condition is known as base case.

5. Consider the following code snippet:

```
void my_recursive_function()
{
    my_recursive_function();
}
int main()
{
    my_recursive_function();
    return 0;
}
```

What will happen when the above snippet is executed?

- a) The code will be executed successfully and no output will be generated
- b) The code will be executed successfully and random output will be generated
- c) The code will show a compile time error
- d) The code will run for some time and stop when the stack overflows

Answer: d

Explanation: Every function call is stored in the stack memory. In this case, there is no terminating condition(base case). So, my_recursive_function() will be called continuously till the stack overflows and there is no more space to store the function calls. At this point of time, the program will stop abruptly.

6. What is the output of the following code?

```
void my_recursive_function(int n)
{
    if(n == 0)
        return;
    printf("%d ",n);
    my_recursive_function(n-1);
}
int main()
{
    my_recursive_function(10);
    return 0;
}
```

- a) 10
- b) 1
- c) 10 9 8 ... 1 0
- d) 10 9 8 ... 1

Answer: d

Explanation: The program prints the numbers from 10 to 1.

7. What is the base case for the following code?

```
void my_recursive_function(int n)
{
    if(n == 0)
        return;
    printf("%d ",n);
    my_recursive_function(n-1);
}
int main()
{
    my_recursive_function(10);
    return 0;
}
```

- a) return
- b) printf("%d ", n)

- c) if(n == 0)
- d) my_recursive_function(n-1)

Answer: c

Explanation: For the base case, the recursive function is not called. So, “if(n == 0)” is the base case.

8. How many times is the recursive function called, when the following code is executed?

```
void my_recursive_function(int n)
{
    if(n == 0)
        return;
    printf("%d ",n);
    my_recursive_function(n-1);
}
int main()
{
    my_recursive_function(10);
    return 0;
}
```

- a) 9
- b) 10
- c) 11
- d) 12

Answer: c

Explanation: The recursive function is called 11 times.

9. What does the following recursive code do?

```
void my_recursive_function(int n)
{
    if(n == 0)
        return;
    my_recursive_function(n-1);
    printf("%d ",n);
}
int main()
{
    my_recursive_function(10);
    return 0;
}
```

- a) Prints the numbers from 10 to 1
- b) Prints the numbers from 10 to 0

- c) Prints the numbers from 1 to 10
- d) Prints the numbers from 0 to 10

Answer: c

Explanation: The above code prints the numbers from 1 to 10.

10. Which of the following statements is true?
- a) Recursion is always better than iteration
 - b) Recursion uses more memory compared to iteration
 - c) Recursion uses less memory compared to iteration
 - d) Iteration is always better and simpler than recursion

Answer: b

Explanation: Recursion uses more memory compared to iteration because every time the recursive function is called, the function call is stored in stack.

11. What will be the output of the following code?

```
int cnt=0;
void my_recursive_function(int n)
{
    if(n == 0)
        return;
    cnt++;
    my_recursive_function(n/10);
}
int main()
{
    my_recursive_function(123456789);
    printf("%d",cnt);
    return 0;
}
```

- a) 123456789
- b) 10
- c) 0
- d) 9

Answer: d

Explanation: The program prints the number of digits in the number 123456789, which is 9.

12. What will be the output of the following code?

```
void my_recursive_function(int n)
{
    if(n == 0)
    {
        printf("False");
        return;
    }
    if(n == 1)
    {
        printf("True");
        return;
    }
    if(n%2==0)
        my_recursive_function(n/2);
    else
    {
        printf("False");
        return;
    }
}
int main()
{
    my_recursive_function(100);
    return 0;
}
```

- a) True
- b) False

Answer: b

Explanation: The function checks if a number is a power of 2. Since 100 is not a power of 2, it prints false.

13. What is the output of the following code?

```
int cnt = 0;
void my_recursive_function(char *s, int i )
{
    if(s[i] == '\0')
        return;
    if(s[i] == 'a' || s[i] == 'e' || s[i]
    ] == 'i' || s[i] == 'o' || s[i] == 'u')
        cnt++;
    my_recursive_function(s,i+1);
}
int main()
{
    my_recursive_function("thisisrecursi
on",0);
```

```

    printf("%d", cnt);
    return 0;
}

```

- a) 6
- b) 9
- c) 5
- d) 10

Answer: a

Explanation: The function counts the number of vowels in a string. In this case the number of vowels is 6.

14. What is the output of the following code?

```

void my_recursive_function(int *arr, int
                           val, int idx, int len)
{
    if(idx == len)
    {
        printf("-1");
        return ;
    }
    if(arr[idx] == val)
    {
        printf("%d", idx);
        return;
    }
    my_recursive_function(arr, val, idx+1, l
en);
}
int main()
{
    int array[10] = {7, 6, 4, 3, 2, 1, 9
, 5, 0, 8};
    int value = 2;
    int len = 10;
    my_recursive_function(array, value,
0, len);
    return 0;
}

```

- a) 3
- b) 4
- c) 5
- d) 6

Answer: b

Explanation: The program searches for a value in the given array and prints the index at which the value is found. In this case, the program searches for value = 2. Since, the

index of 2 is 4(0 based indexing), the program prints 4.

1. In general, which of the following methods isn't used to find the factorial of a number?
 - a) Recursion
 - b) Iteration
 - c) Dynamic programming
 - d) Non iterative / recursive

Answer: d

Explanation: In general we use recursion, iteration and dynamic programming to find the factorial of a number. We can also implement without using iterative / recursive method by using tgammal() method. Most of us never use it generally.

2. Which of the following recursive formula can be used to find the factorial of a number?
 - a) $\text{fact}(n) = n * \text{fact}(n)$
 - b) $\text{fact}(n) = n * \text{fact}(n+1)$
 - c) $\text{fact}(n) = n * \text{fact}(n-1)$
 - d) $\text{fact}(n) = n * \text{fact}(1)$

Answer: c

Explanation: $\text{fact}(n) = n * \text{fact}(n - 1)$ can be used to find the factorial of a number.

3. Consider the following iterative implementation to find the factorial of a number:

```

int main()
{
    int n = 6, i;
    int fact = 1;
    for(i=1;i<=n;i++)
    {
        _____;
        printf("%d", fact);
        return 0;
    }
}

```

Which of the following lines should be inserted to complete the above code?

- a) $\text{fact} = \text{fact} + i$
- b) $\text{fact} = \text{fact} * i$
- c) $i = i * \text{fact}$
- d) $i = i + \text{fact}$

Answer: b

Explanation: The line “fact = fact * i” should be inserted to complete the above code.

4. Consider the following recursive implementation to find the factorial of a number:

```
int fact(int n)
{
    if(_____)
        return 1;
    return n * fact(n - 1);
}

int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) $n = 0$
- b) $n \neq 0$
- c) $n == 0$
- d) $n == 1$

Answer: c

Explanation: The line “ $n == 0$ ” should be inserted to complete the above code.

Note: “ $n == 1$ ” cannot be used because it does not take care of the case when $n = 0$, i.e when we want to find the factorial of 0.

5. The time complexity of the following recursive implementation to find the factorial of a number is _____

```
int fact(int n)
{
    if(_____)
        return 1;
    return n * fact(n - 1);
}

int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
}
```

```
    return 0;
}
```

- a) $O(1)$
- b) $O(n)$
- c) $O(n^2)$
- d) $O(n^3)$

Answer: b

Explanation: The time complexity of the above recursive implementation to find the factorial of a number is $O(n)$.

6. What is the space complexity of the following recursive implementation to find the factorial of a number?

```
int fact(int n)
{
    if(_____)
        return 1;
    return n * fact(n - 1);
}

int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

- a) $O(1)$
- b) $O(n)$
- c) $O(n^2)$
- d) $O(n^3)$

Answer: a

Explanation: The space complexity of the above recursive implementation to find the factorial of a number is $O(1)$.

7. Consider the following recursive implementation to find the factorial of a number:

```
int fact(int n)
{
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}
```

```
int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines is the base case?

- a) return 1
- b) return n * fact(n-1)
- c) if(n == 0)
- d) if(n == 1)

Answer: c

Explanation: The line “if(n == 0)” is the base case.

8. What is the output of the following code?

```
int fact(int n)
{
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}

int main()
{
    int n = 0;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

- a) 0
- b) 1
- c) 2
- d) 3

Answer: b

Explanation: The program prints 0!, which is 1.

9. What is the output of the following code?

```
int fact(int n)
{
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}

int main()
{
```

```
    int n = 1;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

- a) 0
- b) 1
- c) 2
- d) 3

Answer: b

Explanation: The program prints 1!, which is 1.

10. How many times will the function fact() be called when the following code is executed?

```
int fact(int n)
{
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}

int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

- a) 4
- b) 5
- c) 6
- d) 7

Answer: c

Explanation: The fact() function will be called 6 times with the following arguments: fact(5), fact(4), fact(3), fact(2), fact(1), fact(0).

11. What is the output of the following code?

```
int fact(int n)
{
    if(n == 0)
        return 1;
    return n * fact(n - 1);
}
```

```
int main()
{
    int n = 5;
    int ans = fact(n);
    printf("%d",ans);
    return 0;
}
```

- a) 24
- b) 120
- c) 720
- d) 1

Answer: b

Explanation: The function prints 5!, which is 120.

1. Suppose the first fibonnaci number is 0 and the second is 1. What is the sixth fibonnaci number?

- a) 5
- b) 6
- c) 7
- d) 8

Answer: a

Explanation: The sixth fibonnaci number is 5.

2. Which of the following is not a fibonnaci number?

- a) 8
- b) 21
- c) 55
- d) 14

Answer: d

Explanation: 14 is not a fibonnaci number.

3. Which of the following option is wrong?

- a) Fibonacci number can be calculated by using Dynamic programming
- b) Fibonacci number can be calculated by using Recursion method
- c) Fibonacci number can be calculated by using Iteration method
- d) No method is defined to calculate Fibonacci number

Answer: d

Explanation: Fibonacci number can be calculated by using Dynamic Programming, Recursion method, Iteration Method.

4. Consider the following iterative implementation to find the nth fibonacci number?

```
int main()
{
    int n = 10,i;
    if(n == 1)
        printf("0");
    else if(n == 2)
        printf("1");
    else
    {
        int a = 0, b = 1, c;
        for(i = 3; i <= n; i++)
        {
            c = a + b;
            _____;
            _____;
        }
        printf("%d",c);
    }
    return 0;
}
```

Which of the following lines should be added to complete the above code?

- a)

c = b

b = a

- b)

a = b

b = c

- c)

b = c

a = b

- d)

a = b

```
b = a
```

Answer: b

Explanation: The lines “ $a = b$ ” and “ $b = c$ ” should be added to complete the above code.

5. Which of the following recurrence relations can be used to find the nth fibonacci number?

- a) $F(n) = F(n) + F(n - 1)$
- b) $F(n) = F(n) + F(n + 1)$
- c) $F(n) = F(n - 1)$
- d) $F(n) = F(n - 1) + F(n - 2)$

Answer: d

Explanation: The relation $F(n) = F(n - 1) + F(n - 2)$ can be used to find the nth fibonacci number.

6. Consider the following recursive implementation to find the nth fibonacci number:

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return _____;
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) $fibo(n - 1)$
- b) $fibo(n - 1) + fibo(n - 2)$
- c) $fibo(n) + fibo(n - 1)$
- d) $fibo(n - 2) + fibo(n - 1)$

Answer: b

Explanation: The line $fibo(n - 1) + fibo(n -$

2) should be inserted to complete the above code.

7. Consider the following recursive implementation to find the nth fibonnaci number:

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following is the base case?

- a) $if(n == 1)$
- b) $else if(n == 2)$
- c) $return fibo(n - 1) + fibo(n - 2)$
- d) both $if(n == 1)$ and $else if(n == 2)$

Answer: d

Explanation: Both $if(n == 1)$ and $else if(n == 2)$ are the base cases.

8. What is the time complexity of the following recursive implementation to find the nth fibonacci number?

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

- a) O(1)
- b) O(2^n)
- c) O(n^2)
- d) O(2^n)

Answer: d

Explanation: The time complexity of the above recursive implementation to find the nth fibonacci number is O(2^n).

9. What is the space complexity of the following recursive implementation to find the nth fibonacci number?

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

- a) O(1)
- b) O(2^n)
- c) O(n^2)
- d) O(2^n)

Answer: a

Explanation: The space complexity of the above recursive implementation to find the nth fibonacci number is O(1).

10. What is the output of the following code?

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
```

```
{
    int n = -1;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

- a) 0
- b) 1
- c) Compile time error
- d) Runtime error

Answer: d

Explanation: Since negative numbers are not handled by the program, the function fibo() will be called infinite times and the program will produce a runtime error when the stack overflows.

11. What is the output of the following code?

```
int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}
```

- a) 1
- b) 2
- c) 3
- d) 5

Answer: c

Explanation: The program prints the 5th fibonacci number, which is 3.

12. How many times will the function fibo() be called when the following code is executed?

```
int fibo(int n)
{
```

```

    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 5;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}

```

- a) 5
- b) 6
- c) 8
- d) 9

Answer: d

Explanation: The function fibo() will be called 9 times, when the above code is executed.

13. What is the output of the following code?

```

int fibo(int n)
{
    if(n == 1)
        return 0;
    else if(n == 2)
        return 1;
    return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n = 10;
    int ans = fibo(n);
    printf("%d",ans);
    return 0;
}

```

- a) 21
- b) 34
- c) 55
- d) 13

Answer: b

Explanation: The program prints the 10th fibonacci number, which is 34.

1. Which of the following option is wrong about natural numbers?

- a) Sum of first n natural numbers can be calculated by using Iteration method
- b) Sum of first n natural numbers can be calculated by using Recursion method
- c) Sum of first n natural numbers can be calculated by using Binomial coefficient method
- d) No method is prescribed to calculate sum of first n natural number

Answer: d

Explanation: All of the above mentioned methods can be used to find the sum of first n natural numbers.

2. Which of the following gives the sum of the first n natural numbers?

- a) $nC2$
- b) $(n-1)C2$
- c) $(n+1)C2$
- d) $(n+2)C2$

Answer: c

Explanation: The sum of first n natural numbers is given by $n*(n+1)/2$, which is equal to $(n+1)C2$.

3. Consider the following iterative solution to find the sum of first n natural numbers:

```

#include<stdio.h>
int get_sum(int n)
{
    int sm = 0, i;
    for(i = 1; i <= n; i++)
        _____;
    return sm;
}
int main()
{
    int n = 10;
    int ans = get_sum(n);
    printf("%d",ans);
    return 0;
}

```

Which of the following lines completes the above code?

- a) $sm = i$

- b) $sm += i$
- c) $i = sm$
- d) $i += sm$

Answer: b

Explanation: The line “ $sm += i$ ” completes the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int get_sum(int n)
{
    int sm, i;
    for(i = 1; i <= n; i++)
        sm += i;
    return sm;
}

int main()
{
    int n = 10;
    int ans = get_sum(n);
    printf("%d",ans);
    return 0;
}
```

- a) 55
- b) 45
- c) 35
- d) Depends on compiler

Answer: d

Explanation: Since the variable “ sm ” is not initialized to 0, it will produce a garbage value. Some compiler will automatically initialise variables to 0 if not initialised. In that case the value is 55. Hence the value depends on the compiler.

5. What is the time complexity of the following iterative method used to find the sum of the first n natural numbers?

```
#include<stdio.h>
int get_sum(int n)
{
    int sm, i;
    for(i = 1; i <= n; i++)
        sm += i;
    return sm;
}

int main()
```

```
{
    int n = 10;
    int ans = get_sum(n);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O(n)
c) O( $n^2$ )
d) O( $n^3$ )
```

Answer: b

Explanation: The time complexity of the above iterative method used to find the sum of first n natural numbers is $O(n)$.

6. Consider the following code:

```
#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return _____;
}

int main()
{
    int n = 5;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines is the recurrence relation for the above code?

- a) $(n - 1) + \text{recursive_sum}(n)$
- b) $n + \text{recursive_sum}(n)$
- c) $n + \text{recursive_sum}(n - 1)$
- d) $(n - 1) + \text{recursive_sum}(n - 1)$

Answer: c

Explanation: The recurrence relation for the above code is: $n + \text{recursive_sum}(n - 1)$.

7. Consider the following code:

```
#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
```

```

        return n + recursive_sum(n - 1);
}
int main()
{
    int n = 5;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}

```

Which of the following is the base case for the above recursive code?

- a) if($n == 0$)
- b) return 0
- c) return $n + \text{recursive_sum}(n - 1)$
- d) if($n == 1$)

Answer: a

Explanation: “if($n == 0$)” is the base case for the above recursive code.

8. What is the time complexity of the following recursive implementation used to find the sum of the first n natural numbers?

```

#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return n + recursive_sum(n - 1);
}
int main()
{
    int n = 5;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O( $n$ )
c) O( $n^2$ )
d) O( $n^3$ )

```

Answer: b

Explanation: The time complexity of the above recursive implementation used to find the sum of first n natural numbers is $O(n)$.

9. Which of the following methods used to find the sum of first n natural numbers has the

least time complexity?

- a) Recursion
- b) Iteration
- c) Binomial coefficient
- d) All have equal time complexity

Answer: c

Explanation: Recursion and iteration take $O(n)$ time to find the sum of first n natural numbers while binomial coefficient takes $O(1)$ time.

10. What is the output of the following code?

```

#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return n + recursive_sum(n - 1);
}
int main()
{
    int n = 5;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}

a) 10
b) 15
c) 21
d) 14

```

Answer: b

Explanation: The above code prints the sum of first 5 natural numbers, which is 15.

11. How many times is the function `recursive_sum()` called when the following code is executed?

```

#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return n + recursive_sum(n - 1);
}
int main()
{
    int n = 5;

```

```

int ans = recursive_sum(n);
printf("%d",ans);
return 0;
}

```

- a) 4
- b) 5
- c) 6
- d) 7

Answer: c

Explanation: The function recursive_sum is called 6 times when the following code is executed.

12. What is the output of the following code?

```

#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return n + recursive_sum(n - 1);
}
int main()
{
    int n = 0;
    int ans = recursive_sum(n);
    printf("%d",ans);
    return 0;
}

```

- a) -1
- b) 0
- c) 1
- d) runtime error

Answer: b

Explanation: The program prints the sum of first 0 natural numbers, which is 0.

13. What is the output of the following code?

```

#include<stdio.h>
int recursive_sum(int n)
{
    if(n == 0)
        return 0;
    return n + recursive_sum(n - 1);
}
int main()
{
    int n = -4;

```

```

int ans = recursive_sum(n);
printf("%d",ans);
return 0;
}

```

- a) 0
- b) -10
- c) 1
- d) runtime error

Answer: d

Explanation: The above code doesn't handle the case of negative numbers and so the function recursive_sum() will be called again and again till the stack overflows and the program produces a runtime error.

1. Which of the following is not another name for GCD(Greatest Common Divisor)?

- a) LCM
- b) GCM
- c) GCF
- d) HCF

Answer: a

Explanation: : LCM (Least Common Multiple) and GCD are not same. GCM (Greatest Common Measure), GCF (Greatest Common Factor), HCF (Highest Common Factor) are other names for GCD.

2. What is the GCD of 8 and 12?

- a) 8
- b) 12
- c) 2
- d) 4

Answer: d

Explanation: GCD is largest positive integer that divides each of the integer. So the GCD of 8 and 12 is 4.

3. If GCD of two number is 8 and LCM is 144, then what is the second number if first number is 72?

- a) 24
- b) 2

- c) 3
d) 16

Answer: d

Explanation: As $A * B = \text{GCD}(A, B) * \text{LCM}(A, B)$. So $B = (144 * 8)/72 = 16$.

4. Which of the following is also known as GCD?

- a) Highest Common Divisor
- b) Highest Common Multiple
- c) Highest Common Measure
- d) Lowest Common Multiple

Answer: a

Explanation: GCM (Greatest Common Measure), GCF (Greatest Common Factor), HCF (Highest Common Factor) and HCF (Highest Common Divisor) are also known as GCD.

5. Which of the following is coprime number?

- a) 54 and 24
- b) 4 and 8
- c) 6 and 12
- d) 9 and 28

Answer: d

Explanation: Coprime numbers have GCD 1. So 9 and 28 are coprime numbers. While 54 and 24 have GCD 6, 4 and 8 have GCD 4, 6 and 12 have GCD 6.

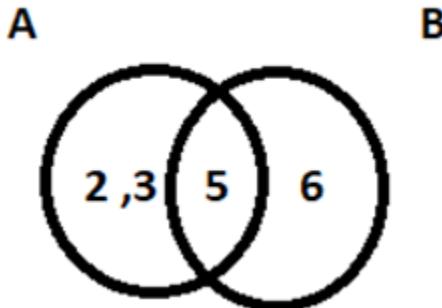
6. In terms of Venn Diagram, which of the following expression gives GCD (Given $A \cap B \neq \emptyset$)?

- a) Multiplication of $A \cup B$ terms
- b) Multiplication of $A \cap B$ terms
- c) Multiplication of $A * B$ terms
- d) Multiplication of $A - B$ terms

Answer: b

Explanation: In terms of Venn Diagram, the GCD is given by the intersection of two sets. So $A \cap B$ gives the GCD. While $A \cup B$ gives the LCM.

7. What is the GCD according to the given Venn Diagram?



- a) 2
- b) 3
- c) 5
- d) 6

Answer: c

Explanation: In terms of Venn Diagram, the GCD is given by the intersection of two sets. So $A \cap B$ gives the GCD. While $A \cup B$ gives the LCM. So here $A \cap B$ is 5.

8. What is the GCD of a and b?

- a) $a + b$
- b) $\text{gcd}(a-b, b)$ if $a > b$
- c) $\text{gcd}(a+b, a-b)$
- d) $a - b$

Answer: b

Explanation: As per Euclid's Algorithm, $\text{gcd}(a, b) = \text{gcd}(a-b, b)$ if $a > b$ or $\text{gcd}(a, b) = \text{gcd}(a, b-a)$ if $b > a$.

9. What is the GCD of 48, 18, 0?

- a) 24
- b) 2
- c) 3
- d) 6

Answer: d

Explanation: GCD is largest positive integer that divides each of the integers. So the GCD of 48, 18, 0 is 6.

10. Is 9 and 28 coprime number?

- a) True
- b) False

Answer: a

Explanation: Coprime numbers have GCD 1. So 9 and 28 are coprime numbers.

11. If gcd (a, b) is defined by the expression, $d=a*p + b*q$ where d, p, q are positive integers and a, b is both not zero, then what is the expression called?

- a) Bezout's Identity
- b) Multiplicative Identity
- c) Sum of Product
- d) Product of Sum

Answer: a

Explanation: If gcd (a, b) is defined by the expression, $d=a*p + b*q$ where d, p, q are positive integers and a, b is both not zero, then the expression is called Bezout's Identity and p, q can be calculated by extended form of Euclidean algorithm.

12. Is gcd an associative function.

- a) True
- b) False

Answer: a

Explanation: The gcd function is an associative function as $\text{gcd} (a, \text{gcd} (b, c)) = \text{gcd} (\text{gcd} (a, b), c)$.

13. Which is the correct term of the given relation, $\text{gcd} (a, b) * \text{lcm} (a, b) = ?$

- a) $|a*b|$
- b) $a + b$
- c) $a - b$
- d) a / b

Answer: a

Explanation: The gcd is closely related to lcm as $\text{gcd} (a, b) * \text{lcm} (a, b) = |a*b|$.

14. Who gave the expression for the probability and expected value of gcd?

- a) James E. Nymann
- b) Riemann
- c) Thomas
- d) Euler

Answer: a

Explanation: In the year 1972, James E. Nymann showed some result to show the probability and expected value of gcd.

15. What is the computational complexity of Binary GCD algorithm where a and b are integers?

- a) $O (\log a + \log b)^2$
- b) $O (\log (a + b))$
- c) $O (\log ab)$
- d) $O (\log a-b)$

Answer: a

Explanation: From the Binary GCD algorithm, it is found that the computational complexity is $O (\log a + \log b)^2$ as the total number of steps in the execution is at most the total sum of number of bits of a and b.

1. LCM is also called as _____

- a) GCD
- b) SCM
- c) GCF
- d) HCF

Answer: b

Explanation: GCD (Greatest Common Divisor), GCF (Greatest Common Factor), HCF (Highest Common Factor) is not an alias for LCM. LCM is also called as Smallest Common Multiple(SCM).

2. What is the LCM of 8 and 13?

- a) 8
- b) 12
- c) 20
- d) 104

Answer: d

Explanation: 104 is the smallest positive integer that is divisible by both 8 and 12.

3. Which is the smallest number of 3 digits that is divisible by 2, 4, 8?

- a) 100
- b) 102

- c) 116
d) 104

Answer: d

Explanation: LCM of 2, 4, 8 is 8. So check for the number that is divisible by 8. So 104 is the smallest number that is divisible by 2, 4, 8.

4. Which of the following is also known as LCM?

- a) Lowest Common Divisor
b) Least Common Multiple
c) Lowest Common Measure
d) Highest Common Multiple

Answer: a

Explanation: Least Common Multiple is also known as LCM or Lowest Common Multiple.

5. What is the LCM of two coprime numbers?

- a) 1
b) 0
c) Addition of two coprime numbers
d) Multiplication of two coprime numbers

Answer: d

Explanation: Coprime numbers have GCD 1. While LCM of coprime numbers is the product of those two coprime numbers.

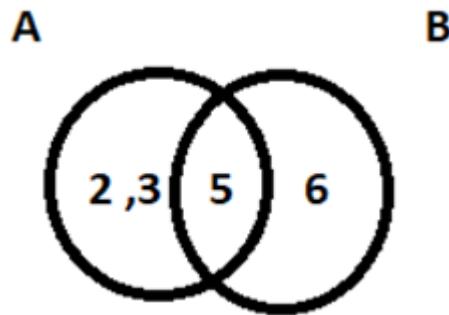
6. In terms of Venn Diagram, which of the following expression gives LCM (Given $A \cap B \neq \emptyset$)?

- a) Multiplication of $A \cup B$ terms
b) Multiplication of $A \cap B$ terms
c) Multiplication of $A * B$ terms
d) Multiplication of $A - B$ terms

Answer: a

Explanation: In terms of Venn Diagram, the LCM is given by the Union of two sets. So $A \cup B$ gives the LCM. While $A \cap B$ gives the GCD.

7. What is the LCM according to the given Venn Diagram?



- a) 2
b) 3
c) 180
d) 6

Answer: c

Explanation: In terms of Venn Diagram, the LCM is given by the Union of two sets. So $A \cup B$ gives the LCM. So product of all the terms is 180.

8. What is the lcm (a, b)?

- a) $a + b$
b) $\text{gcd}(a-b, b)$ if $a > b$
c) $\text{lcm}(b, a)$
d) $a - b$

Answer: c

Explanation: Since the LCM function is commutative, so $\text{lcm}(a, b) = \text{lcm}(b, a)$.

9. What is the LCM of 48, 18, 6?

- a) 12^2
b) $12 * 2$
c) 3
d) 6

Answer: a

Explanation: The LCM of 48, 18, 6 is 144 and 12^2 is 144.

10. Is 9 and 28 coprime number.

- a) True
b) False

Answer: a

Explanation: Coprime numbers have GCD 1

and LCM is the product of the two given terms. So 9 and 28 are coprime numbers.

11. What is the following expression, $\text{lcm}(a, \text{lcm}(b, c))$ equal to?

- a) $\text{lcm}(a, b, c)$
- b) a^*b^*c
- c) $a + b + c$
- d) $\text{lcm}(\text{lcm}(a, b), c)$

Answer: d

Explanation: Since LCM function follows associativity, hence $\text{lcm}(a, \text{lcm}(b, c))$ is equal to $\text{lcm}(\text{lcm}(a, b), c)$.

12. Is lcm an associative function.

- a) True
- b) False

Answer: a

Explanation: The lcm function is an associative function as $\text{lcm}(a, \text{lcm}(b, c))$ is equal to $\text{lcm}(\text{lcm}(a, b), c)$.

13. Which is the correct term of the given relation, $\text{lcm}(a, b) * \text{gcd}(a, b) = ?$

- a) $|a^*b|$
- b) $a + b$
- c) $a - b$
- d) a / b

Answer: a

Explanation: The lcm is closely related to gcd as $\text{lcm}(a, b) * \text{gcd}(a, b) = |a^*b|$.

14. What is the following expression, $\text{lcm}(a, \text{gcd}(a, b))$ equal to?

- a) a
- b) b
- c) a^*b
- d) $a + b$

Answer: a

Explanation: Since the lcm function follows absorption laws so $\text{lcm}(a, \text{gcd}(a, b))$ equal to a.

15. Which algorithm is the most efficient numerical algorithm to obtain lcm?

- a) Euler's Algorithm
- b) Euclid's Algorithm
- c) Chebyshev Function
- d) Partial Division Algorithm

Answer: b

Explanation: The most efficient way of calculating the lcm of a given number is using Euclid's algorithm which computes the lcm in much lesser time compared to other algorithms.

1. Which of the following methods can be used to find the sum of digits of a number?

- a) Recursion
- b) Iteration
- c) Greedy algorithm
- d) Both recursion and iteration

Answer: d

Explanation: Both recursion and iteration can be used to find the sum of digits of a number.

2. What can be the maximum sum of digits for a 4 digit number?

- a) 1
- b) 16
- c) 36
- d) 26

Answer: c

Explanation: The sum of digits will be maximum when all the digits are 9. Thus, the sum will be maximum for the number 9999, which is 36.

3. What can be the minimum sum of digits for a 4 digit number?

- a) 0
- b) 1
- c) 16
- d) 36

Answer: b

Explanation: The sum of digits will be minimum for the number 1000 and the sum is 1.

4. Consider the following iterative implementation to find the sum of digits of a number:

```
#include<stdio.h>
int sum_of_digits(int n)
{
    int sm = 0;
    while(n != 0)
    {
        _____;
        n /= 10;
    }
    return sm;
}
int main()
{
    int n = 1234;
    int ans = sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) $sm += n$
- b) $sm += n \% 10$
- c) $sm += n - 10$
- d) $sm += n / 10$

Answer: b

Explanation: The line “ $sm += n \% 10$ ” adds the last digit(LSB) of the number to the current sum. Thus, the line “ $sm += n \% 10$ ” should be added to complete the above code.

5. What is the output of the following code?

```
#include<stdio.h>
int sum_of_digits(int n)
{
    int sm = 0;
    while(n != 0)
    {
        sm += n \% 10;
        n /= 10;
    }
    return sm;
```

```
}
```

```
int main()
{
    int n = 1234;
    int ans = sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

- a) 1
- b) 3
- c) 7
- d) 10

Answer: d

Explanation: The above code prints the sum of digits of the number 1234, which is 10.

6. Consider the following recursive implementation to find the sum of digits of a number:

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return _____;
}
int main()
{
    int n = 1201;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) $(n / 10) + \text{recursive_sum_of_digits}(n \% 10)$
- b) $(n) + \text{recursive_sum_of_digits}(n \% 10)$
- c) $(n \% 10) + \text{recursive_sum_of_digits}(n / 10)$
- d) $(n \% 10) + \text{recursive_sum_of_digits}(n \% 10)$

Answer: c

Explanation: The line “ $(n \% 10) + \text{recursive_sum_of_digits}(n / 10)$ ” should be inserted to complete the above code.

7. What is the time complexity of the following recursive implementation to find the sum of digits of a number n?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return _____;
}
int main()
{
    int n = 1201;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

- a) O(n)
- b) O(1)
- c) O(len(n)), where len(n) is the number of digits in n
- d) O(1/2)

Answer: c

Explanation: The time complexity of the above recursive implementation to find the sum of digits of a number is O(len(n)).

8. What is the output of the following code?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
    int n = 1234321;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

- a) 10
- b) 16

- c) 15
- d) 14

Answer: b

Explanation: The above code prints the sum of digits of the number 1234321, which is 16.

9. How many times is the function recursive_sum_of_digits() called when the following code is executed?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
    int n = 1201;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}
```

- a) 6
- b) 7
- c) 5
- d) 9

Answer: c

Explanation: The function recursive_sum_of_digits() is called 8 times, when the following code is executed.

10. You have to find the sum of digits of a number given that the number is always greater than 0. Which of the following base cases can replace the base case for the below code?

```
#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return n % 10 + recursive_sum_of_digits(n/10);
}
```

```

int main()
{
    int n = 1201;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}

```

- a) if(n == 0) return 1
- b) if(n == 1) return 0
- c) if(n == 1) return 1
- d) no need to modify the base case

Answer: d

Explanation: None of the above mentioned base cases can replace the base case if(n == 0) return 0.

11. What is the output of the following code?

```

#include<stdio.h>
int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return n % 10 + recursive_sum_of_digits(n/10);
}
int main()
{
    int n = 10000;
    int ans = recursive_sum_of_digits(n);
    printf("%d",ans);
    return 0;
}

```

- a) 0
- b) 1
- c) runtime error
- d) -1

Answer: b

Explanation: The program prints the sum of digits of the number 10000, which is 1.

12. What is the output of the following code?

```

#include<stdio.h>
int cnt =0;
int my_function(int n, int sm)
{

```

```

    int i, tmp_sm;
    for(i=1;i<=n;i++)
    {
        tmp_sm = recursive_sum_of_digits(i);
        if(tmp_sm == sm)
            cnt++;
    }
    return cnt;
}

int recursive_sum_of_digits(int n)
{
    if(n == 0)
        return 0;
    return n % 10 + recursive_sum_of_digits(n/10);
}

int main()
{
    int n = 20, sum = 3;
    int ans = my_function(n,sum);
    printf("%d",ans);
    return 0;
}

a) 0
b) 1
c) 2
d) 3

```

Answer: c

Explanation: The code prints the count of numbers between 1 and 20 such that the sum of their digits is 3. There are only two such numbers: 3 and 12.

1. Consider the following iterative implementation used to reverse a string:

```

#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
    int len = strlen(s);
    int i,j;
    i=0;
    j=len-1;
    while(_____)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
    }
}

```

```

        j--;
    }
}

```

Which of the following lines should be inserted to complete the above code?

- a) $i > j$
- b) $i < len$
- c) $j > 0$
- d) $i < j$

Answer: d

Explanation: The line “ $i < j$ ” should be inserted to complete the above code.

2. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
    int len = strlen(s);
    int i,j;
    i=0;
    j=len-1;
    while(i < j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
        j--;
    }
}
int main()
{
    char s[100] = "reverse";
    reverse_string(s);
    printf("%s",s);
    return 0;
}

```

- a) ersevre
- b) esrever
- c) eserver
- d) eresevr

Answer: b

Explanation: The program reverses the string “reverse” and prints “esrever”.

3. What is the time complexity of the above code used to reverse a string?

- a) $O(1)$
- b) $O(n)$
- c) $O(n^2)$
- d) $O(n^3)$

Answer: b

Explanation: The time complexity of the above code used to reverse a string is $O(n)$.

4. What does the following code do?

```

#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
    int len = strlen(s);
    int i,j;
    i=0;
    j=len-1;
    while(i < j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
        j--;
    }
}
int main()
{
    char s[100] = "abcdefg";
    char t[100];
    strcpy(t,s);
    reverse_string(s);
    if(strcmp(t,s) == 0)
        printf("Yes");
    else
        printf("No");
    return 0;
}

```

- a) Copies a string to another string
- b) Compares two strings
- c) Reverses a string
- d) Checks if a string is a palindrome

Answer: d

Explanation: The main purpose of the above code is to check if a string is a palindrome.

5. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
void reverse_string(char *s)
{
    int len = strlen(s);
    int i,j;
    i=0;
    j=len-1;
    while(i < j)
    {
        char tmp = s[i];
        s[i] = s[j];
        s[j] = tmp;
        i++;
        j--;
    }
}
int main()
{
    char s[100] = "rotator";
    char t[100];
    strcpy(t,s);
    reverse_string(s);
    if(strcmp(t,s) == 0)
        printf("Yes");
    else
        printf("No");
    return 0;
}
```

- a) Yes
- b) No
- c) Runtime error
- d) Compile time error

Answer: a

Explanation: The program checks if a string is a palindrome. Since the string rotator is a palindrome, it prints “yes”.

6. Consider the following recursive implementation used to reverse a string:

```
void recursive_reverse_string(char *s, int left, int right)
{
    if(left < right)
    {
        char tmp = s[left];
        s[left] = s[right];
        s[right] = tmp;
        _____;
    }
}
```

Which of the following lines should be inserted to complete the above code?

- a) recursive_reverse_string(s, left+1, right+1)
- b) recursive_reverse_string(s, left-1, right-1)
- c) recursive_reverse_string(s, left+1, right-1)
- d) recursive_reverse_string(s, left-1, right+1)

Answer: c

Explanation: The line “recursive_reverse_string(s, left+1, right-1)” should be inserted to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
void recursive_reverse_string(char *s, int left, int right)
{
    if(left < right)
    {
        char tmp = s[left];
        s[left] = s[right];
        s[right] = tmp;
        recursive_reverse_string(s, left+1, right-1);
    }
}
int main()
{
    char s[100] = "recursion";
    int len = strlen(s);
    recursive_reverse_string(s,0,len-1);
    printf("%s",s);
    return 0;
}
```

- a) recursion
- b) nsoirucer
- c) noisrcuer
- d) noisrucer

Answer: d

Explanation: The program prints the reversed string of “recursion”, which is “noisrucer”.

8. How many times is the function recursive_reverse_string() called when the following code is executed?

```
#include<stdio.h>
#include<string.h>
void recursive_reverse_string(char *s, int left, int right)
{
    if(left < right)
    {
        char tmp = s[left];
        s[left] = s[right];
        s[right] = tmp;
        recursive_reverse_string(s, left+1, right-1);
    }
}
int main()
{
    char s[100] = "madam";
    int len = strlen(s);
    recursive_reverse_string(s, 0, len-1);
    printf("%s", s);
    return 0;
}

a) 3
b) 4
c) 5
d) 6
```

Answer: a

Explanation: The function recursive_reverse_string() is called 3 times when the above code is executed.

9. What is the time complexity of the above recursive implementation used to reverse a string?
- O(1)
 - O(n)
 - O(n^2)
 - O(n^3)

Answer: b

Explanation: The time complexity of the above recursive implementation used to reverse a string is O(n).

10. In which of the following cases is the reversal of a string not equal to the original string?
- Palindromic strings
 - Strings of length 1

- c) Empty String
d) Strings of length 2

Answer: d

Explanation: String “ab” is a string of length 2 whose reversal is not the same as the given one. Palindromic Strings, String of length 1 and Empty Strings case – the reversal is the same as the one given.

1. Which of the following is the binary representation of 100?

- 1010010
- 1110000
- 1100100
- 1010101

Answer: c

Explanation: $100 = 64 + 32 + 4 = 2^6 + 2^5 + 2^2 = 1100100$.

2. Consider the following iterative code used to convert a decimal number to its equivalent binary:

```
#include<stdio.h>
void dec_to_bin(int n)
{
    int arr[31], len = 0, i;
    if(n == 0)
    {
        arr[0] = 0;
        len = 1;
    }
    while(n != 0)
    {
        arr[len++] = n % 2;
        _____;
    }
    for(i=len-1; i>=0; i--)
        printf("%d", arr[i]);
}
int main()
{
    int n = 10;
    dec_to_bin(n);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) n-
- b) n /= 2
- c) n /= 10
- d) n++

Answer: b

Explanation: The line “n /= 2” should be inserted to complete the above code.

3. What is the output of the following code?

```
#include<stdio.h>
void dec_to_bin(int n)
{
    int arr[31],len = 0,i;
    if(n == 0)
    {
        arr[0] = 0;
        len = 1;
    }
    while(n != 0)
    {
        arr[len++] = n % 2;
        n /= 2;
    }
    for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
    int n = 63;
    dec_to_bin(n);
    return 0;
}
```

- a) 111111
- b) 111011
- c) 101101
- d) 101010

Answer: a

Explanation: The program prints the binary equivalent of 63, which is 111111.

4. What is the output of the following code?

```
#include<stdio.h>
void dec_to_bin(int n)
{
    int arr[31],len = 0,i;
    if(n == 0)
```

```
{
    arr[0] = 0;
    len = 1;
}
while(n != 0)
{
    arr[len++] = n % 2;
    n /= 2;
}
for(i=len-1; i>=0; i--)
    printf("%d",arr[i]);
}
```

```
int main()
{
    int n = 0;
    dec_to_bin(n);
    return 0;
}
```

- a) 0
- b) 1
- c) Runtime error
- d) Garbage value

Answer: a

Explanation: The program prints the binary equivalent of 0, which is 0.

5. What is the time complexity of the following code used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
void dec_to_bin(int n)
{
    int arr[31],len = 0,i;
    if(n == 0)
    {
        arr[0] = 0;
        len = 1;
    }
    while(n != 0)
    {
        arr[len++] = n % 2;
        n /= 2;
    }
    for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
}
int main()
{
    int n = 0;
    dec_to_bin(n);
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(logn)

Answer: d

Explanation: The time complexity of the above code used to convert a decimal number to its binary equivalent is O(logn).

6. Consider the following recursive implementation used to convert a decimal number to its binary equivalent:

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    _____;
    recursive_dec_to_bin(n/2);
}
int main()
{
    int n = 100,i;
    recursive_dec_to_bin(n);
    for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) arr[len] = n
- b) arr[len] = n % 2
- c) arr[len++] = n % 2
- d) arr[len++] = n

Answer: c

Explanation: The line “arr[len++] = n % 2” should be inserted to complete the above code.

7. Consider the following code:

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
```

Which of the following lines is the base case for the above code?

- a) if(n == 0 && len == 0)
- b) if(n == 0)
- c) if(n == 0 && len == 0) and if(n == 0)
- d) if(n == 1)

Answer: c

Explanation: Both of the above mentioned lines are the base cases for the above code.

8. What is the output of the following code?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
int main()
{
    int n = -100,i;
    recursive_dec_to_bin(n);
    for(i=len-1; i>=0; i--)
        printf("%d",arr[i]);
    return 0;
}
```

- a) -1100100
- b) 1100100

- c) 2's complement of 1100100
d) Garbage value

Answer: d

Explanation: The program doesn't handle negative inputs and so produces a garbage value.

9. What is the time complexity of the recursive implementation used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: d

Explanation: The time complexity of the recursive implementation used to convert a decimal number to its binary equivalent is O(logn).

10. What is the space complexity of the recursive implementation used to convert a decimal number to its binary equivalent?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
```

```
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
```

- a) O(1)
b) O(n)
c) O(n^2)
d) O(logn)

Answer: d

Explanation: The space complexity of the recursive implementation used to convert a decimal number to its binary equivalent is O(logn).

11. What is the output of the following code?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
int main()
{
    int n = 111, i;
    recursive_dec_to_bin(n);
    for(i=len-1; i>=0; i--)
        printf("%d", arr[i]);
    return 0;
}
```

- a) 1110111
b) 1001111
c) 1101111
d) 1010111

Answer: c

Explanation: The program prints the binary equivalent of 111, which is 1101111.

12. How many times is the function recursive_dec_to_bin() called when the

following code is executed?

```
#include<stdio.h>
int arr[31], len = 0;
void recursive_dec_to_bin(int n)
{
    if(n == 0 && len == 0)
    {
        arr[len++] = 0;
        return;
    }
    if(n == 0)
        return;
    arr[len++] = n % 2;
    recursive_dec_to_bin(n/2);
}
int main()
{
    int n = 111, i;
    recursive_dec_to_bin(n);
    for(i=len-1; i>=0; i--)
        printf("%d", arr[i]);
    return 0;
}
```

- a) 7
- b) 8
- c) 9
- d) 10

Answer: b

Explanation: The function recursive_dec_to_bin() is called 8 times when the above code is executed.

1. Consider the following iterative implementation used to find the length of a linked list:

```
struct Node
{
    int val;
    struct Node *next;
}*head;
int get_len()
{
    struct Node *temp = head->next;
    int len = 0;
    while(____)
    {
        len++;
        temp = temp->next;
    }
}
```

```
    return len;
}
```

Which of the following conditions should be checked to complete the above code?

- a) temp->next != 0
- b) temp == 0
- c) temp != 0
- d) temp->next == 0

Answer: c

Explanation: The condition “temp != 0” should be checked to complete the above code.

2. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int get_len()
{
    struct Node *temp = head->next;
    int len = 0;
    while(temp != 0)
    {
        len++;
        temp = temp->next;
    }
    return len;
}
int main()
{
    int arr[10] = {1,2,3,4,5}, n = 5, i
;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
    struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
        sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = get_len();
    printf("%d", len);
}
```

```
    return 0;
}
```

- a) 4
- b) 5
- c) 6
- d) 7

Answer: b

Explanation: The program prints the length of the linked list, which is 5.

3. What is the time complexity of the following iterative implementation used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int get_len()
{
    struct Node *temp = head->next;
    int len = 0;
    while(temp != 0)
    {
        len++;
        temp = temp->next;
    }
    return len;
}
int main()
{
    int arr[10] = {1,2,3,4,5}, n = 5, i
;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = get_len();
    printf("%d",len);
}
```

```
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(logn)

Answer: b

Explanation: The time complexity of the above iterative implementation used to find the length of a linked list is O(n).

4. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int get_len()
{
    struct Node *temp = head->next;
    int len = 0;
    while(temp != 0)
    {
        len++;
        temp = temp->next;
    }
    return len;
}
int main()
{
    int arr[10] = {1,2,3,4,5}, n = 5, i
;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head->next = 0;
    int len = get_len();
    printf("%d",len);
    return 0;
}
```

- a) 0
- b) Garbage value
- c) Compile time error
- d) Runtime error

Answer: a

Explanation: The program prints the length

of the linked list, which is 0.

5. Which of the following can be the base case for the recursive implementation used to find the length of linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int get_len()
{
    struct Node *temp = head->next;
    int len = 0;
    while(temp != 0)
    {
        len++;
        temp = temp->next;
    }
    return len;
}
int main()
{
    int arr[10] = {1,2,3,4,5}, n = 5, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    int len = get_len();
    printf("%d",len);
    return 0;
}
```

a) if(current_node == 0) return 1
 b) if(current_node->next == 0) return 1
 c) if(current_node->next == 0) return 0
 d) if(current_node == 0) return 0

Answer: d

Explanation: The line “if(current_node == 0) return 0” can be used as a base case in the recursive implementation used to find the length of a linked list. Note: The line “if(current_node->next == 0) return 1” cannot be used because it won’t work when the length of the linked list is zero.

6. Which of the following lines should be inserted to complete the following recursive

implementation used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
    if(current_node == 0)
        return 0;
    return _____;
}
int main()
{
    int arr[10] = {1,2,3,4,5}, n = 5, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = recursive_get_len(head->next);
    printf("%d",len);
    return 0;
}
```

- a) recursive_get_len(current_node)
 b) 1 + recursive_get_len(current_node)
 c) recursive_get_len(current_node->next)
 d) 1 + recursive_get_len(current_node->next)

Answer: d

Explanation: The line “1 + recursive_get_len(current_node->next)” should be inserted to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
    if(current_node == 0)
        return 0;
    return 1 + recursive_get_len(current_node->next);
}
int main()
{
    int arr[10] = {-1,2,3,-3,4,5,0}, n
= 7, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = recursive_get_len(head->next);
    printf("%d",len);
    return 0;
}
```

- a) 6
- b) 7
- c) 8
- d) 9

Answer: b

Explanation: The program prints the length of the linked list, which is 7.

8. What is the time complexity of the following code used to find the length of a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
```

```
{
    int val;
    struct Node *next;
}*head;
int recursive_get_len(struct Node *current_node)
{
    if(current_node == 0)
        return 0;
    return 1 + recursive_get_len(current_node->next);
}
int main()
{
    int arr[10] = {-1,2,3,-3,4,5,0}, n
= 7, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = recursive_get_len(head->next);
    printf("%d",len);
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: To find the length of the linked list, the program iterates over the linked list once. So, the time complexity of the above code is O(n).

9. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
```

```

        struct Node *next;
}*head;
int recursive_get_len(struct Node *current
t_node)
{
    if(current_node == 0)
        return 0;
    return 1 + recursive_get_len(current
t_node->next);
}
int main()
{
    int arr[10] = {-1,2,3,-3,4,5}, n =
6, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = recursive_get_len(head->next);
    printf("%d",len);
    return 0;
}

```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: b

Explanation: The program prints the length of the linked list, which is 6.

10. How many times is the function recursive_get_len() called when the following code is executed?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node *next;
}*head;
int recursive_get_len(struct Node *current

```

```

t_node)
{
    if(current_node == 0)
        return 0;
    return 1 + recursive_get_len(current
t_node->next);
}
int main()
{
    int arr[10] = {-1,2,3,-3,4,5}, n =
6, i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head->next = 0;
    temp = head;
    for(i=0; i<n; i++)
    {
        newNode = (struct Node*)malloc(
sizeof(struct Node));
        newNode->val = arr[i];
        newNode->next = 0;
        temp->next = newNode;
        temp = temp->next;
    }
    int len = recursive_get_len(head->next);
    printf("%d",len);
    return 0;
}

```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: c

Explanation: The function is called “len + 1” times. Since the length of the linked list in the above code is 6, the function is called $6 + 1 = 7$ times.

1. Consider the following iterative implementation to find the length of the string:

```

#include<stdio.h>
int get_len(char *s)
{
    int len = 0;
    while(_____)
        len++;
    return len;
}

```

```

}

int main()
{
    char *s = "harsh";
    int len = get_len(s);
    printf("%d",len);
    return 0;
}

```

Which of the following lines should be inserted to complete the above code?

- a) s[len-1] != 0
- b) s[len+1] != 0
- c) s[len] != '\0'
- d) s[len] == '\0'

Answer: c

Explanation: The line “s[len] != ‘\0’” should be inserted to complete the above code.

2. What is the output of the following code?

```

#include<stdio.h>
int get_len(char *s)
{
    int len = 0;
    while(s[len] != '\0')
        len++;
    return len;
}
int main()
{
    char *s = "lengthofstring";
    int len = get_len(s);
    printf("%d",len);
    return 0;
}

a) 14
b) 0
c) Compile time error
d) Runtime error

```

Answer: a

Explanation: The program prints the length of the string “lengthofstring”, which is 14.

3. What is the time complexity of the following code used to find the length of the string?

```

#include<stdio.h>
int get_len(char *s)
{
    int len = 0;
    while(s[len] != '\0')
        len++;
    return len;
}
int main()
{
    char *s = "lengthofstring";
    int len = get_len(s);
    printf("%d",len);
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(logn)

```

Answer: b

Explanation: The time complexity of the code used to find the length of the string is O(n).

4. What is the output of the following code?

```

#include<stdio.h>
int get_len(char *s)
{
    int len = 0;
    while(s[len] != '\0')
        len++;
    return len;
}
int main()
{
    char *s = "";
    int len = get_len(s);
    printf("%d",len);
    return 0;
}

```

- a) 0
- b) 1
- c) Runtime error
- d) Garbage value

Answer: a

Explanation: The program prints the length of an empty string, which is 0.

5. Which of the following can be the base case for the recursive implementation used to find the length of a string?

```
#include<stdio.h>
int get_len(char *s)
{
    int len = 0;
    while(s[len] != '\0')
        len++;
    return len;
}
int main()
{
    char *s = "";
    int len = get_len(s);
    printf("%d",len);
    return 0;
}
```

- a) if(string[len] == 1) return 1
- b) if(string[len+1] == 1) return 1
- c) if(string[len] == '\0') return 0
- d) if(string[len] == '\0') return 1

Answer: c

Explanation: “if(string[len] == '\0') return 0” can be used as base case in the recursive implementation used to find the length of the string.

6. Consider the following recursive implementation used to find the length of a string:

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
    if(s[len] == 0)
        return 0;
    return _____;
}
int main()
{
    char *s = "abcdef";
    int len = recursive_get_len(s,0);
    printf("%d",len);
    return 0;
}
```

Which of the following lines should be inserted to complete the above code?

- a) 1

- b) len
- c) recursive_get_len(s, len+1)
- d) 1 + recursive_get_len(s, len+1)

Answer: d

Explanation: The line “1 + recursive_get_len(s, len+1)” should be inserted to complete the code.

7. What is the output of the following code?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
    if(s[len] == 0)
        return 0;
    return 1 + recursive_get_len(s, len
+1);
}
int main()
{
    char *s = "abcdef";
    int len = recursive_get_len(s,0);
    printf("%d",len);
    return 0;
}
```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: b

Explanation: The above code prints the length of the string “abcdef”, which is 6.

8. What is the time complexity of the following recursive implementation used to find the length of the string?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
    if(s[len] == 0)
        return 0;
    return 1 + recursive_get_len(s, len
+1);
}
int main()
{
    char *s = "abcdef";
    int len = recursive_get_len(s,0);
    printf("%d",len);
}
```

```

        return 0;
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above recursive implementation used to find the length of the string is O(n).

9. How many times is the function recursive_get_len() called when the following code is executed?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
    if(s[len] == 0)
        return 0;
    return 1 + recursive_get_len(s, len
+1);
}
int main()
{
    char *s = "adghjkl";
    int len = recursive_get_len(s,0);
    printf("%d",len);
    return 0;
}
```

- a) 6
- b) 7
- c) 8
- d) 9

Answer: c

Explanation: The function recursive_get_len() is called 8 times when the above code is executed.

10. What is the output of the following code?

```
#include<stdio.h>
int recursive_get_len(char *s, int len)
{
    if(s[len] == 0)
        return 0;
    return 1 + recursive_get_len(s, len
+1);
```

```

    }
int main()
{
    char *s = "123-1-2-3";
    int len = recursive_get_len(s,0);
    printf("%d",len);
    return 0;
}
```

- a) 3
- b) 6
- c) 9
- d) 10

Answer: c

Explanation: The above program prints the length of the string “123-1-2-3”, which is 9.

1. If Matrix A is of order X*Y and Matrix B is of order M*N, then what is the order of the Matrix A*B given that Y=M?

- a) Y*N
- b) X*M
- c) X*N
- d) Y*M

Answer: c

Explanation: The Matrix A*B is of order X*N as it is given that Y=M i.e. number of columns in Matrix A is equal to total number of rows in matrix B. So the Matrix A*B must have X number of rows and N number of columns.

2. How many recursive calls are there in Recursive matrix multiplication through Simple Divide and Conquer Method?

- a) 2
- b) 6
- c) 9
- d) 8

Answer: d

Explanation: For the multiplication two square matrix recursively using Simple Divide and Conquer Method, there are 8 recursive calls performed for high time complexity.

3. What is the time complexity of matrix multiplied recursively by Divide and Conquer Method?

- a) $O(n)$
- b) $O(n^2)$
- c) $O(n^3)$
- d) $O(n!)$

Answer: c

Explanation: The time complexity of recursive multiplication of two square matrices by the Divide and Conquer method is found to be $O(n^3)$ since there are total of 8 recursive calls.

4. What is the time complexity of matrix multiplied recursively by Strassen's Method?

- a) $O(n^{\log 7})$
- b) $O(n^2)$
- c) $O(n^3)$
- d) $O(n!)$

Answer: a

Explanation The time complexity of recursive multiplication of two square matrices by Strassen's Method is found to be $O(n^{\log 7})$ since there are total 7 recursive calls.

5. How many recursive calls are there in Recursive matrix multiplication by Strassen's Method?

- a) 5
- b) 7
- c) 8
- d) 4

Answer: b

Explanation: For the multiplication two square matrix recursively using Strassen's Method, there are 7 recursive calls performed for high time complexity.

6. Matrix A is of order 3*4 and Matrix B is of order 4*5. How many elements will be there in a matrix A*B multiplied recursively.

- a) 12
- b) 15

- c) 16
- d) 20

Answer: b

Explanation: The resultant matrix will be of order 3*5 when multiplied recursively and therefore the matrix will have $3*5=15$ elements.

7. If Matrix X is of order A*B and Matrix Y is of order C*D, and B=C then the order of the Matrix X*Y is A*D?

- a) True
- b) False

Answer: a

Explanation: Given that B=C, so the two matrix can be recursively multiplied. Therefore, the order of the Matrix X*Y is A*D.

8. What is the time complexity of the fastest known matrix multiplication algorithm?

- a) $O(n^{\log 7})$
- b) $O(n^{2.37})$
- c) $O(n^3)$
- d) $O(n!)$

Answer: b

Explanation: The Coppersmith-Winograd algorithm multiplies the matrices in $O(n^{2.37})$ time. Several improvements have been made in the algorithm since 2010.

9. Is Coppersmith-Winograd algorithm better than Strassen's algorithm in terms of time complexity?

- a) True
- b) False

Answer: a

Explanation: Since The Coppersmith-Winograd algorithm multiplies the matrices in $O(n^{2.37})$ time. The time complexity of recursive multiplication of two square matrices by Strassen's Method is found to be $O(n^{2.80})$. Therefore, Coppersmith-Winograd

algorithm better than Strassen's algorithm in terms of time complexity.

1. Which of the following statement is true about stack?

- a) Pop operation removes the top most element
- b) Pop operation removes the bottom most element
- c) Push operation adds new element at the bottom
- d) Push operation removes the top most element

Answer: a

Explanation: As stack is based on LIFO(Last In First Out) principle so the deletion takes place from the topmost element. Thus pop operator removes topmost element.

2. What is the space complexity of program to reverse stack recursively?

- a) O(1)
- b) O(log n)
- c) O(n)
- d) O(n log n)

Answer: c

Explanation: The recursive program to reverse stack uses memory of the order n to store function call stack.

3. Stack can be reversed without using extra space by _____

- a) using recursion
- b) using linked list to implement stack
- c) using an extra stack
- d) it is not possible

Answer: b

Explanation: If linked list is used for implementing stack then it can be reversed without using any extra space.

4. Which of the following is considered as the top of the stack in the linked list implementation of the stack?

- a) Last node
- b) First node
- c) Random node
- d) Middle node

Answer: b

Explanation: First node is considered as the top element when stack is implemented using linked list.

5. What is the time complexity of the program to reverse stack when linked list is used for its implementation?

- a) O(n)
- b) O(n log n)
- c) O(n^2)
- d) O(log n)

Answer: a

Explanation: As a linked list takes O(n) time for getting reversed thus linked list version of stack will also take the same time.

6. Which of the following takes O(n) time in worst case in array implementation of stack?

- a) pop
- b) push
- c) isEmpty
- d) pop, push and isEmpty takes constant time

Answer: d

Explanation: Functions pop, push and isEmpty all are implemented in constant time in worst case.

7. What will be the time complexity of the code to reverse stack recursively?

- a) O(n)
- b) O(n log n)
- c) O(log n)
- d) O(n^2)

Answer: d

Explanation: The recurrence relation for the recursive code to reverse stack will be given by $T(n)=T(n-1)+n$. This is calculated to be equal to $O(n^2)$.

8. Which of the following functions correctly represents the recursive approach to reverse a stack?

a)

```
int reverse()
{
    if(s.size()<0)
    {
        int x = s.top();
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

b)

```
int reverse()
{
    if(s.size()>=0)
    {
        int x = s.top();
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

c)

```
int reverse()
{
    if(s.size()>0)
    {
        int x = s.top();
        s.pop();
        reverse();
        BottomInsert(x);
    }
}
```

d)

```
int reverse()
{
    if(s.size()>0)
    {
        int x = s.top();
        BottomInsert(x);
        s.pop();
        reverse();
    }
}
```

Answer: c

Explanation: We keep on holding the elements in call stack until we reach the bottom of the stack. Then we insert elements at the bottom. This reverses our stack.

9. Which of the following correctly represents the function to insert elements at the bottom of stack?

a)

```
int BottomInsert(int x)
{
    if(s.size()!=0) s.push(x);
    else
    {
        int a = s.top();
        s.pop();
        BottomInsert(x);
        s.push(a);
    }
}
```

b)

```
int BottomInsert(int x)
{
    if(s.size()==0) s.push(x);
    else
    {
        int a = s.top();
        s.pop();
        s.push(a);
        BottomInsert(x);
    }
}
```

c)

```
int BottomInsert(int x)
{
    if(s.size()==0) s.push(x);
    else
    {
        int a = s.top();
        s.pop();
        BottomInsert(x);
        s.push(a);
    }
}
```

d)

```

int BottomInsert(int x)
{
    if(s.size()==0) s.push(x);
    else
    {
        s.pop();
        int a = s.top();
        BottomInsert(x);
        s.push(a);
    }
}

```

Answer: c

Explanation: We hold all the elements in the call stack until we reach the bottom of stack and then the first if statement is executed as the stack is empty at this stage. Finally we push back all the elements held in the call stack.

10. Which of the following code correctly represents the function to reverse stack without using recursion?

a)

```

#include <stack>
void reverseStack(stack<int> &input, stack<int> &extra)
{
    while(input.size()!=0)
    {
        extra.push(input.top());
        input.pop();
    }
    input.swap(extra);
}

```

b)

```

#include <stack>
void reverseStack(stack<int> &input, stack<int> &extra)
{
    while(input.size()!=0)
    {
        extra.push(input.top());
        input.pop();
    }
    extra.swap(input);
}

```

c)

```

#include <stack>
void reverseStack(stack<int> &input, stack<int> &extra)
{
    while(input.size()!=0)
    {
        input.pop();
        extra.push(input.top());
    }
    extra.swap(input);
}

```

d)

```

#include <stack>
void reverseStack(stack<int> &input, stack<int> &extra)
{
    while(input.size()==0)
    {
        input.pop();
        extra.push(input.top());
    }
    extra.swap(input);
}

```

Answer: b

Explanation: We are using one extra stack to reverse the given stack. First the elements of the original stack are pushed into the other stack which creates a reversed version of the original stack. Then we swap this stack with the original stack.

1. Which of the following sorting algorithm has best case time complexity of $O(n^2)$?

- a) bubble sort
- b) selection sort
- c) insertion sort
- d) stupid sort

Answer: b

Explanation: Selection sort is not an adaptive sorting algorithm. It finds the index of minimum element in each iteration even if the given array is already sorted. Thus its best case time complexity becomes $O(n^2)$.

2. Which of the following is the biggest advantage of selection sort?
- its has low time complexity
 - it has low space complexity
 - it is easy to implement
 - it requires only n swaps under any condition

Answer: d

Explanation: Selection sort works by obtaining least value element in each iteration and then swapping it with the current index. So it will take n swaps under any condition which will be useful when memory write operation is expensive.

3. What will be the recurrence relation of the code of recursive selection sort?
- $T(n) = 2T(n/2) + n$
 - $T(n) = 2T(n/2) + c$
 - $T(n) = T(n-1) + n$
 - $T(n) = T(n-1) + c$

Answer: c

Explanation: Function to find the minimum element index takes n time. The recursive call is made to one less element than in the previous call so the overall recurrence relation becomes $T(n) = T(n-1) + n$.

4. Which of the following sorting algorithm is NOT stable?
- Selection sort
 - Brick sort
 - Bubble sort
 - Merge sort

Answer: a

Explanation: Out of the given options selection sort is the only algorithm which is not stable. It is because the order of identical elements in sorted output may be different from input array.

5. What will be the best case time complexity of recursive selection sort?
- $O(n)$
 - $O(n^2)$

- $O(\log n)$
- $O(n \log n)$

Answer: b

Explanation: Selection sort's algorithm is such that it finds the index of minimum element in each iteration even if the given array is already sorted. Thus its best case time complexity becomes $O(n^2)$.

6. Recursive selection sort is a comparison based sort.
- true
 - false

Answer: a

Explanation: In selection sort we need to compare elements in order to find the minimum element in each iteration. So we can say that it uses comparisons in order to sort the array. Thus it qualifies as a comparison based sort.

7. What is the average case time complexity of recursive selection sort?
- $O(n)$
 - $O(n \log n)$
 - $O(n^2)$
 - $O(\log n)$

Answer: c

Explanation: The overall recurrence relation of recursive selection sort is given by $T(n) = T(n-1) + n$. It is found to be equal to $O(n^2)$. It is unvaried throughout the three cases.

8. What is the bidirectional variant of selection sort?
- cocktail sort
 - bogo sort
 - gnome sort
 - bubble sort

Answer: a

Explanation: A bidirectional variant of selection sort is called cocktail sort. It's an algorithm which finds both the minimum and maximum values in the array in every pass.

This reduces the number of scans of the array by a factor of 2.

9. Choose correct C++ code for recursive selection sort from the following.

a)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
    if (i == 0)
        return i;
    int k = minIndex(a, i + 1, j);
    return (a[i] < a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{
    if (index == n)
        return;
    int x = minIndex(a, index, n-1);
    if (x == index)
    {
        swap(a[x], a[index]);
    }
    recursiveSelectionSort(a, n, inde
x + 1);
}
```

```
int main()
{
    int arr[] = {5,3,2,4,1};
    int n = sizeof(arr)/sizeof(arr[0])
);
```

```
recursiveSelectionSort(arr, n);
return 0;
}
```

b)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
    if (i == j)
        return i;
    int k = minIndex(a, i + 1, j);
    return (a[i] < a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{
    if (index == n)
        return;
```

```
if (index == n)
    return;
```

```
int x = minIndex(a, index, n-1);
if (x != index)
{
    swap(a[x], a[index]);
}
recursiveSelectionSort(a, n, inde
x + 1);
}
```

```
int main()
{
    int arr[] = {5,3,2,4,1};
    int n = sizeof(arr)/sizeof(arr[0])
);
recursiveSelectionSort(arr, n);
return 0;
}
```

c)

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
    if (i == j)
        return i;
    int k = minIndex(a, i + 1, j);
    return (a[i] > a[k])? i : k;
}
void recursiveSelectionSort(int a[], int n, int index = 0)
{
    if (index == n)
        return;
    int x = minIndex(a, index, n-1);
    if (x != index)
    {
        swap(a[x], a[index]);
    }
    recursiveSelectionSort(a, n, inde
x + 1);
}
```

```
int main()
{
    int arr[] = {5,3,2,4,1};
    int n = sizeof(arr)/sizeof(arr[0])
);
recursiveSelectionSort(arr, n);
return 0;
}
```

```
d)
```

```
#include <iostream>
using namespace std;
int minIndex(int a[], int i, int j)
{
```

```

        if (i == j)
            return i;
        int k = minIndex(a, i + 1, j);
        return (a[i] > a[k])? i : k;
    }
void recursiveSelectionSort(int a[], int
n, int index = 0)
{
    if (index == 0)
        return;
    int x = minIndex(a, index, n-1);
    if (x == index)
    {
        swap(a[x], a[index]);
    }
    recursiveSelectionSort(a, n, inde
x + 1);
}
int main()
{
    int arr[] = {5,3,2,4,1};
    int n = sizeof(arr)/sizeof(arr[0])
);
    recursiveSelectionSort(arr, n);
    return 0;
}

```

Answer: b

Explanation: Using the function recursiveSelectionSort() we find the element that needs to be placed at the current index. For finding the minimum element index we use another function minIndex(). After finding the minimum element index the current element is swapped with this element in the function recursiveSelectionSort().

10. What is the number of swaps required to sort the array arr={5,3,2,4,1} using recursive selection sort?

- a) 0
- b) 1
- c) 2
- d) 3

Answer: c

Explanation: The first swap takes place between 1 and 5. The second swap takes place between 3 and 2 which sorts our array.

1. Which of the following methods can be used to find the largest and smallest element in an array?
 - a) Recursion
 - b) Iteration
 - c) Both recursion and iteration
 - d) No method is suitable

Answer: c

Explanation: Both recursion and iteration can be used to find the largest and smallest element in an array.

2. Consider the following iterative code snippet to find the largest element:

```

int get_max_element(int *arr,int n)
{
    int i, max_element = arr[0];
    for(i = 1; i < n; i++)
        if(_____)
            max_element = arr[i];
    return max_element;
}

```

Which of the following lines should be inserted to complete the above code?

- a) arr[i] > max_element
- b) arr[i] < max_element
- c) arr[i] == max_element
- d) arr[i] != max_element

Answer: a

Explanation: The line “arr[i] > max_element” should be inserted to complete the above code snippet.

3. Consider the following code snippet to find the smallest element in an array:

```

int get_min_element(int *arr, int n)
{
    int i, min_element = arr[0];
    for(i = 1; i < n; i++)
        if(_____)
            min_element = arr[i];
    return min_element;
}

```

Which of the following lines should be inserted to complete the above code?

- a) $\text{arr}[i] > \text{min_element}$
- b) $\text{arr}[i] < \text{min_element}$
- c) $\text{arr}[i] == \text{min_element}$
- d) $\text{arr}[i] != \text{min_element}$

Answer: b

Explanation: The line “ $\text{arr}[i] < \text{min_element}$ ” should be inserted to complete the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
    int i, max_element = arr[0];
    for(i = 1; i < n; i++)
        if(arr[i] > max_element)
            max_element = arr[i];
    return max_element;
}
int get_min_element(int *arr, int n)
{
    int i, min_element = arr[0];
    for(i = 1; i < n; i++)
        if(arr[i] < min_element)
            min_element = arr[i];
    return min_element;
}
int main()
{
    int n = 7, arr[7] = {1,1,1,0,-1,-1,-1};
    int max_element = get_max_element(ar
r,n);
    int min_element = get_min_element(ar
r,n);
    printf("%d %d",max_element,min_eleme
nt);
    return 0;
}
```

- a) 5 3
- b) 3 5
- c) 8 1
- d) 1 8

Answer: c

Explanation: The program prints the values of the largest and the smallest elements in the array, which are 8 and 1 respectively.

5. What is the output of the following code?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
    int i, max_element = arr[0];
    for(i = 1; i < n; i++)
        if(arr[i] > max_element)
            max_element = arr[i];
    return max_element;
}
int get_min_element(int *arr, int n)
{
    int i, min_element;
    for(i = 1; i < n; i++)
        if(arr[i] < min_element)
            min_element = arr[i];
    return min_element;
}
int main()
{
    int n = 7, arr[7] = {1,1,1,0,-1,-1,-1};
    int max_element = get_max_element(ar
r,n);
    int min_element = get_min_element(ar
r,n);
    printf("%d %d",max_element,min_eleme
nt);
    return 0;
}
```

- a) 1 -1
- b) -1 1
- c) 1 Garbage value
- d) Depends on the compiler

Answer: c

Explanation: Since the `min_element` variable is not initialised, some compilers will auto initialise as 0 which produces -1 as output whereas some compilers won't initialise automatically. In that case, the result will be a garbage value hence the output depends on the compiler.

6. What is the time complexity of the following iterative implementation used to find the largest and smallest element in an array?

```
#include<stdio.h>
int get_max_element(int *arr,int n)
{
    int i, max_element = arr[0];
    for(i = 1; i < n; i++)
```

```

        if(arr[i] > max_element)
            max_element = arr[i];
        return max_element;
    }
    int get_min_element(int *arr, int n)
    {
        int i, min_element;
        for(i = 1; i < n; i++)
            if(arr[i] < min_element)
                min_element = arr[i];
        return min_element;
    }
    int main()
    {
        int n = 7, arr[7] = {1,1,1,0,-1,-1,-1};
        int max_element = get_max_element(ar
r,n);
        int min_element = get_min_element(ar
r,n);
        printf("%d %d",max_element,min_eleme
nt);
        return 0;
    }

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O($n/2$)

Answer: b

Explanation: The time complexity of the above iterative implementation used to find the largest and the smallest element in an array is O(n).

7. Consider the following recursive implementation to find the largest element in an array.

```

int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int recursive_max_element(int *arr, int l
en, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return _____;
}

```

Which of the following lines should be inserted to complete the above code?

- a) max_of_two(arr[idx], recursive_max_element(arr, len, idx))
- b) recursive_max_element(arr, len, idx)
- c) max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1))
- d) recursive_max_element(arr, len, idx + 1)

Answer: c

Explanation: The line “max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1))” should be inserted to complete the above code.

8. What is the output of the following code?

```

#include<stdio.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_max_element(int *arr, int l
en, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return max_of_two(arr[idx], recursi
ve_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int l
en, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return min_of_two(arr[idx], recursi
ve_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 10, idx = 0, arr[] = {5,2,6,7
,8,9,3,-1,1,10};
    int max_element = recursive_max_eleme
nt(arr,n,idx);
    int min_element = recursive_min_eleme
nt(arr,n,idx);

```

```

        printf("%d %d",max_element,min_element);
    }
    return 0;
}

```

- a) -1 10
- b) 10 -1
- c) 1 10
- d) 10 1

Answer: b

Explanation: The program prints the values of the largest and the smallest element in the array, which are 10 and -1 respectively.

9. What is the time complexity of the following recursive implementation used to find the largest and the smallest element in an array?

```

#include<stdio.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int len, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 10, idx = 0, arr[] = {5,2,6,7,8,9,3,-1,1,10};
    int max_element = recursive_max_element(arr,n,idx);
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d %d",max_element,min_element);
    return 0;
}

```

```

nt(arr,n,idx);
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d %d",max_element,min_element);
}
return 0;
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above recursive implementation used to find the largest and smallest element in an array is $O(n)$.

10. How many times is the function recursive_min_element() called when the following code is executed?

```

int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_min_element(int *arr, int len, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 10, idx = 0, arr[] = {5,2,6,7,8,9,3,-1,1,10};
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d",min_element);
    return 0;
}

a) 9
b) 10
c) 11
d) 12

```

Answer: b

Explanation: The function recursive_min_element() is called 10 times when the above code is executed.

11. What is the output of the following code?

```
#include<stdio.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_max_element(int *arr, int len, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return max_of_two(arr[idx], recursive_max_element(arr, len, idx + 1));
}
int recursive_min_element(int *arr, int len, int idx)
{
    if(idx == len - 1)
        return arr[idx];
    return min_of_two(arr[idx], recursive_min_element(arr, len, idx + 1));
}
int main()
{
    int n = 5, idx = 0, arr[] = {1,1,1,1,1};
    int max_element = recursive_max_element(arr,n,idx);
    int min_element = recursive_min_element(arr,n,idx);
    printf("%d %d",max_element,min_element);
    return 0;
}

a) 1 1
b) 0 0
c) compile time error
d) runtime error
```

Answer: a

Explanation: The program prints the values of the largest and the smallest element in the array, which are 1 and 1.

1. Which of the following methods can be used to find the largest and smallest number in a linked list?

- a) Recursion
- b) Iteration
- c) Both Recursion and iteration
- d) Impossible to find the largest and smallest numbers

Answer: c

Explanation: Both recursion and iteration can be used to find the largest and smallest number in a linked list.

2. Consider the following code snippet to find the largest element in a linked list:

```
struct Node{
    int val;
    struct Node *next;
}*head;
int get_max()
{
    struct Node* temp = head->next;
    int max_num = temp->val;
    while(____)
    {
        if(temp->val > max_num)
            max_num = temp->val;
        temp = temp->next;
    }
    return max_num;
}
```

Which of the following lines should be inserted to complete the above code?

- a) temp->next != 0
- b) temp != 0
- c) head->next != 0
- d) head != 0

Answer: b

Explanation: The line “temp != 0” should be inserted to complete the above code.

3. Consider the following code snippet to find the smallest element in a linked list:

```
struct Node
{
    int val;
    struct Node* next;
}*head;
int get_min()
{
    struct Node* temp = head->next;
    int min_num = temp->val;
    while(temp != 0)
    {
        if(______)
            min_num = temp->val;
        temp = temp->next;
    }
    return min_num;
}
```

Which of the following lines should be inserted to complete the above code?

- a) temp > min_num
- b) val > min_min
- c) temp->val < min_num
- d) temp->val > min_num

Answer: c

Explanation: The line “temp->val = min_num” should be inserted to complete the above code.

4. What is the output of the following code:

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int get_max()
{
    struct Node* temp = head->next;
    int max_num = temp->val;
    while(temp != 0)
    {
        if(temp->val > max_num)
            max_num = temp->val;
        temp = head->next;
    }
    return max_num;
}
```

```
int main()
{
    int n = 9, arr[9] ={5,1,3,4,5,2,3,3
,1},i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(s
izeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int max_num = get_max();
    printf("%d %d",max_num);
    return 0;
}
```

- a) 5
- b) 1
- c) runtime error
- d) garbage value

Answer: c

Explanation: The variable temp will always point to the first element in the linked list due to the line “temp = head->next” in the while loop. So, it will be an infinite while loop and the program will produce a runtime error.

5. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int get_max()
{
    struct Node* temp = head->next;
    int max_num = temp->val;
    while(temp != 0)
    {
        if(temp->val > max_num)
            max_num = temp->val;
        temp = temp->next;
    }
    return max_num;
}
```

```

}

int get_min()
{
    struct Node* temp = head->next;
    int min_num = temp->val;
    while(temp != 0)
    {
        if(temp->val < min_num)
            min_num = temp->val;
        temp = temp->next;
    }
    return min_num;
}

int main()
{
    int i, n = 9, arr[9] ={8,3,3,4,5,2,
5,6,7};
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(s
izeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int max_num = get_max();
    int min_num = get_min();
    printf("%d %d",max_num,min_num);
    return 0;
}

```

- a) 2 2
- b) 8 8
- c) 2 8
- d) 8 2

Answer: d

Explanation: The program prints the largest and smallest elements in the linked list, which are 8 and 2 respectively.

6. What is the time complexity of the following iterative code used to find the smallest and largest element in a linked list?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{

```

```

        int val;
        struct Node* next;
    }*head;
int get_max()
{
    struct Node* temp = head->next;
    int max_num = temp->val;
    while(temp != 0)
    {
        if(temp->val > max_num)
            max_num = temp->val;
        temp = temp->next;
    }
    return max_num;
}

int get_min()
{
    struct Node* temp = head->next;
    int min_num = temp->val;
    while(temp != 0)
    {
        if(temp->val < min_num)
            min_num = temp->val;
        temp = temp->next;
    }
    return min_num;
}

int main()
{
    int i, n = 9, arr[9] ={8,3,3,4,5,2,
5,6,7};
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(
struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(s
izeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int max_num = get_max();
    int min_num = get_min();
    printf("%d %d",max_num,min_num);
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(n3)

```

Answer: b

Explanation: The time complexity of the above iterative code used to find the largest and smallest element in a linked list is O(n).

7. Consider the following recursive implementation to find the largest element in a linked list:

```
struct Node
{
    int val;
    struct Node* next;
}*head;
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int recursive_get_max(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return max_of_two(_____, _____);
}
```

Which of the following arguments should be passed to the function max_of_two() to complete the above code?

- a) temp->val, recursive_get_max(temp->next)
- b) temp, temp->next
- c) temp->val, temp->next->val
- d) temp->next->val, temp

Answer: a

Explanation: The arguments {temp->val, recursive_get_max(temp->next)} should be passed to the function max_of_two() to complete the above code.

8. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int max_of_two(int a, int b)
{
    if(a > b)
```

```
        return a;
    return b;
}
int recursive_get_max(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return max_of_two(temp->val, recursive_get_max(temp->next));
}
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_get_min(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return min_of_two(temp->val, recursive_get_min(temp->next));
}
int main()
{
    int n = 9, arr[9] ={1,3,2,4,5,0,5,6,7},i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int max_num = recursive_get_max(head->next);
    int min_num = recursive_get_min(head->next);
    printf("%d %d",max_num,min_num);
    return 0;
}
```

- a) 7 1
- b) 0 7
- c) 7 0
- d) 1 1

Answer: c

Explanation: The program prints the largest

and the smallest elements in the linked list, which are 7 and 0 respectively.

9. What is the time complexity of the recursive implementation used to find the largest and smallest element in a linked list?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int recursive_get_max(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return max_of_two(temp->val,recursive_get_max(temp->next));
}
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_get_min(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
    int n = 9, arr[9] ={1,3,2,4,5,0,5,6,7},i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp =temp->next;
    }
}
```

```
        temp = temp->next;
    }
    int max_num = recursive_get_max(head->next);
    int min_num = recursive_get_min(head->next);
    printf("%d %d",max_num,min_num);
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above recursive implementation used to find the largest and smallest element in linked list is O(n).

10. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_get_min(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
    int n = 5, arr[5] ={1,1,1,1,1},i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {
        newNode =(struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp =temp->next;
    }
}
```

```

        sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int min_num = recursive_get_min(head
->next);
    printf("%d",min_num);
    return 0;
}

```

- a) 1
- b) 0
- c) compile time error
- d) runtime error

Answer: a

Explanation: The program prints the smallest element in the linked list, which is 1.

11. How many times will the function recursive_get_min() be called when the following code is executed?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int min_of_two(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int recursive_get_min(struct Node* temp)
{
    if(temp->next == 0)
        return temp->val;
    return min_of_two(temp->val,recursive_get_min(temp->next));
}
int main()
{
    int n = 5, arr[5] ={1,1,1,1,1},i;
    struct Node *temp, *newNode;
    head = (struct Node*)malloc(sizeof(struct Node));
    head -> next =0;
    temp = head;
    for(i=0;i<n;i++)
    {

```

```

        newNode =(struct Node*)malloc(
        sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next =newNode;
        temp = temp->next;
    }
    int min_num = recursive_get_min(head
->next);
    printf("%d",min_num);
    return 0;
}

```

- a) 4
- b) 5
- c) 6
- d) 7

Answer: b

Explanation: The function recursive_get_min() will be called 5 times when the above code is executed.

1. Which of the following techniques can be used to search an element in an unsorted array?

- a) Iterative linear search
- b) Recursive binary search
- c) Iterative binary search
- d) Normal binary search

Answer: a

Explanation: Iterative linear search can be used to search an element in an unsorted array.

Note: Binary search can be used only when the array is sorted.

2. Consider the array {1,1,1,1,1}. Select the wrong option?

- a) Iterative linear search can be used to search for the elements in the given array
- b) Recursive linear search can be used to search for the elements in the given array
- c) Recursive binary search can be used to search for the elements in the given array
- d) No method is defined to search for an element in the given array

Answer: d

Explanation: Iterative linear search, Recursive linear search and Recursive binary search can be applied to search for an element in the above given array.

3. What does the following code do?

```
#include<stdio.h>
int search_num(int *arr, int num, int len)
{
    int i;
    for(i = 0; i < len; i++)
        if(arr[i] == num)
            return i;
        return -1;
}
int main()
{
    int arr[5] ={1,2,3,4,5},num=3,len =
5;
    int indx = search_num(arr,num,len);
    printf("Index of %d is %d",num,indx
);
    return 0;
}
```

- a) Search and returns the index of all the occurrences of the number that is searched
- b) Search and returns the index of the first occurrence of the number that is searched
- c) Search and returns of the last occurrence of the number that is searched
- d) Returns the searched element from the given array

Answer: b

Explanation: The code finds the index of the first occurrence of the number that is searched.

4. What is the output of the following code?

```
#include<stdio.h>
int search_num(int *arr, int num, int len
)
{
    int i;
    for(i = 0; i < len; i++)
        if(arr[i] == num)
            return i;
        return -1;
```

```
}
int main()
{
    int arr[5] ={1,3,3,3,5},num=3,len =
5;
    int indx = search_num(arr,num,len);
    printf("Index of %d is %d",num,indx
);
    return 0;
}
```

- a) Index of 3 is 0
- b) Index of 3 is 1
- c) Index of 3 is 2
- d) Index of 3 is 3

Answer: b

Explanation: The program prints the index of the first occurrence of 3, which is 1.

5. What is the time complexity of the following code used to search an element in an array?

```
#include<stdio.h>
int search_num(int *arr, int num, int len
)
{
    int i;
    for(i = 0; i < len; i++)
        if(arr[i] == num)
            return i;
        return -1;
}
int main()
{
    int arr[5] ={1,3,3,3,5},num=3,len =
5;
    int indx = search_num(arr,num,len);
    printf("Index of %d is %d",num,indx
);
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the

above code used to search an element in an array is O(n).

6. Consider the following recursive implementation of linear search:

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
    if(idx == len)
        return -1;
    if(arr[idx] == num)
        return idx;
    return _____;
}
int main()
{
    int arr[5] ={1,3,3,3,5},num=2,len = 5;
    int indx = recursive_search_num(arr,num,0,len);
    printf("Index of %d is %d",num,indx);
    return 0;
}
```

Which of the following recursive calls should be added to complete the above code?

- a) recursive_search_num(arr, num+1, idx, len);
- b) recursive_search_num(arr, num, idx, len);
- c) recursive_search_num(arr, num, idx+1, len);
- d) recursive_search_num(arr, num+1, idx+1, len);

Answer: c

Explanation: The recursive call “recursive_search_num(arr, num, idx+1, len)” should be added to complete the above code.

7. What is the output of the following code?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
    if(idx == len)
        return -1;
    if(arr[idx] == num)
        return idx;
    return recursive_search_num(arr, num
```

```
, idx+1, len);
}
int main()
{
    int arr[8] ={1,2,3,3,3,5,6,7},num=5
    ,len = 8;
    int indx = recursive_search_num(arr
    ,num,0,len);
    printf("Index of %d is %d",num,indx
    );
    return 0;
}
```

- a) Index of 5 is 5
- b) Index of 5 is 6
- c) Index of 5 is 7
- d) Index of 5 is 8

Answer: a

Explanation: The program prints the index of 5, which is 5.

8. How many times is the function recursive_search_num() called when the following code is executed?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
    if(idx == len)
        return -1;
    if(arr[idx] == num)
        return idx;
    return recursive_search_num(arr, num
    , idx+1, len);
}
int main()
{
    int arr[8] ={1,2,3,3,3,5,6,7},num=5
    ,len = 8;
    int indx = recursive_search_num(arr
    ,num,0,len);
    printf("Index of %d is %d",num,indx
    );
    return 0;
}
```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: b

Explanation: The function recursive_search_num() is called 6 times when the above code is executed.

9. What is the time complexity of the following recursive implementation of linear search?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
    if(idx == len)
        return -1;
    if(arr[idx] == num)
        return idx;
    return recursive_search_num(arr, num, idx+1, len);
}
int main()
{
    int arr[8] = {1,2,3,3,3,5,6,7},num=5
    ,len = 8;
    int indx = recursive_search_num(arr, num, 0, len);
    printf("Index of %d is %d",num,indx);
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above recursive implementation of linear search is O(n).

1. What is the output of the following code?

```
#include<stdio.h>
int recursive_search_num(int *arr, int num, int idx, int len)
{
    if(idx == len)
        return -1;
    if(arr[idx] == num)
        return idx;
    return recursive_search_num(arr, num,
```

```
, idx+1, len);
}
int main()
{
    int arr[8] = {-11,2,-3,0,3,5,-6,7},num = -2,len = 8;
    int indx = recursive_search_num(arr, num, 0, len);
    printf("Index of %d is %d",num,indx);
    return 0;
}
```

- a) Index of -2 is 1
- b) Index of -2 is 0
- c) Index of -2 is -1
- d) None of the mentioned

Answer: c

Explanation: The program prints the index of the first occurrence of -2. Since, -2 doesn't exist in the array, the program prints -1.

2. What does the following code do?

```
#include<stdio.h>
int cnt = 0;
int recursive_search_num(int *arr, int num, int idx, int len)
{
    int cnt = 0;
    if(idx == len)
        return cnt;
    if(arr[idx] == num)
        cnt++;
    return cnt + recursive_search_num(arr, num, idx+1, len);
}
int main()
{
    int arr[8] = {0,0,0,0,3,5,-6,7},num = 0,len = 8;
    int ans = recursive_search_num(arr, num, 0, len);
    printf("%d",ans);
    return 0;
}
```

- a) Adds all the indexes of the number 0
- b) Finds the first last occurrence of the number 0
- c) Counts the number of occurrences of the number 0
- d) None of the mentioned

Answer: c

Explanation: The above code counts the number of occurrences of the number 0.

3. Consider the following recursive implementation of the binary search:

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
        return mid;
    else if(arr[mid] < num)
        _____;
    else
        hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
    int arr[8] ={0,0,0,0,3,5,6,7},num =
7,len = 8;
    int indx = recursive_binary_search(
arr,num,0,len-1);
    printf("Index of %d is %d",num,indx
);
    return 0;
}
```

Which of the following lines should be added to complete the above code?

- a) $hi = mid - 1$
- b) $mid = (lo + hi)/2$
- c) $mid = lo - 1$
- d) $lo = mid + 1$

Answer: d

Explanation: The line “ $lo = mid + 1$ ” should be added to complete the above code.

4. What is the output of the following code?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
```

```
        if(arr[mid] == num)
            return mid;
        else if(arr[mid] < num)
            lo = mid + 1;
        else
            hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
    int arr[8] = {1,2,3,4,5,6,7,8},num
= 7,len = 8;
    int indx = recursive_binary_search(
arr,num,0,len-1);
    printf("Index of %d is %d",num,indx
);
    return 0;
}
```

a) Index of 7 is 4

b) Index of 7 is 5

c) Index of 7 is 6

d) Index of 7 is 7

Answer: c

Explanation: The program prints the index of number 7, which is 6.

5. What is the output of the following code?

```
#include<stdio.h>
int recursive_binary_search(int *arr, int num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
        return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr, num, lo, hi);
}
int main()
{
    int arr[8] = {0,0,0,0,3,5,6,7},num
= 0,len = 8;
    int indx = recursive_binary_search(
arr,num,0,len-1);
    printf("Index of %d is %d",num,indx
);
```

```

        return 0;
    }
}

```

- a) Index of 0 is 0
- b) Index of 0 is 1
- c) Index of 0 is 2
- d) Index of 0 is 3

Answer: d

Explanation: In this case, when the function recursive_binary_search() is called for the first time we have: lo = 0 and hi = 7. So, the value of mid is:

$\text{mid} = (\text{lo} + \text{hi})/2 = (0 + 7)/2 = 3$. Since, arr[mid] = arr[3] = 0, the function returns the value of mid, which is 3.

6. What is the time complexity of the above recursive implementation of binary search?

- a) O(n)
- b) O(2^n)
- c) O(logn)
- d) O(n!)

Answer: c

Explanation: The time complexity of the above recursive implementation of binary search is O(logn).

7. In which of the below cases will the following code produce a wrong output?

```

int recursive_binary_search(int *arr, int
num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
        return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr,
num, lo, hi);
}

```

- a) Array: {0,0,0,0,0,0} Search: -10
- b) Array: {1,2,3,4,5} Search: 0

- c) Array: {5,4,3,2,1} Search: 1
- d) Array: {-5,-4,-3,-2,-1} Search: -1

Answer: c

Explanation: Since the array {5,4,3,2,1} is sorted in descending order, it will produce a wrong output.

8. How many times is the function recursive_binary_search() called when the following code is executed?

```

#include<stdio.h>
int recursive_binary_search(int *arr, int
num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
        return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr,
num, lo, hi);
}
int main()
{
    int arr[5] = {1,2,3,4,5},num = 1,len
= 5;
    int indx = recursive_binary_search(
arr,num,0,len-1);
    printf("Index of %d is %d",num,indx
);
    return 0;
}

```

- a) 0
- b) 1
- c) 2
- d) 3

Answer: c

Explanation: The function recursive_binary_search() is called 2 times, when the above code is executed.

9. What is the output of the following code?

```

#include<stdio.h>
int recursive_binary_search(int *arr, int

```

```

num, int lo, int hi)
{
    if(lo > hi)
        return -1;
    int mid = (lo + hi)/2;
    if(arr[mid] == num)
        return mid;
    else if(arr[mid] < num)
        lo = mid + 1;
    else
        hi = mid - 1;
    return recursive_binary_search(arr,
num, lo, hi);
}
int main()
{
    int arr[5] = {5,4,3,2,1},num = 1,length = 5;
    int indx = recursive_binary_search(
arr,num,0,length-1);
    printf("Index of %d is %d",num,indx);
    return 0;
}

```

- a) Index of 1 is 4
- b) Index of 1 is 5
- c) Index of 1 is -1
- d) Index of 1 is 0

Answer: c

Explanation: Since the array is sorted in descending order, the above implementation of binary search will not work for the given array. Even though 1 is present in the array, binary search won't be able to search it and it will produce -1 as an answer.

1. Which of the following methods can be used to search an element in a linked list?
- a) Iterative linear search
 - b) Iterative binary search
 - c) Recursive binary search
 - d) Normal binary search

Answer: a

Explanation: Iterative linear search can be used to search an element in a linked list. Binary search can be used only when the list is sorted.

2. Consider the following code snippet to search an element in a linked list:

```

struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(int value)
{
    struct Node *temp = head->next;
    while(temp != 0)
    {
        if(temp->val == value)
            return 1;
        _____;
    }
    return 0;
}

```

Which of the following lines should be inserted to complete the above code?

- a) temp = next
- b) temp->next = temp
- c) temp = temp->next
- d) return 0

Answer: c

Explanation: The line “temp = temp->next” should be inserted to complete the above code.

3. What does the following code do?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(int value)
{
    struct Node *temp = head->next;
    while(temp != 0)
    {
        if(temp->val == value)
            return 1;
        temp = temp->next;
    }
    return 0;
}
int main()
{

```

```

int arr[5] = {1,2,3,4,5};
int n = 5,i;
head = (struct Node*)malloc(sizeof(
struct Node));
head->next = 0;
struct Node *temp;
temp = head;
for(i=0; i<n; i++)
{
    struct Node *newNode = (struct
Node*)malloc(sizeof(struct Node));
    newNode->next = 0;
    newNode->val = arr[i];
    temp->next = newNode;
    temp = temp->next;
}
int ans = linear_search(60);
if(ans == 1)
printf("Found");
else
printf("Not found");
return 0;
}

```

- a) Finds the index of the first occurrence of a number in a linked list
- b) Finds the index of the last occurrence of a number in a linked list
- c) Checks if a number is present in a linked list
- d) Checks whether the given list is sorted or not

Answer: c

Explanation: The above code checks if a number is present in a linked list.

4. What is the output of the following code?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(int value)
{
    struct Node *temp = head->next;
    while(temp != 0)
    {
        if(temp->val == value)
            return 1;
        temp = temp->next;
    }
}

```

```

    }
    return 0;
}
int main()
{
    int arr[5] = {1,2,3,4,5};
    int n = 5,i;
    head = (struct Node*)malloc(sizeof(s
truct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct
Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(-1);
    if(ans == 1)
    printf("Found");
    else
    printf("Not found");
    return 0;
}

```

- a) Found
- b) Not found
- c) Compile time error
- d) Runtime error

Answer: b

Explanation: Since the number -1 is not present in the linked list, the program prints not found.

5. What is the time complexity of the following implementation of linear search on a linked list?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(int value)
{
    struct Node *temp = head->next;
    while(temp != 0)
    {

```

```

        if(temp->val == value)
            return 1;
        temp = temp->next;
    }
    return 0;
}
int main()
{
    int arr[5] = {1,2,3,4,5};
    int n = 5,i;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(-1);
    if(ans == 1)
        printf("Found");
    else
        printf("Not found");
    return 0;
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above implementation of linear search on a linked list is O(n).

6. What is the output of the following code?

```

#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(int value)
{
    struct Node *temp = head->next;

```

```

    while(temp->next != 0)
    {
        if(temp->val == value)
            return 1;
        temp = temp->next;
    }
    return 0;
}
int main()
{
    int arr[6] = {1,2,3,4,5,6};
    int n = 6,i;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(60);
    if(ans == 1)
        printf("Found");
    else
        printf("Not found");
    return 0;
}

```

- a) Found
- b) Not found
- c) Compile time error
- d) Runtime error

Answer: b

Explanation: The condition in the while loop “temp->next == 0”, checks if the current element is the last element. If the current element is the last element, the value of the current element is not compared with the value to be searched. So, even though the number 6 is present in the linked list, it will print not found.

7. Can binary search be applied on a sorted linked list in O(Logn) time?

- a) No
- b) Yes

Answer: a

Explanation: Since linked list doesn't allow random access, binary search cannot be applied on a sorted linked list in O(Logn) time.

8. What will be time complexity when binary search is applied on a linked list?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity will be O(n) when binary search is applied on a linked list.

9. Consider the following recursive implementation of linear search on a linked list:

```
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
    if(temp == 0)
        return 0;
    if(temp->val == value)
        return 1;
    return _____;
}
```

Which of the following lines should be inserted to complete the above code?

- a) 1
- b) 0
- c) linear_search(temp, value)
- d) linear_search(temp->next, value)

Answer: d

Explanation: The line “linear_search(temp->next, value)”, should be inserted to complete the above code.

10. What is the output of the following code?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
    if(temp == 0)
        return 0;
    if(temp->val == value)
        return 1;
    return linear_search(temp->next, value);
}
int main()
{
    int arr[6] = {1,2,3,4,5,6};
    int n = 6,i;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(head->next,6);
    if(ans == 1)
        printf("Found");
    else
        printf("Not found");
    return 0;
}
```

- a) Found
- b) Not found
- c) Compile time error
- d) Runtime error

Answer: a

Explanation: Since the element 6 is present in the linked list, the program prints “Found”.

11. How many times is the function linear_search() called when the following

code is executed?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
    if(temp == 0)
        return 0;
    if(temp->val == value)
        return 1;
    return linear_search(temp->next, value);
}
int main()
{
    int arr[6] = {1,2,3,4,5,6};
    int n = 6,i;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(head->next,6);
    if(ans == 1)
        printf("Found");
    else
        printf("Not found");
    return 0;
}
```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: b

Explanation: The function linear_search() is called 6 times when the above code is executed.

12. What is the time complexity of the following recursive implementation of linear search?

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int val;
    struct Node* next;
}*head;
int linear_search(struct Node *temp,int value)
{
    if(temp == 0)
        return 0;
    if(temp->val == value)
        return 1;
    return linear_search(temp->next, value);
}
int main()
{
    int arr[6] = {1,2,3,4,5,6};
    int n = 6,i;
    head = (struct Node*)malloc(sizeof(struct Node));
    head->next = 0;
    struct Node *temp;
    temp = head;
    for(i=0; i<n; i++)
    {
        struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->next = 0;
        newNode->val = arr[i];
        temp->next = newNode;
        temp = temp->next;
    }
    int ans = linear_search(head->next,6);
    if(ans == 1)
        printf("Found");
    else
        printf("Not found");
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(n3)
```

Answer: b

Explanation: The time complexity of the above recursive implementation of linear search is $O(n)$.

1. What will be the output for following code?

```
#include<stdio.h>
int func(int x, int y)
{
    if (y == 0)
        return 1;
    else if (y%2 == 0)
        return func(x, y/2)*func(
x, y/2);
    else
        return x*func(x, y/2)*fun
c(x, y/2);
}
int main()
{
    int x = 2;
    int y = 3;

    printf("%d", func(x, y));
    return 0;
}
```

- a) 9
- b) 6
- c) 8
- d) 5

Answer: c

Explanation: The given program calculates the value of x raised to power y . Thus $2^3 = 8$.

2. What will be the time complexity of the following code which raises an integer x to the power y ?

```
#include<stdio.h>
int power(int x, int y)
{
    if (y == 0)
        return 1;
    else if (y%2 == 0)
        return power(x, y/2)*powe
r(x, y/2);
    else
        return x*power(x, y/2)*po
```

```
wer(x, y/2);
}
int main()
{
    int x = 2;
    int y = 3;

    printf("%d", power(x, y));
    return 0;
}
```

- a) $O(n)$
- b) $O(\log n)$
- c) $O(n \log n)$
- d) $O(n^2)$

Answer: a

Explanation: The recurrence relation for the above code is given by $T(n)=2T(n/2)+c$. By using master theorem we can calculate the result for this relation. It is found to be equal to $O(n)$.

3. What is the space complexity of the given code?

```
#include<stdio.h>
int power(int x, int y)
{
    if (y == 0)
        return 1;
    else if (y%2 == 0)
        return power(x, y/2)*powe
r(x, y/2);
    else
        return x*power(x, y/2)*po
wer(x, y/2);
}
int main()
{
    int x = 2;
    int y = 3;

    printf("%d", power(x, y));
    return 0;
}
```

- a) $O(1)$
- b) $O(n)$
- c) $O(\log n)$
- d) $O(n \log n)$

Answer: a

Explanation: The space complexity of the given code will be equal to O(1) as it uses only constant space in the memory.

4. Recursive program to raise an integer x to power y uses which of the following algorithm?
- Dynamic programming
 - Backtracking
 - Divide and conquer
 - Greedy algorithm

Answer: c

Explanation: The recursive approach uses divide and conquer algorithm as we break the problem into smaller parts and then solve the smaller parts and finally combine their results to get the overall solution.

5. What is the least time in which we can raise a number x to power y?
- $O(x)$
 - $O(y)$
 - $O(\log x)$
 - $O(\log y)$

Answer: d

Explanation: We can optimize the code for finding power of a number by calculating x raised to power $y/2$ only once and using it depending on whether y is even or odd.

6. What will be the time complexity of the following code?

```
#include<stdio.h>
int power(int x, int y)
{
    int temp;
    if (y == 0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else
        return x*temp*temp;
}
int main()
{
    int x = 2;
```

```
    int y = 3;
    printf("%d", power(x, y));
    return 0;
}
```

- $O(1)$
- $O(n)$
- $O(\log n)$
- $O(n \log n)$

Answer: c

Explanation: The given code is the optimized version for finding the power of a number. It forms a recurrence relation given by $T(n)=T(n/2)+c$ which can be solved using master theorem. It is calculated to be equal to $O(\log n)$.

7. What is the advantage of iterative code for finding power of number over recursive code?

- Iterative code requires less time
- Iterative code requires less space
- Iterative code is more compiler friendly
- It has no advantage

Answer: b

Explanation: Both iterative and recursive approach can be implemented in $\log n$ time but the recursive code requires memory in call stack which makes it less preferable.

8. Which of the following correctly implements iterative code for finding power of a number?

a)

```
#include <stdio.h>
int power(int x,  int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y & 1)
            res = res * x;
        y = y >> 1;
        x = x * x;
    }
    return res;
```

```

}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}

```

b)

```

#include <stdio.h>
int power(int x, int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y && 1)
            res = res * x;
        y = y >> 1;
        x = x * x;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}

```

c)

```

#include <stdio.h>
int power(int x, int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y && 1)
            res = x * x;
        y = y >> 1;
        x = x * y;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}

```

d)

```

#include <stdio.h>
int power(int x, int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y & 1)
            res = x * x;
        y = y >> 1;
        x = x * y;
    }
    return res;
}
int main()
{
    int x = 3;
    unsigned int y = 5;
    printf("%d", power(x, y));
    return 0;
}

```

Answer: a

Explanation: It represents the iterative version of required code. It has a time and space complexity of O(log n) and O(1) respectively.

9. Recursive approach to find power of a number is preferred over iterative approach.

- a) True
- b) False

Answer: b

Explanation: The recursive code requires memory in call stack which makes it less preferable as compared to iterative approach.

10. What will be the output for following code?

```

float power(float x, int y)
{
    float temp;
    if( y==0)
        return 1;
    temp = power(x, y/2);
    if (y%2 == 0)
        return temp*temp;
    else

```

```

    {
        if(y > 0)
            return x*temp*temp
    p;
        else
            return (temp*temp
    )/x;
    }
int main()
{
    float x = 2;
    int y = -3;
    printf("%f", power(x, y));
    return 0;
}

```

- a) Error
- b) 1/4
- c) 4
- d) 0.25

Answer: d

Explanation: The given code is capable of handling negative powers too. Thus, the output will be $2^{-2} = 0.25$.

Sanfoundry Global Education & Learning Series – Data Structures & Algorithms.

1. What is the objective of tower of hanoi puzzle?
 - a) To move all disks to some other rod by following rules
 - b) To divide the disks equally among the three rods by following rules
 - c) To move all disks to some other rod in random order
 - d) To divide the disks equally among three rods in random order

Answer: a

Explanation: Objective of tower of hanoi problem is to move all disks to some other rod by following the following rules-1) Only one disk can be moved at a time. 2) Disk can only be moved if it is the uppermost disk of the stack. 3) No disk should be placed over a smaller disk.

2. Which of the following is NOT a rule of tower of hanoi puzzle?
 - a) No disk should be placed over a smaller disk
 - b) Disk can only be moved if it is the uppermost disk of the stack
 - c) No disk should be placed over a larger disk
 - d) Only one disk can be moved at a time

Answer: c

Explanation: The rule is to not put a disk over a smaller one. Putting a smaller disk over larger one is allowed.

3. The time complexity of the solution tower of hanoi problem using recursion is

-
- a) $O(n^2)$
 - b) $O(2^n)$
 - c) $O(n \log n)$
 - d) $O(n)$

Answer: b

Explanation: Time complexity of the problem can be found out by solving the recurrence relation: $T(n)=2T(n-1)+c$. Result of this relation is found to be equal to 2^n . It can be solved using substitution.

4. Recurrence equation formed for the tower of hanoi problem is given by _____
 - a) $T(n) = 2T(n-1)+n$
 - b) $T(n) = 2T(n/2)+c$
 - c) $T(n) = 2T(n-1)+c$
 - d) $T(n) = 2T(n/2)+n$

Answer: c

Explanation: As there are 2 recursive calls to $n-1$ disks and one constant time operation so the recurrence relation will be given by $T(n) = 2T(n-1)+c$.

5. Minimum number of moves required to solve a tower of hanoi problem with n disks is

-
- a) 2^n
 - b) 2^{n-1}

- c) n^2
d) n^2-1

Answer: b

Explanation: Minimum number of moves can be calculated by solving the recurrence relation – $T(n)=2T(n-1)+c$. Alternatively we can observe the pattern formed by the series of number of moves 1,3,7,15.....Either way it turns out to be equal to 2^n-1 .

6. Space complexity of recursive solution of tower of hanoi puzzle is _____

- a) $O(1)$
b) $O(n)$
c) $O(\log n)$
d) $O(n \log n)$

Answer: b

Explanation: Space complexity of tower of hanoi problem can be found out by solving the recurrence relation $T(n)=T(n-1)+c$. Result of this relation is found out to be n . It can be solved using substitution.

7. Which of the following functions correctly represent the solution to tower of hanoi puzzle?

a)

```
void ToH(int n,int a,int b,int c)
{
    If(n>0)
    {
        ToH(n-1,a,c,b);
        cout<<"move a disk from" <<a<<" to"
        <<c;
        ToH(n-1,b,a,c);
    }
}
```

b)

```
void ToH(int n,int a,int b,int c
{
    If(n>0)
    {
        ToH(n-1,a,b,c);
        cout<<"move a disk from" <<a<<" to"
        <<c;
        ToH(n-1,b,a,c);
```

}

c)

```
void ToH(int n,int a,int b,int c)
{
    If(n>0)
    {
        ToH(n-1,a,c,b);
        cout<<"move a disk from" <<a<<" to"
        <<c;
        ToH(n-1,a,b,c);
    }
}
```

d)

```
void ToH(int n,int a,int b,int c)
{
    If(n>0)
    {
        ToH(n-1,b,a,c);
        cout<<"move a disk from" <<a<<" to"
        <<c;
        ToH(n-1,a,c,b);
    }
}
```

Answer: a

Explanation: The first recursive call moves $n-1$ disks from a to b using c. Then we move a disk from a to c. Finally the second recursive call moves $n-1$ disks from b to c using a.

8. Recursive solution of tower of hanoi problem is an example of which of the following algorithm?

- a) Dynamic programming
b) Backtracking
c) Greedy algorithm
d) Divide and conquer

Answer: d

Explanation: The recursive approach uses divide and conquer algorithm as we break the problem into smaller parts and then solve the smaller parts and finally combine their results to get the overall solution.

9. Tower of hanoi problem can be solved iteratively.

- a) True
- b) False

Answer: a

Explanation: Iterative solution to tower of hanoi puzzle also exists. Its approach depends on whether the total numbers of disks are even or odd.

10. Minimum time required to solve tower of hanoi puzzle with 4 disks assuming one move takes 2 seconds, will be _____

- a) 15 seconds
- b) 30 seconds
- c) 16 seconds
- d) 32 seconds

Answer: b

Explanation: Number of moves = $2^4 - 1 = 16 - 1 = 15$

Time for 1 move = 2 seconds

Time for 15 moves = $15 \times 2 = 30$ seconds.

Sanfoundry Global Education & Learning Series – Data Structures & Algorithms.

1. Master's theorem is used for?

- a) solving recurrences
- b) solving iterative relations
- c) analysing loops
- d) calculating the time complexity of any code

Answer: a

Explanation: Master's theorem is a direct method for solving recurrences. We can solve any recurrence that falls under any one of the three cases of master's theorem.

2. How many cases are there under Master's theorem?

- a) 2
- b) 3
- c) 4
- d) 5

Answer: b

Explanation: There are primarily 3 cases under master's theorem. We can solve any recurrence that falls under any one of these three cases.

3. What is the result of the recurrences which fall under first case of Master's theorem (let the recurrence be given by $T(n)=aT(n/b)+f(n)$ and $f(n)=n^c$)?

- a) $T(n) = O(n^{\log_b a})$
- b) $T(n) = O(n^c \log n)$
- c) $T(n) = O(f(n))$
- d) $T(n) = O(n^2)$

Answer: a

Explanation: In first case of master's theorem the necessary condition is that $c < \log_b a$. If this condition is true then $T(n) = O(n^{\log_b a})$.

4. What is the result of the recurrences which fall under second case of Master's theorem (let the recurrence be given by $T(n)=aT(n/b)+f(n)$ and $f(n)=n^c$)?

- a) $T(n) = O(n \log_b a)$
- b) $T(n) = O(n^c \log n)$
- c) $T(n) = O(f(n))$
- d) $T(n) = O(n^2)$

Answer: b

Explanation: In second case of master's theorem the necessary condition is that $c = \log_b a$. If this condition is true then $T(n) = O(n^c \log n)$

5. What is the result of the recurrences which fall under third case of Master's theorem (let the recurrence be given by $T(n)=aT(n/b)+f(n)$ and $f(n)=n^c$)?

- b) $T(n) = O(n \log_b a)$
- b) $T(n) = O(n^c \log n)$
- c) $T(n) = O(f(n))$
- d) $T(n) = O(n^2)$

Answer: c

Explanation: In third case of master's theorem the necessary condition is that $c > \log_b a$. If this condition is true then $T(n) = O(f(n))$.

6. We can solve any recurrence by using Master's theorem.

- a) true
- b) false

Answer: b

Explanation: No we cannot solve all the recurrences by only using master's theorem. We can solve only those which fall under the three cases prescribed in the theorem.

7. Under what case of Master's theorem will the recurrence relation of merge sort fall?

- a) 1
- b) 2
- c) 3
- d) It cannot be solved using master's theorem

Answer: b

Explanation: The recurrence relation of merge sort is given by $T(n) = 2T(n/2) + O(n)$. So we can observe that $c = \log_b a$ so it will fall under case 2 of master's theorem.

8. Under what case of Master's theorem will the recurrence relation of stooge sort fall?

- a) 1
- b) 2
- c) 3
- d) It cannot be solved using master's theorem

Answer: a

Explanation: The recurrence relation of stooge sort is given as $T(n) = 3T(2/3n) + O(1)$. It is found to be equal to $O(n^{2.7})$ using master's theorem first case.

9. Which case of master's theorem can be extended further?

- a) 1
- b) 2

c) 3

d) No case can be extended

Answer: b

Explanation: The second case of master's theorem can be extended for a case where $f(n) = n^c (\log n)^k$ and the resulting recurrence becomes $T(n) = O(n^c (\log n)^{k+1})$.

10. What is the result of the recurrences which fall under the extended second case of Master's theorem (let the recurrence be given by $T(n) = aT(n/b) + f(n)$ and $f(n) = n^c (\log n)^k$)?

- a) $T(n) = O(n \log_b a)$
- b) $T(n) = O(n^c \log n)$
- c) $T(n) = O(n^c (\log n)^{k+1})$
- d) $T(n) = O(n^2)$

Answer: c

Explanation: In the extended second case of master's theorem the necessary condition is that $c = \log_b a$. If this condition is true then $T(n) = O(n^c (\log n)^{k+1})$.

11. Under what case of Master's theorem will the recurrence relation of binary search fall?

- a) 1
- b) 2
- c) 3
- d) It cannot be solved using master's theorem

Answer: b

Explanation: The recurrence relation of binary search is given by $T(n) = T(n/2) + O(1)$. So we can observe that $c = \log_b a$ so it will fall under case 2 of master's theorem.

1. Solve the following recurrence using Master's theorem.

$$T(n) = 4T(n/2) + n^2$$

- a) $T(n) = O(n)$
- b) $T(n) = O(\log n)$
- c) $T(n) = O(n^2 \log n)$
- d) $T(n) = O(n^2)$

Answer: c

Explanation: The given recurrence can be solved by using the second case of Master's theorem.

$$T(n) = O(nc \log n) = \text{Here } nc = n^2$$

So the solution becomes $T(n) = O(n^2 \log n)$.

2. Solve the following recurrence using Master's theorem.

$$T(n) = T(n/2) + 2^n$$

- a) $T(n) = O(n^2)$
- b) $T(n) = O(n^2 \log n)$
- c) $T(n) = O(2^n)$
- d) cannot be solved

Answer: c

Explanation: The given recurrence can be solved by using the third case (as $f(n) > \log 21$) of Master's theorem.

$$T(n) = O(f(n)) = \text{Here } f(n) = 2n.$$

So the solution becomes $T(n) = O(2n)$.

3. Solve the following recurrence using Master's theorem.

$$T(n) = 16T(n/4) + n$$

- a) $T(n) = O(n)$
- b) $T(n) = O(\log n)$
- c) $T(n) = O(n^2 \log n)$
- d) $T(n) = O(n^2)$

Answer: d

Explanation: The given recurrence can be solved by using the first case of Master's theorem. So the solution becomes $T(n) = O(n^2)$.

4. Solve the following recurrence using Master's theorem.

$$T(n) = 2T(n/2) + n/\log n$$

- a) $T(n) = O(n)$
- b) $T(n) = O(\log n)$
- c) $T(n) = O(n^2 \log n)$
- d) cannot be solved using master's theorem

Answer: d

Explanation: The given recurrence cannot be

solved by using the Master's theorem. It is because this recurrence relation does not fit into any case of Master's theorem.

5. Solve the following recurrence using Master's theorem.

$$T(n) = 0.7 T(n/2) + 1/n$$

- a) $T(n) = O(n)$
- b) $T(n) = O(\log n)$
- c) $T(n) = O(n^2 \log n)$
- d) cannot be solved using master's theorem

Answer: d

Explanation: The given recurrence cannot be solved by using the Master's theorem. It is because in this recurrence relation $a < 1$ so master's theorem cannot be applied.

6. Solve the following recurrence using Master's theorem.

$$T(n) = 4 T(n/2) + n!$$

- a) $T(n) = O(n!)$
- b) $T(n) = O(n! \log n)$
- c) $T(n) = O(n^2 \log n)$
- d) cannot be solved using master's theorem

Answer: a

Explanation: The given recurrence can be solved by using the third case of Master's theorem. So the solution becomes $T(n) = O(n!)$.

7. Solve the following recurrence using Master's theorem.

$$T(n) = 4T(n/4) + n \log n$$

- a) $T(n) = O(n (\log n)^2)$
- b) $T(n) = O(n \log n)$
- c) $T(n) = O(n^2 \log n)$
- d) cannot be solved using master's theorem

Answer: a

Explanation: The given recurrence can be solved by using the extended second case of Master's theorem. So the solution becomes $T(n) = O(n (\log n)^2)$.

8. What will be the recurrence relation of the following code?

```
Int sum(int n)
{
    If(n==1)
        return 1;
    else
        return n+sum(n-1);
}
```

- a) $T(n) = T(n/2) + n$
- b) $T(n) = T(n-1) + n$
- c) $T(n) = T(n-1) + O(1)$
- d) $T(n) = T(n/2) + O(1)$

Answer: c

Explanation: As after every recursive call the integer up to which the sum is to be calculated decreases by 1. So the recurrence relation for the given code will be $T(n) = T(n-1) + O(1)$.

9. What will be the recurrence relation of the following code?

```
int xpowy(int x, int n)
if (n==0) return 1;
if (n==1) return x;
if ((n % 2) == 0)
return xpowy(x*x, n/2);
else
return xpowy(x*x, n/2) * x;
```

- a) $T(n) = T(n/2) + n$
- b) $T(n) = T(n-1) + n$
- c) $T(n) = T(n-1) + O(1)$
- d) $T(n) = T(n/2) + O(1)$

Answer: d

Explanation: As after every recursive call the integer up to which the power is to be calculated decreases by half. So the recurrence relation for the given code will be $T(n) = T(n/2) + O(1)$. It can be solved by using master's theorem.

10. What will be the time complexity of the following code?

```
int xpowy(int x, int n)
{
    if (n==0)
        return 1;
    if (n==1)
        return x;
    if ((n % 2) == 0)
        return xpowy(x*x, n/2);
    else
        return xpowy(x*x, n/2) * x;
}
```

- a) $O(\log n)$
- b) $O(n)$
- c) $O(n \log n)$
- d) $O(n^2)$

Answer: a

Explanation: As the recurrence relation of the code is given by $T(n) = T(n/2) + O(1)$ so it can be solved by using master's theorem second case.

UNIT II BRUTE FORCE AND DIVIDE-AND- CONQUER

1. Which of the following is a sub string of “SANFOUNDRY”?

- a) SANO
- b) FOUND
- c) SAND
- d) FOND

Answer: b

Explanation: A sub string is a subset of another string. So “FOUND” is the only possible sub string out of the given options.

2. What will be the output of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
```

```

{
    int m = strlen(str2);
    int n = strlen(str1);
    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (str1[i + j] != str2[j])
                break;
        if (j == m)
            cout << i << endl;
    }
}
int main()
{
    char str1[] = "1253234";
    char str2[] = "323";
    func(str2, str1);
    return 0;
}

a) O(n)
b) O(m)
c) O(m * n)
d) O(m + n)

```

Answer: c

Explanation: The given code describes the naive method of finding a pattern in a string. So the output will be 3 as the given sub string begins at that index in the pattern.

3. What will be the worst case time complexity of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
{
    int m = strlen(str2);
    int n = strlen(str1);
    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (str1[i + j] != str2[j])
                break;
    }
}
```

```

= str2[j])                                break;
                                                if (j == m)
                                                    cout << i << endl;
                                            ;
                                        }
}
int main()
{
    char str1[] = "1253234";
    char str2[] = "323";
    func(str2, str1);
    return 0;
}
```

- a) O(n)
- b) O(m)
- c) O(m * n)
- d) O(m + n)

Answer: c

Explanation: The given code describes the naive method of pattern searching. By observing the nested loop in the code we can say that the time complexity of the loop is $O(m*n)$.

4. What will be the auxiliary space complexity of the following code?

```
#include<bits/stdc++.h>
using namespace std;

void func(char* str2, char* str1)
{
    int m = strlen(str2);
    int n = strlen(str1);
    for (int i = 0; i <= n - m; i++)
    {
        int j;
        for (j = 0; j < m; j++)
            if (str1[i + j] != str2[j])
                break;
        if (j == m)
            cout << i << endl;
    }
}
```

```

int main()
{
    char str1[] = "1253234";
    char str2[] = "323";
    func(str2, str1);
    return 0;
}

```

- a) O(n)
- b) O(1)
- c) O(log n)
- d) O(m)

Answer: b

Explanation: The given code describes the naive method of pattern searching. Its auxiliary space requirement is O(1).

5. What is the worst case time complexity of KMP algorithm for pattern searching (m = length of text, n = length of pattern)?

- a) O(n)
- b) O(n*m)
- c) O(m)
- d) O(log n)

Answer: c

Explanation: KMP algorithm is an efficient pattern searching algorithm. It has a time complexity of O(m) where m is the length of text.

6. What will be the best case time complexity of the following code?

```

#include<bits/stdc++.h>
using namespace std;
void func(char* str2, char* str1)
{
    int m = strlen(str2);
    int n = strlen(str1);

    for (int i = 0; i <= n - m; i++)
    {
        int j;

        for (j = 0; j < m; j++)
            if (str1[i + j] != str2[j])
                break;
    }
}

```

```

if (j == m)
    cout << i << endl
;
}

int main()
{
    char str1[] = "1253234";
    char str2[] = "323";
    func(str2, str1);
    return 0;
}

```

- a) O(n)
- b) O(m)
- c) O(m * n)
- d) O(m + n)

Answer: b

Explanation: The given code describes the naive method of pattern searching. The best case of the code occurs when the first character of the pattern does not appear in the text at all. So in such a case, only one iteration is required thus time complexity will be O(m).

7. What is the time complexity of Z algorithm for pattern searching (m = length of text, n = length of pattern)?

- a) O(n + m)
- b) O(m)
- c) O(n)
- d) O(m * n)

Answer: a

Explanation: Z algorithm is an efficient pattern searching algorithm as it searches the pattern in linear time. It has a time complexity of O(m + n) where m is the length of text and n is the length of the pattern.

8. What is the auxiliary space complexity of Z algorithm for pattern searching (m = length of text, n = length of pattern)?

- a) O(n + m)
- b) O(m)
- c) O(n)
- d) O(m * n)

Answer: b

Explanation: Z algorithm is an efficient pattern searching algorithm as it searches the pattern in linear time. It uses auxiliary space of $O(m)$ for maintaining Z array.

9. The naive pattern searching algorithm is an in place algorithm.
 a) true
 b) false

Answer: a

Explanation: The auxiliary space complexity required by naive pattern searching algorithm is $O(1)$. So it qualifies as an in place algorithm.

10. Rabin Karp algorithm and naive pattern searching algorithm have the same worst case time complexity.
 a) true
 b) false

Answer: a

Explanation: The worst case time complexity of Rabin Karp algorithm is $O(m*n)$ but it has a linear average case time complexity. So Rabin Karp and naive pattern searching algorithm have the same worst case time complexity.

1. Which of the following sorting algorithms is the fastest?
 a) Merge sort
 b) Quick sort
 c) Insertion sort
 d) Shell sort

Answer: b

Explanation: Quick sort is the fastest known sorting algorithm because of its highly optimized inner loop.

2. Quick sort follows Divide-and-Conquer strategy.
 a) True
 b) False

Answer: a

Explanation: In quick sort, the array is divided into sub-arrays and then it is sorted (divide-and-conquer strategy).

3. What is the worst case time complexity of a quick sort algorithm?
 a) $O(N)$
 b) $O(N \log N)$
 c) $O(N^2)$
 d) $O(\log N)$

Answer: c

Explanation: The worst case performance of a quick sort algorithm is mathematically found to be $O(N^2)$.

4. Which of the following methods is the most effective for picking the pivot element?
 a) first element
 b) last element
 c) median-of-three partitioning
 d) random element

Answer: c

Explanation: Median-of-three partitioning is the best method for choosing an appropriate pivot element. Picking a first, last or random element as a pivot is not much effective.

5. Find the pivot element from the given input using median-of-three partitioning method.
 8, 1, 4, 9, 6, 3, 5, 2, 7, 0.
 a) 8
 b) 7
 c) 9
 d) 6

Answer: d

Explanation: Left element=8, right element=0,
 Centre=[position(left+right)/2]=6.

6. Which is the safest method to choose a pivot element?
 a) choosing a random element as pivot
 b) choosing the first element as pivot

- c) choosing the last element as pivot
- d) median-of-three partitioning method

Answer: a

Explanation: This is the safest method to choose the pivot element since it is very unlikely that a random pivot would consistently provide a poor partition.

7. What is the average running time of a quick sort algorithm?

- a) $O(N^2)$
- b) $O(N)$
- c) $O(N \log N)$
- d) $O(\log N)$

Answer: c

Explanation: The best case and average case analysis of a quick sort algorithm are mathematically found to be $O(N \log N)$.

8. Which of the following sorting algorithms is used along with quick sort to sort the sub arrays?

- a) Merge sort
- b) Shell sort
- c) Insertion sort
- d) Bubble sort

Answer: c

Explanation: Insertion sort is used along with quick sort to sort the sub arrays. It is used only at the end.

9. Quick sort uses join operation rather than merge operation.

- a) true
- b) false

Answer: a

Explanation: Quick sort uses join operation since join is a faster operation than merge.

10. How many sub arrays does the quick sort algorithm divide the entire array into?

- a) one
- b) two

- c) three
- d) four

Answer: b

Explanation: The entire array is divided into two partitions, 1st sub array containing elements less than the pivot element and 2nd sub array containing elements greater than the pivot element.

11. Which is the worst method of choosing a pivot element?

- a) first element as pivot
- b) last element as pivot
- c) median-of-three partitioning
- d) random element as pivot

Answer: a

Explanation: Choosing the first element as pivot is the worst method because if the input is pre-sorted or in reverse order, then the pivot provides a poor partition.

12. Which among the following is the best cut-off range to perform insertion sort within a quick sort?

- a) $N=0-5$
- b) $N=5-20$
- c) $N=20-30$
- d) $N>30$

Answer: b

Explanation: A good cut-off range is anywhere between $N=5$ and $N=20$ to avoid nasty degenerate cases.

1. The shortest distance between a line and a point is achieved when?

- a) a line is drawn at 90 degrees to the given line from the given point
- b) a line is drawn at 180 degrees to the given line from the given point
- c) a line is drawn at 60 degrees to the given line from the given point
- d) a line is drawn at 270 degrees to the given line from the given point

Answer: a

Explanation: The shortest distance between a line and a point is achieved when a line is drawn at 90 degrees to the given line from the given point.

2. What is the shortest distance between the line given by $ax + by + c = 0$ and the point (x_1, y_1) ?

$$\frac{|ax_1+by_1+c|}{\sqrt{a^2+b^2}}$$

- a) $\frac{\sqrt{a^2+b^2}}{|ay_1+bx_1+c|}$
- b) $\frac{|ay_1+bx_1+c|}{\sqrt{a^2+b^2}}$
- c) $\frac{|ax_1+by_1|}{\sqrt{a^2+b^2}}$
- d) ax_1+by_1+c

Answer: a

Explanation: The shortest distance between a line and a point is given by the formula $(ax_1+by_1+c)/(\sqrt{a^2+b^2})$. This formula can be derived using the formula of area of a triangle.

3. What is the shortest distance between the line given by $-2x + 3y + 4 = 0$ and the point $(5,6)$?

- a) 4.5 units
- b) 5.4 units
- c) 4.3 units
- d) 3.3 units

Answer: d

Explanation: The shortest distance between a line and a point is given by the formula $(ax_1+by_1+c)/(\sqrt{a^2+b^2})$. Using this formula we get the answer 3.3 units.

4. What is the general formula for finding the shortest distance between two parallel lines given by $ax+by+c_1=0$ and $ax+by+c_2=0$?

$$\frac{|ax_1+by_1+c_1|}{\sqrt{a^2+b^2}}$$

- a) $\frac{\sqrt{a^2+b^2}}{|ax_1+by_1+c_1|}$

b) $\frac{|c_1 - c_2|}{\sqrt{a^2+b^2}}$

c) $\frac{|c_1 + c_2|}{\sqrt{a^2+b^2}}$

- d) $c_1 + c_2$

Answer: b

Explanation: The general formula for finding the shortest distance between two parallel lines given by $ax+by+c_1$ and $ax+by+c_2$ is $(c_1-c_2)/(\sqrt{a^2+b^2})$. We can find this by considering the distance of any one point on one of the line to the other line.

5. What is the distance between the lines $3x-4y+7=0$ and $3x-4y+5=0$?

- a) 1 unit
- b) 0.5 unit
- c) 0.8 unit
- d) 0.4 unit

Answer: d

Explanation: As the given lines are parallel so the distance between them can be calculated by using the formula $(c_1 - c_2)/(\sqrt{a^2+b^2})$. So we get the distance as 0.4 unit.

6. What will be the slope of the line given by $ax + by + c = 0$?

- a) $-a/b$
- b) $-b/a$
- c) $-c/a$
- d) a/c

Answer: a

Explanation: The slope of a line given by the equation $ax + by + c = 0$ has the slope of $-a/b$. So two lines having the same ratio of the coefficient of x and y will be parallel to each other.

7. What will be the slope of the line given by $10x + 5y + 8 = 0$?

- a) -5

- b) -2
- c) -1.25
- d) 5

Answer: b

Explanation: The slope of a line given by the equation $ax + by + c = 0$ has the slope of $-a/b$. So the slope of the given line will be -2.

8. What will be the co-ordinates of foot of perpendicular line drawn from the point (-1,3) to the line $3x-4y-16=0$?

- a) (1/5,2/5)
- b) (2/25,5/25)
- c) (68/25,-49/25)
- d) (-49/25,68/25)

Answer: c

Explanation: The foot of perpendicular can be found by equating the distance between the two points and the distance between point and line. This is found to be (68/25,-49/25).

9. Which of the following is used to find the absolute value of the argument in C++?

- a) abs()
- b) fabs()
- c) mod()
- d) ab()

Answer: b

Explanation: In C++ the absolute value of an argument can be found by using the function fabs(). It is available under the header file math.h.

10. What will be the slope of the line perpendicular to the line $6x-3y-16=0$?

- a) 1/2
- b) -1/2
- c) 2
- d) -2

Answer: b

Explanation: For two lines to be perpendicular the product of their slopes should be equal to -1. So as the slope of given

line is 2 so the slope of line perpendicular to it will be $-1/2$.

11. Find the output of the following code.

```
#include<math.h>
#include<iostream>
using namespace std;
void distance(float x, float y, float a,
float b, float c)
{
    float d = fabs((a * x + b * y + c
)) / (sqrt(a * a + b * b));
    cout<<d;
    return;
}
int main()
{
    float x = -2;
    float y = -3;
    float a = 5;
    float b = -2;
    float c = - 4;
    distance(x, y, a, b, c);
    return 0;
}
```

- a) 2.8
- b) 1.8
- c) 1.4
- d) 2.4

Answer: c

Explanation: The given code calculates the shortest distance between line and a point. So the output will be 1.4.

1. Which of the following areas do closest pair problem arise?

- a) computational geometry
- b) graph colouring problems
- c) numerical problems
- d) string matching

Answer: a

Explanation: Closest pair problem arises in two most important areas- computational geometry and operational research.

2. Which approach is based on computing the distance between each pair of distinct points

- and finding a pair with the smallest distance?
- Brute force
 - Exhaustive search
 - Divide and conquer
 - Branch and bound

Answer: a

Explanation: Brute force is a straight forward approach that solves closest pair problem using that algorithm.

3. What is the runtime efficiency of using brute force technique for the closest pair problem?
- $O(N)$
 - $O(N \log N)$
 - $O(N^2)$
 - $O(N^3 \log N)$

Answer: c

Explanation: The efficiency of closest pair algorithm by brute force technique is mathematically found to be $O(N^2)$.

4. The most important condition for which closest pair is calculated for the points (p_i, p_j) is?
- $i > j$
 - $i \neq j$
 - $i = j$
 - $i < j$

Answer: d

Explanation: To avoid computing the distance between the same pair of points twice, we consider only the pair of points (p_i, p_j) for which $i < j$.

5. What is the basic operation of closest pair algorithm using brute force technique?
- Euclidean distance
 - Radius
 - Area
 - Manhattan distance

Answer: a

Explanation: The basic operation of closest

pair algorithm is Euclidean distance and its formula is given by $d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

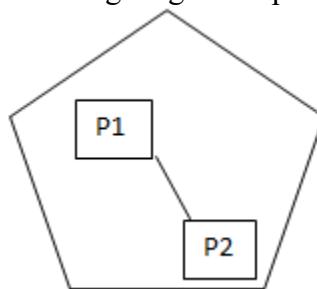
6. Which of the following is similar to Euclidean distance?

- Manhattan distance
- Pythagoras metric
- Chebyshev distance
- Heuristic distance

Answer: b

Explanation: In older times, Euclidean distance metric is also called a Pythagoras metric which is the length of the line segment connecting two points.

7. Which of the following strategies does the following diagram depict?



- Divide and conquer strategy
- Brute force
- Exhaustive search
- Backtracking

Answer: b

Explanation: Brute force is a straight forward approach to solve critical problems. Here, we use brute force technique to find the closest distance between p1 and p2.

8. Manhattan distance is an alternative way to define a distance between two points.

- true
- false

Answer: a

Explanation: Manhattan distance is an alternative way to calculate distance. It is the distance between two points measured along axes at right angles.

9. What is the optimal time required for solving the closest pair problem using divide and conquer approach?

- a) $O(N)$
- b) $O(\log N)$
- c) $O(N \log N)$
- d) $O(N^2)$

Answer: c

Explanation: The optimal time for solving using a divide and conquer approach is mathematically found to be $O(N \log N)$.

10. In divide and conquer, the time is taken for merging the subproblems is?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(\log N)$

Answer: b

Explanation: The time taken for merging the smaller subproblems in a divide and conquer approach is mathematically found to be $O(N \log N)$.

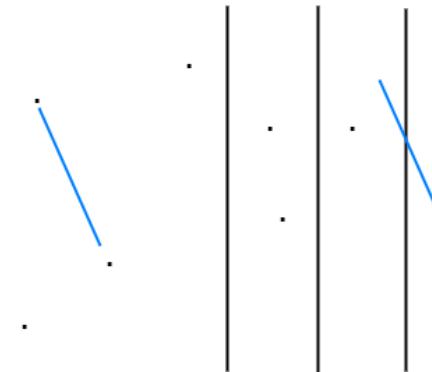
11. The optimal time obtained through divide and conquer approach using merge sort is the best case efficiency.

- a) true
- b) false

Answer: a

Explanation: The optimal time obtained through divide and conquer approach is the best class efficiency and it is given by $\Omega(N \log N)$.

12. Which of the following strategies does the following diagram depict?

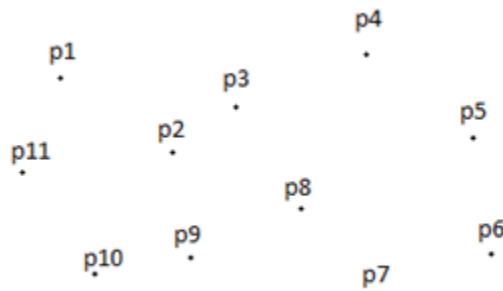


- a) Brute force
- b) Divide and conquer
- c) Exhaustive search
- d) Branch and bound

Answer: b

Explanation: The above diagram depicts the implementation of divide and conquer. The problem is divided into sub problems and are separated by a line.

13. Which of the points are closer to each other?



- a) p1 and p11
- b) p3 and p8
- c) p2 and p3
- d) p9 and p10

Answer: c

Explanation: From symmetry, we determine that the closest pair is p2 and p3. But the exact calculations have to be done using Euclid's algorithm.

-
1. Cross product is a mathematical operation performed between _____
- a) 2 scalar numbers

- b) a scalar and a vector
- c) 2 vectors
- d) any 2 numbers

Answer: c

Explanation: Cross product is a mathematical operation that is performed on 2 vectors in a 3D plane. It has many applications in computer programming and physics.

2. Cross product is also known as?

- a) scalar product
- b) vector product
- c) dot product
- d) multiplication

Answer: b

Explanation: Cross product is also known as a vector product. It is a mathematical operation that is performed on 2 vectors in 3D plane.

3. What is the magnitude of resultant of cross product of two parallel vectors a and b?

- a) $|a|.|b|$
- b) $|a|.|b| \cos(180)$
- c) $|a|.|b| \sin(180)$
- d) 1

Answer: c

Explanation: The resultant of cross product of 2 parallel vectors is always 0 as the angle between them is 0 or 180 degrees. So the answer is $|a|.|b| \sin(180)$.

4. What is the general formula for finding the magnitude of the cross product of two vectors a and b with angle θ between them?

- a) $|a|.|b|$
- b) $|a|.|b| \cos(\theta)$
- c) $|a|.|b| \sin(\theta)$
- d) $|a|.|b| \tan(\theta)$

Answer: c

Explanation: The general formula for finding the magnitude of cross product of two vectors

is $|a|.|b| \sin(\theta)$. Its direction is perpendicular to the plane containing a and b.

5. The concept of cross product is applied in the field of computer graphics.

- a) true
- b) false

Answer: a

Explanation: The concept of cross product find its application in the field of computer graphics. It can be used to find the winding of polygon about a point.

6. Which of the following equals the $a \times b$ (a and b are two vectors)?

- a) $-(a \times b)$
- b) $a.b$
- c) $b \times a$
- d) $-(b \times a)$

Answer: d

Explanation: The vector product $a \times b$ is equal to $-(b \times a)$. The minus sign shows that these vectors have opposite directions.

7. Cross product of two vectors can be used to find?

- a) area of rectangle
- b) area of square
- c) area of parallelogram
- d) perimeter of rectangle

Answer: c

Explanation: Cross product of two vectors can be used to find the area of parallelogram. For this, we need to consider the vectors as the adjacent sides of the parallelogram.

8. The resultant vector from the cross product of two vectors is _____

- a) perpendicular to any one of the two vectors involved in cross product
- b) perpendicular to the plane containing both vectors
- c) parallel to any one of the two vectors involved in cross product

- d) parallel to the plane containing both vectors

Answer: b

Explanation: The resultant vector from the cross product of two vectors is perpendicular to the plane containing both vectors. In other words, it should be perpendicular to both the vectors involved in the cross product.

9. What will be the cross product of the vectors $2\mathbf{i} + 3\mathbf{j} + \mathbf{k}$ and $3\mathbf{i} + 2\mathbf{j} + \mathbf{k}$?

- a) $\mathbf{i} + 2\mathbf{j} + \mathbf{k}$
- b) $2\mathbf{i} + 3\mathbf{j} + \mathbf{k}$
- c) $\mathbf{i} + \mathbf{j} - 5\mathbf{k}$
- d) $2\mathbf{i} - \mathbf{j} - 5\mathbf{k}$

Answer: c

Explanation: We can find the cross product of the given vectors by solving the determinant.

$$\begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 2 & 3 & 1 \\ 3 & 2 & 1 \end{vmatrix}$$

10. What will be the cross product of the vectors $2\mathbf{i} + 3\mathbf{j} + \mathbf{k}$ and $6\mathbf{i} + 9\mathbf{j} + 3\mathbf{k}$?

- a) $\mathbf{i} + 2\mathbf{j} + \mathbf{k}$
- b) $\mathbf{i} - \mathbf{j} - 5\mathbf{k}$
- c) 0
- d) $2\mathbf{i} - \mathbf{j} - 5\mathbf{k}$

Answer: c

Explanation: The given vectors are parallel to each other. The cross product of parallel vectors is 0 because $\sin(0)$ is 0.

11. Find the output of the following code.

```
#include <bits/stdc++.h>
using namespace std;
void crossP(int A[], int B[], int cross[])
{
    cross[0] = A[1] * B[2] - A[2] * B[1];
    cross[1] = A[0] * B[2] - A[2] * B[0];
}
```

```
cross[2] = A[0] * B[1] - A[1] * B[0];
}
int main()
{
    int A[] = { 1, 2, 4 };
    int B[] = { 2, 3, 2 };
    int cross[3];
    crossP(A, B, cross);
    for (int i = 0; i < 3; i++)
        cout << cross[i] << " ";
    return 0;
}

a) 1 2 5
b) -1 -5 -3
c) -6 -8 -1
d) -8 -6 -1
```

Answer: d

Explanation: The given code calculates the cross product of the vectors stored in arrays A and B respectively. So the output will be -8 -6 -1.

12. Which of the following operation will give a vector that is perpendicular to both vectors a and b?

- a) $\mathbf{a} \times \mathbf{b}$
- b) $\mathbf{a} \cdot \mathbf{b}$
- c) $\mathbf{b} \times \mathbf{a}$
- d) both $\mathbf{a} \times \mathbf{b}$ and $\mathbf{b} \times \mathbf{a}$

Answer: d

Explanation: The resultant vector from the cross product of two vectors is perpendicular to the plane containing both vectors. So both $\mathbf{a} \times \mathbf{b}$ and $\mathbf{b} \times \mathbf{a}$ will give a vector that is perpendicular to both vectors a and b.

1. _____ is a method of constructing a smallest polygon out of n given points.
- a) closest pair problem
 - b) quick hull problem
 - c) path compression
 - d) union-by-rank

Answer: b

Explanation: Quick hull is a method of

constructing a smallest convex polygon out of n given points in a plane.

2. What is the other name for quick hull problem?

- a) convex hull
- b) concave hull
- c) closest pair
- d) path compression

Answer: a

Explanation: The other name for quick hull problem is convex hull problem whereas the closest pair problem is the problem of finding the closest distance between two points.

3. How many approaches can be applied to solve quick hull problem?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: Most commonly, two approaches are adopted to solve quick hull problem- brute force approach and divide and conquer approach.

4. What is the average case complexity of a quick hull algorithm?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(\log N)$

Answer: b

Explanation: The average case complexity of quickhull algorithm using divide and conquer approach is mathematically found to be $O(N \log N)$.

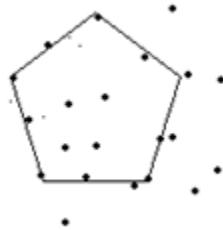
5. What is the worst case complexity of quick hull?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(\log N)$

Answer: c

Explanation: The worst case complexity of quickhull algorithm using divide and conquer approach is mathematically found to be $O(N^2)$.

6. What does the following diagram depict?



- a) closest pair
- b) convex hull
- c) concave hull
- d) path compression

Answer: b

Explanation: The above diagram is a depiction of convex hull, also known as quick hull, since it encloses n points into a convex polygon.

7. Which of the following statement is not related to quickhull algorithm?

- a) finding points with minimum and maximum coordinates
- b) dividing the subset of points by a line
- c) eliminating points within a formed triangle
- d) finding the shortest distance between two points

Answer: d

Explanation: Finding the shortest distance between two points belongs to closest pair algorithm while the rest is quickhull.

8. The quick hull algorithm runs faster if the input uses non-extreme points.

- a) true
- b) false

Answer: a

Explanation: It is proved that the quick hull algorithm runs faster if the input uses non-

extreme points and also, if it uses less memory.

9. To which type of problems does quick hull belong to?

- a) numerical problems
- b) computational geometry
- c) graph problems
- d) string problems

Answer: b

Explanation: Quick hull problem and closest pair algorithms are some of the examples of computational problems.

10. Which of the following algorithms is similar to a quickhull algorithm?

- a) merge sort
- b) shell sort
- c) selection sort
- d) quick sort

Answer: d

Explanation: Quickhull algorithm is similar to a quick sort algorithm with respect to the run time average case and worst case efficiencies.

11. Who formulated quick hull algorithm?

- a) Eddy
- b) Andrew
- c) Chan
- d) Graham

Answer: a

Explanation: Eddy formulated quick hull algorithm. Graham invented graham scan. Andrew formulated Andrew's algorithm and Chan invented Chan's algorithm.

12. The time is taken to find the 'n' points that lie in a convex quadrilateral is?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(\log N)$

Answer: a

Explanation: The time taken to find the 'n' points that lie in a convex quadrilateral is mathematically found to be $O(N)$.

1. Chan's algorithm is used for computing

- a) Closest distance between two points
- b) Convex hull
- c) Area of a polygon
- d) Shortest path between two points

Answer: b

Explanation: Chan's algorithm is an output-sensitive algorithm used to compute the convex hull set of n points in a 2D or 3D space. Closest pair algorithm is used to compute the closest distance between two points.

2. What is the running time of Chan's algorithm?

- a) $O(\log n)$
- b) $O(n \log n)$
- c) $O(n \log h)$
- d) $O(\log h)$

Answer: c

Explanation: The running time of Chan's algorithm is calculated to be $O(n \log h)$ where h is the number of vertices of the convex hull.

3. Who formulated Chan's algorithm?

- a) Timothy
- b) Kirkpatrick
- c) Frank Nielsen
- d) Seidel

Answer: a

Explanation: Chan's algorithm was formulated by Timothy Chan. Kirkpatrick and Seidel formulated the Kirkpatrick-Seidel algorithm. Frank Nielsen developed a paradigm relating to Chan's algorithm.

4. The running time of Chan's algorithm is obtained from combining two algorithms.

- a) True
- b) False

Answer: a

Explanation: The $O(n \log h)$ running time of Chan's algorithm is obtained by combining the running time of Graham's scan [$O(n \log n)$] and Jarvis match [$O(nh)$].

5. Which of the following is called the “ultimate planar convex hull algorithm”?

- a) Chan's algorithm
- b) Kirkpatrick-Seidel algorithm
- c) Gift wrapping algorithm
- d) Jarvis algorithm

Answer: b

Explanation: Kirkpatrick-Seidel algorithm is called as the ultimate planar convex hull algorithm. Its running time is the same as that of Chan's algorithm (i.e.) $O(n \log h)$.

6. Which of the following algorithms is the simplest?

- a) Chan's algorithm
- b) Kirkpatrick-Seidel algorithm
- c) Gift wrapping algorithm
- d) Jarvis algorithm

Answer: a

Explanation: Chan's algorithm is very practical for moderate sized problems whereas Kirkpatrick-Seidel algorithm is not. Although, they both have the same running time. Gift wrapping algorithm is a non-output sensitive algorithm and has a longer running time.

7. What is the running time of Hershberger algorithm?

- a) $O(\log n)$
- b) $O(n \log n)$
- c) $O(n \log h)$
- d) $O(\log h)$

Answer: b

Explanation: Hershberger's algorithm is an output sensitive algorithm whose running

time was originally $O(n \log n)$. He used Chan's algorithm to speed up to $O(n \log h)$ where h is the number of edges.

8. Which of the following statements is not a part of Chan's algorithm?

- a) eliminate points not in the hull
- b) recompute convex hull from scratch
- c) merge previously calculated convex hull
- d) reuse convex hull from the previous iteration

Answer: b

Explanation: Chan's algorithm implies that the convex hulls of larger points can be arrived at by merging previously calculated convex hulls. It makes the algorithm simpler instead of recomputing every time from scratch.

9. Which of the following factors account more to the cost of Chan's algorithm?

- a) computing a single convex hull
- b) locating points that constitute a hull
- c) computing convex hull in groups
- d) merging convex hulls

Answer: c

Explanation: The majority of the cost of the algorithm lies in the pre-processing (i.e.) computing convex hull in groups. To reduce cost, we reuse convex hulls from previous iterations.

10. Chan's algorithm can be used to compute the lower envelope of a trapezoid.

- a) true
- b) false

Answer: a

Explanation: An extension of Chan's algorithm can be used for proving solutions to complex problems like computing the lower envelope $L(S)$ where S is a set of ' n ' line segments in a trapezoid.

1. Which of the following is false in the case of a spanning tree of a graph G?
- It is tree that spans G
 - It is a subgraph of the G
 - It includes every vertex of the G
 - It can be either cyclic or acyclic

Answer: d

Explanation: A graph can have many spanning trees. Each spanning tree of a graph G is a subgraph of the graph G, and spanning trees include every vertex of the graph. Spanning trees are always acyclic.

2. Every graph has only one minimum spanning tree.
- True
 - False

Answer: b

Explanation: Minimum spanning tree is a spanning tree with the lowest cost among all the spanning trees. Sum of all of the edges in the spanning tree is the cost of the spanning tree. There can be many minimum spanning trees for a given graph.

3. Consider a complete graph G with 4 vertices. The graph G has _____ spanning trees.
- 15
 - 8
 - 16
 - 13

Answer: c

Explanation: A graph can have many spanning trees. And a complete graph with n vertices has $n^{(n-2)}$ spanning trees. So, the complete graph with 4 vertices has $4^{(4-2)} = 16$ spanning trees.

4. The travelling salesman problem can be solved using _____
- A spanning tree
 - A minimum spanning tree
 - Bellman – Ford algorithm
 - DFS traversal

Answer: b

Explanation: In the travelling salesman problem we have to find the shortest possible route that visits every city exactly once and returns to the starting point for the given a set of cities. So, travelling salesman problem can be solved by contracting the minimum spanning tree.

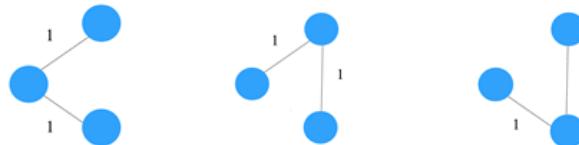
5. Consider the graph M with 3 vertices. Its adjacency matrix is shown below. Which of the following is true?

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

- Graph M has no minimum spanning tree
- Graph M has a unique minimum spanning tree of cost 2
- Graph M has 3 distinct minimum spanning trees, each of cost 2
- Graph M has 3 spanning trees of different costs

Answer: c

Explanation: Here all non-diagonal elements in the adjacency matrix are 1. So, every vertex is connected every other vertex of the graph. And, so graph M has 3 distinct minimum spanning trees.



6. Consider a undirected graph G with vertices { A, B, C, D, E }. In graph G, every edge has distinct weight. Edge CD is edge with minimum weight and edge AB is edge with maximum weight. Then, which of the following is false?

- Every minimum spanning tree of G must contain CD
- If AB is in a minimum spanning tree, then its removal must disconnect G

- c) No minimum spanning tree contains AB
d) G has a unique minimum spanning tree

Answer: c

Explanation: Every MST will contain CD as it is smallest edge. So, Every minimum spanning tree of G must contain CD is true. And G has a unique minimum spanning tree is also true because the graph has edges with distinct weights. So, no minimum spanning tree contains AB is false.

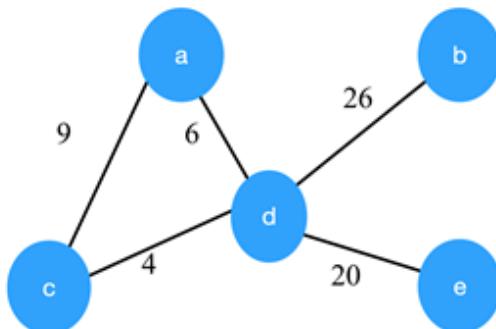
7. If all the weights of the graph are positive, then the minimum spanning tree of the graph is a minimum cost subgraph.

- a) True
b) False

Answer: a

Explanation: A subgraph is a graph formed from a subset of the vertices and edges of the original graph. And the subset of vertices includes all endpoints of the subset of the edges. So, we can say MST of a graph is a subgraph when all weights in the original graph are positive.

8. Consider the graph shown below. Which of the following are the edges in the MST of the given graph?

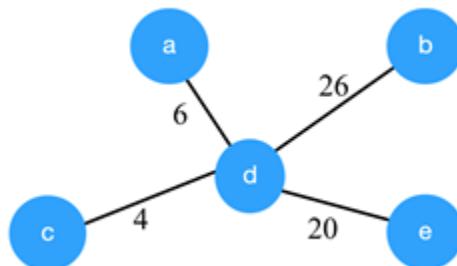


- a) (a-c)(c-d)(d-b)(d-b)
b) (c-a)(a-d)(d-b)(d-e)
c) (a-d)(d-c)(d-b)(d-e)
d) (c-a)(a-d)(d-c)(d-b)(d-e)

Answer: c

Explanation: The minimum spanning tree of the given graph is shown below. It has cost

56.



9. Which of the following is not the algorithm to find the minimum spanning tree of the given graph?

- a) Boruvka's algorithm
b) Prim's algorithm
c) Kruskal's algorithm
d) Bellman–Ford algorithm

Answer: d

Explanation: The Boruvka's algorithm, Prim's algorithm and Kruskal's algorithm are the algorithms that can be used to find the minimum spanning tree of the given graph. The Bellman-Ford algorithm is used to find the shortest path from the single source to all other vertices.

10. Which of the following is false?

- a) The spanning trees do not have any cycles
b) MST have $n - 1$ edges if the graph has n edges
c) Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph
d) Removing one edge from the spanning tree will not make the graph disconnected

Answer: d

Explanation: Every spanning tree has $n - 1$ edges if the graph has n edges and has no cycles. The MST follows the cut property, Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph.

1. Kruskal's algorithm is used to _____
- find minimum spanning tree
 - find single source shortest path
 - find all pair shortest path algorithm
 - traverse the graph

Answer: a

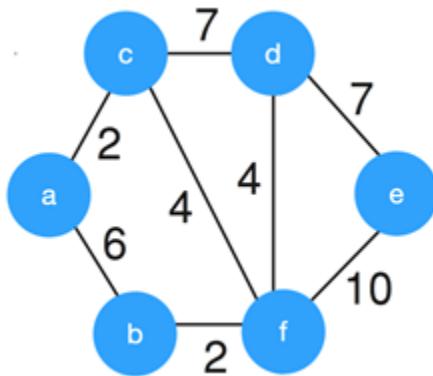
Explanation: The Kruskal's algorithm is used to find the minimum spanning tree of the connected graph. It constructs the MST by finding the edge having the least possible weight that connects two trees in the forest.

2. Kruskal's algorithm is a _____
- divide and conquer algorithm
 - dynamic programming algorithm
 - greedy algorithm
 - approximation algorithm

Answer: c

Explanation: Kruskal's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In the greedy method, we attempt to find an optimal solution in stages.

3. Consider the given graph.



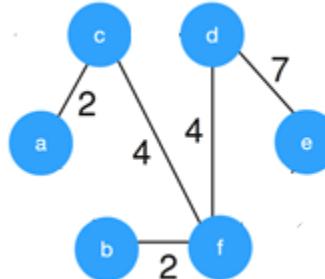
What is the weight of the minimum spanning tree using the Kruskal's algorithm?

- 24
- 23
- 15
- 19

Answer: d

Explanation: Kruskal's algorithm constructs the minimum spanning tree by constructing

by adding the edges to spanning tree one-by-one. The MST for the given graph is,



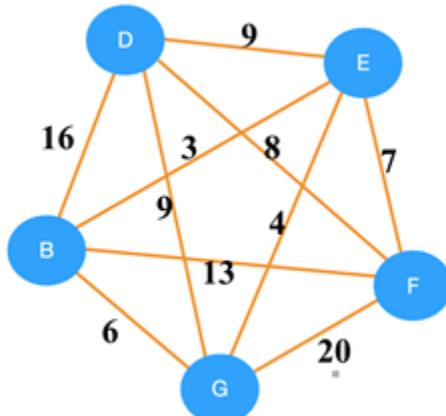
So, the weight of the MST is 19.

4. What is the time complexity of Kruskal's algorithm?
- $O(\log V)$
 - $O(E \log V)$
 - $O(E^2)$
 - $O(V \log E)$

Answer: b

Explanation: Kruskal's algorithm involves sorting of the edges, which takes $O(E \log E)$ time, where E is a number of edges in graph and V is the number of vertices. After sorting, all edges are iterated and union-find algorithm is applied. union-find algorithm requires $O(\log V)$ time. So, overall Kruskal's algorithm requires $O(E \log V)$ time.

5. Consider the following graph. Using Kruskal's algorithm, which edge will be selected first?



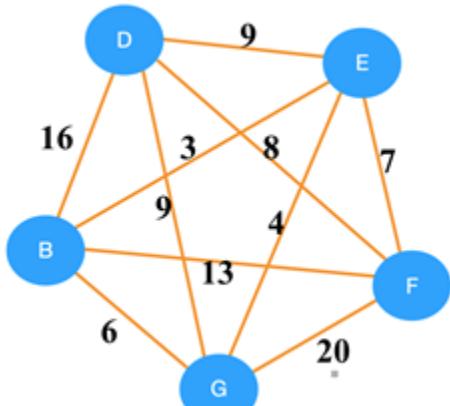
- GF
- DE

- c) BE
d) BG

Answer: c

Explanation: In Kruskal's algorithm the edges are selected and added to the spanning tree in increasing order of their weights. Therefore, the first edge selected will be the minimal one. So, correct option is BE.

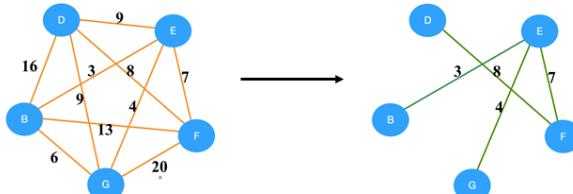
6. Which of the following edges form minimum spanning tree on the graph using kruskals algorithm?



- a) (B-E)(G-E)(E-F)(D-F)
b) (B-E)(G-E)(E-F)(B-G)(D-F)
c) (B-E)(G-E)(E-F)(D-E)
d) (B-E)(G-E)(E-F)(D-F)(D-G)

Answer: a

Explanation: Using Krushkal's algorithm on the given graph, the generated minimum spanning tree is shown below.



So, the edges in the MST are, (B-E)(G-E)(E-F)(D-F).

7. Which of the following is true?

- a) Prim's algorithm can also be used for disconnected graphs
b) Kruskal's algorithm can also run on the

- disconnected graphs
c) Prim's algorithm is simpler than Kruskal's algorithm
d) In Kruskal's sort edges are added to MST in decreasing order of their weights

Answer: b

Explanation: Prim's algorithm iterates from one node to another, so it can not be applied for disconnected graph. Kruskal's algorithm can be applied to the disconnected graphs to construct the minimum cost forest. Kruskal's algorithm is comparatively easier and simpler than prim's algorithm.

8. Which of the following is false about the Kruskal's algorithm?

- a) It is a greedy algorithm
b) It constructs MST by selecting edges in increasing order of their weights
c) It can accept cycles in the MST
d) It uses union-find data structure

Answer: c

Explanation: Kruskal's algorithm is a greedy algorithm to construct the MST of the given graph. It constructs the MST by selecting edges in increasing order of their weights and rejects an edge if it may form the cycle. So, using Kruskal's algorithm is never formed.

9. Kruskal's algorithm is best suited for the dense graphs than the prim's algorithm.

- a) True
b) False

Answer: b

Explanation: Prim's algorithm outperforms the Kruskal's algorithm in case of the dense graphs. It is significantly faster if graph has more edges than the Kruskal's algorithm.

10. Consider the following statements.

- S1. Kruskal's algorithm might produce a non-minimal spanning tree.
S2. Kruskal's algorithm can efficiently implemented using the disjoint-set data structure.

- a) S1 is true but S2 is false
- b) Both S1 and S2 are false
- c) Both S1 and S2 are true
- d) S2 is true but S1 is false

Answer: d

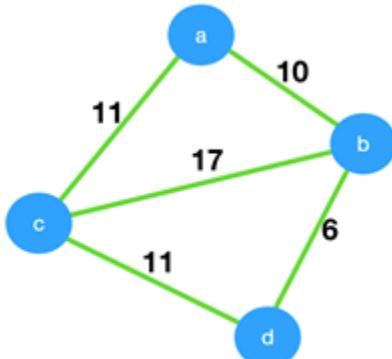
Explanation: In Kruskal's algorithm, the disjoint-set data structure efficiently identifies the components containing a vertex and adds the new edges. And Kruskal's algorithm always finds the MST for the connected graph.

1. Which of the following is true?
- a) Prim's algorithm initialises with a vertex
 - b) Prim's algorithm initialises with a edge
 - c) Prim's algorithm initialises with a vertex which has smallest edge
 - d) Prim's algorithm initialises with a forest

Answer: a

Explanation: Steps in Prim's algorithm: (I) Select any vertex of given graph and add it to MST (II) Add the edge of minimum weight from a vertex not in MST to the vertex in MST; (III) If MST is complete the stop, otherwise go to step (II).

2. Consider the given graph.

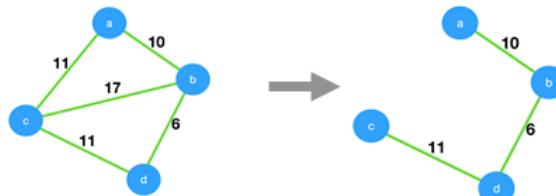


What is the weight of the minimum spanning tree using the Prim's algorithm, starting from vertex a?

- a) 23
- b) 28
- c) 27
- d) 11

Answer: c

Explanation: In Prim's algorithm, we select a vertex and add it to the MST. Then we add the minimum edge from the vertex in MST to vertex not in MST. From, figure shown below weight of MST = 27.



3. Worst case is the worst case time complexity of Prim's algorithm if adjacency matrix is used?

- a) $O(\log V)$
- b) $O(V^2)$
- c) $O(E^2)$
- d) $O(V \log E)$

Answer: b

Explanation: Use of adjacency matrix provides the simple implementation of the Prim's algorithm. In Prim's algorithm, we need to search for the edge with a minimum weight for that vertex. So, worst case time complexity will be $O(V^2)$, where V is the number of vertices.

4. Prim's algorithm is a _____

- a) Divide and conquer algorithm
- b) Greedy algorithm
- c) Dynamic Programming
- d) Approximation algorithm

Answer: b

Explanation: Prim's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In greedy method, we attempt to find an optimal solution in stages.

5. Prim's algorithm resembles Dijkstra's algorithm.

- a) True
- b) False

Answer: a

Explanation: In Prim's algorithm, the MST is constructed starting from a single vertex and adding in new edges to the MST that link the partial tree to a new vertex outside of the MST. And Dijkstra's algorithm also rely on the similar approach of finding the next closest vertex. So, Prim's algorithm resembles Dijkstra's algorithm.

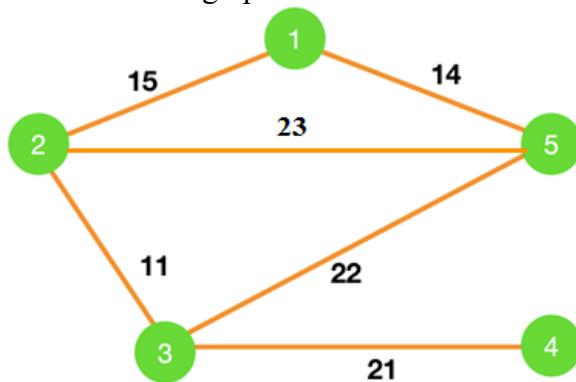
6. Kruskal's algorithm is best suited for the sparse graphs than the prim's algorithm.

- a) True
- b) False

Answer: a

Explanation: Prim's algorithm and Kruskal's algorithm perform equally in case of the sparse graphs. But Kruskal's algorithm is simpler and easy to work with. So, it is best suited for sparse graphs.

7. Consider the graph shown below.



Which of the following edges form the MST of the given graph using Prim'a algorithm, starting from vertex 4.

- a) (4-3)(5-3)(2-3)(1-2)
- b) (4-3)(3-5)(5-1)(1-2)
- c) (4-3)(3-5)(5-2)(1-5)
- d) (4-3)(3-2)(2-1)(1-5)

Answer: d

Explanation: The MST for the given graph using Prim's algorithm starting from vertex 4 is,



So, the MST contains edges (4-3)(3-2)(2-1)(1-5).

8. Prim's algorithm is also known as

- a) Dijksta–Scholten algorithm
- b) Boruvka's algorithm
- c) Floyd–Warshall algorithm
- d) DJP Algorithm

Answer: d

Explanation: The Prim's algorithm was developed by Vojtěch Jarník and it was latter discovered by the duo Prim and Dijkstra. Therefore, Prim's algorithm is also known as DJP Algorithm.

9. Prim's algorithm can be efficiently implemented using _____ for graphs with greater density.

- a) d-ary heap
- b) linear search
- c) fibonacci heap
- d) binary search

Answer: a

Explanation: In Prim's algorithm, we add the minimum weight edge for the chosen vertex which requires searching on the array of weights. This searching can be efficiently implemented using binary heap for dense graphs. And for graphs with greater density, Prim's algorithm can be made to run in linear time using d-ary heap(generalization of binary heap).

10. Which of the following is false about Prim's algorithm?

- a) It is a greedy algorithm
- b) It constructs MST by selecting edges in increasing order of their weights
- c) It never accepts cycles in the MST
- d) It can be implemented using the Fibonacci heap

Answer: b

Explanation: Prim's algorithm can be implemented using Fibonacci heap and it never accepts cycles. And Prim's algorithm follows greedy approach. Prim's algorithms span from one vertex to another.

1. Dijkstra's Algorithm is used to solve _____ problems.

- a) All pair shortest path
- b) Single source shortest path
- c) Network flow
- d) Sorting

Answer: b

Explanation: Dijkstra's Algorithm is used for solving single source shortest path problems. In this algorithm, a single node is fixed as a source node and shortest paths from this node to all other nodes in graph is found.

2. Which of the following is the most commonly used data structure for implementing Dijkstra's Algorithm?

- a) Max priority queue
- b) Stack
- c) Circular queue
- d) Min priority queue

Answer: d

Explanation: Minimum priority queue is the most commonly used data structure for implementing Dijkstra's Algorithm because the required operations to be performed in Dijksta's Algorithm match with specialty of a minimum priority queue.

3. What is the time complexity of Dijikstra's algorithm?

- a) O(N)
- b) O(N^3)
- c) O(N^2)
- d) O(logN)

Answer: c

Explanation: Time complexity of Dijksta's

algorithm is $O(N^2)$ because of the use of doubly nested for loops. It depends on how the table is manipulated.

4. Dijkstra's Algorithm cannot be applied on

- a) Directed and weighted graphs
- b) Graphs having negative weight function
- c) Unweighted graphs
- d) Undirected and unweighted graphs

Answer: b

Explanation: Dijksta's Algorithm cannot be applied on graphs having negative weight function because calculation of cost to reach a destination node from the source node becomes complex.

5. What is the pseudo code to compute the shortest path in Dijksta's algorithm?

a)

```
if(!T[w].Known)
    if(T[v].Dist + C(v,w) < T[w].Dist
) {
    Decrease(T[w].Dist to T[
v].Dist +C(v,w));
    T[w].path=v; }
```

b)

```
if(T[w].Known)
    if(T[v].Dist + C(v,w) < T[w].Dist
) {
        Increase (T[w].Dist to T
[v].Dist +C(v,w));
        T[w].path=v; }
```

c)

```
if(!T[w].Known)
    if(T[v].Dist + C(v,w) > T[w].Dist
) {
        Decrease(T[w].Dist to T[
v].Dist +C(v,w));
        T[w].path=v; }
```

d)

```
if(T[w].Known)
    if(T[v].Dist + C(v,w) < T[w].Dist
) {
```

```

    Increase(T[w].Dist to T[
v].Dist);
T[w].path=v; }

```

Answer: a

Explanation: If the known value of the adjacent vertex(w) is not set then check whether the sum of distance from source vertex(v) and cost to travel from source to adjacent vertex is less than the existing distance of the adjacent node. If so, perform decrease key operation.

6. How many priority queue operations are involved in Dijkstra's Algorithm?

- a) 1
- b) 3
- c) 2
- d) 4

Answer: b

Explanation: The number of priority queue operations involved is 3. They are insert, extract-min and decrease key.

7. How many times the insert and extract min operations are invoked per vertex?

- a) 1
- b) 2
- c) 3
- d) 0

Answer: a

Explanation: Insert and extract min operations are invoked only once per vertex because each vertex is added only once to the set and each edge in the adjacency list is examined only once during the course of algorithm.

8. The maximum number of times the decrease key operation performed in Dijkstra's algorithm will be equal to

-
- a) Total number of vertices
 - b) Total number of edges

- c) Number of vertices – 1
- d) Number of edges – 1

Answer: b

Explanation: If the total number of edges in all adjacency list is E, then there will be a total of E number of iterations, hence there will be a total of at most E decrease key operations.

9. What is running time of Dijkstra's algorithm using Binary min- heap method?

- a) O(V)
- b) O(VlogV)
- c) O(E)
- d) O(ElogV)

Answer: d

Explanation: Time required to build a binary min heap is O(V). Each decrease key operation takes O(logV) and there are still at most E such operations. Hence total running time is O(ElogV).

10. The running time of Bellmann Ford algorithm is lower than that of Dijkstra's Algorithm.

- a) True
- b) False

Answer: b

Explanation: The number of iterations involved in Bellmann Ford Algorithm is more than that of Dijkstra's Algorithm.

11. Dijkstra's Algorithm run on a weighted, directed graph $G=\{V,E\}$ with non-negative weight function w and source s, terminates with $d[u]=\delta(s,u)$ for all vertices u in V.

- a) True
- b) False

Answer: a

Explanation: The equality $d[u]=\delta(s,u)$ holds good when vertex u is added to set S and this equality is maintained thereafter by the upper bound property.

12. Given pseudo code of Dijkstra's Algorithm.

```

1. //Initialise single source(G,s)
2. S=0
3. Q=V[G]
4. While Q != 0
5.   Do u=extract-min(Q)
6.     S=S union {u}
7.     For each vertex v in adj[u]
8.       Do relax(u,v,w)
  
```

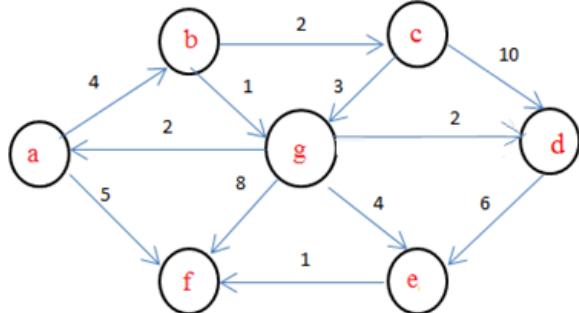
What happens when while loop in line 4 is changed to while $Q > 1$?

- a) While loop gets executed for v times
- b) While loop gets executed for $v-1$ times
- c) While loop gets executed only once
- d) While loop does not get executed

Answer: b

Explanation: In the normal execution of Dijkstra's Algorithm, the while loop gets executed V times. The change in the while loop statement causes it to execute only $V - 1$ times.

13. Consider the following graph.



If b is the source vertex, what is the minimum cost to reach f vertex?

- a) 8
- b) 9
- c) 4
- d) 6

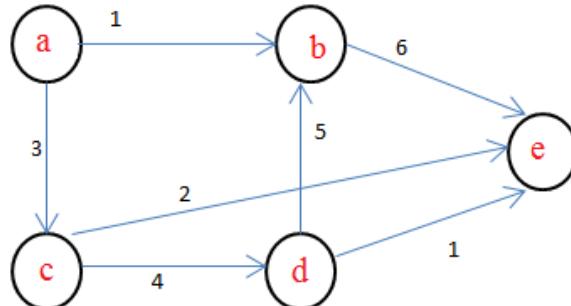
Answer: d

Explanation: The minimum cost to reach f vertex from b vertex is 6 by having vertices g and e as intermediates.

b to g, cost is 1

g to e, cost is 4
e to f, cost is 1
hence total cost $1+4+1=6$.

14. In the given graph:



Identify the shortest path having minimum cost to reach vertex E if A is the source vertex

- a) a-b-e
- b) a-c-e
- c) a-c-d-e
- d) a-c-d-b-e

Answer: b

Explanation: The minimum cost required to travel from vertex A to E is via vertex C
A to C, cost= 3
C to E, cost= 2
Hence the total cost is 5.

15. Dijkstra's Algorithm is the prime example for _____

- a) Greedy algorithm
- b) Branch and bound
- c) Back tracking
- d) Dynamic programming

Answer: a

Explanation: Dijkstra's Algorithm is the prime example for greedy algorithms because greedy algorithms generally solve a problem in stages by doing what appears to be the best thing at each stage.

1. The Bellmann Ford algorithm returns _____ value.

- a) Boolean
- b) Integer

- c) String
- d) Double

Answer: a

Explanation: The Bellmann Ford algorithm returns Boolean value whether there is a negative weight cycle that is reachable from the source.

2. Bellmann ford algorithm provides solution for _____ problems.

- a) All pair shortest path
- b) Sorting
- c) Network flow
- d) Single source shortest path

Answer: d

Explanation: Bellmann ford algorithm is used for finding solutions for single source shortest path problems. If the graph has no negative cycles that are reachable from the source then the algorithm produces the shortest paths and their weights.

3. Bellmann Ford algorithm is used to indicate whether the graph has negative weight cycles or not.

- a) True
- b) False

Answer: a

Explanation: Bellmann Ford algorithm returns true if the graph does not have any negative weight cycles and returns false when the graph has negative weight cycles.

4. How many solution/solutions are available for a graph having negative weight cycle?

- a) One solution
- b) Two solutions
- c) No solution
- d) Infinite solutions

Answer: c

Explanation: If the graph has any negative weight cycle then the algorithm indicates that no solution exists for that graph.

5. What is the running time of Bellmann Ford Algorithm?

- a) $O(V)$
- b) $O(V^2)$
- c) $O(E \log V)$
- d) $O(VE)$

Answer: d

Explanation: Bellmann Ford algorithm runs in time $O(VE)$, since the initialization takes $O(V)$ for each of $V-1$ passes and the for loop in the algorithm takes $O(E)$ time. Hence the total time taken by the algorithm is $O(VE)$.

6. How many times the for loop in the Bellmann Ford Algorithm gets executed?

- a) V times
- b) $V-1$
- c) E
- d) $E-1$

Answer: b

Explanation: The for loop in the Bellmann Ford Algorithm gets executed for $V-1$ times. After making $V-1$ passes, the algorithm checks for a negative weight cycle and returns appropriate boolean value.

7. Dijikstra's Algorithm is more efficient than Bellmann Ford Algorithm.

- a) True
- b) False

Answer: a

Explanation: The running time of Bellmann Ford Algorithm is $O(VE)$ whereas Dijikstra's Algorithm has running time of only $O(V^2)$.

8. Identify the correct Bellmann Ford Algorithm.

- a)

```
for i=1 to V[g]-1
    do for each edge (u,v) in E[g]
        do Relax(u,v,w)
    for each edge (u,v) in E[g]
        do if d[v]>d[u]+w(u,v)
            then return False
return True
```

b)

```

for i=1 to V[g]-1
    for each edge (u,v) in E[g]
        do if d[v]>d[u]+w(u,v)
            then return False
    return True

```

c)

```

for i=1 to V[g]-1
    do for each edge (u,v) in E[g]
        do Relax(u,v,w)
    for each edge (u,v) in E[g]
        do if d[v]<d[u]+w(u,v)
            then return true
    return True

```

d)

```

for i=1 to V[g]-1
    do for each edge (u,v) in E[g]
        do Relax(u,v,w)
    return True

```

Answer: a

Explanation: After initialization, the algorithm makes $V-1$ passes over the edges of the graph. Each pass is one iteration of the for loop and consists of relaxing each edge of the graph once. Then it checks for the negative weight cycle and returns an appropriate Boolean value.

9. What is the basic principle behind Bellmann Ford Algorithm?

- a) Interpolation
- b) Extrapolation
- c) Regression
- d) Relaxation

Answer: d

Explanation: Relaxation methods which are also called as iterative methods in which an approximation to the correct distance is replaced progressively by more accurate values till an optimum solution is found.

10. Bellmann Ford Algorithm can be applied for _____

- a) Undirected and weighted graphs
- b) Undirected and unweighted graphs
- c) Directed and weighted graphs
- d) All directed graphs

Answer: c

Explanation: Bellmann Ford Algorithm can be applied for all directed and weighted graphs. The weight function in the graph may either be positive or negative.

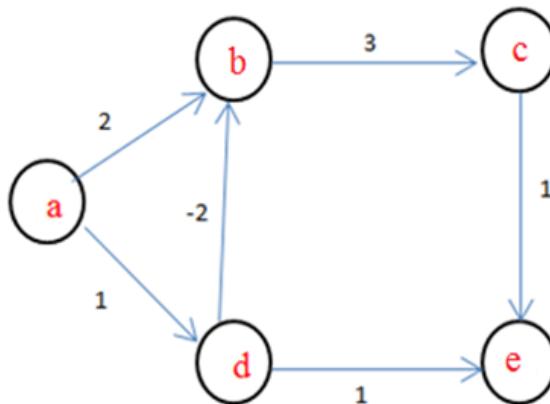
11. Bellmann Ford algorithm was first proposed by _____

- a) Richard Bellmann
- b) Alfonso Shimbe
- c) Lester Ford Jr
- d) Edward F. Moore

Answer: b

Explanation: Alfonso Shimbe proposed Bellmann Ford algorithm in the year 1955. Later it was published by Richard Bellmann in 1957 and Lester Ford Jr in the year 1956. Hence it is called Bellmann Ford Algorithm.

12. Consider the following graph:



What is the minimum cost to travel from node A to node C

- a) 5
- b) 2
- c) 1
- d) 3

Answer: b

Explanation: The minimum cost to travel from node A to node C is 2.

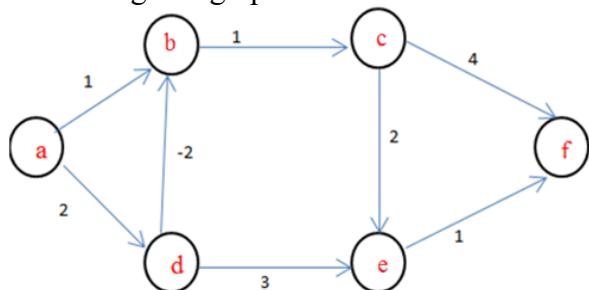
A-D, cost=1

D-B, cost=-2

B-C, cost=3

Hence the total cost is 2.

13. In the given graph:



Identify the path that has minimum cost to travel from node a to node f

- a) a-b-c-f
- b) a-d-e-f
- c) a-d-b-c-f
- d) a-d-b-c-e-f

Answer: d

Explanation: The minimum cost taken by the path a-d-b-c-e-f is 4.

a-d, cost=2

d-b, cost=-2

b-c, cost=1

c-e, cost= 2

e-f, cost=1

Hence the total cost is 4.

14. Bellmann Ford Algorithm is an example for

- a) Dynamic Programming
- b) Greedy Algorithms
- c) Linear Programming
- d) Branch and Bound

Answer: a

Explanation: In Bellmann Ford Algorithm the shortest paths are calculated in bottom up manner which is similar to other dynamic programming problems.

15. A graph is said to have a negative weight cycle when?

- a) The graph has 1 negative weighted edge
- b) The graph has a cycle
- c) The total weight of the graph is negative
- d) The graph has 1 or more negative weighted edges

Answer: c

Explanation: When the total weight of the graph sums up to a negative number then the graph is said to have a negative weight cycle. Bellmann Ford Algorithm provides no solution for such graphs.

1. Floyd Warshall's Algorithm is used for solving _____

- a) All pair shortest path problems
- b) Single Source shortest path problems
- c) Network flow problems
- d) Sorting problems

Answer: a

Explanation: Floyd Warshall's Algorithm is used for solving all pair shortest path problems. It means the algorithm is used for finding the shortest paths between all pairs of vertices in a graph.

2. Floyd Warshall's Algorithm can be applied on _____

- a) Undirected and unweighted graphs
- b) Undirected graphs
- c) Directed graphs
- d) Acyclic graphs

Answer: c

Explanation: Floyd Warshall Algorithm can be applied in directed graphs. From a given directed graph, an adjacency matrix is framed and then all pair shortest path is computed by the Floyd Warshall Algorithm.

3. What is the running time of the Floyd Warshall Algorithm?

- a) Big-oh(V)
- b) Theta(V^2)

- c) Big-Oh(VE)
- d) Theta(V^3)

Answer: d

Explanation: The running time of the Floyd Warshall algorithm is determined by the triply nested for loops. Since each execution of the for loop takes $O(1)$ time, the algorithm runs in time Theta(V^3).

4. What approach is being followed in Floyd Warshall Algorithm?

- a) Greedy technique
- b) Dynamic Programming
- c) Linear Programming
- d) Backtracking

Answer: b

Explanation: Floyd Warshall Algorithm follows dynamic programming approach because the all pair shortest paths are computed in bottom up manner.

5. Floyd Warshall Algorithm can be used for finding _____

- a) Single source shortest path
- b) Topological sort
- c) Minimum spanning tree
- d) Transitive closure

Answer: d

Explanation: One of the ways to compute the transitive closure of a graph in Theta(N^3) time is to assign a weight of 1 to each edge of E and then run the Floyd Warshall Algorithm.

6. What procedure is being followed in Floyd Warshall Algorithm?

- a) Top down
- b) Bottom up
- c) Big bang
- d) Sandwich

Answer: b

Explanation: Bottom up procedure is being used to compute the values of the matrix elements $dij(k)$. The input is an $n \times n$ matrix.

The procedure returns the matrix $D(n)$ of the shortest path weights.

7. Floyd- Warshall algorithm was proposed by _____

- a) Robert Floyd and Stephen Warshall
- b) Stephen Floyd and Robert Warshall
- c) Bernad Floyd and Robert Warshall
- d) Robert Floyd and Bernad Warshall

Answer: a

Explanation: Floyd- Warshall Algorithm was proposed by Robert Floyd in the year 1962. The same algorithm was proposed by Stephen Warshall during the same year for finding the transitive closure of the graph.

8. Who proposed the modern formulation of Floyd-Warshall Algorithm as three nested loops?

- a) Robert Floyd
- b) Stephen Warshall
- c) Bernard Roy
- d) Peter Ingerman

Answer: d

Explanation: The modern formulation of Floyd-Warshall Algorithm as three nested for-loops was proposed by Peter Ingerman in the year 1962.

9. Complete the program.

```

n=rows[W]
D(0)=W
for k=1 to n
    do for i=1 to n
        do for j=1 to n
            do _____
_____
return D(n)

```

- a) $dij(k)=\min(dij(k-1), dik(k-1) - dkj(k-1))$
- b) $dij(k)=\max(dij(k-1), dik(k-1) - dkj(k-1))$
- c) $dij(k)=\min(dij(k-1), dik(k-1) + dkj(k-1))$
- d) $dij(k)=\max(dij(k-1), dik(k-1) + dkj(k-1))$

Answer: c

Explanation: In order to compute the shortest path from vertex i to vertex j , we need to find

the minimum of 2 values which are $dij(k-1)$ and sum of $dik(k-1)$ and $dkj(k-1)$.

10. What happens when the value of k is 0 in the Floyd Warshall Algorithm?

- a) 1 intermediate vertex
- b) 0 intermediate vertex
- c) N intermediate vertices
- d) N-1 intermediate vertices

Answer: b

Explanation: When $k=0$, a path from vertex i to vertex j has no intermediate vertices at all. Such a path has at most one edge and hence $dij(0) = w_{ij}$.

11. Using logical operator's instead arithmetic operators saves time and space.

- a) True
- b) False

Answer: a

Explanation: In computers, logical operations on single bit values execute faster than arithmetic operations on integer words of data.

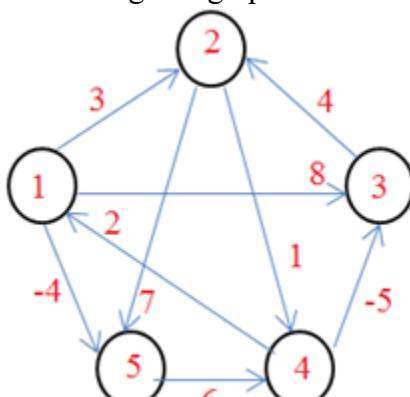
12. The time taken to compute the transitive closure of a graph is $\Theta(n^2)$.

- a) True
- b) False

Answer: b

Explanation: The time taken to compute the transitive closure of a graph is $\Theta(n^3)$. Transitive closure can be computed by assigning weight of 1 to each edge and by running Floyd Warshall Algorithm.

13. In the given graph



What is the minimum cost to travel from vertex 1 to vertex 3?

- a) 3
- b) 2
- c) 10
- d) -3

Answer: d

Explanation: The minimum cost required to travel from node 1 to node 5 is -3.

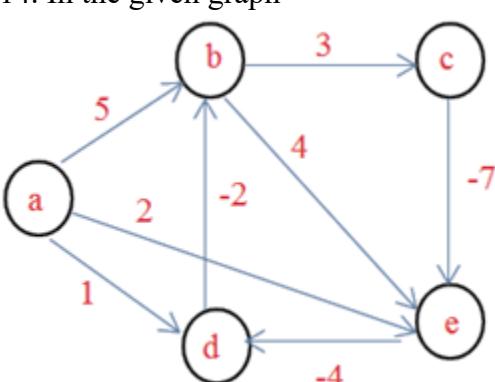
1-5, cost is -4

5-4, cost is 6

4-3, cost is -5

Hence total cost is $-4 + 6 + -5 = -3$.

14. In the given graph



How many intermediate vertices are required to travel from node a to node e at a minimum cost?

- a) 2
- b) 0
- c) 1
- d) 3

Answer: c

Explanation: The minimum cost to travel from node a to node e is 1 by passing via nodes b and c.

a-b, cost 5

b-c, cost 3

c-e, cost -7

Hence the total cost is 1.

15. What is the formula to compute the transitive closure of a graph?

- a) $tij(k) = tij(k-1) \text{ AND } (tik(k-1) \text{ OR } tkj(k-1))$
- b) $tij(k) = tij(k-1) \text{ OR } (tik(k-1) \text{ AND } tkj(k-1))$
- c) $tij(k) = tij(k-1) \text{ AND } (tik(k-1) \text{ AND } tkj(k-1))$
- d) $tij(k) = tij(k-1) \text{ OR } (tik(k-1) \text{ OR } tkj(k-1))$

Answer: b

Explanation: Transitive closure of a graph can be computed by using Floyd Warshall algorithm. This method involves substitution of logical operations (logical OR and logical AND) for arithmetic operations min and + in Floyd Warshall Algorithm.

Floyd Warshall Algorithm: $dij(k) = \min(dij(k-1), dik(k-1) + dkj(k-1))$

Transitive closure: $tij(k) = tij(k-1) \text{ OR } (tik(k-1) \text{ AND } tkj(k-1))$.

UNIT III DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE

1. Which of the following is/are property/properties of a dynamic programming problem?
 - a) Optimal substructure
 - b) Overlapping subproblems
 - c) Greedy approach
 - d) Both optimal substructure and overlapping subproblems

Answer: d

Explanation: A problem that can be solved using dynamic programming possesses overlapping subproblems as well as optimal substructure properties.

2. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses _____ property.

- a) Overlapping subproblems
- b) Optimal substructure
- c) Memoization
- d) Greedy

Answer: b

Explanation: Optimal substructure is the property in which an optimal solution is found for the problem by constructing optimal solutions for the subproblems.

3. If a problem can be broken into subproblems which are reused several times, the problem possesses _____ property.

- a) Overlapping subproblems
- b) Optimal substructure
- c) Memoization
- d) Greedy

Answer: a

Explanation: Overlapping subproblems is the property in which value of a subproblem is used several times.

4. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called

- a) Dynamic programming
- b) Greedy
- c) Divide and conquer
- d) Recursion

Answer: c

Explanation: In divide and conquer, the problem is divided into smaller non-overlapping subproblems and an optimal

solution for each of the subproblems is found. The optimal solutions are then combined to get a global optimal solution. For example, mergesort uses divide and conquer strategy.

5. When dynamic programming is applied to a problem, it takes far less time as compared to other methods that don't take advantage of overlapping subproblems.

- a) True
- b) False

Answer: a

Explanation: Dynamic programming calculates the value of a subproblem only once, while other methods that don't take advantage of the overlapping subproblems property may calculate the value of the same subproblem several times. So, dynamic programming saves the time of recalculation and takes far less time as compared to other methods that don't take advantage of the overlapping subproblems property.

6. A greedy algorithm can be used to solve all the dynamic programming problems.

- a) True
- b) False

Answer: b

Explanation: A greedy algorithm gives optimal solution for all subproblems, but when these locally optimal solutions are combined it may NOT result into a globally optimal solution. Hence, a greedy algorithm CANNOT be used to solve all the dynamic programming problems.

7. In dynamic programming, the technique of storing the previously calculated values is called _____

- a) Saving value property
- b) Storing value property
- c) Memoization
- d) Mapping

Answer: c

Explanation: Memoization is the technique

in which previously calculated values are stored, so that, these values can be used to solve other subproblems.

8. When a top-down approach of dynamic programming is applied to a problem, it usually _____

- a) Decreases both, the time complexity and the space complexity
- b) Decreases the time complexity and increases the space complexity
- c) Increases the time complexity and decreases the space complexity
- d) Increases both, the time complexity and the space complexity

Answer: b

Explanation: The top-down approach uses the memoization technique which stores the previously calculated values. Due to this, the time complexity is decreased but the space complexity is increased.

9. Which of the following problems is NOT solved using dynamic programming?

- a) 0/1 knapsack problem
- b) Matrix chain multiplication problem
- c) Edit distance problem
- d) Fractional knapsack problem

Answer: d

Explanation: The fractional knapsack problem is solved using a greedy algorithm.

10. Which of the following problems should be solved using dynamic programming?

- a) Mergesort
- b) Binary search
- c) Longest common subsequence
- d) Quicksort

Answer: c

Explanation: The longest common subsequence problem has both, optimal substructure and overlapping subproblems. Hence, dynamic programming should be used to solve this problem.

1. The following sequence is a fibonacci sequence:
 0, 1, 1, 2, 3, 5, 8, 13, 21,.....
 Which technique can be used to get the nth fibonacci term?
 a) Recursion
 b) Dynamic programming
 c) A single for loop
 d) Recursion, Dynamic Programming, For loops

Answer: d

Explanation: Each of the above mentioned methods can be used to find the nth fibonacci term.

2. Consider the recursive implementation to find the nth fibonacci number:

```
int fibo(int n)
if n <= 1
    return n
return _____
```

Which line would make the implementation complete?

- a) fibo(n) + fibo(n)
- b) fibo(n) + fibo(n - 1)
- c) fibo(n - 1) + fibo(n + 1)
- d) fibo(n - 1) + fibo(n - 2)

Answer: d

Explanation: Consider the first five terms of the fibonacci sequence: 0,1,1,2,3. The 6th term can be found by adding the two previous terms, i.e. $fibo(6) = fibo(5) + fibo(4) = 3 + 2 = 5$. Therefore, the nth term of a fibonacci sequence would be given by:
 $fibo(n) = fibo(n-1) + fibo(n-2)$.

3. What is the time complexity of the recursive implementation used to find the nth fibonacci term?
- a) O(1)
 - b) $O(n^2)$
 - c) $O(n!)$
 - d) Exponential

Answer: d

Explanation: The recurrence relation is given by $fibo(n) = fibo(n - 1) + fibo(n - 2)$. So, the time complexity is given by:

$$T(n) = T(n - 1) + T(n - 2)$$

Approximately,

$$T(n) = 2 * T(n - 1)$$

$$= 4 * T(n - 2)$$

$$= 8 * T(n - 3)$$

:

:

$$= 2^k * T(n - k)$$

This recurrence will stop when $n - k = 0$

i.e. $n = k$

$$\text{Therefore, } T(n) = 2^n * O(0) = 2^n$$

Hence, it takes exponential time.

It can also be proved by drawing the recursion tree and counting the number of leaves.

4. Suppose we find the 8th term using the recursive implementation. The arguments passed to the function calls will be as follows:

`fibonacci(8)`

`fibonacci(7) + fibonacci(6)`

`fibonacci(6) + fibonacci(5) + fibonacci(5) + fibonacci(4)`

`fibonacci(5) + fibonacci(4) + fibonacci(4) + fibonacci(3) + fibonacci(4) + fibonacci(3) + fibonacci(3) + fibonacci(2)`

:

:

:

Which property is shown by the above function calls?

- a) Memoization
- b) Optimal substructure
- c) Overlapping subproblems
- d) Greedy

Answer: c

Explanation: From the function calls, we can see that fibonacci(4) is calculated twice and fibonacci(3) is calculated thrice. Thus, the same subproblem is solved many times and hence the function calls show the overlapping subproblems property.

5. What is the output of the following program?

```
#include<stdio.h>
int fibo(int n)
{
    if(n<=1)
        return n;
    return fibo(n-1) + fibo(n-2);
}
int main()
{
    int r = fibo(50000);
    printf("%d",r);
    return 0;
}
```

- a) 1253556389
- b) 5635632456
- c) Garbage value
- d) Runtime error

Answer: d

Explanation: The value of n is 50000. The function is recursive and it's time complexity is exponential. So, the function will be called almost 2^{50000} times. Now, even though NO variables are stored by the function, the space required to store the addresses of these function calls will be enormous. Stack memory is utilized to store these addresses and only a particular amount of stack memory can be used by any program. So, after a certain function call, no more stack space will be available and it will lead to stack overflow causing runtime error.

6. What is the space complexity of the recursive implementation used to find the nth fibonacci term?

- a) O(1)
- b) O(n)

c) $O(n^2)$

d) $O(n^3)$

Answer: a

Explanation: The recursive implementation doesn't store any values and calculates every value from scratch. So, the space complexity is $O(1)$.

7. Consider the following code to find the nth fibonacci term:

```
int fibo(int n)
    if n == 0
        return 0
    else
        prevFib = 0
        curFib = 1
        for i : 1 to n-1
            nextFib = prevFib + curFib
            _____
            _____
return curFib
```

Complete the above code.

a)

prevFib = curFib

curFib = curFib

b)

prevFib = nextFib

curFib = prevFib

c)

prevFib = curFib

curFib = nextFib

d)

prevFib = nextFib

nextFib = curFib

Answer: c

Explanation: The lines, prevFib = curFib and curFib = nextFib, make the code complete.

8. What is the time complexity of the following for loop method used to compute the nth fibonacci term?

```
int fibo(int n)
{
    if n == 0
        return 0
    else
        prevFib = 0
        curFib = 1
        for i : 1 to n-1
            nextFib = prevFib + curFib
            prevFib = curFib
            curFib = nextFib
        return curFib
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) Exponential

Answer: b

Explanation: To calculate the nth term, the for loop runs $(n - 1)$ times and each time a for loop is run, it takes a constant time.

Therefore, the time complexity is of the order of n.

9. What is the space complexity of the following for loop method used to compute the nth fibonacci term?

```
int fibo(int n)
{
    if n == 0
        return 0
    else
        prevFib = 0
        curFib = 1
        for i : 1 to n-1
            nextFib = prevFib + curFib
            prevFib = curFib
            curFib = nextFib
        return curFib
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) Exponential

Answer: a

Explanation: To calculate the nth term, we just store the previous term and the current term and then calculate the next term using these two terms. It takes a constant space to store these two terms and hence O(1) is the answer.

10. What will be the output when the following code is executed?

```
#include<stdio.h>
int fibo(int n)
{
    if(n==0)
        return 0;
    int i;
    int prevFib=0,curFib=1;
    for(i=1;i<=n-1;i++)
    {
        int nextFib = prevFib + curFib
    ;
        prevFib = curFib;
        curFib = nextFib;
    }
    return curFib;
}
int main()
{
    int r = fibo(10);
    printf("%d",r);
    return 0;
}
```

- a) 34
- b) 55
- c) Compile error
- d) Runtime error

Answer: b

Explanation: The output is the 10th fibonacci number, which is 55.

11. Consider the following code to find the nth fibonacci term using dynamic programming:

1. int fibo(int n)
2. int fibo_terms[100000] //arr to store the fibonacci numbers
3. fibo_terms[0] = 0
4. fibo_terms[1] = 1
- 5.

```

6.   for i: 2 to n
7.       fibo_terms[i] = fibo_terms[i - 1]
] + fibo_terms[i - 2]
8.
9.   return fibo_terms[n]

```

Which property is shown by line 7 of the above code?

- a) Optimal substructure
- b) Overlapping subproblems
- c) Both overlapping subproblems and optimal substructure
- d) Greedy substructure

Answer: a

Explanation: We find the nth fibonacci term by finding previous fibonacci terms, i.e. by solving subproblems. Hence, line 7 shows the optimal substructure property.

12. Consider the following code to find the nth fibonacci term using dynamic programming:

```

1. int fibo(int n)
2.     int fibo_terms[100000] //arr to
    store the fibonacci numbers
3.     fibo_terms[0] = 0
4.     fibo_terms[1] = 1
5.
6.     for i: 2 to n
7.         fibo_terms[i] = fibo_terms[i - 1]
] + fibo_terms[i - 2]
8.
9.     return fibo_terms[n]

```

Which technique is used by line 7 of the above code?

- a) Greedy
- b) Recursion
- c) Memoization
- d) Overlapping subproblems

Answer: c

Explanation: Line 7 stores the current value that is calculated, so that the value can be used later directly without calculating it from scratch. This is memoization.

13. What is the time complexity of the following dynamic programming

implementation used to compute the nth fibonacci term?

```

1. int fibo(int n)
2.     int fibo_terms[100000] //arr to
    store the fibonacci numbers
3.     fibo_terms[0] = 0
4.     fibo_terms[1] = 1
5.
6.     for i: 2 to n
7.         fibo_terms[i] = fibo_terms[i - 1]
] + fibo_terms[i - 2]
8.
9.     return fibo_terms[n]

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) Exponential

Answer: b

Explanation: To calculate the nth term, the for loop runs ($n - 1$) times and each time a for loop is run, it takes a constant time.

Therefore, the time complexity is of the order of n .

14. What is the space complexity of the following dynamic programming implementation used to compute the nth fibonacci term?

```

1. int fibo(int n)
2.     int fibo_terms[100000] //arr to
    store the fibonacci numbers
3.     fibo_terms[0] = 0
4.     fibo_terms[1] = 1
5.
6.     for i: 2 to n
7.         fibo_terms[i] = fibo_terms[i - 1]
] + fibo_terms[i - 2]
8.
9.     return fibo_terms[n]

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) Exponential

Answer: b

Explanation: To calculate the nth term, we

store all the terms from 0 to $n - 1$. So, it takes $O(n)$ space.

15. What will be the output when the following code is executed?

```
#include<stdio.
int fibo(int n)
{
    int i;
    int fibo_terms[100];
    fibo_terms[0]=0;
    fibo_terms[1]=1;
    for(i=2;i<=n;i++)
        fibo_terms[i] = fibo_terms[i-2]
+ fibo_terms[i-1];
    return fibo_terms[n];
}
int main()
{
    int r = fibo(8);
    printf("%d",r);
    return 0;
}
```

- a) 34
- b) 55
- c) Compile error
- d) 21

Answer: d

Explanation: The program prints the 8th fibonacci term, which is 21.

1. You are given infinite coins of denominations $v_1, v_2, v_3, \dots, v_n$ and a sum S . The coin change problem is to find the minimum number of coins required to get the sum S . This problem can be solved using

- a) Greedy algorithm
- b) Dynamic programming
- c) Divide and conquer
- d) Backtracking

Answer: b

Explanation: The coin change problem has overlapping subproblems(same subproblems are solved multiple times) and optimal substructure(the solution to the problem can

be found by finding optimal solutions for subproblems). So, dynamic programming can be used to solve the coin change problem.

2. Suppose you have coins of denominations 1, 3 and 4. You use a greedy algorithm, in which you choose the largest denomination coin which is not greater than the remaining sum. For which of the following sums, will the algorithm NOT produce an optimal answer?

- a) 20
- b) 12
- c) 6
- d) 5

Answer: c

Explanation: Using the greedy algorithm, three coins {4,1,1} will be selected to make a sum of 6. But, the optimal answer is two coins {3,3}.

3. Suppose you have coins of denominations 1,3 and 4. You use a greedy algorithm, in which you choose the largest denomination coin which is not greater than the remaining sum. For which of the following sums, will the algorithm produce an optimal answer?

- a) 14
- b) 10
- c) 6
- d) 100

Answer: d

Explanation: Using the greedy algorithm, three coins {4,1,1} will be selected to make a sum of 6. But, the optimal answer is two coins {3,3}. Similarly, four coins {4,4,1,1} will be selected to make a sum of 10. But, the optimal answer is three coins {4,3,3}. Also, five coins {4,4,4,1,1} will be selected to make a sum of 14. But, the optimal answer is four coins {4,4,3,3}. For a sum of 100, twenty-five coins {all 4's} will be selected and the optimal answer is also twenty-five coins {all 4's}.

4. Fill in the blank to complete the code.

```
#include<stdio.h>
int main()
{
    int coins[10]={1,3,4}, lookup[100000];
    int i,j,tmp,num_coins = 3,sum=100;
    lookup[0]=0;
    for(i = 1; i <= sum; i++)
    {
        int min_coins = i;
        for(j = 0;j < num_coins; j++)
        {
            tmp = i - coins[j];
            if(tmp < 0)
                continue;
            if(lookup[tmp] < min_coins)
                min_coins = lookup[tmp];
        }
        lookup[i] = min_coins + 1;
    }
    printf("%d",lookup[sum]);
    return 0;
}
```

- a) $\text{lookup}[\text{tmp}] = \text{min_coins}$
- b) $\text{min_coins} = \text{lookup}[\text{tmp}]$
- c) break
- d) continue

Answer: b

Explanation: $\text{min_coins} = \text{lookup}[\text{tmp}]$ will complete the code.

5. You are given infinite coins of N denominations v1, v2, v3, ..., vn and a sum S. The coin change problem is to find the minimum number of coins required to get the sum S. What is the time complexity of a dynamic programming implementation used to solve the coin change problem?

- a) O(N)
- b) O(S)
- c) $O(N^2)$
- d) $O(S^N)$

Answer: d

Explanation: The time complexity is $O(S^N)$.

6. Suppose you are given infinite coins of N denominations v1, v2, v3, ..., vn and a sum S.

The coin change problem is to find the minimum number of coins required to get the sum S. What is the space complexity of a dynamic programming implementation used to solve the coin change problem?

- a) O(N)
- b) O(S)
- c) $O(N^2)$
- d) $O(S^N)$

Answer: b

Explanation: To get the optimal solution for a sum S, the optimal solution is found for each sum less than equal to S and each solution is stored. So, the space complexity is $O(S)$.

7. You are given infinite coins of denominations 1, 3, 4. What is the total number of ways in which a sum of 7 can be achieved using these coins if the order of the coins is not important?

- a) 4
- b) 3
- c) 5
- d) 6

Answer: c

Explanation: A sum of 7 can be achieved in the following ways:

{1,1,1,1,1,1}, {1,1,1,1,3}, {1,3,3}, {1,1,1,4}, {3,4}.

Therefore, the sum can be achieved in 5 ways.

8. You are given infinite coins of denominations 1, 3, 4. What is the minimum number of coins required to achieve a sum of 7?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: A sum of 7 can be achieved by using a minimum of two coins {3,4}.

9. You are given infinite coins of denominations 5, 7, 9. Which of the following sum CANNOT be achieved using these coins?

- a) 50
- b) 21
- c) 13
- d) 23

Answer: c

Explanation: One way to achieve a sum of 50 is to use ten coins of 5. A sum of 21 can be achieved by using three coins of 7. One way to achieve a sum of 23 is to use two coins of 7 and one coin of 9. A sum of 13 cannot be achieved.

10. You are given infinite coins of denominations 3, 5, 7. Which of the following sum CANNOT be achieved using these coins?

- a) 15
- b) 16
- c) 17
- d) 4

Answer: d

Explanation: Sums can be achieved as follows:

$$\begin{aligned} 15 &= \{5,5,5\} \\ 16 &= \{3,3,5,5\} \\ 17 &= \{3,7,7\} \end{aligned}$$

we can't achieve for sum=4 because our available denominations are 3,5,7 and sum of any two denominations is greater than 4.

11. What is the output of the following program?

```
#include<stdio.h>
int main()
{
    int coins[10]={1,3,4},lookup[100];
    int i,j,tmp,num_coins = 3,sum=10;
    lookup[0]=0;
    for(i=1;i<=sum;i++)
    {
        int min_coins = i;
        for(j=0;j<num_coins;j++)
        {
```

```
            tmp=i-coins[j];
            if(tmp<0)
                continue;
            if(lookup[tmp] < min_coins)
                min_coins=lookup[tmp];
        }
        lookup[i] = min_coins + 1;
    }
    printf("%d",lookup[sum]);
    return 0;
}
```

- a) 2
- b) 3
- c) 4
- d) 5

Answer: b

Explanation: The program prints the minimum number of coins required to get a sum of 10, which is 3.

12. What is the output of the following program?

```
#include<stdio.h>
int main()
{
    int coins[10]={1,3,4},lookup[100];
    int i,j,tmp,num_coins = 3,sum=14;
    lookup[0]=0;
    for(i=1;i<=sum;i++)
    {
        int min_coins = i;
        for(j=0;j<num_coins;j++)
        {
            tmp=i-coins[j];
            if(tmp<0)
                continue;
            if(lookup[tmp] < min_coins)
                min_coins=lookup[tmp];
        }
        lookup[i] = min_coins + 1;
    }
    printf("%d",lookup[sum]);
    return 0;
}
```

- a) 2
- b) 3
- c) 4
- d) 5

Answer: c

Explanation: The program prints the minimum number of coins required to get a sum of 14, which is 4.

1. Given a one-dimensional array of integers, you have to find a sub-array with maximum sum. This is the maximum sub-array sum problem. Which of these methods can be used to solve the problem?
 - a) Dynamic programming
 - b) Two for loops (naive method)
 - c) Divide and conquer
 - d) Dynamic programming, naïve method and Divide and conquer methods

Answer: d

Explanation: Dynamic programming, naïve method and Divide and conquer methods can be used to solve the maximum sub array sum problem.

2. Find the maximum sub-array sum for the given elements.

{2, -1, 3, -4, 1, -2, -1, 5, -4}

- a) 3
- b) 5
- c) 8
- d) 6

Answer: b

Explanation: The maximum sub-array sum is 5.

3. Find the maximum sub-array sum for the given elements.

{-2, -1, -3, -4, -1, -2, -1, -5, -4}

- a) -3
- b) 5
- c) 3
- d) -1

Answer: d

Explanation: All the elements are negative. So, the maximum sub-array sum will be equal to the largest element. The largest element is

-1 and therefore, the maximum sub-array sum is -1.

4. Consider the following naive method to find the maximum sub-array sum:

```
#include<stdio.h>
int main()
{
    int arr[1000]={2, -1, 3, -4, 1, -2,
-1, 5, -4}, len=9;
    int cur_max, tmp_max, strt_idx, sub_
arr_idx;
    cur_max = arr[0];
    for(strt_idx = 0; strt_idx < len; st
rt_idx++)
    {
        tmp_max=0;
        for(sub_arr_idx = strt_idx; sub_
arr_idx < len; sub_arr_idx++)
        {
            tmp_max +=arr[sub_arr_idx]
;
            if(tmp_max > cur_max)
                _____;
        }
    }
    printf("%d",cur_max);
    return 0;
}
```

Which line should be inserted to complete the above code?

- a) tmp_max = cur_max
- b) break
- c) continue
- d) cur_max = tmp_max

Answer: d

Explanation: If the tmp_max element is greater than the cur_max element, we make cur_max equal to tmp_max, i.e. cur_max = tmp_max.

5. What is the time complexity of the following naive method used to find the maximum sub-array sum in an array containing n elements?

```
#include<stdio.h>
int main()
{
    int arr[1000]={2, -1, 3, -4, 1, -2,
```

```

-1, 5, -4}, len=9;
    int cur_max, tmp_max, strt_idx, sub_
arr_idx;
    cur_max = arr[0];
    for(strt_idx = 0; strt_idx < len; st
rt_idx++)
    {
        tmp_max=0;
        for(sub_arr_idx = strt_idx; sub
_arr_idx < len; sub_arr_idx++)
        {
            tmp_max +=arr[sub_arr_idx]
;
            if(tmp_max > cur_max)
                _____;
        }
    printf("%d",cur_max);
    return 0;
}

```

- a) O(n^2)
- b) O(n)
- c) O(n^3)
- d) O(1)

Answer: a

Explanation: The naive method uses two for loops. The outer loop runs from 0 to n, i.e. $i = 0 : n$.

The inner loop runs from i to n, i.e. $j = i : n$. Therefore, the inner loop runs:

n times when the outer loop runs the first time.

(n-1) times when the outer loop runs the second time.

:

:

:

2 times when the outer loop runs (n-1)th time.
1 time when the outer loop runs nth time.

Therefore, time complexity = $n + (n-1) + (n-2) + \dots + 2 + 1 = n * (n + 1) / 2 = O(n^2)$.

6. What is the space complexity of the following naive method used to find the maximum sub-array sum in an array containing n elements?

```
#include<stdio.h>
int main()
```

```

{
    int arr[1000]={2, -1, 3, -4, 1, -2,
-1, 5, -4}, len=9;
    int cur_max, tmp_max, strt_idx, sub_
arr_idx;
    cur_max = arr[0];
    for(strt_idx = 0; strt_idx < len; st
rt_idx++)
    {
        tmp_max=0;
        for(sub_arr_idx = strt_idx; sub
_arr_idx < len; sub_arr_idx++)
        {
            tmp_max +=arr[sub_arr_idx]
;
            if(tmp_max > cur_max)
                _____;
        }
    printf("%d",cur_max);
    return 0;
}

```

- a) O(n^2)
- b) O(1)
- c) O(n^3)
- d) O(n)

Answer: b

Explanation: The naive method uses only a constant space. So, the space complexity is O(1).

7. What is the output of the following naive method used to find the maximum sub-array sum?

```
#include<stdio.h>
int main()
{
    int arr[1000] = {-2, -5, 6, -2, 3, -1, 0,-5, 6}, len = 9;
    int cur_max, tmp_max, strt_idx, sub_
arr_idx;
    cur_max = arr[0];
    for(strt_idx = 0; strt_idx < len; st
rt_idx++)
    {
        tmp_max = 0;
        for(sub_arr_idx = strt_idx; su
b_arr_idx < len; sub_arr_idx++)
        {
            tmp_max += arr[sub_arr_id
x];
        }
    }
}
```

```

        if(tmp_max > cur_max)
            cur_max = tmp_max;
    }
    printf("%d",cur_max);
    return 0;
}

```

- a) 6
- b) 9
- c) 7
- d) 4

Answer: c

Explanation: The naive method prints the maximum sub-array sum, which is 7.

8. What is the time complexity of the divide and conquer algorithm used to find the maximum sub-array sum?
- a) $O(n)$
 - b) $O(\log n)$
 - c) $O(n \log n)$
 - d) $O(n^2)$

Answer: c

Explanation: The time complexity of the divide and conquer algorithm used to find the maximum sub-array sum is $O(n \log n)$.

9. What is the space complexity of the divide and conquer algorithm used to find the maximum sub-array sum?
- a) $O(n)$
 - b) $O(1)$
 - c) $O(n!)$
 - d) $O(n^2)$

Answer: b

Explanation: The divide and conquer algorithm uses a constant space. So, the space complexity is $O(1)$.

1. Which line should be inserted in the blank to complete the following dynamic programming implementation of the maximum sub-array sum problem?

```

#include<stdio.h>
int max_num(int a,int b)
{
    if(a> b)
        return a;
    return b;
}
int maximum_subarray_sum(int *arr, int len)
{
    int sum[len], idx;
    sum[0] = arr[0];
    for(idx = 1; idx < len; idx++)
        sum[idx] = _____
    ;
    int mx = sum[0];
    for(idx = 0; idx < len; idx++)
        if(sum[idx] > mx)
            mx =sum[idx];
    return mx;
}
int main()
{
    int arr[] = {-2, -5, 6, -2, 3, -1, 0, -5, 6}, len = 9;
    int ans = maximum_subarray_sum(arr, len);
    printf("%d",ans);
    return 0;
}

a) max_num(sum[idx - 1] + arr[idx], arr[idx])
b) sum[idx - 1] + arr[idx].
c) min_num(sum[idx - 1] + arr[idx], arr[idx])
d) arr[idx].

```

Answer: a

Explanation: The array “sum” is used to store the maximum sub-array sum. The appropriate way to do this is by using: $sum[idx] = max_num(sum[idx - 1] + arr[idx], arr[idx])$.

2. What is the time complexity of the following dynamic programming algorithm used to find the maximum sub-array sum?

```

#include<stdio.h>
int max_num(int a,int b)
{
    if(a> b)
        return a;
    return b;
}

```

```

int maximum_subarray_sum(int *arr, int len)
{
    int sum[len], idx;
    sum[0] = arr[0];
    for(idx = 1; idx < len; idx++)
        sum[idx] = max_num(sum[idx - 1]
+ arr[idx], arr[idx]);
    int mx = sum[0];
    for(idx = 0; idx < len; idx++)
        if(sum[idx] > mx)
            mx = sum[idx];
    return mx;
}
int main()
{
    int arr[] = {-2, -5, 6, -2, 3, -1,
0, -5, 6}, len = 9;
    int ans = maximum_subarray_sum(arr,
len);
    printf("%d", ans);
    return 0;
}

```

- a) O(n)
- b) O(logn)
- c) O(nlogn)
- d) O(n^2)

Answer: a

Explanation: The time complexity of the above dynamic programming algorithm used to solve maximum sub-array sum is O(n).

3. What is the space complexity of the following dynamic programming algorithm used to find the maximum sub-array sum?

```

#include<stdio.h>
int max_num(int a,int b)
{
    if(a > b)
        return a;
    return b;
}
int maximum_subarray_sum(int *arr, int len)
{
    int sum[len], idx;
    sum[0] = arr[0];
    for(idx = 1; idx < len; idx++)
        sum[idx] = max_num(sum[idx - 1]
+ arr[idx], arr[idx]);
    int mx = sum[0];

```

```

        for(idx = 0; idx < len; idx++)
            if(sum[idx] > mx)
                mx = sum[idx];
    return mx;
}
int main()
{
    int arr[] = {-2, -5, 6, -2, 3, -1,
0, -5, 6}, len = 9;
    int ans = maximum_subarray_sum(arr,
len);
    printf("%d", ans);
    return 0;
}

a) O(n)
b) O(1)
c) O(n!)
d) O( $n^2$ )

```

Answer: a

Explanation: The above dynamic programming algorithm uses space equal to the length of the array to store the sum values. So, the space complexity is O(n).

4. Consider the following code snippet:

```

1. int sum[len], idx;
2. sum[0] = arr[0];
3. for(idx = 1; idx < len; idx++)
4.     sum[idx] = max(sum[idx - 1] + a
rr[idx], arr[idx]);
5. int mx = sum[0];
6. for(idx = 0; idx < len; idx++)
7.     if(sum[idx] > mx)
8.         mx = sum[idx];
9. return mx;

```

Which property is shown by line 4 of the above code snippet?

- a) Optimal substructure
- b) Overlapping subproblems
- c) Both overlapping subproblems and optimal substructure
- d) Greedy substructure

Answer: a

Explanation: The current sum (i.e. sum[idx]) uses the previous sum (i.e. sum[idx - 1]) to get an optimal value. So, line 4 shows the optimal substructure property.

5. Consider the following code snippet:

```

1. int sum[len], idx;
2. sum[0] = arr[0];
3. for(idx = 1; idx < len; idx++)
4.     sum[idx] = max(sum[idx - 1] + a
rr[idx], arr[idx]);
5. int mx = sum[0];
6. for(idx = 0; idx < len; idx++)
7.     if(sum[idx] > mx)
8.         mx =sum[idx];
9. return mx;

```

Which method is used by line 4 of the above code snippet?

- a) Divide and conquer
- b) Recursion
- c) Both memoization and divide and conquer
- d) Memoization

Answer: d

Explanation: The array “sum” is used to store the previously calculated values, so that they aren’t recalculated. So, line 4 uses the memoization technique.

6. Find the maximum sub-array sum for the following array:

{3, 6, 7, 9, 3, 8}

- a) 33
- b) 36
- c) 23
- d) 26

Answer: b

Explanation: All the elements of the array are positive. So, the maximum sub-array sum is equal to the sum of all the elements, which is 36.

7. What is the output of the following program?

```
#include<stdio.h>
int max_num(int a,int b)
{
    if(a> b)
        return a;
    return b;
}
int maximum_subarray_sum(int *arr, int le
```

```

n)
{
    int sum[len], idx;
    sum[0] = arr[0];
    for(idx = 1; idx < len; idx++)
        sum[idx] = max_num(sum[idx - 1] +
arr[idx], arr[idx]);
    int mx = sum[0];
    for(idx = 0; idx < len; idx++)
        if(sum[idx] > mx)
            mx =sum[idx];
    return mx;
}
int main()
{
    int arr[] = {-20, 23, 10, 3, -10, 11
, -5},len = 7;
    int ans = maximum_subarray_sum(arr,
len);
    printf("%d",ans);
    return 0;
}

a) 27
b) 37
c) 36
d) 26

```

Answer: b

Explanation: The program prints the value of maximum sub-array sum, which is 37.

8. What is the value stored in sum[4] after the following program is executed?

```
#include<stdio.h>
int max_num(int a,int b)
{
    if(a> b)
        return a;
    return b;
}
int maximum_subarray_sum(int *arr, int le
n)
{
    int sum[len], idx;
    sum[0] = arr[0];
    for(idx = 1; idx < len; idx++)
        sum[idx] = max_num(sum[idx - 1]
+ arr[idx], arr[idx]);
    int mx = sum[0];
    for(idx = 0; idx < len; idx++)
        if(sum[idx] > mx)
            mx =sum[idx];
    return mx;
}
```

```

}
int main()
{
    int arr[] = {-2, 14, 11, -13, 10, -5, 11, -6, 3, -5}, len = 10;
    int ans = maximum_subarray_sum(arr, len);
    printf("%d", ans);
    return 0;
}

```

- a) 28
- b) 25
- c) 22
- d) 12

Answer: c

Explanation: After the program is executed the value stored in sum[4] is 22.

Note: You are asked to find the value stored in sum[4] and NOT the output of the program.

1. Kadane's algorithm is used to find

- a) Longest increasing subsequence
- b) Longest palindrome subsequence
- c) Maximum sub-array sum
- d) Longest decreasing subsequence

Answer: c

Explanation: Kadane's algorithm is used to find the maximum sub-array sum for a given array.

2. Kadane's algorithm uses which of the following techniques?

- a) Divide and conquer
- b) Dynamic programming
- c) Recursion
- d) Greedy algorithm

Answer: b

Explanation: Kadane's algorithm uses dynamic programming.

3. For which of the following inputs would Kadane's algorithm produce the INCORRECT output?

- a) {0,1,2,3}
- b) {-1,0,1}
- c) {-1,-2,-3,0}
- d) {-4,-3,-2,-1}

Answer: d

Explanation: Kadane's algorithm works if the input array contains at least one non-negative element. Every element in the array {-4,-3,-2,-1} is negative. Hence Kadane's algorithm won't work.

4. For which of the following inputs would Kadane's algorithm produce a WRONG output?

- a) {1,0,-1}
- b) {-1,-2,-3}
- c) {1,2,3}
- d) {0,0,0}

Answer: b

Explanation: Kadane's algorithm doesn't work for all negative numbers. So, the answer is {-1,-2,-3}.

5. Complete the following code for Kadane's algorithm:

```

#include<stdio.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int kadane_algo(int *arr, int len)
{
    int ans, sum, idx;
    ans = 0;
    sum = 0;
    for(idx = 0; idx < len; idx++)
    {
        sum = max_num(0, sum + arr[idx]);
        ans = _____;
    }
    return ans;
}
int main()
{
    int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3}, len=7;
    int ans = kadane_algo(arr, len);

```

```

    printf("%d",ans);
    return 0;
}

a) max_num(sum, sum + arr[idx])
b) sum
c) sum + arr[idx]
d) max_num(sum,ans)

```

Answer: d

Explanation: The maximum of sum and ans, is stored in ans. So, the answer is max_num(sum, ans).

6. What is the time complexity of Kadane's algorithm?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(5)

Answer: b

Explanation: The time complexity of Kadane's algorithm is O(n) because there is only one for loop which scans the entire array exactly once.

7. What is the space complexity of Kadane's algorithm?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) None of the mentioned

Answer: a

Explanation: Kadane's algorithm uses a constant space. So, the space complexity is O(1).

8. What is the output of the following implementation of Kadane's algorithm?

```

#include<stdio.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int kadane_algo(int *arr, int len)

```

```

    {
        int ans, sum, idx;
        ans = 0;
        sum = 0;
        for(idx = 0; idx < len; idx++)
        {
            sum = max_num(0,sum + arr[idx]);
            ans = max_num(sum,ans);
        }
        return ans;
    }
    int main()
    {
        int arr[] = {2, 3, -3, -1, 2, 1, 5,
-3}, len = 8;
        int ans = kadane_algo(arr,len);
        printf("%d",ans);
        return 0;
    }

a) 6
b) 7
c) 8
d) 9

```

Answer: d

Explanation: Kadane's algorithm produces the maximum sub-array sum, which is equal to 9.

9. What is the output of the following implementation of Kadane's algorithm?

```

#include<stdio.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int kadane_algo(int *arr, int len)
{
    int ans, sum, idx;
    ans = 0;
    sum = 0;
    for(idx = 0; idx < len; idx++)
    {
        sum = max_num(0,sum + arr[idx]);
    }
    ans = max_num(sum,ans);
    return ans;
}
int main()

```

```

{
    int arr[] = {-2, -3, -3, -1, -2, -1
, -5, -3},len = 8;
    int ans = kadane_algo(arr,len);
    printf("%d",ans);
    return 0;
}

```

- a) 1
- b) -1
- c) -2
- d) 0

Answer: d

Explanation: Kadane's algorithm produces a wrong output when all the elements are negative. The output produced is 0.

10. Consider the following implementation of Kadane's algorithm:

```

1. #include<stdio.h>
2. int max_num(int a, int b)
3. {
4.     if(a > b)
5.         return a;
6.     return b;
7. }
8. int kadane_algo(int *arr, int len)
9. {
10.     int ans = 0, sum = 0, idx;
11.     for(idx = 0; idx < len; idx++)
12.     {
13.         sum = max_num(0,sum + arr
[idx]);
14.         ans = max_num(sum,ans);
15.     }
16.     return ans;
17. }
18. int main()
19. {
20.     int arr[] = {-2, -3, -3, -1, -2
, -1, -5, -3},len = 8;
21.     int ans = kadane_algo(arr,len);
22.     printf("%d",ans);
23.     return 0;
24. }

```

What changes should be made to the Kadane's algorithm so that it produces the right output even when all array elements are negative?

Change 1 = Line 10: int sum = arr[0], ans = arr[0]
Change 2 = Line 13: sum = max_num(arr[idx],sum+arr[idx])

- a) Only Change 1 is sufficient
- b) Only Change 2 is sufficient
- c) Both Change 1 and Change 2 are necessary
- d) No change is required

Answer: c

Explanation: Both change 1 and change 2 should be made to Kadane's algorithm so that it produces the right output even when all the array elements are negative.

1. The longest increasing subsequence problem is a problem to find the length of a subsequence from a sequence of array elements such that the subsequence is sorted in increasing order and its length is maximum. This problem can be solved using

- a) Recursion
- b) Dynamic programming
- c) Brute force
- d) Recursion, Dynamic programming, Brute force

Answer: d

Explanation: The longest increasing subsequence problem can be solved using all of the mentioned methods.

2. Find the longest increasing subsequence for the given sequence:

{10, -10, 12, 9, 10, 15, 13, 14}
a) {10, 12, 15}
b) {10, 12, 13, 14}
c) {-10, 12, 13, 14}
d) {-10, 9, 10, 13, 14}

Answer: d

Explanation: The longest increasing subsequence is {-10, 9, 10, 13, 14}.

3. Find the length of the longest increasing subsequence for the given sequence:

- {-10, 24, -9, 35, -21, 55, -41, 76, 84};
 a) 5
 b) 4
 c) 3
 d) 6

Answer: d

Explanation: The longest increasing subsequence is {-10, 24, 35, 55, 76, 84} and it's length is 6.

4. For any given sequence, there will ALWAYS be a unique increasing subsequence with the longest length.
 a) True
 b) False

Answer: b

Explanation: For a given sequence, it is possible that there is more than one subsequence with the longest length. Consider, the following sequence:

{10,11,12,1,2,3}:

There are two longest increasing subsequences: {1,2,3} and {10,11,12}.

5. The number of increasing subsequences with the longest length for the given sequence are:

- {10, 9, 8, 7, 6, 5}
 a) 3
 b) 4
 c) 5
 d) 6

Answer: d

Explanation: Each array element individually forms a longest increasing subsequence and so, the length of the longest increasing subsequence is 1. So, the number of increasing subsequences with the longest length is 6.

6. In the brute force implementation to find the longest increasing subsequence, all the subsequences of a given sequence are found. All the increasing subsequences are then selected and the length of the longest

subsequence is found. What is the time complexity of this brute force implementation?

- a) O(n)
 b) O(n^2)
 c) O(n!)
 d) O(2^n)

Answer: d

Explanation: The time required to find all the subsequences of a given sequence is 2^n , where 'n' is the number of elements in the sequence. So, the time complexity is O(2^n).

7. Complete the following dynamic programming implementation of the longest increasing subsequence problem:

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing subsequence
    LIS[0]=1;
    for(i = 1; i < len; i++)
    {
        tmp_max = 0;
        for(j = 0; j < i; j++)
        {
            if(arr[j] < arr[i])
            {
                if(LIS[j] > tmp_max)
                    _____;
            }
        }
        LIS[i] = tmp_max + 1;
    }
    int max = LIS[0];
    for(i = 0; i < len; i++)
        if(LIS[i] > max)
            max = LIS[i];
    return max;
}
int main()
{
    int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
    int ans = longest_inc_sub(arr, len);
    printf("%d",ans);
```

```
        return 0;
}
```

- a) tmp_max = LIS[j]
- b) LIS[i] = LIS[j]
- c) LIS[j] = tmp_max
- d) tmp_max = LIS[i]

Answer: a

Explanation: tmp_max is used to store the maximum length of an increasing subsequence for any 'j' such that: arr[j] < arr[i] and $0 < j < i$. So, tmp_max = LIS[j] completes the code.

8. What is the time complexity of the following dynamic programming implementation used to find the length of the longest increasing subsequence?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing subsequence
    LIS[0]=1;
    for(i = 1; i < len; i++)
    {
        tmp_max = 0;
        for(j = 0; j < i; j++)
        {
            if(arr[j] < arr[i])
            {
                if(LIS[j] > tmp_max)
                    tmp_max = LIS[j];
            }
        }
        LIS[i] = tmp_max + 1;
    }
    int max = LIS[0];
    for(i = 0; i < len; i++)
        if(LIS[i] > max)
            max = LIS[i];
    return max;
}
int main()
{
    int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
    int ans = longest_inc_sub(arr, len);
    printf("%d",ans);
}
```

```
        return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(nlogn)

Answer: c

Explanation: The time complexity of the above dynamic programming implementation used to find the length of the longest increasing subsequence is $O(n^2)$.

9. What is the space complexity of the following dynamic programming implementation used to find the length of the longest increasing subsequence?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing subsequence
    LIS[0]=1;
    for(i = 1; i < len; i++)
    {
        tmp_max = 0;
        for(j = 0; j < i; j++)
        {
            if(arr[j] < arr[i])
            {
                if(LIS[j] > tmp_max)
                    tmp_max = LIS[j];
            }
        }
        LIS[i] = tmp_max + 1;
    }
    int max = LIS[0];
    for(i = 0; i < len; i++)
        if(LIS[i] > max)
            max = LIS[i];
    return max;
}
int main()
{
    int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
    int ans = longest_inc_sub(arr, len);
    printf("%d",ans);
}
```

```
    return 0;
}
```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(nlogn)

Answer: b

Explanation: The above dynamic programming implementation uses space equal to the length of the sequence. So, the space complexity of the above dynamic programming implementation used to find the length of the longest increasing subsequence is O(n).

10. What is the output of the following program?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing subsequence
    LIS[0]=1;
    for(i = 1; i < len; i++)
    {
        tmp_max = 0;
        for(j = 0; j < i; j++)
        {
            if(arr[j] < arr[i])
            {
                if(LIS[j] > tmp_max)
                    tmp_max = LIS[j];
            }
            LIS[i] = tmp_max + 1;
        }
        int max = LIS[0];
        for(i = 0; i < len; i++)
            if(LIS[i] > max)
                max = LIS[i];
        return max;
    }
    int main()
    {
        int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
        int ans = longest_inc_sub(arr, len);
        printf("%d",ans);
    }
}
```

```
    return 0;
}
```

- a) 3
- b) 4
- c) 5
- d) 6

Answer: d

Explanation: The program prints the length of the longest increasing subsequence, which is 6.

11. What is the value stored in LIS[5] after the following program is executed?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing subsequence
    LIS[0]=1;
    for(i = 1; i < len; i++)
    {
        tmp_max = 0;
        for(j = 0; j < i; j++)
        {
            if(arr[j] < arr[i])
            {
                if(LIS[j] > tmp_max)
                    tmp_max = LIS[j];
            }
            LIS[i] = tmp_max + 1;
        }
        int max = LIS[0];
        for(i = 0; i < len; i++)
            if(LIS[i] > max)
                max = LIS[i];
        return max;
    }
    int main()
    {
        int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
        int ans = longest_inc_sub(arr, len);
        printf("%d",ans);
        return 0;
    }
}
```

- a) 2
- b) 3
- c) 4
- d) 5

Answer: c

Explanation: The value stored in LIS[5] after the program is executed is 4.

1. Given a rod of length n and the selling prices of all pieces smaller than equal to n, find the most beneficial way of cutting the rod into smaller pieces. This problem is called the rod cutting problem. Which of these methods can be used to solve the rod cutting problem?

- a) Brute force
- b) Dynamic programming
- c) Recursion
- d) Brute force, Dynamic programming and Recursion

Answer: d

Explanation: Brute force, Dynamic programming and Recursion can be used to solve the rod cutting problem.

2. You are given a rod of length 5 and the prices of each length are as follows:

length	price
1	2
2	5
3	6
4	9
5	9

What is the maximum value that you can get after cutting the rod and selling the pieces?

- a) 10
- b) 11
- c) 12
- d) 13

Answer: c

Explanation: The pieces {1,2,2} give the maximum value of 12.

3. Consider the brute force implementation of the rod cutting problem in which all the possible cuts are found and the maximum value is calculated. What is the time complexity of this brute force implementation?

- a) $O(n^2)$
- b) $O(n^3)$
- c) $O(n\log n)$
- d) $O(2^n)$

Answer: d

Explanation: The brute force implementation finds all the possible cuts. This takes $O(2^n)$ time.

4. You are given a rod of length 10 and the following prices.

length	price
1	2
2	5
3	6
4	9
5	9
6	17
7	17
8	18
9	20
10	22

Which of these pieces give the maximum price?

- a) {1,2,7}
- b) {10}
- c) {2,2,6}
- d) {1,4,5}

Answer: c

Explanation: The pieces {2,2,6} give the maximum value of 27.

5. Consider the following recursive implementation of the rod cutting problem:

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int rod_cut(int *prices, int len)
{
    int max_price = INT_MIN; // INT_MIN is the min value an integer can take
    int i;
    if(len <= 0 )
        return 0;
    for(i = 0; i < len; i++)
        max_price = max_of_two(max_price,
______); // subtract 1 because index starts from 0
    return max_price;
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 17,
, 18, 20, 22},len_of_rod = 10;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}
```

Complete the above code.

- a) max_price, prices[i] + rod_cut(prices,len - i - 1)
- b) max_price, prices[i - 1].
- c) max_price, rod_cut(prices, len - i - 1)
- d) max_price, prices[i - 1] + rod_cut(prices,len - i - 1)

Answer: a

Explanation: max_price, prices[i] + rod_cut(prices,len - i - 1) completes the above code.

6. What is the time complexity of the following recursive implementation?

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int rod_cut(int *prices, int len)
{
    int max_price = INT_MIN; // INT_MIN is the min value an integer can take
    int i;
    if(len <= 0 )
        return 0;
    for(i = 0; i < len; i++)
        max_price = max_of_two(max_price
, prices[i] + rod_cut(prices,len - i - 1));
    // subtract 1 because index starts from 0
    return max_price;
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 17
, 18, 20, 22},len_of_rod = 10;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}
```

a) O(n)

b) O(n^2)

c) O(n^3)

d) O(2^n)

Answer: d

Explanation: The time complexity of the above recursive implementation is O(2^n).

7. What is the space complexity of the following recursive implementation?

```
#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int rod_cut(int *prices, int len)
{
```

```

        int max_price = INT_MIN; // INT_MIN
is the min value an integer can take
        int i;
        if(len <= 0 )
            return 0;
        for(i = 0; i < len; i++)
            max_price = max_of_two(max_price
, prices[i] + rod_cut(prices,len - i - 1)
); // subtract 1 because index starts fro
m 0
        return max_price;
    }
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 1
, 18, 20, 22},len_of_rod = 10;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}

```

- a) O(1)
- b) O(logn)
- c) O(nlogn)
- d) O(n!)

Answer: a

Explanation: The space complexity of the above recursive implementation is O(1) because it uses a constant space.

8. What will be the value stored in max_value when the following code is executed?

```

#include<stdio.h>
#include<limits.h>
int max_of_two(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int rod_cut(int *prices, int len)
{
    int max_price = INT_MIN; // INT_MI
N is the min value an integer can take
    int i;
    if(len <= 0 )
        return 0;
    for(i = 0; i < len; i++)
        max_price = max_of_two(prices[i]
] + rod_cut(prices,len - i - 1), max_price
); // subtract 1 because index starts fr
om 0

```

```

        return max_price;
    }
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 1
, 18, 20, 22},len_of_rod = 3;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}

```

- a) 5
- b) 6
- c) 7
- d) 8

Answer: c

Explanation: The value stored in max_value after the code is executed is equal to 7.

9. For every rod cutting problem there will be a unique set of pieces that give the maximum price.

- a) True
- b) False

Answer: b

Explanation: Consider a rod of length 3. The prices are {2,3,6} for lengths {1,2,3} respectively. The pieces {1,1,1} and {3} both give the maximum value of 6.

10. Consider the following dynamic programming implementation of the rod cutting problem:

```

#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
    int max_val[len + 1];
    int i,j,tmp_price,tmp_idx;
    max_val[0] = 0;
    for(i = 1; i <= len; i++)
    {
        int tmp_max = INT_MIN; // mini
mum value an integer can hold
        for(j = 1; j <= i; j++)
        {
            tmp_idx = i - j;
            tmp_price =
        }; //subtract 1 because index of prices s

```

```

        tarts from 0
        if(tmp_price > tmp_max)
            tmp_max = tmp_price;
    }
    max_val[i] = tmp_max;
}
return max_val[len];
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 1
7, 18, 20, 22},len_of_rod = 5;
    int ans = rod_cut(prices, len_of_r
od);
    printf("%d",ans);
    return 0;
}

```

Which line will complete the ABOVE code?

- a) prices[j-1] + max_val[tmp_idx]
- b) prices[j] + max_val[tmp_idx]
- c) prices[j-1] + max_val[tmp_idx - 1]
- d) prices[j] + max_val[tmp_idx - 1]

Answer: a

Explanation: prices[j-1] + max_val[tmp_idx] completes the code.

11. What is the time complexity of the following dynamic programming implementation of the rod cutting problem?

```

#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
    int max_val[len + 1];
    int i,j,tmp_price,tmp_idx;
    max_val[0] = 0;
    for(i = 1; i <= len; i++)
    {
        int tmp_max = INT_MIN; // mini
        mum value an integer can hold
        for(j = 1; j <= i; j++)
        {
            tmp_idx = i - j;
            tmp_price = prices[j-1]
            + max_val[tmp_idx]; //subtract 1 because
            index of prices starts from 0
            if(tmp_price > tmp_max)
                tmp_max = tmp_price;
        }
        max_val[i] = tmp_max;
    }
    return max_val[len];
}

```

```

    }
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 1
7, 18, 20, 22},len_of_rod = 5;
    int ans = rod_cut(prices, len_of_r
od);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(2n)

```

Answer: c

Explanation: The time complexity of the above dynamic programming implementation of the rod cutting problem is O(n²).

12. What is the space complexity of the following dynamic programming implementation of the rod cutting problem?

```

#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
    int max_val[len + 1];
    int i,j,tmp_price,tmp_idx;
    max_val[0] = 0;
    for(i = 1; i <= len; i++)
    {
        int tmp_max = INT_MIN; // mini
        mum value an integer can hold
        for(j = 1; j <= i; j++)
        {
            tmp_idx = i - j;
            tmp_price = prices[j-1]
            + max_val[tmp_idx]; //subtract 1 because
            index of prices starts from 0
            if(tmp_price > tmp_max)
                tmp_max = tmp_price;
        }
        max_val[i] = tmp_max;
    }
    return max_val[len];
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 1
7, 18, 20, 22},len_of_rod = 5;
    int ans = rod_cut(prices, len_of_r
od);
    printf("%d",ans);
    return 0;
}

```

```

od);
    printf("%d",ans);
    return 0;
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(2^n)

Answer: b

Explanation: The space complexity of the above dynamic programming implementation of the rod cutting problem is O(n) because it uses a space equal to the length of the rod.

13. What is the output of the following program?

```

#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
    int max_val[len + 1];
    int i,j,tmp_price,tmp_idx;
    max_val[0] = 0;
    for(i = 1; i <= len; i++)
    {
        int tmp_max = INT_MIN; // minimum value an integer can hold
        for(j = 1; j <= i; j++)
        {
            tmp_idx = i - j;
            tmp_price = prices[j-1] +
max_val[tmp_idx];//subtract 1 because index of prices starts from 0
            if(tmp_price > tmp_max)
                tmp_max = tmp_price;
        }
        max_val[i] = tmp_max;
    }
    return max_val[len];
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 17
, 18, 20, 22},len_of_rod = 5;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}

```

- a) 9
- b) 10
- c) 11
- d) 12

Answer: d

Explanation: The program prints the maximum price that can be achieved by cutting the rod into pieces, which is equal to 27.

14. What is the value stored in max_val[5] after the following program is executed?

```

#include<stdio.h>
#include<limits.h>
int rod_cut(int *prices, int len)
{
    int max_val[len + 1];
    int i,j,tmp_price,tmp_idx;
    max_val[0] = 0;
    for(i = 1; i <= len; i++)
    {
        int tmp_max = INT_MIN; // minimum value an integer can hold
        for(j = 1; j <= i; j++)
        {
            tmp_idx = i - j;
            tmp_price = prices[j-1] +
max_val[tmp_idx];//subtract 1 because index of prices starts from 0
            if(tmp_price > tmp_max)
                tmp_max = tmp_price;
        }
        max_val[i] = tmp_max;
    }
    return max_val[len];
}
int main()
{
    int prices[]={2, 5, 6, 9, 9, 17, 17
, 18, 20, 22},len_of_rod = 5;
    int ans = rod_cut(prices, len_of_rod);
    printf("%d",ans);
    return 0;
}
a) 12
b) 27
c) 10
d) 17

```

Answer: a

Explanation: The value stored in max_val[5] after the program is executed is 12.

1. You are given an array of elements where each array element represents the MAXIMUM number of jumps that can be made in the forward direction from that element. You have to find the minimum number of jumps that are required to reach the end of the array. Which of these methods can be used to solve the problem?
 - a) Dynamic Programming
 - b) Greedy Algorithm
 - c) Recursion
 - d) Recursion and Dynamic Programming

Answer: d

Explanation: Both recursion and dynamic programming can be used to solve minimum number of jumps problem.

2. Consider the following array:

{1, 3, 5, 8, 9, 2, 6, 7, 6}

What is the minimum number of jumps required to reach the end of the array?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: c

Explanation: The jumps made will be: {1 -> 2 -> 4 -> 9}. So, the number of jumps is three.

3. Consider the following recursive implementation:

```
#include<stdio.h>
#include<limits.h>
int min_jumps(int *arr, int strt, int end)
{
    int idx;
    if(strt == end)
        return 0;
    if(arr[strt] == 0) // jump cannot be
made
```

```
        return INT_MAX;
    int min = INT_MAX;
    for(idx = 1; idx <= arr[strt] && str
t + idx <= end; idx++)
    {
        int jumps = min_jumps(____,____
,____) + 1;
        if(jumps < min)
            min = jumps;
    }
    return min;
}
int main()
{
    int arr[] ={1, 3, 5, 8, 9, 2, 6, 7,
6},len = 9;
    int ans = min_jumps(arr, 0, len-1);
    printf("%d\n",ans);
    return 0;
}
```

Which of these arguments should be passed by the min_jumps function represented by the blanks?

- a) arr, strt + idx, end
- b) arr + idx, strt, end
- c) arr, strt, end
- d) arr, strt, end + idx

Answer: a

Explanation: arr, strt + idx, end should be passed as arguments.

4. For a given array, there can be multiple ways to reach the end of the array using minimum number of jumps.

- a) True
- b) False

Answer: a

Explanation: Consider the array {1,2,3,4,5}. It is possible to reach the end in the following ways: {1 -> 2 -> 3 -> 5} or {1 -> 2 -> 4 -> 5}. In both the cases the number of jumps is 3, which is minimum. Hence, it is possible to reach the end of the array in multiple ways using minimum number of jumps.

5. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int min_jumps(int *arr, int strt, int end)
{
    int idx;
    if(strt == end)
        return 0;
    if(arr[strt] == 0) // jump cannot be made
        return INT_MAX;
    int min = INT_MAX;
    for(idx = 1; idx <= arr[strt] && start + idx <= end; idx++)
    {
        int jumps = min_jumps(arr, start + idx, end) + 1;
        if(jumps < min)
            min = jumps;
    }
    return min;
}
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 4, 3, 2, 1}, len = 9;
    int ans = min_jumps(arr, 0, len-1);
    printf("%d\n", ans);
    return 0;
}
```

a) 4
b) 5
c) 6
d) 7

Answer: a

Explanation: The program prints the minimum number of jumps required to reach the end of the array. One way to reach the end using minimum number of jumps is {1 -> 2 -> 4 -> 8 -> 9}. So, the number of jumps is 4.

6. For any array, given that at most one element is non-zero, it is ALWAYS possible to reach the end of the array using minimum jumps.

- a) True
- b) False

Answer: b

Explanation: Consider the array {1,0,2,3,4}.

In this case, only one element is 0 but it is not possible to reach the end of the array.

7. Consider the following dynamic programming implementation of the minimum jumps problem:

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
    int j, idx, jumps[len];
    jumps[len - 1] = 0;
    for(idx = len - 2; idx >= 0; idx--)
    {
        int tmp_min = INT_MAX;
        for(j = 1; j <= arr[idx] && idx + j < len; j++)
        {
            if(jumps[idx + j] + 1 < tmp_min)
                tmp_min = jumps[idx + j] + 1;
        }
        jumps[idx] = tmp_min;
    }
    return jumps[0];
}
int main()
{
    int arr[] = {1, 1, 1, 1, 1, 1, 1, 1, 1}, len = 9;
    int ans = min_jump(arr, len);
    printf("%d\n", ans);
    return 0;
}
```

Which of the following “for” loops can be used instead of the inner for loop so that the output doesn’t change?

- a) for(j = 1; j < arr[idx] + len; j++)
- b) for(j = 0; j < arr[idx] - len; j++)
- c) for(j = idx + 1; j < len && j <= arr[idx] + idx; j++)
- d) No change is required

Answer: d

Explanation: None of the above mentioned “for” loops can be used instead of the inner for loop. Note, for(j = idx + 1; j < len && j <= arr[idx] + idx; j++) covers the same range as the inner for loop but it produces the wrong output because the indexing inside the

loops changes as “j” takes different values in the two “for” loops.

8. What is the time complexity of the following dynamic programming implementation used to find the minimum number of jumps?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
    int j, idx, jumps[len];
    jumps[len - 1] = 0;
    for(idx = len - 2; idx >= 0; idx--)
    {
        int tmp_min = INT_MAX;
        for(j = 1; j <= arr[idx] && idx + j < len; j++)
        {
            if(jumps[idx + j] + 1 < tmp_min)
                tmp_min = jumps[idx + j] + 1;
        }
        jumps[idx] = tmp_min;
    }
    return jumps[0];
}
int main()
{
    int arr[] = {1, 1, 1, 1, 1, 1, 1, 1, 1}, len = 9;
    int ans = min_jump(arr, len);
    printf("%d\n", ans);
    return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) None of the mentioned

Answer: c

Explanation: The time complexity of the above dynamic programming implementation is $O(n^2)$.

9. What is the space complexity of the following dynamic programming implementation used to find the minimum number of jumps?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
    int j, idx, jumps[len];
    jumps[len - 1] = 0;
    for(idx = len - 2; idx >= 0; idx--)
    {
        int tmp_min = INT_MAX;
        for(j = 1; j <= arr[idx] && idx + j < len; j++)
        {
            if(jumps[idx + j] + 1 < tmp_min)
                tmp_min = jumps[idx + j] + 1;
        }
        jumps[idx] = tmp_min;
    }
    return jumps[0];
}
int main()
{
    int arr[] = {1, 1, 1, 1, 1, 1, 1, 1, 1}, len = 9;
    int ans = min_jump(arr, len);
    printf("%d\n", ans);
    return 0;
}
```

a) O(1)
b) O(n)
c) O(n^2)
d) O(5)

Answer: b

Explanation: The space complexity of the above dynamic programming implementation is $O(n)$.

10. What is the output of the following program?

```
#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
    int j, idx, jumps[len];
    jumps[len - 1] = 0;
    for(idx = len - 2; idx >= 0; idx--)
    {
        int tmp_min = INT_MAX;
        for(j = 1; j <= arr[idx] && idx + j < len; j++)
```

```

    {
        if(jumps[idx + j] + 1
< tmp_min)
            tmp_min = jumps[idx
+ j] + 1;
        }
        jumps[idx] = tmp_min;
    }
    return jumps[0];
}
int main()
{
    int arr[] ={1, 1, 1, 1, 1, 1, 1, 1,
1},len = 9;
    int ans = min_jump(arr,len);
    printf("%d\n",ans);
    return 0;
}

```

- a) 7
- b) 8
- c) 9
- d) 10

Answer: b

Explanation: The program prints the minimum jumps required to reach the end of the array, which is 8 and so the output is 8.

11. What is the output of the following program?

```

#include<stdio.h>
#include<limits.h>
int min_jump(int *arr, int len)
{
    int j, idx, jumps[len];
    jumps[len - 1] = 0;
    for(idx = len - 2; idx >= 0; idx--)
    {
        int tmp_min = INT_MAX;
        for(j = 1; j <= arr[idx] && idx
+ j < len; j++)
        {
            if(jumps[idx + j] + 1 < t
mp_min)
                tmp_min = jumps[idx + j
] + 1;
        }
        jumps[idx] = tmp_min;
    }
    return jumps[0];
}
int main()
{

```

```

        int arr[] ={9, 9, 9, 9, 9, 9, 9,
9},len = 9;
        int ans = min_jump(arr,len);
        printf("%d\n",ans);
        return 0;
    }

```

- a) 1
- b) 6
- c) 2
- d) 7

Answer: a

Explanation: The program prints the minimum jumps required to reach the end of the array, which is 1 and so the output is 1.

1. The Knapsack problem is an example of

- a) Greedy algorithm
- b) 2D dynamic programming
- c) 1D dynamic programming
- d) Divide and conquer

Answer: b

Explanation: Knapsack problem is an example of 2D dynamic programming.

2. Which of the following methods can be used to solve the Knapsack problem?

- a) Brute force algorithm
- b) Recursion
- c) Dynamic programming
- d) Brute force, Recursion and Dynamic Programming

Answer: d

Explanation: Brute force, Recursion and Dynamic Programming can be used to solve the knapsack problem.

3. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?

- a) 160
- b) 200

- c) 170
d) 90

Answer: a

Explanation: The maximum value you can get is 160. This can be achieved by choosing the items 1 and 3 that have a total weight of 60.

4. Which of the following problems is equivalent to the 0-1 Knapsack problem?
- You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of $\{w_1, w_2, w_3, \dots, w_n\}$ and a value of $\{v_1, v_2, v_3, \dots, v_n\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value
 - You are studying for an exam and you have to study N questions. The questions take $\{t_1, t_2, t_3, \dots, t_n\}$ time(in hours) and carry $\{m_1, m_2, m_3, \dots, m_n\}$ marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized
 - You are given infinite coins of denominations $\{v_1, v_2, v_3, \dots, v_n\}$ and a sum S. You have to find the minimum number of coins required to get the sum S
 - You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of $\{10, 20, 15, 40\}$ and a value of $\{1, 2, 3, 4\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

Answer: b

Explanation: In this case, questions are used instead of items. Each question has a score which is same as each item having a value. Also, each question takes a time t which is same as each item having a weight w. You have to maximize the score in time T which is same as maximizing the value using a bag of weight W.

5. What is the time complexity of the brute force algorithm used to solve the Knapsack

- problem?
a) $O(n)$
b) $O(n!)$
c) $O(2^n)$
d) $O(n^3)$

Answer: c

Explanation: In the brute force algorithm all the subsets of the items are found and the value of each subset is calculated. The subset of items with the maximum value and a weight less than equal to the maximum allowed weight gives the answer. The time taken to calculate all the subsets is $O(2^n)$.

6. The 0-1 Knapsack problem can be solved using Greedy algorithm.

- True
- False

Answer: b

Explanation: The Knapsack problem cannot be solved using the greedy algorithm.

7. Consider the following dynamic programming implementation of the Knapsack problem:

```
#include<stdio.h>
int find_max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
    int ans[n + 1][W + 1];
    int itm,w;
    for(itm = 0; itm <= n; itm++)
        ans[itm][0] = 0;
    for(w = 0;w <= W; w++)
        ans[0][w] = 0;
    for(itm = 1; itm <= n; itm++)
    {
        for(w = 1; w <= W; w++)
        {
            if(wt[itm - 1] <= w)
                ans[itm][w] = _____;
            else
                ans[itm][w] = _____;
        }
    }
}
```

```

        else
            ans[item][w] = ans[item -
1][w];
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100
, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}

```

Which of the following lines completes the above code?

- a) find_max(ans[item - 1][w - wt[item - 1]] + val[item - 1], ans[item - 1][w])
- b) find_max(ans[item - 1][w - wt[item - 1]], ans[item - 1][w])
- c) ans[item][w] = ans[item - 1][w];
- d) ans[item+1][w] = ans[item - 1][w];

Answer: a

Explanation: find_max(ans[item - 1][w - wt[item - 1]] + val[item - 1], ans[item - 1][w]) completes the above code.

8. What is the time complexity of the following dynamic programming implementation of the Knapsack problem with n items and a maximum weight of W?

```

#include<stdio.h>
int find_max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int knapsack(int W, int *wt, int *val,int
n)
{
    int ans[n + 1][W + 1];
    int item,w;
    for(item = 0; item <= n; item++)
        ans[item][0] = 0;
    for(w = 0;w <= W; w++)
        ans[0][w] = 0;
    for(item = 1; item <= n; item++)
    {
        for(w = 1; w <= W; w++)

```

```

        {
            if(wt[item - 1] <= w)
                ans[item][w] = find_max(
ans[item - 1][w - wt[item - 1]] + val[item -
1], ans[item - 1][w]);
            else
                ans[item][w] = ans[item -
1][w];
        }
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100
, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}

a) O(n)
b) O(n + w)
c) O(nW)
d) O(n2)

```

Answer: c

Explanation: The time complexity of the above dynamic programming implementation of the Knapsack problem is O(nW).

9. What is the space complexity of the following dynamic programming implementation of the Knapsack problem?

```

#include<stdio.h>
int find_max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int knapsack(int W, int *wt, int *val,int
n)
{
    int ans[n + 1][W + 1];
    int item,w;
    for(item = 0; item <= n; item++)
        ans[item][0] = 0;
    for(w = 0;w <= W; w++)
        ans[0][w] = 0;
    for(item = 1; item <= n; item++)
    {
        for(w = 1; w <= W; w++)

```

```

    {
        if(wt[item - 1] <= w)
            ans[item][w] = find_max(
ans[item - 1][w - wt[item - 1]] + val[item -
1], ans[item - 1][w]);
        else
            ans[item][w] = ans[item -
1][w];
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100
, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}

```

- a) O(n)
- b) O(n + w)
- c) O(nW)
- d) O(n²)

Answer: c

Explanation: The space complexity of the above dynamic programming implementation of the Knapsack problem is O(nW).

10. What is the output of the following code?

```

#include<stdio.h>
int find_max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int knapsack(int W, int *wt, int *val,int n)
{
    int ans[n + 1][W + 1];
    int item,w;
    for(item = 0; item <= n; item++)
        ans[item][0] = 0;
    for(w = 0;w <= W; w++)
        ans[0][w] = 0;
    for(item = 1; item <= n; item++)
    {
        for(w = 1; w <= W; w++)
        {
            if(wt[item - 1] <= w)
                ans[item][w] = find_max(

```

```

ans[item - 1][w - wt[item - 1]] + val[item -
1], ans[item - 1][w]);
            else
                ans[item][w] = ans[item -
1][w];
        }
    }
    return ans[n][W];
}
int main()
{
    int w[] = {10,20,30}, v[] = {60, 100
, 120}, W = 50;
    int ans = knapsack(W, w, v, 3);
    printf("%d",ans);
    return 0;
}

```

- a) 120
- b) 100
- c) 180
- d) 220

Answer: d

Explanation: The output of the above code is 220.

1. Which of the following methods can be used to solve the matrix chain multiplication problem?

- a) Dynamic programming
- b) Brute force
- c) Recursion
- d) Dynamic Programming, Brute force, Recursion

Answer: d

Explanation: Dynamic Programming, Brute force, Recursion methods can be used to solve the matrix chain multiplication problem.

2. Which of the following is the recurrence relation for the matrix chain multiplication problem where mat[i-1] * mat[i] gives the dimension of the ith matrix?

- a) dp[i,j] = 1 if i=j
dp[i,j] = min{dp[i,k] + dp[k+1,j]}
- b) dp[i,j] = 0 if i=j
dp[i,j] = min{dp[i,k] + dp[k+1,j]}

- c) $dp[i,j] = 1$ if $i=j$
 $dp[i,j] = \min\{dp[i,k] + dp[k+1,j]\} + mat[i-1]*mat[k]*mat[j].$
- d) $dp[i,j] = 0$ if $i=j$
 $dp[i,j] = \min\{dp[i,k] + dp[k+1,j]\} + mat[i-1]*mat[k]*mat[j].$

Answer: d

Explanation: The recurrence relation is given by:

- $dp[i,j] = 0$ if $i=j$
- $dp[i,j] = \min\{dp[i,k] + dp[k+1,j]\} + mat[i-1]*mat[k]*mat[j].$

3. Consider the two matrices P and Q which are 10×20 and 20×30 matrices respectively. What is the number of multiplications required to multiply the two matrices?

- a) $10*20$
- b) $20*30$
- c) $10*30$
- d) $10*20*30$

Answer: d

Explanation: The number of multiplications required is $10*20*30$.

4. Consider the matrices P, Q and R which are 10×20 , 20×30 and 30×40 matrices respectively. What is the minimum number of multiplications required to multiply the three matrices?

- a) 18000
- b) 12000
- c) 24000
- d) 32000

Answer: a

Explanation: The minimum number of multiplications are 18000. This is the case when the matrices are parenthesized as $(P*Q)*R$.

5. Consider the matrices P, Q, R and S which are 20×15 , 15×30 , 30×5 and 5×40 matrices respectively. What is the minimum number of multiplications required to multiply the four matrices?

- a) 6050
- b) 7500
- c) 7750
- d) 12000

Answer: c

Explanation: The minimum number of multiplications required is 7750.

6. Consider the brute force implementation in which we find all the possible ways of multiplying the given set of n matrices. What is the time complexity of this implementation?

- a) $O(n!)$
- b) $O(n^3)$
- c) $O(n^2)$
- d) Exponential

Answer: d

Explanation: The time complexity of finding all the possible ways of multiplying a set of n matrices is given by $(n-1)^{\text{th}}$ Catalan number which is exponential.

7. Consider the following dynamic programming implementation of the matrix chain problem:

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, int n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len + 1; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1; k++)
            {
                int tmp = _____;
                if(tmp < arr[row][col]
```

```

])           arr[row][col] = tmp;
        }
    }
    return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,5,30,10,40};
    int ans = mat_chain_multiplication(m
at,5);
    printf("%d",ans);
    return 0;
}

```

Which of the following lines should be inserted to complete the above code?

- a) arr[row][k] – arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col];
- b) arr[row][k] + arr[k + 1][col] – mat[row – 1] * mat[k] * mat[col];
- c) arr[row][k] + arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col];
- d) arr[row][k] – arr[k + 1][col] – mat[row – 1] * mat[k] * mat[col];

Answer: c

Explanation: The line arr[row][k] + arr[k + 1][col] + mat[row – 1] * mat[k] * mat[col] should be inserted to complete the above code.

8. What is the time complexity of the following dynamic programming implementation of the matrix chain problem?

```

#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len +
1; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
        }
    }
}

```

```

for(k = row; k <= col - 1;
{
    int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
    if(tmp < arr[row][col])
        arr[row][col] = tmp;
}
return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,5,30,10,40};
    int ans = mat_chain_multiplication(m
at,5);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(n3)

```

Answer: d

Explanation: The time complexity of the above dynamic programming implementation of the matrix chain multiplication is O(n³).

9. What is the space complexity of the following dynamic programming implementation of the matrix chain problem?

```

#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len +
1; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1;

```

```

k++)
{
    int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
    if(tmp < arr[row][col])
        arr[row][col] = tmp;
}
return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,5,30,10,40};
    int ans = mat_chain_multiplication(m
at,5);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O(n)
c) O( $n^2$ )
d) O( $n^3$ )

```

Answer: c

Explanation: The space complexity of the above dynamic programming implementation of the matrix chain multiplication is $O(n^2)$.

10. What is the output of the following code?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len + 1
; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1;
k++)
            {
                int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
                if(tmp < arr[row][col])
                    arr[row][col] = tmp;
            }
        }
    }
}
```

```

+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
if(tmp < arr[row][col])
arr[row][col] = tmp;
}
return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,30,40,50};
    int ans = mat_chain_multiplication(m
at,4);
    printf("%d",ans);
    return 0;
}
```

- a) 64000
- b) 70000
- c) 120000
- d) 150000

Answer: a

Explanation: The program prints the minimum number of multiplications required, which is 64000.

11. What is the value stored in arr[2][3] when the following code is executed?

```
#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len + 1
; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1;
k++)
            {
                int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
                if(tmp < arr[row][co
```

```

    1])
        arr[row][col] = tmp;
    }
}
return arr[1][n - 1];
}
int main()
{
    int mat[6] = {20,30,40,50};
    int ans = mat_chain_multiplication(m
at,4);
    printf("%d",ans);
    return 0;
}

a) 64000
b) 60000
c) 24000
d) 12000

```

Answer: b

Explanation: The value stored in arr[2][3] when the above code is executed is 60000.

12. What is the output of the following code?

```

#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len + 1
; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1;
k++)
            {
                int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
                if(tmp < arr[row][col
])
                    arr[row][col] = tmp;
            }
        }
    }
}

```

```

    return arr[1][n-1];
}
int main()
{
    int mat[6] = {10,10,10,10,10,10};
    int ans = mat_chain_multiplication(m
at,6);
    printf("%d",ans);
    return 0;
}

a) 2000
b) 3000
c) 4000
d) 5000

```

Answer: c

Explanation: The program prints the minimum number of multiplications required to multiply the given matrices, which is 4000.

13. What is the output of the following code?

```

#include<stdio.h>
#include<limits.h>
int mat_chain_multiplication(int *mat, in
t n)
{
    int arr[n][n];
    int i,k,row,col,len;
    for(i=1;i<n;i++)
        arr[i][i] = 0;
    for(len = 2; len < n; len++)
    {
        for(row = 1; row <= n - len + 1
; row++)
        {
            col = row + len - 1;
            arr[row][col] = INT_MAX;
            for(k = row; k <= col - 1;
k++)
            {
                int tmp = arr[row][k]
+ arr[k + 1][col] + mat[row - 1] * mat[k]
* mat[col];
                if(tmp < arr[row][col
])
                    arr[row][col] = tmp;
            }
        }
    }
    return arr[1][n-1];
}
int main()
{

```

```

int mat[6] = {20,25,30,35,40};
int ans = mat_chain_multiplication(m
at,5);
printf("%d",ans);
return 0;
}

```

- a) 32000
- b) 28000
- c) 64000
- d) 70000

Answer: c

Explanation: The output of the program is 64000.

1. Which of the following methods can be used to solve the longest common subsequence problem?
 - a) Recursion
 - b) Dynamic programming
 - c) Both recursion and dynamic programming
 - d) Greedy algorithm

Answer: c

Explanation: Both recursion and dynamic programming can be used to solve the longest subsequence problem.

2. Consider the strings “PQRSTPQRS” and “PRATPBRQRPS”. What is the length of the longest common subsequence?

- a) 9
- b) 8
- c) 7
- d) 6

Answer: c

Explanation: The longest common subsequence is “PRTPQRS” and its length is 7.

3. Which of the following problems can be solved using the longest subsequence problem?

- a) Longest increasing subsequence
- b) Longest palindromic subsequence

- c) Longest bitonic subsequence
- d) Longest decreasing subsequence

Answer: b

Explanation: To find the longest palindromic subsequence in a given string, reverse the given string and then find the longest common subsequence in the given string and the reversed string.

4. Longest common subsequence is an example of _____
 - a) Greedy algorithm
 - b) 2D dynamic programming
 - c) 1D dynamic programming
 - d) Divide and conquer

Answer: b

Explanation: Longest common subsequence is an example of 2D dynamic programming.

5. What is the time complexity of the brute force algorithm used to find the longest common subsequence?

- a) $O(n)$
- b) $O(n^2)$
- c) $O(n^3)$
- d) $O(2^n)$

Answer: d

Explanation: The time complexity of the brute force algorithm used to find the longest common subsequence is $O(2^n)$.

6. Consider the following dynamic programming implementation of the longest common subsequence problem:

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;

```

```

len1 = strlen(str1);
len2 = strlen(str2);
int arr[len1 + 1][len2 + 1];
for(i = 0; i <= len1; i++)
    arr[i][0] = 0;
for(i = 0; i <= len2; i++)
    arr[0][i] = 0;
for(i = 1; i <= len1; i++)
{
    for(j = 1; j <= len2; j++)
    {
        if(str1[i-1] == str2[j - 1])
            _____;
        else
            arr[i][j] = max_num(ar
r[i - 1][j], arr[i][j - 1]);
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = " abcedfg", str2[] =
"bcdfh";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}

```

Which of the following lines completes the above code?

- a) $\text{arr}[i][j] = 1 + \text{arr}[i][j]$.
- b) $\text{arr}[i][j] = 1 + \text{arr}[i - 1][j - 1]$.
- c) $\text{arr}[i][j] = \text{arr}[i - 1][j - 1]$.
- d) $\text{arr}[i][j] = \text{arr}[i][j]$.

Answer: b

Explanation: The line, $\text{arr}[i][j] = 1 + \text{arr}[i - 1][j - 1]$ completes the above code.

7. What is the time complexity of the following dynamic programming implementation of the longest common subsequence problem where length of one string is “m” and the length of the other string is “n”?

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
}

```

```

        return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len2; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(ar
r[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = " abcedfg", str2[] =
"bcdfh";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}

```

- a) O(n)
- b) O(m)
- c) O(m + n)
- d) O(mn)

Answer: d

Explanation: The time complexity of the above dynamic programming implementation of the longest common subsequence is O(mn).

8. What is the space complexity of the following dynamic programming implementation of the longest common subsequence problem where length of one string is “m” and the length of the other string is “n”?

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len2; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = " abcdedfg", str2[] = "bcdfh";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}

```

- a) O(n)
- b) O(m)
- c) O(m + n)
- d) O(mn)

Answer: d

Explanation: The space complexity of the above dynamic programming implementation of the longest common subsequence is O(mn).

9. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len2; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = "hbcfgmnapq", str2[] = "cbhgrsfnmq";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}

```

- a) 3
- b) 4
- c) 5
- d) 6

Answer: b

Explanation: The program prints the length of the longest common subsequence, which is 4.

10. Which of the following is the longest common subsequence between the strings “hbcfgmnapq” and “cbhgrsfnmq” ?

- a) hgmq
- b) cfnq
- c) bfmq
- d) fgmna

Answer: d

Explanation: The length of the longest common subsequence is 4. But 'fgmna' is not the longest common subsequence as its length is 5.

11. What is the value stored in arr[2][3] when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len2; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = "hbcfgmnapq", str2[]
= "cbhgrsfnmq";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}
```

- a) 1
- b) 2
- c) 3
- d) 4

Answer: a

Explanation: The value stored in arr[2][3] is 1.

12. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lcs(char *str1, char *str2)
{
    int i,j,len1,len2;
    len1 = strlen(str1);
    len2 = strlen(str2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len2; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(ar
r[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len1][len2];
}
int main()
{
    char str1[] = "abcd", str2[] = "efg
h";
    int ans = lcs(str1,str2);
    printf("%d",ans);
    return 0;
}
```

- a) 3
- b) 2

- c) 1
d) 0

Answer: d

Explanation: The program prints the length of the longest common subsequence, which is 0.

1. Which of the following methods can be used to solve the longest palindromic subsequence problem?
 a) Dynamic programming
 b) Recursion
 c) Brute force
 d) Dynamic programming, Recursion, Brute force

Answer: d

Explanation: Dynamic programming, Recursion, Brute force can be used to solve the longest palindromic subsequence problem.

2. Which of the following is not a palindromic subsequence of the string “ababcdabba”?
 a) abcba
 b) abba
 c) abbbba
 d) adba

Answer: d

Explanation: ‘adba’ is not a palindromic sequence.

3. For which of the following, the length of the string is not equal to the length of the longest palindromic subsequence?
 a) A string that is a palindrome
 b) A string of length one
 c) A string that has all the same letters(e.g. aaaaaa)
 d) Some strings of length two

Answer: d

Explanation: A string of length 2 for eg: ab is not a palindrome.

4. What is the length of the longest palindromic subsequence for the string “ababcdabba”?
 a) 6
 b) 7
 c) 8
 d) 9

Answer: b

Explanation: The longest palindromic subsequence is “abbabba” and its length is 7.

5. What is the time complexity of the brute force algorithm used to find the length of the longest palindromic subsequence?
 a) $O(1)$
 b) $O(2^n)$
 c) $O(n)$
 d) $O(n^2)$

Answer: b

Explanation: In the brute force algorithm, all the subsequences are found and the length of the longest palindromic subsequence is calculated. This takes exponential time.

6. For every non-empty string, the length of the longest palindromic subsequence is at least one.
 a) True
 b) False

Answer: a

Explanation: A single character of any string can always be considered as a palindrome and its length is one.

7. Longest palindromic subsequence is an example of _____
 a) Greedy algorithm
 b) 2D dynamic programming
 c) 1D dynamic programming
 d) Divide and conquer

Answer: b

Explanation: Longest palindromic subsequence is an example of 2D dynamic programming.

8. Consider the following code:

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    _____;
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len][len];
}
int main()
{
    char str1[] = "ababcdabba";
    int ans = lps(str1);
    printf("%d",ans);
    return 0;
}
```

Which of the following lines completes the above code?

- a) strrev(str2);
- b) str2 = str1
- c) len2 = strlen(str2)
- d) strlen(str2)

Answer: a

Explanation: To find the longest palindromic

subsequence, we need to reverse the copy of the string, which is done by strrev.

9. What is the time complexity of the following dynamic programming implementation to find the longest palindromic subsequence where the length of the string is n?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    strrev(str2);
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len][len];
}
int main()
{
    char str1[] = "ababcdabba";
    int ans = lps(str1);
    printf("%d",ans);
    return 0;
}
```

a) O(n)

b) O(1)

- c) $O(n^2)$
d) $O(2)$

Answer: c

Explanation: The time complexity of the above dynamic programming implementation to find the longest palindromic subsequence is $O(n^2)$.

10. What is the space complexity of the following dynamic programming implementation to find the longest palindromic subsequence where the length of the string is n?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    strrev(str2);
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len][len];
}
int main()
{
    char str1[] = "ababcdabba";
    int ans = lps(str1);
```

```
    printf("%d",ans);
    return 0;
}
```

- a) $O(n)$
b) $O(1)$
c) $O(n^2)$
d) $O(2)$

Answer: c

Explanation: The space complexity of the above dynamic programming implementation to find the longest palindromic subsequence is $O(n^2)$.

11. What is the value stored in $arr[3][3]$ when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    strrev(str2);
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
    }
    return arr[len][len];
}
int main()
```

```

{
    char str1[] = "ababcdabba";
    int ans = lps(str1);
    printf("%d",ans);
    return 0;
}

a) 2
b) 3
c) 4
d) 5

```

Answer: a

Explanation: The value stored in arr[3][3] when the above code is executed is 2.

12. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    strrev(str2);
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
        return arr[len][len];
}
int main()
{

```

```

    char str1[] = "abcd";
    int ans = lps(str1);
    printf("%d",ans);
    return 0;
}

a) 0
b) 1
c) 2
d) 3

```

Answer: b

Explanation: The program prints the length of the longest palindromic subsequence, which is 1.

13. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int lps(char *str1)
{
    int i,j,len;
    len = strlen(str1);
    char str2[len + 1];
    strcpy(str2, str1);
    strrev(str2);
    int arr[len + 1][len + 1];
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(str1[i-1] == str2[j - 1])
                arr[i][j] = 1 + arr[i - 1][j - 1];
            else
                arr[i][j] = max_num(arr[i - 1][j], arr[i][j - 1]);
        }
        return arr[len][len];
}
int main()
{

```

```

char str1[] = "abdkgkagdjbjccbbba";
int ans = lps(str1);
printf("%d",ans);
return 0;
}

a) 5
b) 7
c) 9
d) 11

```

Answer: c

Explanation: The program prints the length of the longest palindromic subsequence, which is 9.

1. Which of the following methods can be used to solve the edit distance problem?
 - a) Recursion
 - b) Dynamic programming
 - c) Both dynamic programming and recursion
 - d) Greedy Algorithm

Answer: c

Explanation: Both dynamic programming and recursion can be used to solve the edit distance problem.

2. The edit distance satisfies the axioms of a metric when the costs are non-negative.
 - a) True
 - b) False

Answer: a

Explanation: $d(s,s) = 0$, since each string can be transformed into itself without any change. $d(s_1, s_2) > 0$ when $s_1 \neq s_2$, since the transformation would require at least one operation.

$$\begin{aligned} d(s_1, s_2) &= d(s_2, s_1) \\ d(s_1, s_3) &\leq d(s_1, s_2) + d(s_2, s_3) \end{aligned}$$

Thus, the edit distance satisfies the axioms of a metric.

3. Which of the following is an application of the edit distance problem?
 - a) Approximate string matching
 - b) Spelling correction

- c) Similarity of DNA
- d) Approximate string matching, Spelling Correction and Similarity of DNA

Answer: d

Explanation: All of the mentioned are the applications of the edit distance problem.

4. In which of the following cases will the edit distance between two strings be zero?
 - a) When one string is a substring of another
 - b) When the lengths of the two strings are equal
 - c) When the two strings are equal
 - d) The edit distance can never be zero

Answer: c

Explanation: The edit distance will be zero only when the two strings are equal.

5. Suppose each edit (insert, delete, replace) has a cost of one. Then, the maximum edit distance cost between the two strings is equal to the length of the larger string.
 - a) True
 - b) False

Answer: a

Explanation: Consider the strings “abcd” and “efghi”. The string “efghi” can be converted to “abcd” by deleting “i” and converting “efgh” to “abcd”. The cost of transformation is 5, which is equal to the length of the larger string.

6. Consider the strings “monday” and “tuesday”. What is the edit distance between the two strings?
 - a) 3
 - b) 4
 - c) 5
 - d) 6

Answer: b

Explanation: “monday” can be converted to “tuesday” by replacing “m” with “t”, “o” with “u”, “n” with “e” and inserting “s” at the appropriate position. So, the edit distance is 4.

7. Consider the two strings “”(empty string) and “abcd”. What is the edit distance between the two strings?

- a) 0
- b) 4
- c) 2
- d) 3

Answer: b

Explanation: The empty string can be transformed into “abcd” by inserting “a”, “b”, “c” and “d” at appropriate positions. Thus, the edit distance is 4.

8. Consider the following dynamic programming implementation of the edit distance problem:

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1, len2, i, j, min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],
arr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < mi
n)
                    min = arr[i-1][j-1]
                else
                {
                    if(arr[i-1][j-1] + 1
< min)
                        min = arr[i-1][j]
```

```
-1] + 1;
            }
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "defg";
    int ans = edit_distance(s1, s2);
    printf("%d", ans);
    return 0;
}
```

Which of the following lines should be added to complete the above code?

- a) arr[i-1][j] = min
- b) arr[i][j-1] = min
- c) arr[i-1][j-1] = min
- d) arr[i][j] = min

Answer: d

Explanation: The line arr[i][j] = min completes the above code.

9. What is the time complexity of the following dynamic programming implementation of the edit distance problem where “m” and “n” are the lengths of two strings?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1, len2, i, j, min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],
```

```

        min = get_min(arr[i-1][j],
arr[i][j-1]) + 1;
        if(s1[i - 1] == s2[j - 1])
        {
            if(arr[i-1][j-1] < mi
n)
                min = arr[i-1][j-1
];
            else
            {
                if(arr[i-1][j-1] + 1
< min)
                    min = arr[i-1][j
-1] + 1;
            }
            arr[i][j] = min;
        }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "defg";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}

```

- a) O(1)
- b) O(m + n)
- c) O(mn)
- d) O(n)

Answer: c

Explanation: The time complexity of the above dynamic programming implementation of the edit distance problem is O(mn).

10. What is the space complexity of the following dynamic programming implementation of the edit distance problem where “m” and “n” are the lengths of the two strings?

```

#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)

```

```

        int len1,len2,i,j,min;
        len1 = strlen(s1);
        len2 = strlen(s2);
        int arr[len1 + 1][len2 + 1];
        for(i = 0;i <= len1; i++)
            arr[i][0] = i;
        for(i = 0; i <= len2; i++)
            arr[0][i] = i;
        for(i = 1; i <= len1; i++)
        {
            for(j = 1; j <= len2; j++)
            {
                min = get_min(arr[i-1][j],
arr[i][j-1]) + 1;
                if(s1[i - 1] == s2[j - 1])
                {
                    if(arr[i-1][j-1] < mi
n)
                        min = arr[i-1][j-1
];
                }
                else
                {
                    if(arr[i-1][j-1] + 1
< min)
                        min = arr[i-1][j
-1] + 1;
                }
                arr[i][j] = min;
            }
        }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "defg";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}

```

- a) O(1)
- b) O(m + n)
- c) O(mn)
- d) O(n)

Answer: c

Explanation: The space complexity of the above dynamic programming implementation of the edit distance problem is O(mn).

11. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 < min)
                    min = arr[i-1][j-1] + 1;
            }
            arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "defg";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}

a) 1
b) 2
c) 3
d) 4

```

Answer: d

Explanation: The program prints the edit distance between the strings “abcd” and “defg”, which is 4.

12. What is the value stored in arr[2][2] when the following code is executed?

```

#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],arr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 < min)
                    min = arr[i-1][j-1] + 1;
            }
            arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "abcd", s2[] = "defg";
    int ans = edit_distance(s1, s2);

```

```

        printf("%d",ans);
        return 0;
    }
}

```

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: The value stored in arr[2][2] when the above code is executed is 2.

13. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],a
rr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 <
min)
                    min = arr[i-1][j-1]
+ 1;
            }
            arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}

```

```

    }
int main()
{
    char s1[] = "pqrstuv", s2[] = "prstu
v";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}

```

- a) 1
- b) 2
- c) 3
- d) 4

Answer: a

Explanation: The code prints the edit distance between the two strings, which is 1.

1. Wagner–Fischer is a _____ algorithm.

- a) Brute force
- b) Greedy
- c) Dynamic programming
- d) Recursive

Answer: c

Explanation: Wagner–Fischer belongs to the dynamic programming type of algorithms.

2. Wagner–Fischer algorithm is used to find

-
- a) Longest common subsequence
 - b) Longest increasing subsequence
 - c) Edit distance between two strings
 - d) Longest decreasing subsequence

Answer: c

Explanation: Wagner–Fischer algorithm is used to find the edit distance between two strings.

3. What is the edit distance between the strings “abcd” and “acbd” when the allowed operations are insertion, deletion and substitution?

- a) 1
- b) 2

- c) 3
d) 4

Answer: b

Explanation: The string “abcd” can be changed to “acbd” by substituting “b” with “c” and “c” with “b”. Thus, the edit distance is 2.

4. For which of the following pairs of strings is the edit distance maximum?

- a) sunday & monday
b) monday & tuesday
c) tuesday & wednesday
d) wednesday & thursday

Answer: d

Explanation: The edit distances are 2, 4, 4 and 5 respectively. Hence, the maximum edit distance is between the strings wednesday and thursday.

5. Consider the following implementation of the Wagner–Fischer algorithm:

```
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1, len2, i, j, min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0; i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j], a
rr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    _____;
            }
            else
```

```
{           if(arr[i-1][j-1] + 1 <
min)
            min = arr[i-1][j-1]
+ 1;
        }
        arr[i][j] = min;
    }
    return arr[len1][len2];
}
```

Which of the following lines should be inserted to complete the above code?

- a) arr[i][j] = min
b) min = arr[i-1][j-1] – 1;
c) min = arr[i-1][j-1].
d) min = arr[i-1][j-1] + 1;

Answer: c

Explanation: The line min = arr[i-1][j-1] completes the above code.

6. What is the time complexity of the Wagner–Fischer algorithm where “m” and “n” are the lengths of the two strings?

- a) O(1)
b) O(n+m)
c) O(mn)
d) O(nlogm)

Answer: c

Explanation: The time complexity of the Wagner–Fischer algorithm is O(mn).

7. What is the space complexity of the above implementation of Wagner–Fischer algorithm where “m” and “n” are the lengths of the two strings?

- a) O(1)
b) O(n+m)
c) O(mn)
d) O(nlogm)

Answer: c

Explanation: The space complexity of the above Wagner–Fischer algorithm is O(mn).

8. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],
arr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 <
min)
                    min = arr[i-1][j-1] + 1;
            }
            arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "somestring", s2[] = "anotherthing";
    int ans = edit_distance(s1, s2);
    printf("%d",ans);
    return 0;
}

a) 6
b) 7
c) 8
d) 9
```

Answer: a

Explanation: The program prints the edit distance between the strings “somestring” and “anotherthing”, which is 6.

9. What is the value stored in arr[3][3] when the below code is executed?

```
#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],
arr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 <
min)
                    min = arr[i-1][j-1] + 1;
            }
            arr[i][j] = min;
        }
    }
    return arr[len1][len2];
}
int main()
{
    char s1[] = "somestring", s2[] = "anotherthing";
```

```

int ans = edit_distance(s1, s2);
printf("%d",ans);
return 0;
}

a) 1
b) 2
c) 3
d) 4

```

Answer: c

Explanation: The value stored in arr[3][3] is 3.

10. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int edit_distance(char *s1, char *s2)
{
    int len1,len2,i,j,min;
    len1 = strlen(s1);
    len2 = strlen(s2);
    int arr[len1 + 1][len2 + 1];
    for(i = 0;i <= len1; i++)
        arr[i][0] = i;
    for(i = 0; i <= len2; i++)
        arr[0][i] = i;
    for(i = 1; i <= len1; i++)
    {
        for(j = 1; j <= len2; j++)
        {
            min = get_min(arr[i-1][j],a
rr[i][j-1]) + 1;
            if(s1[i - 1] == s2[j - 1])
            {
                if(arr[i-1][j-1] < min)
                    min = arr[i-1][j-1];
            }
            else
            {
                if(arr[i-1][j-1] + 1 <
min)
                    min = arr[i-1][j-1];
            }
            arr[i][j] = min;
        }
    }
}

```

```

        return arr[len1][len2];
    }
    int main()
    {
        char s1[] = "abcd", s2[] = "dcba";
        int ans = edit_distance(s1, s2);
        printf("%d",ans);
        return 0;
    }

```

a) 1

b) 2

c) 3

d) 4

Answer: d

Explanation: The program prints the edit distance between the strings “abcd” and “dcba”, which is 4.

1. Which of the following is NOT a Catalan number?

a) 1

b) 5

c) 14

d) 43

Answer: d

Explanation: Catalan numbers are given by: $(2n!)/((n+1)!n!)$.

For $n = 0$, we get $C_0 = 1$.

For $n = 3$, we get $C_3 = 5$.

For $n = 4$, we get $C_4 = 14$.

For $n = 5$, we get $C_5 = 42$.

2. Which of the following numbers is the 6th Catalan number?

a) 14

b) 429

c) 132

d) 42

Answer: d

Explanation: Catalan numbers are given by: $(2n!)/((n+1)!n!)$.

First Catalan number is given by $n = 0$.

So the 6th Catalan number will be given by $n = 5$, which is 42.

3. Which of the following is not an application of Catalan Numbers?
- Counting the number of Dyck words
 - Counting the number of expressions containing n pairs of parenthesis
 - Counting the number of ways in which a convex polygon can be cut into triangles by connecting vertices with straight lines
 - Creation of head and tail for a given number of tosses

Answer: d

Explanation: Counting the number of Dyck words, Counting the number of expressions containing n pairs of parenthesis, Counting the number of ways in which a convex polygon can be cut into triangles by connecting vertices with straight lines are the applications of Catalan numbers where as creation of head and tails for a given number of tosses is an application of Pascal's triangle.

4. Which of the following methods can be used to find the nth Catalan number?
- Recursion
 - Binomial coefficients
 - Dynamic programming
 - Recursion, Binomial Coefficients, Dynamic programming

Answer: d

Explanation: All of the mentioned methods can be used to find the nth Catalan number.

5. The recursive formula for Catalan number is given by $C_n = \sum C_i * C_{n-i}$. Consider the following dynamic programming implementation for Catalan numbers:

```
#include<stdio.h>
int cat_number(int n)
{
    int i,j,arr[n],k;
    arr[0] = 1;
    for(i = 1; i < n; i++)
    {
        arr[i] = 0;
        for(j = 0,k = i - 1; j < i; j++,
k--)

```

```
    }
    return arr[n-1];
}

int main()
{
    int ans, n = 8;
    ans = cat_number(n);
    printf("%d\n",ans);
    return 0;
}
```

Which of the following lines completes the above code?

- $arr[i] = arr[j] * arr[k];$
- $arr[j] += arr[i] * arr[k];$
- $arr[i] += arr[j] * arr[k].$
- $arr[j] = arr[i] * arr[k];$

Answer: c

Explanation: The line $arr[i] += arr[j] * arr[k]$ reflects the recursive formula $C_n = \sum C_i * C_{n-i}$.

6. Which of the following implementations of Catalan numbers has the smallest time complexity?
- Dynamic programming
 - Binomial coefficients
 - Recursion
 - All have equal time complexity

Answer: b

Explanation: The time complexities are as follows:

Dynamic programming: $O(n^2)$

Recursion: Exponential

Binomial coefficients: $O(n)$.

7. What is the output of the following code?

```
#include<stdio.h>
int cat_number(int n)
{
    int i,j,arr[n],k;
    arr[0] = 1;
    for(i = 1; i < n; i++)
    {
        arr[i] = 0;
        for(j = 0,k = i - 1; j < i; j++,
k--)
```

```

        arr[i] += arr[j] * arr[k];
    }
    return arr[n-1];
}

```

```

int main()
{
    int ans, n = 8;
    ans = cat_number(n);
    printf("%d\n", ans);
    return 0;
}

```

- a) 42
- b) 132
- c) 429
- d) 1430

Answer: c

Explanation: The program prints the 8th Catalan number, which is 429.

8. Which of the following implementations of Catalan numbers has the largest space complexity(Don't consider the stack space)?

- a) Dynamic programming
- b) Binomial coefficients
- c) Recursion
- d) All have equal space complexities

Answer: a

Explanation: The space complexities are as follows:

Dynamic programming: O(n)

Recursion: O(1)

Binomial coefficients: O(1).

9. What will be the value stored in arr[5] when the following code is executed?

```

#include<stdio.h>
int cat_number(int n)
{
    int i,j,arr[n],k;
    arr[0] = 1;
    for(i = 1; i < n; i++)
    {
        arr[i] = 0;
        for(j = 0,k = i - 1; j < i; j++, k--)
            arr[i] += arr[j] * arr[k];
    }
    return arr[n-1];
}

```

```

    }
    int main()
    {
        int ans, n = 10;
        ans = cat_number(n);
        printf("%d\n",ans);
        return 0;
    }

```

- a) 14
- b) 42
- c) 132
- d) 429

Answer: b

Explanation: The 6th Catalan number will be stored in arr[5], which is 42.

10. Which of the following errors will occur when the below code is executed?

```

#include<stdio.h>
int cat_number(int n)
{
    int i,j,arr[n],k;
    arr[0] = 1;
    for(i = 1; i < n; i++)
    {
        arr[i] = 0;
        for(j = 0,k = i - 1; j < i; j++, k--)
            arr[i] += arr[j] * arr[k];
    }
    return arr[n-1];
}
int main()
{
    int ans, n = 100;
    ans = cat_number(n);
    printf("%d\n",ans);
    return 0;
}

```

- a) Segmentation fault
- b) Array size too large
- c) Integer value out of range
- d) Array index out of range

Answer: c

Explanation: The 100th Catalan number is too large to be stored in an integer. So, the error produced will be integer value out of

range.(It will be a logical error and the compiler wont show it).

1. Which of the following methods can be used to solve the assembly line scheduling problem?
 - a) Recursion
 - b) Brute force
 - c) Dynamic programming
 - d) All of the mentioned

Answer: d

Explanation: All of the above mentioned methods can be used to solve the assembly line scheduling problem.

2. What is the time complexity of the brute force algorithm used to solve the assembly line scheduling problem?
 - a) O(1)
 - b) O(n)
 - c) O(n^2)
 - d) O(2^n)

Answer: d

Explanation: In the brute force algorithm, all the possible ways are calculated which are of the order of 2^n .

3. In the dynamic programming implementation of the assembly line scheduling problem, how many lookup tables are required?
 - a) 0
 - b) 1
 - c) 2
 - d) 3

Answer: c

Explanation: In the dynamic programming implementation of the assembly line scheduling problem, 2 lookup tables are required one for storing the minimum time and the other for storing the assembly line number.

4. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19,
4, 9}}
```

```
time_spent[2][4] = {{6, 5, 15, 7}, {5, 1
0, 11, 4}}
```

```
entry_time[2] = {8, 10}
```

```
exit_time[2] = {10, 7}
```

```
num_of_stations = 4
```

For the optimal solution which should be the starting assembly line?

- a) Line 1
- b) Line 2
- c) All of the mentioned
- d) None of the mentioned

Answer: b

Explanation: For the optimal solution, the starting assembly line is line 2.

5. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19,
4, 9}}
```

```
time_spent[2][4] = {{6, 5, 15, 7}, {5, 1
0, 11, 4}}
```

```
entry_time[2] = {8, 10}
```

```
exit_time[2] = {10, 7}
```

```
num_of_stations = 4
```

For the optimal solution, which should be the exit assembly line?

- a) Line 1
- b) Line 2
- c) All of the mentioned
- d) None of the mentioned

Answer: b

Explanation: For the optimal solution, the exit assembly line is line 2.

6. Consider the following assembly line problem:

```
time_to_reach[2][3] = {{17, 2, 7}, {19, 4, 9}}
```

```
time_spent[2][4] = {{6, 5, 15, 7}, {5, 10, 11, 4}}
```

```
entry_time[2] = {8, 10}
```

```
exit_time[2] = {10, 7}
```

```
num_of_stations = 4
```

What is the minimum time required to build the car chassis?

- a) 40
- b) 41
- c) 42
- d) 43

Answer: d

Explanation: The minimum time required is 43.

The path is S_{2,1} -> S_{1,2} -> S_{2,3} -> S_{2,4}, where S_{i,j} : i = line number, j = station number

7. Consider the following code:

```
#include<stdio.h>
int get_min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
int minimum_time_required(int reach[][3],
int spent[][4], int *entry, int *exit, int n)
{
    int t1[n], t2[n], i;
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0][i],
        t2[i-1]+reach[1][i-1]+spent[0][i]);
        _____;
    }
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
```

Which of the following lines should be inserted to complete the above code?

- a) $t2[i] = \text{get_min}(t2[i-1]+\text{spent}[1][i], t1[i-1]+\text{reach}[0][i-1]+\text{spent}[1][i])$
- b) $t2[i] = \text{get_min}(t2[i-1]+\text{spent}[1][i], t1[i-1]+\text{spent}[1][i])$
- c) $t2[i] = \text{get_min}(t2[i-1]+\text{spent}[1][i], t1[i-1]+\text{reach}[0][i-1])$
- d) none of the mentioned

Answer: a

Explanation: The line $t2[i] = \text{get_min}(t2[i-1]+\text{spent}[1][i], t1[i-1]+\text{reach}[0][i-1]+\text{spent}[1][i])$ should be added to complete the above code.

8. What is the time complexity of the above dynamic programming implementation of the assembly line scheduling problem?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The time complexity of the above dynamic programming implementation of the assembly line scheduling problem is O(n).

9. What is the space complexity of the above dynamic programming implementation of the assembly line scheduling problem?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: b

Explanation: The space complexity of the above dynamic programming implementation of the assembly line scheduling problem is O(n).

10. What is the output of the following code?

```
#include<stdio.h>
int get_min(int a, int b)
```

```

{
    if(a<b)
        return a;
    return b;
}
int minimum_time_required(int reach[][3],
int spent[][4], int *entry, int *exit, in
t n)
{
    int t1[n], t2[n], i;
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0]
[i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        t2[i] = get_min(t2[i-1]+spent[1]
[i], t1[i-1]+reach[0][i-1]+spent[1][i]);
    }
    return get_min(t1[n-1]+exit[0], t2[n-
1]+exit[1]);
}
int main()
{
    int time_to_reach[][3] = {{6, 1, 5},
                               {2, 4, 7}};
    int time_spent[][4] = {{6, 5, 4, 7},
                           {5, 10, 2, 6}};
    int entry_time[2] = {5, 6};
    int exit_time[2] = {8, 9};
    int num_of_stations = 4;
    int ans = minimum_time_required(time_
to_reach, time_spent, entry_time, exit_t
ime, num_of_stations);
    printf("%d",ans);
    return 0;
}

a) 32
b) 33
c) 34
d) 35

```

Answer: c

Explanation: The program prints the optimal time required to build the car chassis, which is 34.

11. What is the value stored in t1[2] when the following code is executed?

```
#include<stdio.h>
int get_min(int a, int b)
{
    if(a<b)
        return a;
    return b;
}
```

```

        return a;
    return b;
}
int minimum_time_required(int reach[][3],
int spent[][4], int *entry, int *exit, in
t n)
{
    int t1[n], t2[n], i;
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0]
[i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        t2[i] = get_min(t2[i-1]+spent[1]
[i], t1[i-1]+reach[0][i-1]+spent[1][i]);
    }
    return get_min(t1[n-1]+exit[0], t2[n-
1]+exit[1]);
}
int main()
{
    int time_to_reach[][3] = {{6, 1, 5},
                               {2, 4, 7}};
    int time_spent[][4] = {{6, 5, 4, 7},
                           {5, 10, 2, 6}};
    int entry_time[2] = {5, 6};
    int exit_time[2] = {8, 9};
    int num_of_stations = 4;
    int ans = minimum_time_required(time_
to_reach, time_spent, entry_time, exit_t
ime, num_of_stations);
    printf("%d",ans);
    return 0;
}
```

- a) 16
- b) 18
- c) 20
- d) 22

Answer: c

Explanation: The value stored in t1[2] when the above code is executed is 20.

12. What is the value stored in t2[3] when the following code is executed?

```
#include<stdio.h>
int get_min(int a, int b)
{
    if(a<b)
        return a;
    return b;
}
```

```

int minimum_time_required(int reach[][3],
int spent[][4], int *entry, int *exit, int n)
{
    int t1[n], t2[n];
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0]
[i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        t2[i] = get_min(t2[i-1]+spent[1]
[i], t1[i-1]+reach[0][i-1]+spent[1][i]);
    }
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
int main()
{
    int time_to_reach[][3] = {{6, 1, 5},
                               {2, 4, 7}};
    int time_spent[][4] = {{6, 5, 4, 7},
                           {5, 10, 2, 6}};
    int entry_time[2] = {5, 6};
    int exit_time[2] = {8, 9};
    int num_of_stations = 4;
    int ans = minimum_time_required(time_
to_reach, time_spent, entry_time, exit_time,
num_of_stations);
    printf("%d", ans);
    return 0;
}

```

- a) 19
- b) 23
- c) 25
- d) 27

Answer: c

Explanation: The value stored in t2[3] when the above code is executed is 25.

13. What is the output of the following code?

```

#include<stdio.h>
int get_min(int a, int b)
{
    if(a<b)
        return a;
    return b;
}
int minimum_time_required(int reach[][4],
int spent[][5], int *entry, int *exit, int n)
{

```

```

    int t1[n], t2[n];
    t1[0] = entry[0] + spent[0][0];
    t2[0] = entry[1] + spent[1][0];
    for(i = 1; i < n; i++)
    {
        t1[i] = get_min(t1[i-1]+spent[0]
[i], t2[i-1]+reach[1][i-1]+spent[0][i]);
        t2[i] = get_min(t2[i-1]+spent[1]
[i], t1[i-1]+reach[0][i-1]+spent[1][i]);
    }
    return get_min(t1[n-1]+exit[0], t2[n-1]+exit[1]);
}
int main()
{
    int time_to_reach[][4] = {{16, 10, 5,
                               12}, {12, 4, 17, 8}};
    int time_spent[][5] = {{13, 5, 20, 1
9, 9}, {15, 10, 12, 16, 13}};
    int entry_time[2] = {12, 9};
    int exit_time[2] = {10, 13};
    int num_of_stations = 5;
    int ans = minimum_time_required(time_
to_reach, time_spent, entry_time, exit_time,
num_of_stations);
    printf("%d", ans);
    return 0;
}

```

- a) 62
- b) 69
- c) 75
- d) 88

Answer: d

Explanation: The program prints the optimal time required to build the car chassis, which is 88.

1. Given a string, you have to find the minimum number of characters to be inserted in the string so that the string becomes a palindrome. Which of the following methods can be used to solve the problem?
 - a) Greedy algorithm
 - b) Recursion
 - c) Dynamic programming
 - d) Both recursion and dynamic programming

Answer: d

Explanation: Dynamic programming and recursion can be used to solve the problem.

2. In which of the following cases the minimum no of insertions to form palindrome is maximum?
- String of length one
 - String with all same characters
 - Palindromic string
 - Non palindromic string

Answer: d

Explanation: In string of length one, string with all same characters and a palindromic string the no of insertions is zero since the strings are already palindromes. To convert a non-palindromic string to a palindromic string, the minimum length of string to be added is 1 which is greater than all the other above cases. Hence the minimum no of insertions to form palindrome is maximum in non-palindromic strings.

3. In the worst case, the minimum number of insertions to be made to convert the string into a palindrome is equal to the length of the string.

- True
- False

Answer: b

Explanation: In the worst case, the minimum number of insertions to be made to convert the string into a palindrome is equal to length of the string minus one. For example, consider the string “abc”. The string can be converted to “abcba” by inserting “a” and “b”. The number of insertions is two, which is equal to length minus one.

4. Consider the string “efge”. What is the minimum number of insertions required to make the string a palindrome?

- 0
- 1
- 2
- 3

Answer: b

Explanation: The string can be converted to “efgfe” by inserting “f” or to “egfge” by inserting “g”. Thus, only one insertion is required.

5. Consider the string “abbccbba”. What is the minimum number of insertions required to make the string a palindrome?

- 0
- 1
- 2
- 3

Answer: a

Explanation: The given string is already a palindrome. So, no insertions are required.

6. Which of the following problems can be used to solve the minimum number of insertions to form a palindrome problem?

- Minimum number of jumps problem
- Longest common subsequence problem
- Coin change problem
- Knapsack problems

Answer: b

Explanation: A variation of longest common subsequence can be used to solve the minimum number of insertions to form a palindrome problem.

7. Consider the following dynamic programming implementation:

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
{
    int len = strlen(s), i, j;
    int arr[len + 1][len + 1];
    char rev[len + 1];
    strcpy(rev, s);
    strrev(rev);
    for(i = 0; i <= len; i++)
        for(j = 0; j <= len; j++)
            if(s[i] == rev[j])
                arr[i][j] = arr[i - 1][j - 1];
            else
                arr[i][j] = max(arr[i - 1][j], arr[i][j - 1]) + 1;
}
```

```

        arr[i][0] = 0;
        for(i = 0; i <= len; i++)
            arr[0][i] = 0;
        for(i = 1; i <= len; i++)
        {
            for(j = 1; j <= len; j++)
            {
                if(s[i - 1] == rev[j - 1])
                    arr[i][j] = arr[i - 1][j
- 1] + 1;
                else
                    arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
            }
            return _____;
}
int main()
{
    char s[] = "abcda";
    int ans = min_ins(s);
    printf("%d",ans);
    return 0;
}

```

Which of the following lines should be added to complete the code?

- a) arr[len][len]
- b) len + arr[len][len]
- c) len
- d) len – arr[len][len]

Answer: d

Explanation: arr[len][len] contains the length of the longest palindromic subsequence. So, len – arr[len][len] gives the minimum number of insertions required to form a palindrome.

8. What is the time complexity of the following dynamic programming implementation of the minimum number of insertions to form a palindrome problem?

```

#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
{
    int len = strlen(s), i, j;

```

```

        int arr[len + 1][len + 1];
        char rev[len + 1];
        strcpy(rev, s);
        strrev(rev);
        for(i = 0; i <= len; i++)
            arr[i][0] = 0;
        for(i = 0; i <= len; i++)
            arr[0][i] = 0;
        for(i = 1; i <= len; i++)
        {
            for(j = 1; j <= len; j++)
            {
                if(s[i - 1] == rev[j - 1])
                    arr[i][j] = arr[i - 1][j
- 1] + 1;
                else
                    arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
            }
            return len - arr[len][len];
}
int main()
{
    char s[] = "abcda";
    int ans = min_ins(s);
    printf("%d",ans);
    return 0;
}

a) O(1)
b) O(n)
c) O(n2)
d) O(mn)
```

Answer: c

Explanation: The time complexity of the above dynamic programming implementation is O(n²).

9. What is the space complexity of the following dynamic programming implementation of the minimum number of insertions to form a palindrome problem?

```

#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
```

```

{
    int len = strlen(s), i, j;
    int arr[len + 1][len + 1];
    char rev[len + 1];
    strcpy(rev, s);
    strrev(rev);
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(s[i - 1] == rev[j - 1])
                arr[i][j] = arr[i - 1][j
- 1] + 1;
            else
                arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
        }
        return len - arr[len][len];
    }
    int main()
    {
        char s[] = "abcda";
        int ans = min_ins(s);
        printf("%d", ans);
        return 0;
    }

a) O(1)
b) O(n)
c) O(n2)
d) O(mn)
}

```

a) 1
b) 2
c) 3
d) 4

Answer: c

Explanation: The space complexity of the above dynamic programming implementation is $O(n^2)$.

10. What is the output of the following code?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
{
```

```

    int len = strlen(s), i, j;
    int arr[len + 1][len + 1];
    char rev[len + 1];
    strcpy(rev, s);
    strrev(rev);
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(s[i - 1] == rev[j - 1])
                arr[i][j] = arr[i - 1][j
- 1] + 1;
            else
                arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
        }
        return len - arr[len][len];
    }
    int main()
    {
        char s[] = "abcda";
        int ans = min_ins(s);
        printf("%d", ans);
        return 0;
    }
}
```

a) 1

b) 2

c) 3

d) 4

Answer: b

Explanation: The length of the longest palindromic subsequence is 3. So, the output will be $5 - 3 = 2$.

11. What is the value stored in $\text{arr}[2][4]$ when the following code is executed?

```
#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
{
    int len = strlen(s), i, j;
```

```

int arr[len + 1][len + 1];
char rev[len + 1];
strcpy(rev, s);
strrev(rev);
for(i = 0; i <= len; i++)
    arr[i][0] = 0;
for(i = 0; i <= len; i++)
    arr[0][i] = 0;
for(i = 1; i <= len; i++)
{
    for(j = 1; j <= len; j++)
    {
        if(s[i - 1] == rev[j - 1])
            arr[i][j] = arr[i - 1][j
- 1] + 1;
        else
            arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
    }
    return len - arr[len][len];
}
int main()
{
    char s[] = "abcda";
    int ans = min_ins(s);
    printf("%d", ans);
    return 0;
}

```

- a) 2
- b) 3
- c) 4
- d) 5

Answer: a

Explanation: The value stored in arr[2][4] when the above code is executed is 2.

12. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int min_ins(char *s)
{
    int len = strlen(s), i, j;
    int arr[len + 1][len + 1];
    char rev[len + 1];
    strcpy(rev, s);

```

```

    strrev(rev);
    for(i = 0; i <= len; i++)
        arr[i][0] = 0;
    for(i = 0; i <= len; i++)
        arr[0][i] = 0;
    for(i = 1; i <= len; i++)
    {
        for(j = 1; j <= len; j++)
        {
            if(s[i - 1] == rev[j - 1])
                arr[i][j] = arr[i - 1][j
- 1] + 1;
            else
                arr[i][j] = max(arr[i -
1][j], arr[i][j - 1]);
        }
    }
    return len - arr[len][len];
}
int main()
{
    char s[] = "efgfe";
    int ans = min_ins(s);
    printf("%d", ans);
    return 0;
}

```

- a) 0
- b) 2
- c) 4
- d) 6

Answer: a

Explanation: Since the string “efgfe” is already a palindrome, the number of insertions required is 0.

1. Given a 2D matrix, find a submatrix that has the maximum sum. Which of the following methods can be used to solve this problem?

- a) Brute force
- b) Recursion
- c) Dynamic programming
- d) Brute force, Recursion, Dynamic programming

Answer: d

Explanation: Brute force, Recursion and Dynamic programming can be used to find the submatrix that has the maximum sum.

2. In which of the following cases, the maximum sum rectangle is the 2D matrix itself?
- When all the elements are negative
 - When all the elements are positive
 - When some elements are positive and some negative
 - When diagonal elements are positive and rest are negative

Answer: a

Explanation: When all the elements of a matrix are positive, the maximum sum rectangle is the 2D matrix itself.

3. Consider the following statements and select which of the following statement are true.

Statement 1: The maximum sum rectangle can be 1X1 matrix containing the largest element If the matrix size is 1X1

Statement 2: The maximum sum rectangle can be 1X1 matrix containing the largest element If all the elements are zero

Statement 3: The maximum sum rectangle can be 1X1 matrix containing the largest element If all the elements are negative

- Only statement 1 is correct
- Only statement 1 and Statement 2 are correct
- Only statement 1 and Statement 3 are correct
- Statement 1, Statement 2 and Statement 3 are correct

Answer: d

Explanation: If the matrix size is 1×1 then the element itself is the maximum sum of that 1×1 matrix. If all elements are zero, then the sum of any submatrix of the given matrix is zero. If all elements are negative, then the maximum element in that matrix is the highest sum in that matrix which is again 1×1 submatrix of the given matrix. Hence all three statements are correct.

4. Consider a matrix in which all the elements are non-zero(at least one positive and at least

one negative element). In this case, the sum of the elements of the maximum sum rectangle cannot be zero.

- True
- False

Answer: a

Explanation: If a matrix contains all non-zero elements with at least one positive and at least one negative element, then the sum of elements of the maximum sum rectangle cannot be zero.

5. Consider the 2×3 matrix $\{\{1,2,3\},\{1,2,3\}\}$. What is the sum of elements of the maximum sum rectangle?

- 3
- 6
- 12
- 18

Answer: c

Explanation: Since all the elements of the 2×3 matrix are positive, the maximum sum rectangle is the matrix itself and the sum of elements is 12.

6. Consider the 2×2 matrix $\{\{-1,-2\},\{-3,-4\}\}$. What is the sum of elements of the maximum sum rectangle?

- 0
- 1
- 7
- 12

Answer: b

Explanation: Since all the elements of the 2×2 matrix are negative, the maximum sum rectangle is $\{-1\}$, a 1×1 matrix containing the largest element. The sum of elements of the maximum sum rectangle is -1.

7. Consider the 3×3 matrix $\{\{2,1,-3\},\{6,3,4\},\{-2,3,0\}\}$. What is the sum of the elements of the maximum sum rectangle?

- 13
- 16

- c) 14
d) 19

Answer: c

Explanation: The complete matrix represents the maximum sum rectangle and its sum is 14.

8. What is the time complexity of the brute force implementation of the maximum sum rectangle problem?

- a) $O(n)$
b) $O(n^2)$
c) $O(n^3)$
d) $O(n^4)$

Answer: d

Explanation: The time complexity of the brute force implementation of the maximum sum rectangle problem is $O(n^4)$.

9. The dynamic programming implementation of the maximum sum rectangle problem uses which of the following algorithm?

- a) Hirschberg's algorithm
b) Needleman-Wunsch algorithm
c) Kadane's algorithm
d) Wagner Fischer algorithm

Answer: c

Explanation: The dynamic programming implementation of the maximum sum rectangle problem uses Kadane's algorithm.

10. Consider the following code snippet:

```
int max_sum_rectangle(int arr[][3],int ro
w,int col)
{
    int left, right, tmp[row], mx_sm =
    INT_MIN, idx, val;
    for(left = 0; left < col; left++)
    {
        for(right = left; right < col;
right++)
        {
            if(right == left)
            {
                for(idx = 0; idx < row
; idx++)

```

```
                tmp[idx] = arr[idx][
right];
            }
        }
        else
        {
            for(idx = 0; idx < row
; idx++)
                tmp[idx] += arr[idx
][right];
        }
        val = kadane_algo(tmp, row)
;
        if(val > mx_sm)
            _____;
    }
    return mx_sm;
}
```

Which of the following lines should be inserted to complete the above code?

- a) val = mx_sm
b) return val
c) mx_sm = val
d) return mx_sm

Answer: c

Explanation: The line "mx_sm = val" should be inserted to complete the above code.

11. What is the time complexity of the following dynamic programming implementation of the maximum sum rectangle problem?

```
int max_sum_rectangle(int arr[][3],int ro
w,int col)
{
    int left, right, tmp[row], mx_sm =
    INT_MIN, idx, val;
    for(left = 0; left < col; left++)
    {
        for(right = left; right < col;
right++)
        {
            if(right == left)
            {
                for(idx = 0; idx < row
; idx++)
                    tmp[idx] = arr[idx][
right];
            }
            else
            {
```

```

        for(idx = 0; idx < row ; val = kadane_algo(tmp, row)
; idx++)
            tmp[idx] += arr[idx] ;
        }
    val = kadane_algo(tmp, row) if(val > mx_sm)
; if(val > mx_sm) mx_sm = val;
}
return mx_sm;
}

a) O(row*col)
b) O(row)
c) O(col)
d) O(row*col*col)

```

Answer: d

Explanation: The time complexity of the above dynamic programming implementation of the maximum sum rectangle is $O(\text{row}^*\text{col}^*\text{col})$.

12. What is the space complexity of the following dynamic programming implementation of the maximum sum rectangle problem?

```

int max_sum_rectangle(int arr[][3],int ro
w,int col)
{
    int left, right, tmp[row], mx_sm =
INT_MIN, idx, val;
    for(left = 0; left < col; left++)
    {
        for(right = left; right < col;
right++)
        {
            if(right == left)
            {
                for(idx = 0; idx < row
; idx++)
                    tmp[idx] = arr[idx][
right];
            }
            else
            {
                for(idx = 0; idx < row
; idx++)
                    tmp[idx] += arr[idx][
right];
            }
        }
    }
}
```

- a) O(row^*col)
- b) O(row)
- c) O(col)
- d) O($\text{row}^*\text{col}^*\text{col}$)

Answer: b

Explanation: The space complexity of the above dynamic programming implementation of the maximum sum rectangle problem is $O(\text{row})$.

13. What is the output of the following code?

```

#include<stdio.h>
#include<limits.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int kadane_algo(int *arr, int len)
{
    int ans, sum, idx;
    ans = 0;
    sum = 0;
    for(idx = 0; idx < len; idx++)
    {
        sum = max_num(0,sum + arr[idx])
;
        ans = max_num(sum,ans);
    }
    return ans;
}
int max_sum_rectangle(int arr[][5],int ro
w,int col)
{
    int left, right, tmp[row], mx_sm =
INT_MIN, idx, val;
    for(left = 0; left < col; left++)
    {
        for(right = left; right < col;
right++)
        {
            if(right == left)
            {

```

```

w; idx++)
        for(idx = 0; idx < ro
            tmp[idx] = arr[idx][
right];
    }
    else
    {
        for(idx = 0; idx < r
            tmp[idx] += arr[id
x][right];
    }
    val = kadane_algo(tmp,ro
);
    if(val > mx_sm)
        mx_sm = val;
}
return mx_sm;
}
int main()
{
    int arr[2][5] ={{1, 2, -1, -4, -20
}, {-4, -1, 1, 7, -6}};
    int row = 2, col = 5;
    int ans = max_sum_rectangle(arr,ro
w,col);
    printf("%d",ans);
    return 0;
}

a) 7
b) 8
c) 9
d) 10

```

Answer: b

Explanation: The program prints the sum of elements of the maximum sum rectangle, which is 8.

14. What is the output of the following code?

```

#include<stdio.h>
#include<limits.h>
int max_num(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int kadane_algo(int *arr, int len)
{
    int ans, sum, idx;
    ans =0;
        sum =0;
        for(idx =0; idx < len; idx++)
    {
        sum = max_num(0,sum + arr[idx])
    }
    ans = max_num(sum,ans);
}
return ans;
}
int max_sum_rectangle(int arr[][5],int ro
w,int col)
{
    int left, right, tmp[row], mx_sm =
INT_MIN, idx, val=0;
    for(left = 0; left < col; left++)
    {
        for(right = left; right < col;
right++)
        {
            if(right == left)
            {
                for(idx = 0; idx < row
; idx++)
                    tmp[idx] = arr[idx][
right];
            }
            else
            {
                for(idx = 0; idx < row
; idx++)
                    tmp[idx] += arr[idx][
right];
            }
            val = kadane_algo(tmp,row)
;
            if(val > mx_sm)
                mx_sm = val;
        }
    }
    return mx_sm;
}
int main()
{
    int arr[4][5] ={{ 7,  1, -3, -6, -1
5},
                    { 10, -6,  3,  -4,
11},
                    { -2, -3, -1,  2, -5}
,
                    { 3,  0,  1,  0,  3}}
;
    int row = 4, col = 5;
    int ans = max_sum_rectangle(arr,ro
w,col);
    printf("%d",ans);
    return 0;
}

```

- a) 17
- b) 18
- c) 19
- d) 20

Answer: b

Explanation: The program prints the sum of elements of the maximum sum rectangle, which is 18.

1. Given an array, check if the array can be divided into two subsets such that the sum of elements of the two subsets is equal. This is the balanced partition problem. Which of the following methods can be used to solve the balanced partition problem?

- a) Dynamic programming
- b) Recursion
- c) Brute force
- d) Dynamic programming, Recursion, Brute force

Answer: d

Explanation: All of the mentioned methods can be used to solve the balanced partition problem.

2. In which of the following cases, it is not possible to have two subsets with equal sum?

- a) When the number of elements is odd
- b) When the number of elements is even
- c) When the sum of elements is odd
- d) When the sum of elements is even

Answer: c

Explanation: When the sum of all the elements is odd, it is not possible to have two subsets with equal sum.

3. What is the time complexity of the brute force algorithm used to solve the balanced partition problem?

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(2^n)

Answer: d

Explanation: In the brute force implementation, all the possible subsets will be formed. This takes exponential time.

4. Consider a variation of the balanced partition problem in which we find two subsets such that $|S_1 - S_2|$ is minimum. Consider the array {1, 2, 3, 4, 5}. Which of the following pairs of subsets is an optimal solution for the above problem?

- a) {5, 4} & {3, 2, 1}
- b) {5} & {4, 3, 2, 1}
- c) {4, 2} & {5, 3, 1}
- d) {5, 3} & {4, 2, 1}

Answer: d

Explanation: For $S_1 = \{5, 3\}$ and $S_2 = \{4, 2, 1\}$, $\text{sum}(S_1) - \text{sum}(S_2) = 1$, which is the optimal solution.

5. Consider the following code:

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = _____
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {3, 4, 5, 6, 7, 1}, len
    = 6;
    int ans = balanced_partition(arr, len)
```

```

);
if(ans == 0)
    printf("false");
else
    printf("true");
return 0;
}

```

Which of the following lines should be inserted to complete the above code?

- a) ans[i - arr[j - 1]][j - 1]
- b) ans[i][j]
- c) ans[i][j] || ans[i - arr[j - 1]][j - 1]
- d) ans[i][j] && ans[i - arr[j - 1]][j - 1]

Answer: c

Explanation: The line “ans[i][j] || ans[i - arr[j - 1]][j - 1]” completes the above code.

6. What is the time complexity of the following dynamic programming implementation of the balanced partition problem where “n” is the number of elements and “sum” is their sum?

```

#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {3, 4, 5, 6, 7, 1}, len
= 6;
}

```

```

int ans = balanced_partition(arr, len
);
if(ans == 0)
    printf("false");
else
    printf("true");
return 0;
}

```

- a) O(sum)
- b) O(n)
- c) O(sum * n)
- d) O(sum + n)

Answer: c

Explanation: The time complexity of the above dynamic programming implementation of the balanced partition problem is O(sum * n).

7. What is the space complexity of the following dynamic programming implementation of the balanced partition problem?

```

#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {3, 4, 5, 6, 7, 1}, len
= 6;
}

```

```

        int ans = balanced_partition(arr, len
);
        if(ans == 0)
            printf("false");
        else
            printf("true");
        return 0;
    }
}

```

- a) O(sum)
- b) O(n)
- c) O(sum * n)
- d) O(sum + n)

Answer: c

Explanation: The space complexity of the above dynamic programming implementation of the balanced partition problem is $O(\text{sum} * n)$.

8. What is the output of the following code?

```

#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
ans[i - arr[j - 1]][j - 1];
        }
        return ans[sm/2][len];
    }
    int main()
    {
        int arr[] = {3, 4, 5, 6, 7, 1}, len
= 6;
        int ans = balanced_partition(arr,le
n);
        if(ans == 0)
            printf("false");
    }
}

```

```

        else
            printf("true");
        return 0;
    }
}

```

- a) True
- b) False

Answer: a

Explanation: The partitions are $S_1 = \{6, 7\}$ and $S_2 = \{1, 3, 4, 5\}$ and the sum of each partition is 13. So, the array can be divided into balanced partitions.

9. What is the value stored in $\text{ans}[3][3]$ when the following code is executed?

```

#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {3, 4, 5, 6, 7, 1}, len
= 6;
    int ans = balanced_partition(arr, len
);
    if(ans == 0)
        printf("false");
    else
        printf("true");
    return 0;
}

```

- a) 0
- b) 1
- c) -1
- d) -2

Answer: b

Explanation: The value stored in $\text{ans}[3][3]$ indicates if a sum of 3 can be obtained using a subset of the first 3 elements. Since the sum can be obtained the value stored is 1.

10. What is the sum of each of the balanced partitions for the array {5, 6, 7, 10, 3, 1}?
- a) 16
 - b) 32
 - c) 0
 - d) 64

Answer: a

Explanation: The sum of all the elements of the array is 32. So, the sum of all the elements of each partition should be 16.

11. What is the output of the following code?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
                ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {5, 6, 7, 10, 3, 1}, len
```

```
= 6;
    int ans = balanced_partition(arr, len
);
    if(ans == 0)
        printf("false");
    else
        printf("true");
    return 0;
}
```

- a) True
- b) False

Answer: a

Explanation: The array can be divided into two partitions $S1 = \{10, 6\}$ and $S2 = \{5, 7, 3, 1\}$ and the sum of all the elements of each partition is 16. So, the answer is true.

12. What is the output of the following code?

```
#include<stdio.h>
int balanced_partition(int *arr, int len)
{
    int sm = 0, i, j;
    for(i = 0; i < len; i++)
        sm += arr[i];
    if(sm % 2 != 0)
        return 0;
    int ans[sm/2 + 1][len + 1];
    for(i = 0; i <= len; i++)
        ans[0][i] = 1;
    for(i = 1; i <= sm/2; i++)
        ans[i][0] = 0;
    for(i = 1; i <= sm/2; i++)
    {
        for(j = 1; j <= len; j++)
        {
            ans[i][j] = ans[i][j-1];
            if(i >= arr[j - 1])
                ans[i][j] = ans[i][j] ||
                ans[i - arr[j - 1]][j - 1];
        }
    }
    return ans[sm/2][len];
}
int main()
{
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8,
9}, len = 9;
    int ans = balanced_partition(arr, len
);
    if(ans == 0)
        printf("false");
    else
```

```

    printf("true");
    return 0;
}

```

- a) True
b) False

Answer: b

Explanation: Since the sum of all the elements of the array is 45, the array cannot be divided into two partitions of equal sum and the answer is false.

1. You are given n dice each having f faces. You have to find the number of ways in which a sum of S can be achieved. This is the dice throw problem. Which of the following methods can be used to solve the dice throw problem?

- a) Brute force
b) Recursion
c) Dynamic programming
d) Brute force, Recursion and Dynamic Programming

Answer: d

Explanation: Brute force, Recursion and Dynamic Programming can be used to solve the dice throw problem.

2. You have n dice each having f faces. What is the number of permutations that can be obtained when you roll the n dice together?

- a) $n \cdot n \cdot n \dots f$ times
b) $f \cdot f \cdot f \dots n$ times
c) $n \cdot n \cdot n \dots n$ times
d) $f \cdot f \cdot f \dots f$ times

Answer: b

Explanation: Each die can take f values and there are n dice. So, the total number of permutations is $f \cdot f \cdot f \dots n$ times.

3. You have 3 dice each having 6 faces. What is the number of permutations that can be obtained when you roll the 3 dice together?

- a) 27
b) 36

- c) 216
d) 81

Answer: c

Explanation: The total number of permutations that can be obtained is $6 \cdot 6 \cdot 6 = 216$.

4. You have 2 dice each of them having 6 faces numbered from 1 to 6. What is the number of ways in which a sum of 11 can be achieved?

- a) 0
b) 1
c) 2
d) 3

Answer: c

Explanation: The sum of 11 can be achieved when the dice show {6, 5} or {5, 6}.

5. There are n dice with f faces. The faces are numbered from 1 to f. What is the minimum possible sum that can be obtained when the n dice are rolled together?

- a) 1
b) f
c) n
d) $n \cdot f$

Answer: c

Explanation: The sum will be minimum when all the faces show a value 1. The sum in this case will be n.

6. There are n dice with f faces. The faces are numbered from 1 to f. What is the maximum possible sum that can be obtained when the n dice are rolled together?

- a) 1
b) $f \cdot f$
c) $n \cdot n$
d) $n \cdot f$

Answer: d

Explanation: The sum will be maximum when all the faces show a value f. The sum in this case will be $n \cdot f$.

7. There are 10 dice having 5 faces. The faces are numbered from 1 to 5. What is the number of ways in which a sum of 4 can be achieved?

- a) 0
- b) 2
- c) 4
- d) 8

Answer: a

Explanation: Since there are 10 dice and the minimum value each die can take is 1, the minimum possible sum is 10. Hence, a sum of 4 cannot be achieved.

8. Consider the following dynamic programming implementation of the dice throw problem:

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
        return _____;
    }
    int main()
    {
        int num_of_dice = 3, num_of_faces = 4, sum = 6;
        int ans = get_ways(num_of_dice, num_of_faces, sum);
        printf("%d",ans);
        return 0;
    }
}
```

Which of the following lines should be added to complete the above code?

- a) arr[num_of_dice][S]
- b) arr[dice][sm]
- c) arr[dice][S]
- d) arr[S][dice]

Answer: a

Explanation: The line arr[num_of_dice][S] completes the above code.

9. What is time complexity of the following dynamic programming implementation of the dice throw problem where f is the number of faces, n is the number of dice and s is the sum to be found?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
        return arr[num_of_dice][S];
    }
    int main()
    {
        int num_of_dice = 3, num_of_faces = 4, sum = 6;
        int ans = get_ways(num_of_dice, num_of_faces, sum);
        printf("%d",ans);
        return 0;
    }
}
```

- a) O(n*f)
- b) O(f*s)

- c) $O(n^*s)$
- d) $O(n^*f^*s)$

Answer: d

Explanation: The time complexity of the above dynamic programming implementation is $O(n^*f^*s)$.

10. What is space complexity of the following dynamic programming implementation of the dice throw problem where f is the number of faces, n is the number of dice and s is the sum to be found?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
        return arr[num_of_dice][S];
    }
    int main()
    {
        int num_of_dice = 3, num_of_faces = 4, sum = 6;
        int ans = get_ways(num_of_dice, num_of_faces, sum);
        printf("%d", ans);
        return 0;
    }
}
a) O(n*f)
b) O(f*s)
c) O(n*s)
d) O(n*f*s)
```

Answer: c

Explanation: The space complexity of the above dynamic programming implementation is $O(n^*s)$.

11. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
        return arr[num_of_dice][S];
    }
    int main()
    {
        int num_of_dice = 3, num_of_faces = 4, sum = 6;
        int ans = get_ways(num_of_dice, num_of_faces, sum);
        printf("%d", ans);
        return 0;
    }
}
```

- a) 10
- b) 12
- c) 14
- d) 16

Answer: a

Explanation: The output of the above code is 10.

12. What will be the value stored in $arr[2][2]$ when the following code is executed?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
        return arr[num_of_dice][S];
    }
int main()
{
    int num_of_dice = 3, num_of_faces = 4, sum = 6;
    int ans = get_ways(num_of_dice, num_of_faces, sum);
    printf("%d", ans);
    return 0;
}
```

- a) 0
- b) 1
- c) 2
- d) 3

Answer: b

Explanation: The value stored in arr[2][2] is 1.

13. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
```

```
for(sm = 1; sm <= S; sm++)
    arr[1][sm] = 1;
for(dice = 2; dice <= num_of_dice; dice++)
{
    for(sm = 1; sm <= S; sm++)
    {
        for(face = 1; face <= num_of_faces && face < sm; face++)
            arr[dice][sm] += arr[dice - 1][sm - face];
    }
}
return arr[num_of_dice][S];
}

int main()
{
    int num_of_dice = 4, num_of_faces = 6, sum = 3;
    int ans = get_ways(num_of_dice, num_of_faces, sum);
    printf("%d", ans);
    return 0;
}
```

- a) 0
- b) 1
- c) 2
- d) 3

Answer: a

Explanation: The minimum possible sum is 4. So, the output for sum = 3 is 0.

14. What is the output of the following code?

```
#include<stdio.h>
int get_ways(int num_of_dice, int num_of_faces, int S)
{
    int arr[num_of_dice + 1][S + 1];
    int dice, face, sm;
    for(dice = 0; dice <= num_of_dice; dice++)
        for(sm = 0; sm <= S; sm++)
            arr[dice][sm] = 0;
    for(sm = 1; sm <= S; sm++)
        arr[1][sm] = 1;
    for(dice = 2; dice <= num_of_dice; dice++)
    {
        for(sm = 1; sm <= S; sm++)
        {
            for(face = 1; face <= num_of_faces && face < sm; face++)
                arr[dice][sm] += arr[dice - 1][sm - face];
        }
    }
}
```

```

f_faces && face < sm; face++)
    arr[dice][sm] += arr[dic
e - 1][sm - face];
}
return arr[num_of_dice][S];
}
int main()
{
    int num_of_dice = 2, num_of_faces =
6, sum = 5;
    int ans = get_ways(num_of_dice, num
_of_faces, sum);
    printf("%d",ans);
    return 0;
}

a) 2
b) 3
c) 4
d) 5

```

Answer: c**Explanation:** The output of the above code is 4.

1. You are given a boolean expression which consists of operators $\&$, $|$ and \wedge (AND, OR and XOR) and symbols T or F (true or false). You have to find the number of ways in which the symbols can be parenthesized so that the expression evaluates to true. This is the boolean parenthesization problem. Which of the following methods can be used to solve the problem?

- a) Dynamic programming
- b) Recursion
- c) Brute force
- d) Dynamic programming, Recursion and Brute force

Answer: d**Explanation:** Dynamic programming, Recursion and Brute force can be used to solve the Boolean parenthesization problem.

2. Consider the expression $T \& F | T$. What is the number of ways in which the expression can be parenthesized so that the output is T (true)?

- a) 0
- b) 1
- c) 2
- d) 3

Answer: c**Explanation:** The expression can be parenthesized as $T \& (F | T)$ and $(T \& F) | T$ so that the output is T.

3. Consider the expression $T \& F \wedge T$. What is the number of ways in which the expression can be parenthesized so that the output is T (true)?

- a) 0
- b) 1
- c) 2
- d) 3

Answer: c**Explanation:** The expression can be parenthesized as $(T \& F) \wedge T$ or $T \& (F \wedge T)$, so that the output is T.

4. Consider the expression $T | F \wedge T$. In how many ways can the expression be parenthesized so that the output is F (false)?

- a) 0
- b) 1
- c) 2
- d) 3

Answer: b**Explanation:** The expression can be parenthesized as $(T | F) \wedge T$, so that the output is F (false).

5. Which of the following gives the total number of ways of parenthesizing an expression with $n + 1$ terms?

- a) n factorial
- b) n square
- c) n cube
- d) n th catalan number

Answer: d**Explanation:** The n th Catalan number gives

the total number of ways of parenthesizing an expression with $n + 1$ terms.

6. What is the maximum number of ways in which a boolean expression with $n + 1$ terms can be parenthesized, such that the output is true?

- a) nth catalan number
- b) n factorial
- c) n cube
- d) n square

Answer: a

Explanation: The number of ways will be maximum when all the possible parenthesizations result in a true value. The number of possible parenthesizations is given by the nth catalan number.

7. Consider the following dynamic programming implementation of the boolean parenthesization problem:

```
int count_bool_parenthesization(char *sym
, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len], False[str_len][str_len];
    int row, col, length, l;
    for(row = 0, col = 0; row < str_len
; row++, col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
            False[row][col] = 1;
        }
    }
    for(length = 1; length < str_len; l
length++)
    {
        for(row = 0, col = length; col
< str_len; col++, row++)
        {
            True[row][col] = 0;
            False[row][col] = 0;
            for(l = 0; l < length; l++)
            {
                if(sym[l] == '|')
                {
                    True[row][col] += True[row][l] * False[l][col];
                    False[row][col] += False[row][l] * True[l][col];
                }
                else if(sym[l] == '&')
                {
                    True[row][col] += True[row][l] * True[l][col];
                    False[row][col] += False[row][l] * False[l][col];
                }
                else if(sym[l] == '^')
                {
                    True[row][col] += True[row][l] * True[l][col];
                    False[row][col] += False[row][l] * False[l][col];
                }
            }
        }
    }
}
```

```
{
    int pos = row + 1;
    int t_row_pos = True[ro
w][pos] + False[row][pos];
    int t_pos_col = True[po
s+1][col] + False[pos+1][col];
    if(op[pos] == '|')
    {
        _____;
    }
    if(op[pos] == '&')
    {
        _____;
    }
    if(op[pos] == '^')
    {
        _____;
    }
}
return True[0][str_len-1];
}
```

Which of the following lines should be added to complete the “if(op[pos] == '|’)” part of the code?

- a) False[row][col] += True[row][pos] * False[pos+1][col];
True[row][col] += t_row_pos * t_pos_col + False[row][pos] * False[pos+1][col];
- b) False[row][col] += False[row][pos] * True[pos+1][col];
True[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
- c) False[row][col] += True[row][pos] * True[pos+1][col];
True[row][col] += t_row_pos * t_pos_col + True[row][pos] * True[pos+1][col];
- d) False[row][col] += False[row][pos] * False[pos+1][col];
True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col];

Answer: d

Explanation: The following lines should be added:

```
False[row][col] += False[row][pos] *
False[pos+1][col];
True[row][col] += t_row_pos * t_pos_col +
False[row][pos] * False[pos+1][col];
```

8. Which of the following lines should be added to complete the “if($op[k] == '&'$)” part of the following code?

```
int count_bool_parenthesization(char *sym
, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len], False[st
r_len][str_len];
    int row,col,length,l;
    for(row = 0, col = 0; row < str_len
; row++,col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
            False[row][col] = 1;
        }
    }
    for(length = 1; length < str_len; l
ength++)
    {
        for(row = 0, col = length; col
< str_len; col++, row++)
        {
            True[row][col] = 0;
            False[row][col] = 0;
            for(l = 0; l < length; l++)
            {
                int pos = row + l;
                int t_row_pos = True[ro
w][pos] + False[row][pos];
                int t_pos_col = True[po
s+1][col] + False[pos+1][col];
                if(op[pos] == '|')
                {
                    _____;
                }
                if(op[pos] == '&')
                {
                    _____;
                }
                if(op[pos] == '^')
                {
                    _____;
                }
            }
        }
    }
}
```

```
return True[0][str_len-1];
}

a) True[row][col] += False[row][pos] *
False[pos+1][col];
False[row][col] += t_row_pos * t_pos_col -
True[row][pos] * True[pos+1][col];
b) True[row][col] += True[row][pos] *
True[pos+1][col];
False[row][col] += t_row_pos * t_pos_col -
True[row][pos] * True[pos+1][col];
c) True[row][col] += True[row][pos] *
False[pos+1][col];
False[row][col] += t_row_pos * t_pos_col -
False[row][pos] * True[pos+1][col];
d) True[row][col] += False[row][pos] *
True[pos+1][col];
False[row][col] += t_row_pos * t_pos_col -
False[row][pos] * True[pos+1][col];
```

Answer: b

Explanation: The following lines should be added:

```
True[row][col] += True[row][pos] *
True[pos+1][col];
False[row][col] += t_row_pos * t_pos_col -
True[row][pos] * True[pos+1][col];
```

9. What is the time complexity of the following dynamic programming implementation of the boolean parenthesization problem?

```
int count_bool_parenthesization(char *sym
, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len], False[st
r_len][str_len];
    int row,col,length,l;
    for(row = 0, col = 0; row < str_len
; row++,col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
```

```

        False[row][col] = 1;
    }
}
for(length = 1; length < str_len; length++)
{
    for(row = 0, col = length; col < str_len; col++, row++)
    {
        True[row][col] = 0;
        False[row][col] = 0;
        for(l = 0; l < length; l++)
        {
            int pos = row + l;
            int t_row_pos = True[row][pos] + False[row][pos];
            int t_pos_col = True[pos+1][col] + False[pos+1][col];
            if(op[pos] == '|')
            {
                False[row][col] += False[row][pos] * False[pos+1][col];
                True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col];
            }
            if(op[pos] == '&')
            {
                True[row][col] += True[row][pos] * True[pos+1][col];
                False[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
            }
            if(op[pos] == '^')
            {
                True[row][col] += True[row][pos] * False[pos+1][col] + False[row][pos] * True[pos+1][col];
                False[row][col] += True[row][pos] * True[pos+1][col] + False[row][pos] * False[pos+1][col];
            }
        }
    }
    return True[0][str_len-1];
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: d

Explanation: The time complexity of the above dynamic programming implementation is $O(n^3)$.

10. What is the space complexity of the following dynamic programming implementation of the boolean parenthesization problem?

```

int count_bool_parenthesization(char *sym
, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len], False[str_len][str_len];
    int row,col,length,l;
    for(row = 0, col = 0; row < str_len ; row++,col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
            False[row][col] = 1;
        }
    }
    for(length = 1; length < str_len; length++)
    {
        for(row = 0, col = length; col < str_len; col++, row++)
        {
            True[row][col] = 0;
            False[row][col] = 0;
            for(l = 0; l < length; l++)
            {
                int pos = row + l;
                int t_row_pos = True[row][pos] + False[row][pos];
                int t_pos_col = True[pos+1][col] + False[pos+1][col];
                if(op[pos] == '|')
                {
                    False[row][col] += False[row][pos] * False[pos+1][col];
                    True[row][col] += t_row_pos * t_pos_col - False[row][pos] * False[pos+1][col];
                }
                if(op[pos] == '&')
                {
                    True[row][col] += True[row][pos] * True[pos+1][col];
                    False[row][col] += t_row_pos * t_pos_col - True[row][pos] * True[pos+1][col];
                }
            }
        }
    }
}

```

```

    {
        True[row][col] += True
        [row][pos] * True[pos+1][col];
        False[row][col] += t
        _row_pos * t_pos_col - True[row][pos] * T
        rue[pos+1][col];
    }
    if(op[pos] == '^')
    {
        True[row][col] += True
        [row][pos] * False[pos+1][col] + False[
        row][pos] * True[pos + 1][col];
        False[row][col] += T
        rue[row][pos] * True[pos+1][col] + False[
        row][pos] * False[pos+1][col];
    }
}
return True[0][str_len-1];
}

```

- a) O(1)
- b) O(n)
- c) O(n^2)
- d) O(n^3)

Answer: c

Explanation: The space complexity of the above dynamic programming implementation is $O(n^2)$.

11. What is the output of the following code?

```

#include<stdio.h>
#include<string.h>
int count_bool_parenthesization(char *sym
, char *op)
{
    int str_len = strlen(sym);
    int True[str_len][str_len], False[str
    _len][str_len];
    int row, col, length, l;
    for(row = 0, col = 0; row < str_len;
    row++, col++)
    {
        if(sym[row] == 'T')
        {
            True[row][col] = 1;
            False[row][col] = 0;
        }
        else
        {
            True[row][col] = 0;
        }
    }
}

```

```

    False[row][col] = 1;
}
for(length = 1; length < str_len; le
ngth++)
{
    for(row = 0, col = length; col <
str_len; col++, row++)
    {
        True[row][col] = 0;
        False[row][col] = 0;
        for(l = 0; l < length; l++)
        {
            int pos = row + l;
            int t_row_pos = True[row
][pos] + False[row][pos];
            int t_pos_col = True[pos
+1][col] + False[pos+1][col];
            if(op[pos] == '|')
            {
                False[row][col] += F
                also[row][pos] * False[pos+1][col];
                True[row][col] += t_
                row_pos * t_pos_col - False[row][pos] * F
                also[pos+1][col];
            }
            if(op[pos] == '&')
            {
                True[row][col] += Tr
                ue[row][pos] * True[pos+1][col];
                False[row][col] += t
                _row_pos * t_pos_col - True[row][pos] * T
                rue[pos+1][col];
            }
            if(op[pos] == '^')
            {
                True[row][col] += True
                [row][pos] * False[pos+1][col] + False[
                row][pos] * True[pos + 1][col];
                False[row][col] += T
                rue[row][pos] * True[pos+1][col] + False[
                row][pos] * False[pos+1][col];
            }
        }
    }
}
return True[0][str_len-1];
}
int main()
{
    char sym[] = "TTTT";
    char op[] = "|^^";
    int ans = count_bool_parenthesizatio
n(sym, op);
    printf("%d", ans);
    return 0;
}

```

- a) 1
- b) 2
- c) 3
- d) 4

Answer: d

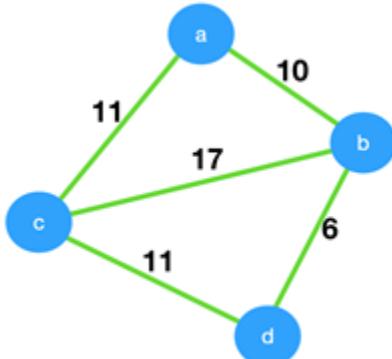
Explanation: The output of the above program is 4.

1. Which of the following is true?
- a) Prim's algorithm initialises with a vertex
 - b) Prim's algorithm initialises with a edge
 - c) Prim's algorithm initialises with a vertex which has smallest edge
 - d) Prim's algorithm initialises with a forest

Answer: a

Explanation: Steps in Prim's algorithm: (I) Select any vertex of given graph and add it to MST (II) Add the edge of minimum weight from a vertex not in MST to the vertex in MST; (III) If MST is complete the stop, otherwise go to step (II).

2. Consider the given graph.



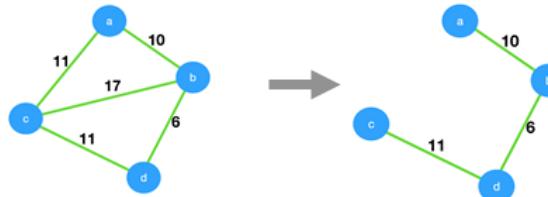
What is the weight of the minimum spanning tree using the Prim's algorithm, starting from vertex a?

- a) 23
- b) 28
- c) 27
- d) 11

Answer: c

Explanation: In Prim's algorithm, we select a vertex and add it to the MST. Then we add

the minimum edge from the vertex in MST to vertex not in MST. From, figure shown below weight of MST = 27.



3. Worst case is the worst case time complexity of Prim's algorithm if adjacency matrix is used?
- a) $O(\log V)$
 - b) $O(V^2)$
 - c) $O(E^2)$
 - d) $O(V \log E)$

Answer: b

Explanation: Use of adjacency matrix provides the simple implementation of the Prim's algorithm. In Prim's algorithm, we need to search for the edge with a minimum for that vertex. So, worst case time complexity will be $O(V^2)$, where V is the number of vertices.

4. Prim's algorithm is a _____
- a) Divide and conquer algorithm
 - b) Greedy algorithm
 - c) Dynamic Programming
 - d) Approximation algorithm

Answer: b

Explanation: Prim's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In greedy method, we attempt to find an optimal solution in stages.

5. Prim's algorithm resembles Dijkstra's algorithm.
- a) True
 - b) False

Answer: a

Explanation: In Prim's algorithm, the MST is

constructed starting from a single vertex and adding in new edges to the MST that link the partial tree to a new vertex outside of the MST. And Dijkstra's algorithm also rely on the similar approach of finding the next closest vertex. So, Prim's algorithm resembles Dijkstra's algorithm.

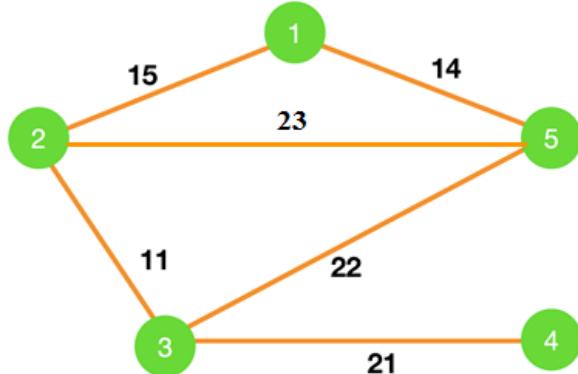
6. Kruskal's algorithm is best suited for the sparse graphs than the prim's algorithm.

- a) True
- b) False

Answer: a

Explanation: Prim's algorithm and Kruskal's algorithm perform equally in case of the sparse graphs. But Kruskal's algorithm is simpler and easy to work with. So, it is best suited for sparse graphs.

7. Consider the graph shown below.



Which of the following edges form the MST of the given graph using Prim's algorithm, starting from vertex 4.

- a) (4-3)(5-3)(2-3)(1-2)
- b) (4-3)(3-5)(5-1)(1-2)
- c) (4-3)(3-5)(5-2)(1-5)
- d) (4-3)(3-2)(2-1)(1-5)

Answer: d

Explanation: The MST for the given graph using Prim's algorithm starting from vertex 4 is,



So, the MST contains edges (4-3)(3-2)(2-1)(1-5).

8. Prim's algorithm is also known as

- a) Dijkstra–Scholten algorithm
- b) Boruvka's algorithm
- c) Floyd–Warshall algorithm
- d) DJP Algorithm

Answer: d

Explanation: The Prim's algorithm was developed by Vojtěch Jarník and it was latter discovered by the duo Prim and Dijkstra. Therefore, Prim's algorithm is also known as DJP Algorithm.

9. Prim's algorithm can be efficiently implemented using _____ for graphs with greater density.

- a) d-ary heap
- b) linear search
- c) fibonacci heap
- d) binary search

Answer: a

Explanation: In Prim's algorithm, we add the minimum weight edge for the chosen vertex which requires searching on the array of weights. This searching can be efficiently implemented using binary heap for dense graphs. And for graphs with greater density, Prim's algorithm can be made to run in linear time using d-ary heap(generalization of binary heap).

10. Which of the following is false about Prim's algorithm?

- a) It is a greedy algorithm
- b) It constructs MST by selecting edges in increasing order of their weights
- c) It never accepts cycles in the MST
- d) It can be implemented using the Fibonacci heap

Answer: b

Explanation: Prim's algorithm can be implemented using Fibonacci heap and it

never accepts cycles. And Prim's algorithm follows greedy approach. Prim's algorithms span from one vertex to another.

1. Kruskal's algorithm is used to _____
- find minimum spanning tree
 - find single source shortest path
 - find all pair shortest path algorithm
 - traverse the graph

Answer: a

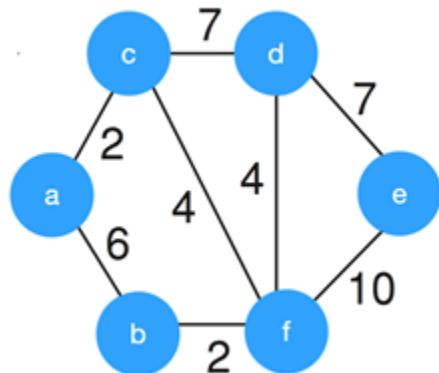
Explanation: The Kruskal's algorithm is used to find the minimum spanning tree of the connected graph. It constructs the MST by finding the edge having the least possible weight that connects two trees in the forest.

2. Kruskal's algorithm is a _____
- divide and conquer algorithm
 - dynamic programming algorithm
 - greedy algorithm
 - approximation algorithm

Answer: c

Explanation: Kruskal's algorithm uses a greedy algorithm approach to find the MST of the connected weighted graph. In the greedy method, we attempt to find an optimal solution in stages.

3. Consider the given graph.



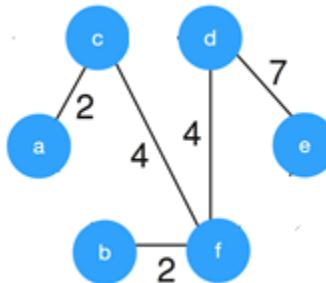
What is the weight of the minimum spanning tree using the Kruskal's algorithm?

- 24
- 23

- 15
- 19

Answer: d

Explanation: Kruskal's algorithm constructs the minimum spanning tree by constructing by adding the edges to spanning tree one-one by one. The MST for the given graph is,



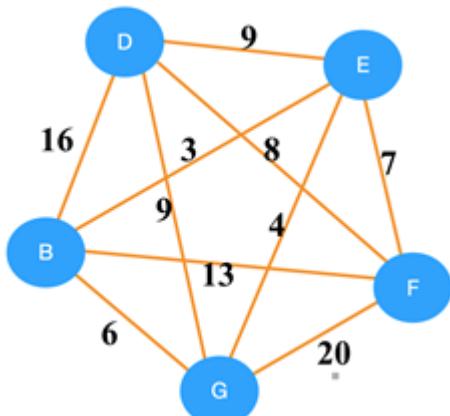
So, the weight of the MST is 19.

4. What is the time complexity of Kruskal's algorithm?
- $O(\log V)$
 - $O(E \log V)$
 - $O(E^2)$
 - $O(V \log E)$

Answer: b

Explanation: Kruskal's algorithm involves sorting of the edges, which takes $O(E \log E)$ time, where E is a number of edges in graph and V is the number of vertices. After sorting, all edges are iterated and union-find algorithm is applied. union-find algorithm requires $O(\log V)$ time. So, overall Kruskal's algorithm requires $O(E \log V)$ time.

5. Consider the following graph. Using Kruskal's algorithm, which edge will be selected first?

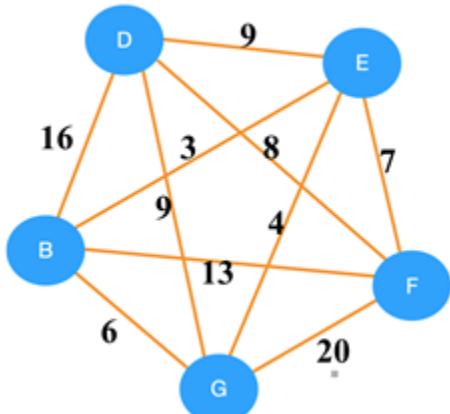


- a) GF
- b) DE
- c) BE
- d) BG

Answer: c

Explanation: In Kruskal's algorithm the edges are selected and added to the spanning tree in increasing order of their weights. Therefore, the first edge selected will be the minimal one. So, correct option is BE.

6. Which of the following edges form minimum spanning tree on the graph using kruskals algorithm?

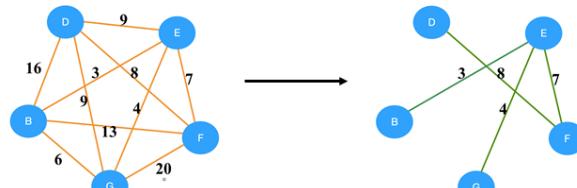


- a) (B-E)(G-E)(E-F)(D-F)
- b) (B-E)(G-E)(E-F)(B-G)(D-F)
- c) (B-E)(G-E)(E-F)(D-E)
- d) (B-E)(G-E)(E-F)(D-F)(D-G)

Answer: a

Explanation: Using Krushkal's algorithm on the given graph, the generated minimum

spanning tree is shown below.



So, the edges in the MST are, (B-E)(G-E)(E-F).

7. Which of the following is true?

- a) Prim's algorithm can also be used for disconnected graphs
- b) Kruskal's algorithm can also run on the disconnected graphs
- c) Prim's algorithm is simpler than Kruskal's algorithm
- d) In Kruskal's sort edges are added to MST in decreasing order of their weights

Answer: b

Explanation: Prim's algorithm iterates from one node to another, so it can not be applied for disconnected graph. Kruskal's algorithm can be applied to the disconnected graphs to construct the minimum cost forest. Kruskal's algorithm is comparatively easier and simpler than prim's algorithm.

8. Which of the following is false about the Kruskal's algorithm?

- a) It is a greedy algorithm
- b) It constructs MST by selecting edges in increasing order of their weights
- c) It can accept cycles in the MST
- d) It uses union-find data structure

Answer: c

Explanation: Kruskal's algorithm is a greedy algorithm to construct the MST of the given graph. It constructs the MST by selecting edges in increasing order of their weights and rejects an edge if it may form the cycle. So, using Kruskal's algorithm is never formed.

9. Kruskal's algorithm is best suited for the dense graphs than the prim's algorithm.

- a) True
- b) False

Answer: b

Explanation: Prim's algorithm outperforms the Kruskal's algorithm in case of the dense graphs. It is significantly faster if graph has more edges than the Kruskal's algorithm.

10. Consider the following statements.
- S1. Kruskal's algorithm might produce a non-minimal spanning tree.
 - S2. Kruskal's algorithm can efficiently implemented using the disjoint-set data structure.
 - a) S1 is true but S2 is false
 - b) Both S1 and S2 are false
 - c) Both S1 and S2 are true
 - d) S2 is true but S1 is false

Answer: d

Explanation: In Kruskal's algorithm, the disjoint-set data structure efficiently identifies the components containing a vertex and adds the new edges. And Kruskal's algorithm always finds the MST for the connected graph.

1. Which of the following is false in the case of a spanning tree of a graph G?
- a) It is tree that spans G
 - b) It is a subgraph of the G
 - c) It includes every vertex of the G
 - d) It can be either cyclic or acyclic

Answer: d

Explanation: A graph can have many spanning trees. Each spanning tree of a graph G is a subgraph of the graph G, and spanning trees include every vertex of the graph. Spanning trees are always acyclic.

2. Every graph has only one minimum spanning tree.
- a) True
 - b) False

Answer: b

Explanation: Minimum spanning tree is a spanning tree with the lowest cost among all the spanning trees. Sum of all of the edges in the spanning tree is the cost of the spanning tree. There can be many minimum spanning trees for a given graph.

3. Consider a complete graph G with 4 vertices. The graph G has _____ spanning trees.
- a) 15
 - b) 8
 - c) 16
 - d) 13

Answer: c

Explanation: A graph can have many spanning trees. And a complete graph with n vertices has $n^{(n-2)}$ spanning trees. So, the complete graph with 4 vertices has $4^{(4-2)} = 16$ spanning trees.

4. The travelling salesman problem can be solved using _____
- a) A spanning tree
 - b) A minimum spanning tree
 - c) Bellman – Ford algorithm
 - d) DFS traversal

Answer: b

Explanation: In the travelling salesman problem we have to find the shortest possible route that visits every city exactly once and returns to the starting point for the given a set of cities. So, travelling salesman problem can be solved by contracting the minimum spanning tree.

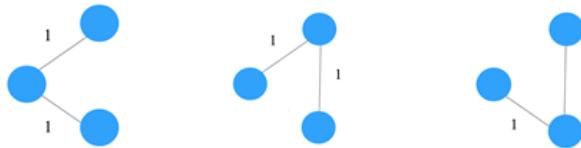
5. Consider the graph M with 3 vertices. Its adjacency matrix is shown below. Which of the following is true?

$$M = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

- a) Graph M has no minimum spanning tree
- b) Graph M has a unique minimum spanning trees of cost 2
- c) Graph M has 3 distinct minimum spanning trees, each of cost 2
- d) Graph M has 3 spanning trees of different costs

Answer: c

Explanation: Here all non-diagonal elements in the adjacency matrix are 1. So, every vertex is connected every other vertex of the graph. And, so graph M has 3 distinct minimum spanning trees.



6. Consider a undirected graph G with vertices { A, B, C, D, E}. In graph G, every edge has distinct weight. Edge CD is edge with minimum weight and edge AB is edge with maximum weight. Then, which of the following is false?

- a) Every minimum spanning tree of G must contain CD
- b) If AB is in a minimum spanning tree, then its removal must disconnect G
- c) No minimum spanning tree contains AB
- d) G has a unique minimum spanning tree

Answer: c

Explanation: Every MST will contain CD as it is smallest edge. So, Every minimum spanning tree of G must contain CD is true. And G has a unique minimum spanning tree is also true because the graph has edges with distinct weights. So, no minimum spanning tree contains AB is false.

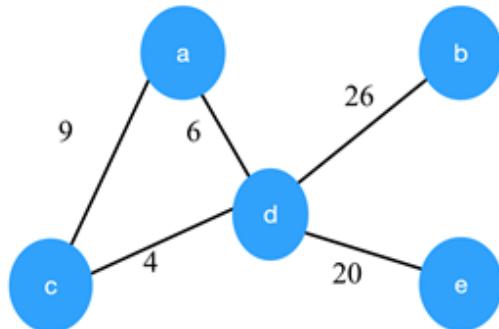
7. If all the weights of the graph are positive, then the minimum spanning tree of the graph is a minimum cost subgraph.

- a) True
- b) False

Answer: a

Explanation: A subgraph is a graph formed from a subset of the vertices and edges of the original graph. And the subset of vertices includes all endpoints of the subset of the edges. So, we can say MST of a graph is a subgraph when all weights in the original graph are positive.

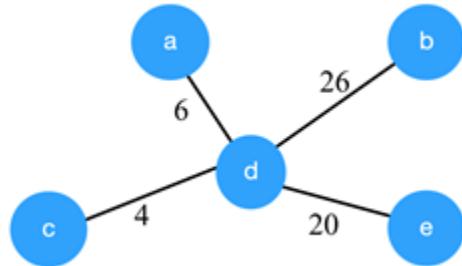
8. Consider the graph shown below. Which of the following are the edges in the MST of the given graph?



- a) (a-c)(c-d)(d-b)(d-b)
- b) (c-a)(a-d)(d-b)(d-e)
- c) (a-d)(d-c)(d-b)(d-e)
- d) (c-a)(a-d)(d-c)(d-b)(d-e)

Answer: c

Explanation: The minimum spanning tree of the given graph is shown below. It has cost 56.



9. Which of the following is not the algorithm to find the minimum spanning tree of the given graph?

- a) Boruvka's algorithm
- b) Prim's algorithm
- c) Kruskal's algorithm
- d) Bellman–Ford algorithm

Answer: d

Explanation: The Boruvka's algorithm, Prim's algorithm and Kruskal's algorithm are the algorithms that can be used to find the minimum spanning tree of the given graph. The Bellman-Ford algorithm is used to find the shortest path from the single source to all other vertices.

10. Which of the following is false?
- The spanning trees do not have any cycles
 - MST have $n - 1$ edges if the graph has n edges
 - Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph
 - Removing one edge from the spanning tree will not make the graph disconnected

Answer: d

Explanation: Every spanning tree has $n - 1$ edges if the graph has n edges and has no cycles. The MST follows the cut property, Edge e belonging to a cut of the graph if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph.

1. Fractional knapsack problem is also known as _____
- 0/1 knapsack problem
 - Continuous knapsack problem
 - Divisible knapsack problem
 - Non continuous knapsack problem

Answer: b

Explanation: Fractional knapsack problem is also called continuous knapsack problem. Fractional knapsack is solved using dynamic programming.

2. Fractional knapsack problem is solved most efficiently by which of the following algorithm?
- Divide and conquer
 - Dynamic programming

- Greedy algorithm
- Backtracking

Answer: c

Explanation: Greedy algorithm is used to solve this problem. We first sort items according to their value/weight ratio and then add item with highest ratio until we cannot add the next item as a whole. At the end, we add the next item as much as we can.

3. What is the objective of the knapsack problem?

- To get maximum total value in the knapsack
- To get minimum total value in the knapsack
- To get maximum weight in the knapsack
- To get minimum weight in the knapsack

Answer: a

Explanation: The objective is to fill the knapsack of some given volume with different materials such that the value of selected items is maximized.

4. Which of the following statement about 0/1 knapsack and fractional knapsack problem is correct?

- In 0/1 knapsack problem items are divisible and in fractional knapsack items are indivisible
- Both are the same
- 0/1 knapsack is solved using a greedy algorithm and fractional knapsack is solved using dynamic programming
- In 0/1 knapsack problem items are indivisible and in fractional knapsack items are divisible

Answer: d

Explanation: In fractional knapsack problem we can partially include an item into the knapsack whereas in 0/1 knapsack we have to either include or exclude the item wholly.

5. Time complexity of fractional knapsack problem is _____

- a) $O(n \log n)$
- b) $O(n)$
- c) $O(n^2)$
- d) $O(nW)$

Answer: a

Explanation: As the main time taking a step is of sorting so it defines the time complexity of our code. So the time complexity will be $O(n \log n)$ if we use quick sort for sorting.

6. Fractional knapsack problem can be solved in time $O(n)$.

- a) True
- b) False

Answer: a

Explanation: It is possible to solve the problem in $O(n)$ time by adapting the algorithm for finding weighted medians.

7. Given items as {value,weight} pairs $\{\{40,20\}, \{30,10\}, \{20,5\}\}$. The capacity of knapsack=20. Find the maximum value output assuming items to be divisible.

- a) 60
- b) 80
- c) 100
- d) 40

Answer: a

Explanation: The value/weight ratio are $\{2,3,4\}$. So we include the second and third items wholly into the knapsack. This leaves only 5 units of volume for the first item. So we include the first item partially.
Final value = $20+30+(40/4)=60$.

8. The result of the fractional knapsack is greater than or equal to 0/1 knapsack.

- a) True
- b) False

Answer: a

Explanation: As fractional knapsack gives extra liberty to include the object partially which is not possible with 0/1 knapsack, thus

we get better results with a fractional knapsack.

9. The main time taking step in fractional knapsack problem is _____

- a) Breaking items into fraction
- b) Adding items into knapsack
- c) Sorting
- d) Looping through sorted items

Answer: c

Explanation: The main time taking step is to sort the items according to their value/weight ratio. It defines the time complexity of the code.

10. Given items as {value,weight} pairs $\{\{60,20\}, \{50,25\}, \{20,5\}\}$. The capacity of knapsack=40. Find the maximum value output assuming items to be divisible and nondivisible respectively.

- a) 100, 80
- b) 110, 70
- c) 130, 110
- d) 110, 80

Answer: d

Explanation: Assuming items to be divisible- The value/weight ratio are $\{3, 2, 4\}$. So we include third and first items wholly. So, now only 15 units of volume are left for second item. So we include it partially.

Final volume =

$$20+60+50 \times (15/25) = 80+30=110$$

Assuming items to be indivisible- In this case we will have to leave one item due to insufficient capacity.

Final volume = $60 + 20 = 80$.

Sanfoundry Global Education & Learning Series – Data Structures & Algorithms.

UNIT IV ITERATIVE IMPROVEMENT

1. Given G is a bipartite graph and the bipartitions of this graphs are U and V respectively. What is the relation between them?

- a) Number of vertices in U = Number of vertices in V
- b) Sum of degrees of vertices in U = Sum of degrees of vertices in V
- c) Number of vertices in U > Number of vertices in V
- d) Nothing can be said

Answer: b

Explanation: We can prove this by induction. By adding one edge, the degree of vertices in U is equal to 1 as well as in V. Let us assume that this is true for $n-1$ edges and add one more edge. Since the given edge adds exactly once to both U and V we can tell that this statement is true for all n vertices.

2. A k-regular bipartite graph is the one in which degree of each vertices is k for all the vertices in the graph. Given that the bipartitions of this graph are U and V respectively. What is the relation between them?

- a) Number of vertices in U=Number of vertices in V
- b) Number of vertices in U not equal to number of vertices in V
- c) Number of vertices in U always greater than the number of vertices in V
- d) Nothing can be said

Answer: a

Explanation: We know that in a bipartite graph sum of degrees of vertices in U= sum of degrees of vertices in V. Given that the graph is a k-regular bipartite graph, we have $k^*(\text{number of vertices in U})=k^*(\text{number of vertices in V})$.

3. There are four students in a class namely A, B, C and D. A tells that a triangle is a bipartite graph. B tells pentagon is a bipartite graph. C tells square is a bipartite graph. D tells heptagon is a bipartite graph. Who

among the following is correct?

- a) A
- b) B
- c) C
- d) D

Answer: c

Explanation: We can prove it in this following way. Let '1' be a vertex in bipartite set X and let '2' be a vertex in the bipartite set Y. Therefore the bipartite set X contains all odd numbers and the bipartite set Y contains all even numbers. Now let us consider a graph of odd cycle (a triangle). There exists an edge from '1' to '2', '2' to '3' and '3' to '1'. The latter case ('3' to '1') makes an edge to exist in a bipartite set X itself. Therefore telling us that graphs with odd cycles are not bipartite.

4. A complete bipartite graph is a one in which each vertex in set X has an edge with set Y. Let n be the total number of vertices. For maximum number of edges, the total number of vertices hat should be present on set X is?

- a) n
- b) $n/2$
- c) $n/4$
- d) data insufficient

Answer: b

Explanation: We can prove this by calculus. Let x be the total number of vertices on set X. Therefore set Y will have $n-x$. We have to maximize $x*(n-x)$. This is true when $x=n/2$.

5. When is a graph said to be bipartite?

- a) If it can be divided into two independent sets A and B such that each edge connects a vertex from to A to B
- b) If the graph is connected and it has odd number of vertices
- c) If the graph is disconnected
- d) If the graph has at least $n/2$ vertices whose degree is greater than $n/2$

Answer: a

Explanation: A graph is said to be bipartite if it can be divided into two independent sets A and B such that each edge connects a vertex from A to B.

6. Are trees bipartite?

- a) Yes
- b) No
- c) Yes if it has even number of vertices
- d) No if it has odd number of vertices

Answer: a

Explanation: Condition needed is that there should not be an odd cycle. But in a tree there are no cycles at all. Hence it is bipartite.

7. A graph has 20 vertices. The maximum number of edges it can have is? (Given it is bipartite)

- a) 100
- b) 140
- c) 80
- d) 20

Answer: a

Explanation: Let the given bipartition X have x vertices, then Y will have $20-x$ vertices. We need to maximize $x*(20-x)$. This will be maxed when $x=10$.

8. Given that a graph contains no odd cycle. Is it enough to tell that it is bipartite?

- a) Yes
- b) No

Answer: a

Explanation: It is required that the graph is connected also. If it is not then it cannot be called a bipartite graph.

9. Can there exist a graph which is both eulerian and is bipartite?

- a) Yes
- b) No
- c) Yes if it has even number of edges
- d) Nothing can be said

Answer: a

Explanation: If a graph is such that there exists a path which visits every edge atleast once, then it is said to be Eulerian. Taking an example of a square, the given question evaluates to yes.

10. A graph is found to be 2 colorable. What can be said about that graph?

- a) The given graph is eulerian
- b) The given graph is bipartite
- c) The given graph is hamiltonian
- d) The given graph is planar

Answer: b

Explanation: A graph is said to be colorable if two vertices connected by an edge are never of the same color. 2 colorable mean that this can be achieved with just 2 colors.

1. Which type of graph has no odd cycle in it?

- a) Bipartite
- b) Histogram
- c) Cartesian
- d) Pie

Answer: a

Explanation: The graph is known as Bipartite if the graph does not contain any odd length cycle in it. Odd length cycle means a cycle with the odd number of vertices in it.

2. What type of graph has chromatic number less than or equal to 2?

- a) Histogram
- b) Bipartite
- c) Cartesian
- d) Tree

Answer: b

Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is chromatic number.

3. Which of the following is the correct type of spectrum of the bipartite graph?

- a) Symmetric
- b) Anti – Symmetric
- c) Circular
- d) Exponential

Answer: a

Explanation: The spectrum of the bipartite graph is symmetric in nature. The spectrum is the property of graph that are related to polynomial, Eigen values, Eigen vectors of the matrix related to graph.

4. Which of the following is not a property of the bipartite graph?

- a) No Odd Cycle
- b) Symmetric spectrum
- c) Chromatic Number Is Less Than or Equal to 2
- d) Asymmetric spectrum

Answer: d

Explanation: A graph is known to be bipartite if it has odd length cycle number. It also has symmetric spectrum and the bipartite graph contains the total chromatic number less than or equal to 2.

5. Which one of the following is the chromatic number of bipartite graph?

- a) 1
- b) 4
- c) 3
- d) 5

Answer: a

Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is the chromatic number.

6. Which graph has a size of minimum vertex cover equal to maximum matching?

- a) Cartesian
- b) Tree

- c) Heap
- d) Bipartite

Answer: d

Explanation: The Konig's theorem given the equivalence relation between the minimum vertex cover and the maximum matching in graph theory. Bipartite graph has a size of minimum vertex cover equal to maximum matching.

7. Which theorem gives the relation between the minimum vertex cover and maximum matching?

- a) Konig's Theorem
- b) Kirchhoff's Theorem
- c) Kuratowski's Theorem
- d) Kelmans Theorem

Answer: a

Explanation: The Konig's theorem given the equivalence relation between the minimum vertex cover and the maximum matching in graph theory. Bipartite graph has a size of minimum vertex cover equal to maximum matching.

8. Which of the following is not a property of perfect graph?

- a) Compliment of Line Graph of Bipartite Graph
- b) Compliment of Bipartite Graph
- c) Line Graph of Bipartite Graph
- d) Line Graph

Answer: d

Explanation: The Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is known as a perfect graph in graph theory. Normal line graph is not a perfect graph whereas line perfect graph is a graph whose line graph is a perfect graph.

9. Which of the following graphs don't have chromatic number less than or equal to 2?

- a) Compliment of Line Graph of Bipartite

- Graph
 b) Compliment of Bipartite Graph
 c) Line Graph of Bipartite Graph
 d) Wheel graph

Answer: d

Explanation: The perfect bipartite graph has chromatic number 2. Also, the Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is known as perfect graph in graph theory. Wheel graph W_n has chromatic number 3 if n is odd and 4 if n is even.

10. Which of the following has maximum clique size 2?

- a) Perfect graph
 b) Tree
 c) Histogram
 d) Cartesian

Answer: a

Explanation: The perfect bipartite graph has clique size 2. Also, the clique size of Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is 2.

11. What is the chromatic number of compliment of line graph of bipartite graph?

- a) 0
 b) 1
 c) 2
 d) 3

Answer: c

Explanation: The perfect bipartite graph has chromatic number 2. So the Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph has chromatic number 2.

12. What is the clique size of the line graph of bipartite graph?

- a) 0

- b) 1
 c) 2
 d) 3

Answer: c

Explanation: The perfect bipartite graph has clique size 2. So the clique size of Compliment of Line Graph of Bipartite Graph, Compliment of Bipartite Graph, Line Graph of Bipartite Graph and every Bipartite Graph is 2.

13. It is possible to have a negative chromatic number of bipartite graph.

- a) True
 b) False

Answer: b

Explanation: A graph is known as bipartite graph if and only if it has the total chromatic number less than or equal to 2. The smallest number of graphs needed to color the graph is the chromatic number. But the chromatic number cannot be negative.

14. Every Perfect graph has forbidden graph characterization.

- a) True
 b) False

Answer: a

Explanation: Berge theorem proves the forbidden graph characterization of every perfect graphs. Because of that reason every bipartite graph is perfect graph.

15. Which structure can be modelled by using Bipartite graph?

- a) Hypergraph
 b) Perfect Graph
 c) Hetero Graph
 d) Directed Graph

Answer: a

Explanation: A combinatorial structure such as Hypergraph can be made using the bipartite graphs. A hypergraph in graph

theory is a type of graph in which edge can join any number of vertices.

1. Which type of graph has all the vertex of the first set connected to all the vertex of the second set?
 - a) Bipartite
 - b) Complete Bipartite
 - c) Cartesian
 - d) Pie

Answer: b

Explanation: The graph is known as Bipartite if the graph does not contain any odd length cycle in it. The complete bipartite graph has all the vertex of first set connected to all the vertex of second set.

2. Which graph is also known as biclique?
 - a) Histogram
 - b) Complete Bipartite
 - c) Cartesian
 - d) Tree

Answer: b

Explanation: A graph is known as complete bipartite graph if and only if it has all the vertex of first set connected to all the vertex of second set. Complete Bipartite graph is also known as Biclique.

3. Which term defines all the complete bipartite graph that are trees?
 - a) Symmetric
 - b) Anti – Symmetric
 - c) Circular
 - d) Stars

Answer: d

Explanation: Star is a complete bipartite graph with one internal node and k leaves. Therefore, all complete bipartite graph which is trees are known as stars in graph theory.

4. How many edges does a n vertex triangle free graph contains?
 - a) n^2

- b) $n^2 + 2$
- c) $n^2 / 4$
- d) n^3

Answer: c

Explanation: A n vertex triangle free graph contains a total of $n^2 / 4$ number of edges. This is stated by Mantel's Theorem which is a special case in Turan's theorem for r=2.

5. Which graph is used to define the claw free graph?
 - a) Bipartite Graph
 - b) Claw Graph
 - c) Star Graph
 - d) Cartesian Graph

Answer: b

Explanation: Star is a complete bipartite graph with one internal node and k leaves. Star with three edges is called a claw. Hence this graph is used to define claw free graph.

6. What is testing of a complete bipartite subgraph in a bipartite graph problem called?
 - a) P Problem
 - b) P-Complete Problem
 - c) NP Problem
 - d) NP-Complete Problem

Answer: d

Explanation: NP stands for nondeterministic polynomial time. In a bipartite graph, the testing of a complete bipartite subgraph in a bipartite graph is an NP-Complete Problem.

7. Which graph cannot contain K3, 3 as a minor of graph?
 - a) Planar Graph
 - b) Outer Planar Graph
 - c) Non Planar Graph
 - d) Inner Planar Graph

Answer: a

Explanation: Minor graph is formed by deleting certain number of edges from a graph or by deleting certain number off

vertices from a graph. Hence Planar graph cannot contain K_{3, 3} as a minor graph.

8. Which of the following is not an Eigen value of the adjacency matrix of the complete bipartite graph?

- a) $(nm)^{1/2}$
- b) $(-nm)^{1/2}$
- c) 0
- d) nm

Answer: d

Explanation: The adjacency matrix is a square matrix that is used to represent a finite graph. Therefore, the Eigen values for the complete bipartite graph is found to be $(nm)^{1/2}$, $(-nm)^{1/2}$, 0.

9. Which complete graph is not present in minor of Outer Planar Graph?

- a) K_{3, 3}
- b) K_{3, 1}
- c) K_{3, 2}
- d) K_{1, 1}

Answer: c

Explanation: Minor graph is formed by deleting certain number of edges from a graph or by deleting certain number off vertices from a graph. Hence Outer Planar graph cannot contain K_{3, 2} as a minor graph.

10. Is every complete bipartite graph a Moore Graph.

- a) True
- b) False

Answer: a

Explanation: In graph theory, Moore graph is defined as a regular graph that has a degree d and diameter k. therefore, every complete bipartite graph is a Moore Graph.

11. What is the multiplicity for the adjacency matrix of complete bipartite graph for 0 Eigen value?

- a) 1
- b) n + m - 2

- c) 0
- d) 2

Answer: b

Explanation: The adjacency matrix is a square matrix that is used to represent a finite graph. The multiplicity of the adjacency matrix off complete bipartite graph with Eigen Value 0 is n + m - 2.

12. Which of the following is not an Eigen value of the Laplacian matrix of the complete bipartite graph?

- a) n + m
- b) n
- c) 0
- d) n*m

Answer: d

Explanation: The laplacian matrix is used to represent a finite graph in the mathematical field of Graph Theory. Therefore, the Eigen values for the complete bipartite graph is found to be n + m, n, m, 0.

13. What is the multiplicity for the laplacian matrix of the complete bipartite graph for n Eigen value?

- a) 1
- b) m-1
- c) n-1
- d) 0

Answer: b

Explanation: The laplacian matrix is used to represent a finite graph in the mathematical field of Graph Theory. The multiplicity of the laplacian matrix of complete bipartite graph with Eigen Value n is m-1.

14. Is it true that every complete bipartite graph is a modular graph.

- a) True
- b) False

Answer: a

Explanation: Yes, the modular graph in graph theory is defined as an undirected

graph in which all three vertices have at least one median vertex. So all complete bipartite graph is called modular graph.

15. How many spanning trees does a complete bipartite graph contain?

- a) n^m
- b) $m^{n-1} * n^{n-1}$
- c) 1
- d) 0

Answer: b

Explanation: Spanning tree of a given graph is defined as the subgraph or the tree with all the given vertices but having minimum number of edges. So, there are a total of $m^{n-1} * n^{n-1}$ spanning trees for a complete bipartite graph.

1. What does Maximum flow problem involve?

- a) finding a flow between source and sink that is maximum
- b) finding a flow between source and sink that is minimum
- c) finding the shortest path between source and sink
- d) computing a minimum spanning tree

Answer: a

Explanation: The maximum flow problem involves finding a feasible flow between a source and a sink in a network that is maximum and not minimum.

2. A network can have only one source and one sink.

- a) False
- b) True

Answer: b

Explanation: A network can have only one source and one sink inorder to find the feasible flow in a weighted connected graph.

3. What is the source?
 - a) Vertex with no incoming edges
 - b) Vertex with no leaving edges
 - c) Centre vertex
 - d) Vertex with the least weight

Answer: a

Explanation: Vertex with no incoming edges is called as a source. Vertex with no leaving edges is called as a sink.

4. Which algorithm is used to solve a maximum flow problem?

- a) Prim's algorithm
- b) Kruskal's algorithm
- c) Dijkstra's algorithm
- d) Ford-Fulkerson algorithm

Answer: d

Explanation: Ford-fulkerson algorithm is used to compute the maximum feasible flow between a source and a sink in a network.

5. Does Ford- Fulkerson algorithm use the idea of?

- a) Naïve greedy algorithm approach
- b) Residual graphs
- c) Minimum cut
- d) Minimum spanning tree

Answer: b

Explanation: Ford-Fulkerson algorithm uses the idea of residual graphs which is an extension of naïve greedy approach allowing undo operations.

6. The first step in the naïve greedy algorithm is?

- a) analysing the zero flow
- b) calculating the maximum flow using trial and error
- c) adding flows with higher values
- d) reversing flow if required

Answer: a

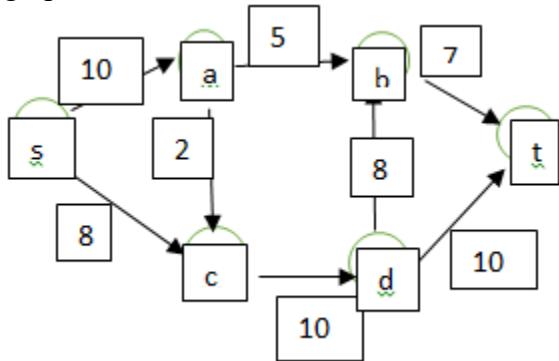
Explanation: The first step in the naïve greedy algorithm is to start with the zero flow followed by adding edges with higher values.

7. Under what condition can a vertex combine and distribute flow in any manner?
- It may violate edge capacities
 - It should maintain flow conservation
 - The vertex should be a source vertex
 - The vertex should be a sink vertex

Answer: b

Explanation: A vertex can combine and distribute flow in any manner but it should not violate edge capacities and it should maintain flow conservation.

8. Find the maximum flow from the following graph.



- 22
- 17
- 15
- 20

Answer: c

Explanation: Initially, zero flow is computed. Then, computing flow = $7+1+5+2=15$. Hence, maximum flow = 15.

9. A simple acyclic path between source and sink which pass through only positive weighted edges is called?

- augmenting path
- critical path
- residual path
- maximum path

Answer: a

Explanation: Augmenting path between source and sink is a simple path without cycles. Path consisting of zero slack edges is called critical path.

10. In what time can an augmented path be found?
- $O(|E| \log |V|)$
 - $O(|E|)$
 - $O(|E|^2)$
 - $O(|E|^2 \log |V|)$

Answer: b

Explanation: An augmenting path can be found in $O(|E|)$ mathematically by an unweighted shortest path algorithm.

11. Dinic's algorithm runs faster than the Ford-Fulkerson algorithm.

- true
- false

Answer: a

Explanation: Dinic's algorithm includes construction of level graphs and residual graphs and finding of augmenting paths along with blocking flow and is faster than the Ford-Fulkerson algorithm.

12. What is the running time of an unweighted shortest path algorithm whose augmenting path is the path with the least number of edges?

- $O(|E|)$
- $O(|E||V|)$
- $O(|E|^2|V|)$
- $O(|E| \log |V|)$

Answer: c

Explanation: Each augmenting step takes $O(|E|)$ using an unweighted shortest path algorithm yielding a $O(|E|2|V|)$ bound on the running time.

13. Who is the formulator of Maximum flow problem?

- Lester R. Ford and Delbert R. Fulkerson
- T.E. Harris and F.S. Ross
- Y.A. Dinitz
- Kruskal

Answer: b

Explanation: The first ever people to

formulate Maximum flow problem were T.E. Harris and F.S. Ross. Lester R. Ford and Delbert R. Fulkerson formulated Ford-Fulkerson algorithm.

14. What is the running time of Dinic's blocking flow algorithm?
- $O(V^2E)$
 - $O(VE^2)$
 - $O(V^3)$
 - $O(E \max |f|)$

Answer: a

Explanation: The running time of Dinic's blocking flow algorithm is $O(V^2E)$. The running of Ford-Fulkerson algorithm is $O(E \max |f|)$.

15. How many constraints does flow have?
- one
 - three
 - two
 - four

Answer: c

Explanation: A flow is a mapping which follows two constraints- conservation of flows and capacity constraints.

1. Stable marriage problem is an example of?
- Branch and bound algorithm
 - Backtracking algorithm
 - Greedy algorithm
 - Divide and conquer algorithm

Answer: b

Explanation: Stable marriage problem is an example for recursive algorithm because it recursively uses backtracking algorithm to find an optimal solution.

2. Which of the following algorithms does Stable marriage problem uses?
- Gale-Shapley algorithm
 - Dijkstra's algorithm

- Ford-Fulkerson algorithm
- Prim's algorithm

Answer: a

Explanation: Stable marriage problem uses Gale-Shapley algorithm. Maximum flow problem uses Ford-Fulkerson algorithm. Prim's algorithm involves minimum spanning tree.

3. An optimal solution satisfying men's preferences is said to be?
- Man optimal
 - Woman optimal
 - Pair optimal
 - Best optimal

Answer: a

Explanation: An optimal solution satisfying men's preferences are said to be man optimal. An optimal solution satisfying woman's preferences are said to be woman optimal.

4. When a free man proposes to an available woman, which of the following happens?
- She will think and decide
 - She will reject
 - She will replace her current mate
 - She will accept

Answer: d

Explanation: When a man proposes to an available woman, she will accept his proposal irrespective of his position on his preference list.

5. If there are n couples who would prefer each other to their actual marriage partners, then the assignment is said to be unstable.
- True
 - False

Answer: a

Explanation: If there are n couples such that a man and a woman are not married, and if they prefer each other to their actual partners, the assignment is unstable.

6. How many 2×2 matrices are used in this problem?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: Two 2×2 matrices are used. One for men representing corresponding woman and ranking and the other for women.

7. What happens when a free man approaches a married woman?

- a) She simply rejects him
- b) She simply replaces her mate with him
- c) She goes through her preference list and accordingly, she replaces her current mate with him
- d) She accepts his proposal

Answer: c

Explanation: If the preference of the man is greater, she replaces her current mate with him, leaving her current mate free.

8. In case of stability, how many symmetric possibilities of trouble can occur?

- a) 1
- b) 2
- c) 4
- d) 3

Answer: b

Explanation: Possibilities- There might be a woman pw, preferred to w by m, who herself prefers m to be her husband and the same applies to man as well.

9. Consider the following ranking matrix.

	W1	W2	W3
M1	1, 2	3, 3	2, 1
M2	3, 1	2, 1	1, 2
M3	2, 3	3, 2	1, 3

Assume that M1 and W2 are married. Now, M2 approaches W2. Which of the following happens?

- a) W2 replaces M1 with M2
- b) W2 rejects M2
- c) W2 accepts both M1 and M2
- d) W2 rejects both M1 and M2

Answer: a

Explanation: W2 is married to M1. But the preference of W2 has M2 before M1. Hence, W2 replaces M1 with M2.

10. Consider the following ranking matrix.

	W1	W2	W3
M1	1, 2	3, 3	2, 1
M2	3, 1	2, 1	1, 2
M3	2, 3	3, 2	1, 3

Assume that M1 and W1 are married and M2 and W3 are married. Now, whom will M3 approach first?

- a) W1
- b) W2
- c) W3
- d) All three

Answer: c

Explanation: M3 will approach W3 first. Since W3 is married and since her preference

list has her current mate before M3, she rejects his proposal.

11. Who formulated a straight forward backtracking scheme for stable marriage problem?

- a) McVitie and Wilson
- b) Gale
- c) Ford and Fulkerson
- d) Dinitz

Answer: a

Explanation: McVitie and Wilson formulated a much faster straight forward backtracking scheme for stable marriage problem. Ford and Fulkerson formulated Maximum flow problem.

12. Can stable marriage cannot be solved using branch and bound algorithm.

- a) True
- b) False

Answer: b

Explanation: Stable marriage problem can be solved using branch and bound approach because branch and bound follows backtracking scheme with a limitation factor.

13. What is the prime task of the stable marriage problem?

- a) To provide man optimal solution
- b) To provide woman optimal solution
- c) To determine stability of marriage
- d) To use backtracking approach

Answer: c

Explanation: The prime task of stable marriage problem is to determine stability of marriage (i.e.) finding a man and a woman who prefer each other to others.

14. Which of the following problems is related to stable marriage problem?

- a) Choice of school by students
- b) N-queen problem
- c) Arranging data in a database
- d) Knapsack problem

Answer: a

Explanation: Choice of school by students is the most related example in the given set of options since both school and students will have a preference list.

15. What is the efficiency of Gale-Shapley algorithm used in stable marriage problem?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(\log N)$

Answer: c

Explanation: The time efficiency of Gale-Shapley algorithm is mathematically found to be $O(N^2)$ where N denotes stable marriage problem.

1. _____ is a matching with the largest number of edges.

- a) Maximum bipartite matching
- b) Non-bipartite matching
- c) Stable marriage
- d) Simplex

Answer: a

Explanation: Maximum bipartite matching matches two elements with a property that no two edges share a vertex.

2. Maximum matching is also called as maximum cardinality matching.

- a) True
- b) False

Answer: a

Explanation: Maximum matching is also called as maximum cardinality matching (i.e.) matching with the largest number of edges.

3. How many colours are used in a bipartite graph?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: A bipartite graph is said to be two-colourable so that every edge has its vertices coloured in different colours.

4. What is the simplest method to prove that a graph is bipartite?
 - a) It has a cycle of an odd length
 - b) It does not have cycles
 - c) It does not have a cycle of an odd length
 - d) Both odd and even cycles are formed

Answer: c

Explanation: It is not difficult to prove that a graph is bipartite if and only if it does not have a cycle of an odd length.

5. A matching that matches all the vertices of a graph is called?
 - a) Perfect matching
 - b) Cardinality matching
 - c) Good matching
 - d) Simplex matching

Answer: a

Explanation: A matching that matches all the vertices of a graph is called perfect matching.

6. What is the length of an augmenting path?
 - a) Even
 - b) Odd
 - c) Depends on graph
 - d) 1

Answer: b

Explanation: The length of an augmenting path in a bipartite graph is always said to be always odd.

7. In a bipartite graph $G=(V,U,E)$, the matching of a free vertex in V to a free vertex in U is called?
 - a) Bipartite matching
 - b) Cardinality matching
 - c) Augmenting
 - d) Weight matching

Answer: c

Explanation: A simple path from a free vertex in V to a free vertex in U whose edges alternate between edges not in M and edges in M is called a augmenting path.

8. A matching M is maximal if and only if there exists no augmenting path with respect to M .
 - a) True
 - b) False

Answer: a

Explanation: According to the theorem discovered by the French mathematician Claude Berge, it means that the current matching is maximal if there is no augmenting path.

9. Which one of the following is an application for matching?
 - a) Proposal of marriage
 - b) Pairing boys and girls for a dance
 - c) Arranging elements in a set
 - d) Finding the shortest traversal path

Answer: b

Explanation: Pairing boys and girls for a dance is a traditional example for matching. Proposal of marriage is an application of stable marriage problem.

10. Which is the correct technique for finding a maximum matching in a graph?
 - a) DFS traversal
 - b) BFS traversal
 - c) Shortest path traversal
 - d) Heap order traversal

Answer: b

Explanation: The correct technique for finding a maximum matching in a bipartite graph is by using a Breadth First Search(BFS).

11. The problem of maximizing the sum of weights on edges connecting matched pairs of vertices is?

- a) Maximum- mass matching
- b) Maximum bipartite matching
- c) Maximum weight matching
- d) Maximum node matching

Answer: c

Explanation: The problem is called as maximum weight matching which is similar to a bipartite matching. It is also called as assignment problem.

12. What is the total number of iterations used in a maximum- matching algorithm?
- a) $[n/2]$
 - b) $[n/3]$
 - c) $[n/2]+n$
 - d) $[n/2]+1$

Answer: d

Explanation: The total number of iterations cannot exceed $[n/2]+1$ where $n=|V|+|U|$ denoting the number of vertices in the graph.

13. What is the efficiency of algorithm designed by Hopcroft and Karp?
- a) $O(n+m)$
 - b) $O(n(n+m))$
 - c) $O(\sqrt{n(n+m)})$
 - d) $O(n^2)$

Answer: c

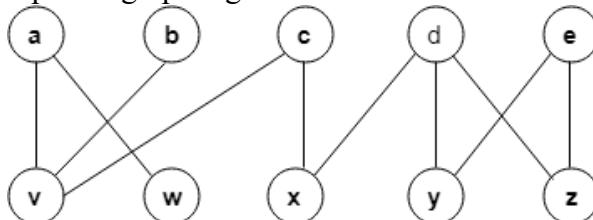
Explanation: The efficiency of algorithm designed by Hopcroft and Karp is mathematically found to be $O(\sqrt{n(n+m)})$.

14. Who was the first person to solve the maximum matching problem?
- a) Jack Edmonds
 - b) Hopcroft
 - c) Karp
 - d) Claude Berge

Answer: a

Explanation: Jack Edmonds was the first person to solve the maximum matching problem in 1965.

15. From the given graph, how many vertices can be matched using maximum matching in bipartite graph algorithm?



- a) 5
- b) 4
- c) 3
- d) 2

Answer: a

Explanation: One of the solutions of the matching problem is given by a-w,b-v,c-x,d-y,e-z. Hence the answer is 5.

UNIT V COPING WITH THE LIMITATIONS OF ALGORITHM POWER

1. The worst-case efficiency of solving a problem in polynomial time is?
- a) $O(p(n))$
 - b) $O(p(n \log n))$
 - c) $O(p(n^2))$
 - d) $O(p(m \log n))$

Answer: a

Explanation: The worst-case efficiency of solving an problem in polynomial time is $O(p(n))$ where $p(n)$ is the polynomial time of input size.

2. Problems that can be solved in polynomial time are known as?
- a) intractable
 - b) tractable
 - c) decision
 - d) complete

Answer: b

Explanation: Problems that can be solved in polynomial time are known as tractable. Problems that cannot be solved in polynomial time are intractable.

3. The sum and composition of two polynomials are always polynomials.

- a) true
- b) false

Answer: a

Explanation: One of the properties of polynomial functions states that the sum and composition of two polynomials are always polynomials.

4. _____ is the class of decision problems that can be solved by non-deterministic polynomial algorithms?

- a) NP
- b) P
- c) Hard
- d) Complete

Answer: a

Explanation: NP problems are called as non-deterministic polynomial problems. They are a class of decision problems that can be solved using NP algorithms.

5. Problems that cannot be solved by any algorithm are called?

- a) tractable problems
- b) intractable problems
- c) undecidable problems
- d) decidable problems

Answer: c

Explanation: Problems that cannot be solved by any algorithm are called undecidable problems. Problems that can be solved in polynomial time are called Tractable problems.

6. The Euler's circuit problem can be solved in?

- a) $O(N)$

- b) $O(N \log N)$
- c) $O(\log N)$
- d) $O(N^2)$

Answer: d

Explanation: Mathematically, the run time of Euler's circuit problem is determined to be $O(N^2)$.

7. To which class does the Euler's circuit problem belong?

- a) P class
- b) NP class
- c) Partition class
- d) Complete class

Answer: a

Explanation: Euler's circuit problem can be solved in polynomial time. It can be solved in $O(N^2)$.

8. Halting problem is an example for?

- a) decidable problem
- b) undecidable problem
- c) complete problem
- d) trackable problem

Answer: b

Explanation: Halting problem by Alan Turing cannot be solved by any algorithm. Hence, it is undecidable.

9. How many stages of procedure does a non-deterministic algorithm consist of?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: A non-deterministic algorithm is a two-stage procedure- guessing stage and verification stage.

10. A non-deterministic algorithm is said to be non-deterministic polynomial if the time-efficiency of its verification stage is polynomial.

- a) true
- b) false

Answer: a

Explanation: One of the properties of NP class problems states that A non-deterministic algorithm is said to be non-deterministic polynomial if the time-efficiency of its verification stage is polynomial.

11. How many conditions have to be met if an NP- complete problem is polynomially reducible?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: A function t that maps all yes instances of decision problems D_1 and D_2 and t should be computed in polynomial time are the two conditions.

12. To which of the following class does a CNF-satisfiability problem belong?

- a) NP class
- b) P class
- c) NP complete
- d) NP hard

Answer: c

Explanation: The CNF satisfiability problem belongs to NP complete class. It deals with Boolean expressions.

13. How many steps are required to prove that a decision problem is NP complete?

- a) 1
- b) 2
- c) 3
- d) 4

Answer: b

Explanation: First, the problem should be NP. Next, it should be proved that every problem in NP is reducible to the problem in question in polynomial time.

14. Which of the following problems is not NP complete?
- a) Hamiltonian circuit
 - b) Bin packing
 - c) Partition problem
 - d) Halting problem

Answer: d

Explanation: Hamiltonian circuit, bin packing, partition problems are NP complete problems. Halting problem is an undecidable problem.

15. The choice of polynomial class has led to the development of an extensive theory called

-
- a) computational complexity
 - b) time complexity
 - c) problem complexity
 - d) decision complexity

Answer: a

Explanation: An extensive theory called computational complexity seeks to classify problems according to their inherent difficulty.

1. Which of the following algorithm can be used to solve the Hamiltonian path problem efficiently?

- a) branch and bound
- b) iterative improvement
- c) divide and conquer
- d) greedy algorithm

Answer: a

Explanation: The Hamiltonian path problem can be solved efficiently using branch and bound approach. It can also be solved using a backtracking approach.

2. The problem of finding a path in a graph that visits every vertex exactly once is called?

- a) Hamiltonian path problem
- b) Hamiltonian cycle problem
- c) Subset sum problem
- d) Turnpike reconstruction problem

Answer: a

Explanation: Hamiltonian path problem is a problem of finding a path in a graph that visits every node exactly once whereas Hamiltonian cycle problem is finding a cycle in a graph.

3. Hamiltonian path problem is _____

- a) NP problem
- b) N class problem
- c) P class problem
- d) NP complete problem

Answer: d

Explanation: Hamiltonian path problem is found to be NP complete. Hamiltonian cycle problem is also an NP- complete problem.

4. There is no existing relationship between a Hamiltonian path problem and Hamiltonian circuit problem.

- a) true
- b) false

Answer: b

Explanation: There is a relationship between Hamiltonian path problem and Hamiltonian circuit problem. The Hamiltonian path in graph G is equal to Hamiltonian cycle in graph H under certain conditions.

5. Which of the following problems is similar to that of a Hamiltonian path problem?

- a) knapsack problem
- b) closest pair problem
- c) travelling salesman problem
- d) assignment problem

Answer: c

Explanation: Hamiltonian path problem is similar to that of a travelling salesman problem since both the problem traverses all the nodes in a graph exactly once.

6. Who formulated the first ever algorithm for solving the Hamiltonian path problem?

- a) Martello
- b) Monte Carlo

- c) Leonard
- d) Bellman

Answer: a

Explanation: The first ever problem to solve the Hamiltonian path was the enumerative algorithm formulated by Martello.

7. In what time can the Hamiltonian path problem can be solved using dynamic programming?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$
- d) $O(N^2 2^N)$

Answer: d

Explanation: Using dynamic programming, the time taken to solve the Hamiltonian path problem is mathematically found to be $O(N^2 2^N)$.

8. In graphs, in which all vertices have an odd degree, the number of Hamiltonian cycles through any fixed edge is always even.

- a) true
- b) false

Answer: a

Explanation: According to a handshaking lemma, in graphs, in which all vertices have an odd degree, the number of Hamiltonian cycles through any fixed edge is always even.

9. Who invented the inclusion-exclusion principle to solve the Hamiltonian path problem?

- a) Karp
- b) Leonard Adleman
- c) Andreas Björklund
- d) Martello

Answer: c

Explanation: Andreas Björklund came up with the inclusion-exclusion principle to reduce the counting of number of Hamiltonian cycles.

10. For a graph of degree three, in what time can a Hamiltonian path be found?

- a) $O(0.251^n)$
- b) $O(0.401^n)$
- c) $O(0.167^n)$
- d) $O(0.151^n)$

Answer: a

Explanation: For a graph of maximum degree three, a Hamiltonian path can be found in time $O(0.251^n)$.

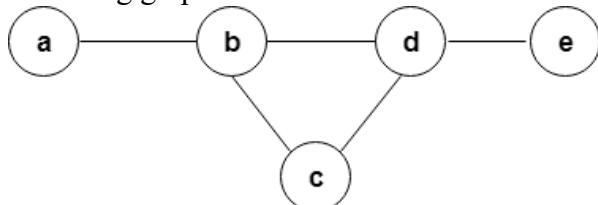
11. What is the time complexity for finding a Hamiltonian path for a graph having N vertices (using permutation)?

- a) $O(N!)$
- b) $O(N! * N)$
- c) $O(\log N)$
- d) $O(N)$

Answer: b

Explanation: For a graph having N vertices traverse the permutations in $N!$ iterations and it traverses the permutations to see if adjacent vertices are connected or not takes N iterations (i.e.) $O(N! * N)$.

12. How many Hamiltonian paths does the following graph have?

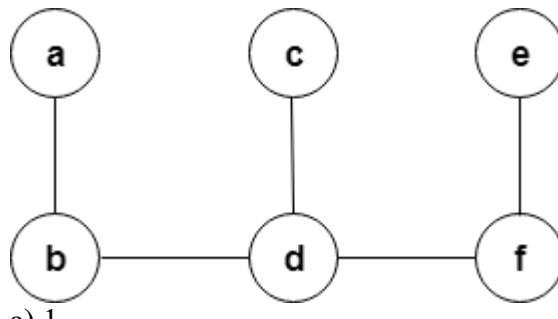


- a) 1
- b) 2
- c) 3
- d) 4

Answer: a

Explanation: The above graph has only one Hamiltonian path that is from a-b-c-d-e.

13. How many Hamiltonian paths does the following graph have?



- a) 1
- b) 2
- c) 0
- d) 3

Answer: c

Explanation: The above graph has no Hamiltonian paths. That is, we cannot traverse the graph with meeting vertices exactly once.

1. Under what condition any set A will be a subset of B?

- a) if all elements of set B are also present in set A
- b) if all elements of set A are also present in set B
- c) if A contains more elements than B
- d) if B contains more elements than A

Answer: b

Explanation: Any set A will be called a subset of set B if all elements of set A are also present in set B. So in such a case set A will be a part of set B.

2. What is a subset sum problem?

- a) finding a subset of a set that has sum of elements equal to a given number
- b) checking for the presence of a subset that has sum of elements equal to a given number and printing true or false based on the result
- c) finding the sum of elements present in a set
- d) finding the sum of all the subsets of a set

Answer: b

Explanation: In subset sum problem check for the presence of a subset that has sum of elements equal to a given number. If such a

subset is present then we print true otherwise false.

3. Which of the following is true about the time complexity of the recursive solution of the subset sum problem?

- a) It has an exponential time complexity
- b) It has a linear time complexity
- c) It has a logarithmic time complexity
- d) it has a time complexity of $O(n^2)$

Answer: a

Explanation: Subset sum problem has both recursive as well as dynamic programming solution. The recursive solution has an exponential time complexity as it will require to check for all subsets in worst case.

4. What is the worst case time complexity of dynamic programming solution of the subset sum problem(sum=given subset sum)?

- a) $O(n)$
- b) $O(\text{sum})$
- c) $O(n^2)$
- d) $O(\text{sum} * n)$

Answer: d

Explanation Subset sum problem has both recursive as well as dynamic programming solution. The dynamic programming solution has a time complexity of $O(n * \text{sum})$ as it is a nested loop with limits from 1 to n and 1 to sum respectively.

5. Subset sum problem is an example of NP-complete problem.

- a) true
- b) false

Answer: a

Explanation: Subset sum problem takes exponential time when we implement a recursive solution. Subset sum problem is known to be a part of NP complete problems.

6. Recursive solution of subset sum problem is faster than dynamic problem solution in terms of time complexity.

- a) true
- b) false

Answer: b

Explanation: The recursive solution to subset sum problem takes exponential time complexity whereas the dynamic programming solution takes polynomial time complexity. So dynamic programming solution is faster in terms of time complexity.

7. Which of the following is not true about subset sum problem?

- a) the recursive solution has a time complexity of $O(2n)$
- b) there is no known solution that takes polynomial time
- c) the recursive solution is slower than dynamic programming solution
- d) the dynamic programming solution has a time complexity of $O(n \log n)$

Answer: d

Explanation: Recursive solution of subset sum problem is slower than dynamic problem solution in terms of time complexity.

Dynamic programming solution has a time complexity of $O(n * \text{sum})$.

8. Which of the following should be the base case for the recursive solution of subset sum problem?

- a)

```
if(sum==0)
```

```
return true;
```

- b)

```
if(sum==0)
```

```
return true;
```

```
if (n ==0 && sum!= 0)
```

```
return false;
```

- c)

```

if (n == 0 && sum != 0)
    return false;

d)

if(sum<0)
    return true;

if (n == 0 && sum!= 0)
    return false;

```

Answer: b

Explanation: The base case condition defines the point at which the program should stop recursion. In this case we need to make sure that, the sum does not become 0 and there should be elements left in our array for recursion to happen.

9. What will be the output for the following code?

```

#include <stdio.h>
bool func(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func(arr, n-1, sum);

    return func(arr, n-1, sum) || func(ar
    r, n-1, sum-arr[n-1]);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int sum = 12;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}

```

- a) 12
- b) 4 6 2

- c) True
- d) False

Answer: c

Explanation: The given code represents the recursive approach of solving the subset sum problem. The output for the code will be true if any subset is found to have sum equal to the desired sum, otherwise false will be printed.

10. What will be the output for the following code?

```

#include <stdio.h>
bool func(int arr[], int n, int sum)
{
    bool subarr[n+1][sum+1];
    for (int i = 0; i <= n; i++)
        subarr[i][0] = true;
    for (int i = 1; i <= sum; i++)
        subarr[0][i] = false;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= sum; j++)
        {
            if(j<arr[i-1])
                subarr[i][j] = subarr[i-1
                ][j];
            if (j >= arr[i-1])
                subarr[i][j] = subarr[i-1
                ][j] ||
                subarr[i - 1][j-arr[i-1]];
        }
    }
    return subarr[n][sum];
}

int main()
{
    int arr[] = {3, 3, 4, 4, 7};
    int sum = 5;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}

```

- a) true
- b) false

- c) 0
- d) error in code

Answer: b

Explanation: The given code represents the dynamic programming approach of solving the subset sum problem. The output for the code will be true if any subset is found to have sum equal to the desired sum, otherwise false will be printed.

11. What will be the worst case time complexity for the following code?

```
#include <stdio.h>
bool func(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func(arr, n-1, sum);
    return func(arr, n-1, sum) || func(ar
r, n-1, sum-arr[n-1]);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int sum = 12;
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n, sum) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

- a) $O(n \log n)$
- b) $O(n^2)$
- c) $O(2^n)$
- d) $O(n^2 \log n)$

Answer: c

Explanation: The given code represents the recursive approach solution of the subset sum problem. It has an exponential time complexity as it will require to check for all subsets in the worst case. It is equal to $O(2^n)$.

1. What is meant by the power set of a set?
- a) subset of all sets
- b) set of all subsets
- c) set of particular subsets
- d) an empty set

Answer: b

Explanation: Power set of a set is defined as the set of all subsets. Ex- if there is a set $S= \{1,3\}$ then power set of set S will be $P=\{\{\}, \{1\}, \{3\}, \{1,3\}\}$.

2. What is the set partition problem?
- a) finding a subset of a set that has sum of elements equal to a given number
- b) checking for the presence of a subset that has sum of elements equal to a given number
- c) checking whether the set can be divided into two subsets of with equal sum of elements and printing true or false based on the result
- d) finding subsets with equal sum of elements

Answer: c

Explanation: In set partition problem we check whether a set can be divided into 2 subsets such that the sum of elements in each subset is equal. If such subsets are present then we print true otherwise false.

3. Which of the following is true about the time complexity of the recursive solution of set partition problem?
- a) It has an exponential time complexity
- b) It has a linear time complexity
- c) It has a logarithmic time complexity
- d) it has a time complexity of $O(n^2)$

Answer: a

Explanation: Set partition problem has both recursive as well as dynamic programming solution. The recursive solution has an exponential time complexity as it will require to check for all subsets in the worst case.

4. What is the worst case time complexity of dynamic programming solution of set partition problem($\text{sum}=\text{sum of set elements}$)?

- a) $O(n)$
- b) $O(\text{sum})$
- c) $O(n^2)$
- d) $O(\text{sum} \cdot n)$

Answer: d

Explanation: Set partition problem has both recursive as well as dynamic programming solution. The dynamic programming solution has a time complexity of $O(n \cdot \text{sum})$ as it has a nested loop with limits from 1 to n and 1 to sum respectively.

5. Set partition problem is an example of NP complete problem.

- a) true
- b) false

Answer: a

Explanation: Set partition problem takes exponential time when we implement a recursive solution. Set partition problem is known to be a part of NP complete problems.

6. Recursive solution of Set partition problem is faster than dynamic problem solution in terms of time complexity.

- a) true
- b) false

Answer: b

Explanation: The recursive solution to set partition problem takes exponential time complexity whereas the dynamic programming solution takes polynomial time complexity. So dynamic programming solution is faster in terms of time complexity.

7. Which of the following is not true about set partition problem?

- a) the recursive solution has a time complexity of $O(2n)$
- b) there is no known solution that takes polynomial time
- c) the recursive solution is slower than dynamic programming solution
- d) the dynamic programming solution has a time complexity of $O(n \log n)$

Answer: d

Explanation: Recursive solution of set partition problem is slower than dynamic problem solution in terms of time complexity. Dynamic programming solution has a time complexity of $O(n \cdot \text{sum})$.

8. Which of the following should be the base case for the recursive solution of a set partition problem?

a)

```
If(sum%2!=0)
    return false;
if(sum==0)
    return true;
```

b)

```
If(sum%2!=0)
    return false;
if(sum==0)
    return true;
if (n ==0 && sum!= 0)
    return false;
```

c)

```
if (n ==0 && sum!= 0)
    return false;
```

d)

```
if(sum<0)
    return true;
if (n ==0 && sum!= 0)
    return false;
```

Answer: b

Explanation: In this case, we need to make sure that, the sum does not become 0 and

there should be elements left in our array for recursion to happen. Also if the sum of elements of the set is an odd number then that set cannot be partitioned into two subsets with an equal sum so under such a condition false should be returned.

9. What will be the output for the given code?

```
#include <stdio.h>
#include <stdbool.h>
bool func1(int arr[], int n, int sum)
{
    if (sum == 0)
        return true;
    if (n == 0 && sum != 0)
        return false;
    if (arr[n-1] > sum)
        return func1(arr, n-1, sum);
    return func1(arr, n-1, sum) || func1(
        arr, n-1, sum-arr[n-1]);
}
bool func (int arr[], int n)
{
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum += arr[i];
    if (sum%2 != 0)
        return false;
    return func1 (arr, n, sum/2);
}
int main()
{
    int arr[] = {4,6, 12, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

- a) true
- b) false
- c) 4 6 2
- d) 12

Answer: a

Explanation: The given code represents the recursive approach of solving the set partition problem. The code checks whether a set can

be divided into 2 subsets such that the sum of elements in each subset is equal. If such a partition is possible then we print true otherwise false. In this case true should be printed.

10. What will be the output for the given code?

```
#include <stdio.h>
bool func (int arr[], int n)
{
    int sum = 0;
    int i, j;
    for (i = 0; i < n; i++)
        sum += arr[i];
    if (sum%2 != 0)
        return false;
    bool partition[sum/2+1][n+1];
    for (i = 0; i <= n; i++)
        partition[0][i] = true;
    for (i = 1; i <= sum/2; i++)
        partition[i][0] = false;
    for (i = 1; i <= sum/2; i++)
    {
        for (j = 1; j <= n; j++)
        {
            partition[i][j] = partition[i][j-1];
            if (i >= arr[j-1])
                partition[i][j] = partition[i][j-1] ||
                    partition[i - arr[j-1]][j-1];
        }
    }
    return partition[sum/2][n];
}
int main()
{
    int arr[] = {3, 3, 4, 4, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    if (func(arr, n) == true)
        printf("true");
    else
        printf("false");
    return 0;
}
```

- a) true
- b) false
- c) 0
- d) error

Answer: b

Explanation: The given code represents the

dynamic programming approach of solving set partition problem. The code checks whether a set can be divided into 2 subsets such that the sum of elements in each subset is equal. If such a partition is possible then we print true otherwise false. In this case, false should be printed.

11. What will be the auxiliary space complexity of dynamic programming solution of set partition problem(sum=sum of set elements)?

- a) $O(n \log n)$

- b) $O(n^2)$
- c) $O(2^n)$
- d) $O(\text{sum} * n)$

Answer: d

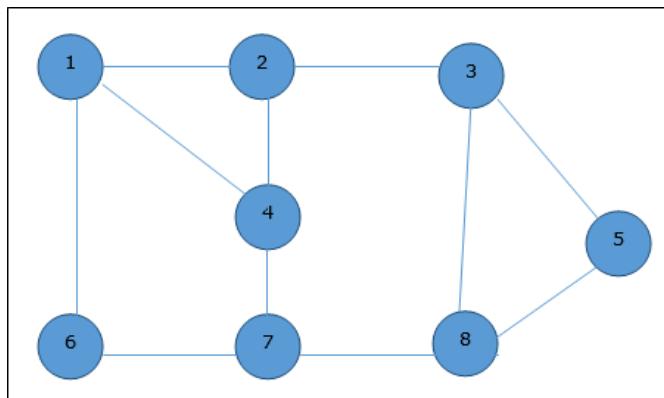
Explanation: The auxiliary space complexity of set partition problem is required in order to store the partition table. It takes up a space of $n * \text{sum}$, so its auxiliary space requirement becomes $O(n * \text{sum})$.

ASSIGNMENT II

1) Find the minimum vertex cover for the following undirected graph by

a) Approximation Algorithm

b) Greedy Approach



2) Given: Input String be “ADBDA” and Pattern to be searched be “BD”

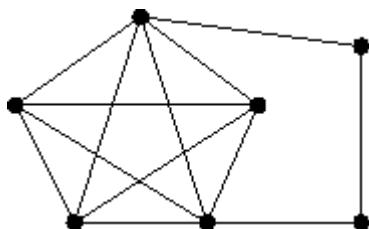
Let ‘d’ be the length of the input string and ‘m’ be length of the pattern to be searched

Let the prime number be $q = 11$ and hash function is given by ,

$$h = d^{m-1} \bmod q$$

Check whether the pattern exists in the given input string or not by Rabin carp method

3) a. Find the max clique of this graph? To which class the clique problem belongs to? Justify it.



b. Given CNF, $(X+Y+Z) (X+Y+Z') (X+Y'+Z)$

(a) Is the above CNF satisfiable? If yes, give an instance of satisfiability.

DESIGN AND ANALYSIS OF ALGORITHMS

Subject Code: CS501PC

Regulations: R16 - JNTUH

Class: III Year B.Tech CSE I Semester



Department of Computer Science and Engineering

BHARAT INSTITUTE OF ENGINEERING AND TECHNOLOGY

Ibrahimpatnam - 501 510, Hyderabad

DESIGN AND ANALYSIS OF ALGORITHMS (CS501PC)

COURSE PLANNER

I. COURSE OVERVIEW:

Introduction to fundamental techniques for designing and analyzing algorithms, including asymptotic analysis; divide-and-conquer algorithms and disjoint set operations; graph algorithms; backtracking algorithms; greedy algorithms; dynamic programming; and branch and bound algorithms; NP-Hard and NP-Complete **problems**;

II. PREREQUISITE(S):

1. Problem Solving Skills
2. Basic Programming
3. Data Structures
4. Formal Languages and Automata Theory

III. COURSE OBJECTIVES:

1	To analyze performance of algorithms.
2	To choose the appropriate data structure and algorithm design method for a specified application.
3	To understand how the choice of data structures and algorithm design methods impacts the performance of programs.
4	To solve problems using algorithm design methods such as the greedy method, divide and conquer, dynamic programming, backtracking and branch and bound.
5	To understand the differences between tractable and intractable problems.
6	To introduce P and NP classes.

IV. COURSE OUTCOMES:

S.No	Description	Bloom's Taxonomy Level
1	Ability to analyze the performance of algorithms.	Analyze (level 4)
2	Ability to choose appropriate algorithm design techniques for solving problems.	Knowledge, Application (level 1, level 3)
3	Ability to understand how the choice of data structures and the algorithm design methods impact the performance of programs.	Understanding, Synthesis (Level 2, level 5)

V. HOW PROGRAM OUTCOMES ARE ASSESSED:

Program Outcomes (PO)		Level	Proficiency assessed by
PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems related to Computer Science and Engineering.	3	Assignments
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems related to Computer Science and Engineering and reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.	3	Assignments
PO3	Design/development of solutions: Design solutions for complex engineering problems related to Computer Science and Engineering and design system components or processes that meet the specified needs with appropriate consideration for the public health and	2	Assignments

	safety, and the cultural, societal, and environmental considerations.		
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.	2	Assignments
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.	--	--
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the Computer Science and Engineering professional engineering practice.	1	Assignments
PO7	Environment and sustainability: Understand the impact of the Computer Science and Engineering professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.	-	--
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.	-	--
PO9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.	-	--
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.	-	--
PO11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.	-	--
PO12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.	2	Research

1: Slight (Low)

2: Moderate

3: Substantial

(Medium)

(High)

- : None

VI. HOW PROGRAM SPECIFIC OUTCOMES ARE ASSESSED:

Program Specific Outcomes (PSO)		Level	Proficiency assessed by
PSO1	Foundation of mathematical concepts: To use mathematical methodologies to crack problem using suitable mathematical analysis, data structure and suitable algorithm.	3	Lectures, Assignments
PSO2	Foundation of Computer System: The ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.	2	Lectures, Assignments
PSO3	Foundations of Software development: The ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.	--	--

VII. SYLLABUS:

UNIT- I

Introduction-Algorithm definition, Algorithm Specification, Performance Analysis-Space complexity, Time complexity, Randomized Algorithms. **Divide and conquer**- General method, applications - Binary search, Merge sort, Quick sort, Strassen's Matrix Multiplication.

UNIT- II

Disjoint set operations, union and find algorithms, AND/OR graphs, Connected Components and Spanning trees, Bi-connected components **Backtracking**-General method, applications-The 8-queen problem, sum of subsets problem, graph coloring, Hamiltonian cycles.

UNIT-

III

Greedy method- General method, applications- Knapsack problem, Job sequencing with deadlines, Minimum cost spanning trees, Single source shortest path problem.

UNIT- IV

Dynamic Programming- General Method, applications- Chained matrix multiplication, All pairs shortest path problem, Optimal binary search trees, 0/1 knapsack problem, Reliability design, Traveling sales person problem.

UNIT- V

Branch and Bound- General Method, applications-0/1 Knapsack problem, LC Branch and Bound solution, FIFO Branch and Bound solution, Traveling sales person problem.

NP-Hard and NP-Complete problems- Basic concepts, Non-deterministic algorithms, NP - Hard and NP- Complete classes, Cook's theorem.

SUGGESTED BOOKS:

TEXT BOOKS:

1. Fundamentals of Computer Algorithms, 2nd Edition, Ellis Horowitz, Sartaj Sahni and S. Rajasekharan, Universities Press.
2. Design and Analysis of Algorithms, P. H. Dave, H.B.Dave,2nd edition, Pearson Education.

REFERENCE BOOKS:

1. Algorithm Design: Foundations, Analysis and Internet examples, M. T. Goodrich and R. Tomassia, John Wiley and sons.
2. Design and Analysis of Algorithms, S. Sridhar, Oxford Univ. Press
3. Design and Analysis of algorithms, Aho, Ullman and Hopcroft, Pearson Education.

4. Foundations of Algorithms,, R. Neapolitan and K. Naimipour, 4th edition, Jones an Bartlett Student edition.

5. Introduction to Algorithms,3rd Edition, T. H. Cormen, C. E.Leiserson, R. L. Rivest, and C. Stein, PHI

NPTEL Web Course:

1. <http://nptel.ac.in/courses/106101060/>
2. https://onlinecourses.nptel.ac.in/noc16_cs04/preview

NPTEL Video Course:

1. <http://www.nptelvideos.in/2012/11/design-analysis-of-algorithms.html>

GATE SYLLABUS:

Searching, sorting, hashing. Asymptotic worst case time and space complexity. Algorithm design techniques: greedy, dynamic programming and divide-and-conquer. Graph search, minimum spanning trees, and shortest paths.

IES SYLLABUS:

Not Applicable

VIII. COURSE PLAN:

Sl No	Week	Topic	Course Learning outcomes	Reference
UNIT-I				
1	1	Introduction- Algorithm definition,	Understand the fundamentals of Design and Analysis of Algorithms	1
2		Algorithm Specification,		
3		Performance Analysis-		
4		Space complexity, Time complexity, Randomized Algorithms.		
5	2	Divide and conquer- General method, applications - Binary search,	Compare the performance of the algorithms Describe the Significance of Divide and Conquer Design method Apply the Concept of Divide and Conquer method to solve the real world problems.	1
6				
7				
8				
9	3	Merge sort,	Assess the student skills.	1
10		Quick sort,		
11		Strassen's Matrix Multiplication		
12		BRIDGE CLASS -1		
UNIT-II				
13	4	Disjoint set operations, union and find algorithms,	Explain the basics of various mathematical concept of searching and	1
14		AND/OR graphs,		
15		Connected Components		
16				

			traversing techniques.	
17	5	and Spanning trees,	Understand the various searching techniques.	
18		Bi-connected components	Discuss the various connected components techniques.	
19		Backtracking -General method,	Define the concept of Backtracking.	
20		Applications-The 8-queen problem,	Compute the real world problems by using Backtracking	
21		sum of subsets problem,		
22	6	graph coloring,		
23		Hamiltonian cycles		
24		BRIDGE CLASS -2	Assess the student skills.	

UNIT-III

25	7	Greedy method - General method,	Describe the concept of Greedy method.		
26		applications- Knapsack problem,	Apply the Concept of Greedy method to solve the real world problems.	1	
27					
28	8	Job sequencing with deadlines,			
29		Mock Test-1			
30		1 st MID EXAMS			
31		Minimum cost spanning trees,	Apply the Concept of Greedy method to solve the real world problems.		
32		Single source shortest path problem.			
33	9	BRIDGE CLASS-3	Assess the student skills.		
34					
35					
36					

UNIT-IV

37	10	Dynamic Programming - General Method, applications- Chained matrix multiplication,	Understand the Dynamic Programming techniques.		
38		All pairs shortest path problem,	Demonstrate the Dynamic programming techniques.	1	
39		Optimal binary search trees,			
40		0/1 knapsack problem,			
41	11	Reliability design,			
42					
43		Traveling sales person problem.			
44					

UNIT-V																	
45	12	Branch and Bound- General Method,									State the purpose of Branch and Bound		1				
46		applications-0/1 Knapsack problem,									Solve the real world problems by using Branch and Bound.						
47		LC Branch and Bound solution, FIFO Branch and Bound solution,															
48		Traveling sales person problem.															
49	13	NP-Hard and NP-Complete problems- Basic concepts,									Distinguish the concept of NP-Hard and NP-Complete Problems.						
50		Non-deterministic algorithms,									Discuss the concept of Non-deterministic algorithms						
51		NP - Hard and NP- Complete classes, Cook's theorem									Distinguish the concept of NP-Hard and NP-Complete Problems.						
52		BRIDGE CLASS-4									Assess the student skills.						
53	14	REVISION									Revise the concepts of all units						
54		PREVIOUS PAPER DISCUSSION									Discuss the questions from previous year question papers						

IX. MAPPING COURSE OUTCOMES LEADING TO THE ACHIEVEMENT OF PROGRAM OUTCOMES AND PROGRAM SPECIFIC OUTCOMES:

COs / POs	Program Outcomes												Program Specific Outcomes		
	PO 1	PO 2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	3	-	-	-	-	-	-	-	-	3	3	-
CO2	3	3	2	3	-	-	-	-	-	-	-	-	2	3	-
CO3	3	3	2	2	-	-	-	-	-	-	-	-	1	1	-

X. QUESTION BANK: (JNTUH)

No	Question	Blooms Taxon	Program Outc

		o my Level	ome
UNIT – I			
PART – A (SHORT ANSWER QUESTIONS)			
1	Define the term algorithm and state the criteria the algorithm should satisfy.	Knowledge	1
2	Compute the average case time complexity of quick sort	Apply	7
3	Describe the role of space complexity and time complexity of a program?	Knowledge	1
4	If $f(n)=5n^2 + 6n + 4$, then prove that $f(n)$ is $O(n^2)$	Apply	3
5	What is meant by divide and conquer? Give the recurrence relation for divide and conquer.	Understand	7
PART – B (LONGANSWER QUESTIONS)			
1	Write binary search algorithm and analyze its time complexity	Understand	7
2	Explain quick sort algorithm and simulate it for the following data 20, 5, 10, 16, 54, 21	Apply	7
3	Illustrate merge sort algorithm and discuss time complexity	Understand	7
4	Describe strassen's matrix multiplication.	Understand	7
5	Sort the list of numbers using merge sort: 78, 32, 42, 62, 98, 12, 34, 83	Apply	7

S. No	Question	Blooms Taxonomy Level	Program Outcome
UNIT – II			
PART – A (SHORT ANSWER QUESTIONS)			
1	Discuss about union operation on sets	Knowledge	5
2	Describe AND/OR graph	Understand	5
3	Explain game tree	Understand	5
4	Define a connected and bi-connected component.	Knowledge	5
5	Define an articulation point?	Knowledge	5
PART – B (LONGANSWER QUESTIONS)			
1	Discuss various tree traversal techniques with examples	Understand	5
2	Discuss about weighting rule for finding UNION of sets and collapsing rule	Understand	5
3	Differentiate divide and conquer and greedy method	Understand	6,7
4	Discuss game trees	Understand	5
5	Compare and contrast BFS and DFS.	analyze	5

No	Question	Blooms Taxon omy Level	Program Out come
UNIT – III			
PART – A (SHORT ANSWER QUESTIONS)			
1	Define greedy method	Know edge	8
2	State Prim;s algorithm	Knowledge	8

3	What is job sequencing with deadlines problem	Know edge	8
4	State the principle of optimality	Know edge	8
5	Define minimum cost spanning tree	Know edge	8
PART – B (LONGANSWER QUESTIONS)			
1	Discuss single source shortest path problem with example	Apply	8
2	Discuss kruskals algorithm with an example	Understand	8
3	Write an algorithm knapsack problem .Give example	Apply	8
4	Explain prims algorithm with an example	Understand	8
5	Compute the optimal solution for knapsack problem using greedy method N=3, $M = 20, (p_1, p_2, p_3) = (25, 24, 15), (w_1, w_2, w_3) = (18, 15, 10)$	Apply	8

No	Question	Blooms Taxonomy Level	Program Outcome
----	----------	-----------------------	-----------------

UNIT – IV

PART – A (SHORT ANSWER QUESTIONS)

1	Write an algorithm for optimal binary search tree Give example	Apply	8
2	Explain 0/1 knapsack problem with example	Understand	8
3	Discuss all pairs shortest path problem with an example	Understand	8
4	Explain 8 – Queens problem	Understand 1	10
5	Define Sum of Subsets problem	Understand 1	10

PART – B (LONGANSWER QUESTIONS)

1	Describe the travelling salesman problem and discuss how to solve it using dynamic programming?	Understand	9
2	Explain the concept Chained matrix multiplication.	Apply	8
3	Solve the solution for 0/1 knapsack problem using dynamic programming($p_1, p_2, p_3, p_4) = (11, 21, 31, 33), (w_1, w_2, w_3, w_4) = (2, 11, 22, 15), M=40, n=4$	Apply	8
4	Use optimal binary search tree algorithm and compute $w_{ij}, c_{ij}, r_{ij}, 0 \leq i \leq j \leq 4, p_1=1/10, p_2=1/5, p_3=1/10, p_4=1/120, q_0=1/5, q_1=1/10, q_2=1/5, q_3=1/20, q_4=1/20$.	Apply	8
5	Discuss all pairs shortest path problem with an example	Understand	8

No	Question	Blooms Taxonomy Level	Program Outcome
----	----------	-----------------------	-----------------

UNIT – V

PART – A (SHORT ANSWER QUESTIONS)

1	Define a dead node	Knowledge	10
2	Differentiate live node and dead node	Knowledge	10
3	Compare NP-hard and NP-completeness	Knowledge	10
4	Define deterministic problem	Understand	12
5	Define maxclique problem?	Understand	12

PART – B (LONGANSWER QUESTIONS)			
1	Explain the principle of FIFO branch and bound	Apply	11
2	Explain the method of reduction to solve travelling sales person problem using branch and bound	Apply	11
3	Write non deterministic algorithm for sorting and searching	Understand	12
5	What is chromatic number decision problem and clique decision problem	Apply	12

XI. OBJECTIVE QUESTIONS: JNTUH

UNIT-I

1. In analysis of algorithm, approximate relationship between the size of the job and the amount of work required to do is expressed by using _____

- (a) Central tendency (b) Differential equation (c) Order of execution (d) Order of magnitude (e) Order of Storage.

Ans :Order of execution

2. Worst case efficiency of binary search is

- (a) $\log_2 n + 1$ (b) n (c) N^2 (d) $2n$ (e) $\log n$.

Ans : $\log_2 n + 1$

3. For analyzing an algorithm, which is better computing time?

- (a) $O(100 \log N)$ (b) $O(N)$ (c) $O(2N)$ (d) $O(N \log N)$ (e) $O(N^2)$.

Ans : $O(100 \log N)$

4. Consider the usual algorithm for determining whether a sequence of parentheses is balanced. What is the maximum number of parentheses that will appear on the stack AT ANY ONE TIME when the algorithm analyzes: ((())())()

- (a) 1 (b) 2 (c) 3 (d) 4

Ans :3

5. Breadth first search _____

- (a) Scans each incident node along with its children. (b) Scans all incident edges before moving to other node. (c) Is same as backtracking (d) Scans all the nodes in random order.

Ans :Scans all incident edges before moving to other node.

6. Which method of traversal does not use stack to hold nodes that are waiting to be processed?

- (a) Dept First (b) D-search (c) Breadth first (d) Back-tracking

Ans :Breadth first

7. The Knapsack problem where the objective function is to minimize the profit is _____

- (a) Greedy (b) Dynamic 0 / 1 (c) Back tracking (d) Branch & Bound 0/1

Ans :Branch & Bound 0/1

8. Choose the correct answer for the following statements:

I. The theory of NP-completeness provides a method of obtaining a polynomial time for NP algorithms.

II. All NP-complete problems are NP-Hard.

- (a) I is FALSE and II is TRUE (b) I is TRUE and II is FALSE (c) Both are TRUE (d) Both are FALSE

Ans :I is FALSE and II is TRUE

9. If all $c(i, j)$'s and $r(i, j)$'s are calculated, then OBST algorithm in worst case takes one of the following time.

- (a) $O(n \log n)$ (b) $O(n^3)$ (c) $O(n^2)$ (d) $O(\log n)$ (e) $O(n^4)$.

Ans : $O(n^3)$

10. The upper bound on the time complexity of the nondeterministic sorting algorithm is

- (a) $O(n)$ (b) $O(n \log n)$ (c) $O(1)$ (d) $O(\log n)$ (e) $O(n^2)$.

Ans: $O(n)$

11. The worst case time complexity of the nondeterministic dynamic knapsack algorithm is
(a) $O(n \log n)$ (b) $O(\log n)$ (c) $O(n^2)$ (d) $O(n)$ (e) $O(1)$.

Ans :O(n)

12. Recursive algorithms are based on

- (a) Divideand conquer approach (b) Top-down approach (c) Bottom-up approach (d)
Hierarchical approach

Ans :Bottom-up approach

13. What do you call the selected keys in the quick sort method?

- (a) Outer key (b)Inner Key (c) Partition key(d) Pivot key (e) Recombine key.

Ans :c

14. How do you determine the cost of a spanning tree?

- (a) By the sum of the costs of the edges of the tree (b) By the sum of the costs of the edges and vertices of the tree
(c) By the sum of the costs of the vertices of the tree (d) By the sum of the costs of the edges of the graph
(e) By the sum of thecosts of the edges and vertices of the graph.

Ans :By the sum of the costs of the edges of the tree8.

15. The time complexity of the normal quick sort, randomized quick sort algorithms in the worst case is

- (a) $O(n^2)$, $O(n \log n)$ (b) $O(n^2)$, $O(n^2)$ (c) $O(n \log n)$, $O(n^2)$ (d) $O(n \log n)$, $O(n \log n)$ (e)
 $O(n \log n)$, $O(n^2 \log n)$.

Ans :O(n2), O(n2)

16. Let there be an array of length 'N', and the selection sort algorithm is used to sort it, how many times a swap function is called to complete the execution?

- (a) $N \log N$ times (b) $\log N$ times (c) N^2 times (d) $N-1$ times (e) N times.

Ans :N-1 times

17. The Sorting methodwhich is used for external sort is

- (a) Bubble sort (b) Quick sort (c) Merge sort (d) Radix sort (e) Selection sort.

Ans :Radix sort

18. The graph colouringalgorithm's time can be bounded by _____

- (a) $O(mnm)$ (b) $O(nm)$ (c) $O(nm \cdot 2n)$ (d) $O(nmn)$.

Ans :O(nmn).

19. Sorting is not possible by using which of the following methods?

- (a) Insertion (b) Selection (c) Deletion (d) Exchange

Ans :Deletion

20. What is the type of the algorithm used in solving the 8 Queens problem?

- (a)Backtracking (b) Dynamic (c) Branch and Bound (d) DandC

Ans :Backtracking

UNIT-II

1. Name the node which has been generated but none of its children nodes have been generated in state space tree of backtracking method.

- (a) Dead node (b) Live node (c) E-Node (d) State Node

Ans: Livenode

2. How many nodes are there in a full state space tree with $n = 6$?

- (a) 65 (b) 64 (c) 63 (d) 32

Ans : 63

3. This algorithm scans the list by swapping the entries whenever pair of adjacent keys are out of desired order.

- (a) Insertion sort. (b) Bubble sort. (c) Shell sort. (d) Quick sort.

Ans: Bubble sort.

5. From the following chose the one which belongs to the algorithm paradigm other than to which others from the following belongs to.

- (a) Minimum & Maximum problem. (b) Knapsack problem. (c) Selection problem.(d) Merge sort.

Ans: Knapsack problem.

6. To calculate $c(i, j)$'s, $w(i, j)$'s and $r(i, j)$'s; the OBST algorithm in worst case takes the following time.

- (a) $O(\log n)$ (b) $O(n^4)$ (c) $O(n^3)$ (d) $O(n \log n)$

Ans: O (n3)

7. What is the type of the algorithm used in solving the 4 Queens problem?

- (a) Greedy (b) Dynamic (c) Branch and Bound (d) Backtracking.

Ans: Backtracking.

8. In Knapsack problem, the best strategy to get the optimal solution, where P_i , W_i is the Profit, Weight associated with each of the X_i object respectively is to

- (a) Arrange the values P_i/W_i in ascending order (b) Arrange the values P_i/X_i in ascending order
(c) Arrange the values P_i/W_i in descending order (d) Arrange the values P_i/X_i in descending order

Ans: Arrange the values P_i/X_i in descending order

9. Greedy job scheduling with deadlines algorithms' complexity is defined as

- (a) $O(N)$ (b) $\Omega(n \log n)$ (c) $O(n^2 \log n)$ (d) $O(n \log n)$

Ans: O(N)

12. From the following choose the one which belongs to the algorithm paradigm other than to which others from the following belongs to.

- (a) Minimum & Maximum problem (b) Knapsack problem (c) Selection problem (d) Merge sort

Ans : Knapsack problem

14. Identify the name of the sorting in which time is not proportional to n^2 .

- (a) Selection sort (b) Bubble sort (c) Quicik sort (d) Insertion sort.

Ans : Insertion sort

15. The optimal solution to a problem is a combination of optimal solutions to its subproblems. This is known as

- (a) Principle of Duality (b) Principle of Feasibility (c) Principle of Optimality (d) Principle of Dynamicity.

Ans : Principle of Optimality

16. Which of the following versions of merge sort algorithm does uses space efficiently?

- (a) Contiguous version (b) Array version (c) Linked version (d) Structure version (e) Heap version.

Ans : Linked version

17. Identify the correct problem for multistage graph from the list given below.

- (a) Resource allocation problem (b) Traveling salesperson problem
(c) Producer consumer problem (d) Barber's problem

Ans : Resource allocation problem

18. How many edges are there in a Hamiltonian cycle if the edge cost is 'c' and the cost of cycle is ' cn '

- (a)c (b) cn (c) n (d) $2c$

Ans :n.

19. A problem L is NP-complete iff L is NP-hard and

- (a) $L \approx NP$ (b) $L \alpha NP$ (c) $L \in NP$ (d) $L = NP$

Ans : $L \in NP$

20. What would be the cost value for any answering node of a sub tree with root 'r' using branch-bound algorithm?

- (a) Maximum (b) Minimum (c) Optimal (d) Average

Ans: Minimum

UNIT-III

1. From the following pick the one which does not belongs to the same paradigm to which others belongs to.

(a) Minimum & Maximum problem (b) Knapsack problem
(c) Selection problem (d) Merge sort

Ans:Knapsack problem

2. Prims algorithm is based on _____ method

 - a. Divide and conquer method c. Dynamic programming
 - b. Greedy method d. Branch and bound

3. Graph Method

Ans. Greedy Method

3. The amount of memory needs to run to completion is known as _____

 - a. Space complexity c. Worst case
 - b. Time complexity d. Best case

Ans: Space complexity

4. The amount of time needs to run to completion is known as _____

 - a. Space complexity c. Worst case
 - b. Time complexity d. Best case

Ans: Time complexity

- Ans. Time complexity

5. _____ is the minimum number of steps that can be executed for the given parameters

 - a. Average case c. Worst case
 - b. Time complexity d. Best case

Ans: Best case

6. _____ is the maximum number of steps that can be executed for the given parameters

 - a. Average case c. Worst case
 - b. Time complexity d. Best case

Ans: Worst case

- Ans: Worst case

7. _____ is the average number of steps that can be executed for the given parameters
a. Average case c. Worst case
b. Time complexity d. Best case

Ans: Average Case

8. Testing of a program consists of 2 phases which are _____ and _____

- a. Average case & Worst case b. Time complexity & Space complexity
 - c. Validation and checking errors d. Debugging and profiling

Ans: Debugging and profiling

9. Worst case time complexity of binary search is _____
a. O(n) b. O(logn)c. $\Theta(n\log n)$ d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

10. Best case time complexity of binary search is _____

 - a. O(n) c. $\Theta(n \log n)$
 - b. O($\log n$) d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

- Ans: O(logn)

11. Average case time complexity of binary search is _____

a. O(n) c. $\Theta(n\log n)$

b. $O(\log n)$ d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

12. Merge sort invented by _____

- a. CARHOARE c. HAMILTON
- b. JOHN VON NEUMANN d. STRASSEN

Ans : JOHN VON NEUMANN

13. Quick sort invented by _____

- a. CARHOARE c. HAMILTON
- b. JOHN VON NEUMANN d. STRASSEN

Ans : CARHOARE

14. Worst case time complexity of Quick sort is _____

- a. $O(n^2 \log 7)$ c. $O(n \log n)$
- b. $O(n^2)$ d. $O(\log n)$

Ans : $O(n^2)$

15. Best case time complexity of Quick sort is _____

- a. $O(n^2 \log n)$ c. $O(n \log n)$
- b. $O(\log n)$ d. $O(\log n^2)$

Ans : $O(n \log n)$

16. Average case time complexity of Quick sort is _____

- a. $\Theta(n \log n)$ b. $O(\log n)$ c. $O(n \log n)$ d. $\Theta(\log n)$

17. Which design strategy stops the execution when it finds the solution otherwise starts the problem from top

- a. Back tracking c. Divide and conquer
- b. Branch and Bound d. Dynamic programming

Ans: Back Tracking

18. Graphical representation of algorithm is _____

- a. Pseudo-code c. Graph Coloring
- b. Flow Chart d. Dynamic programming

Ans: Flow Chart

19. In pseudo-code conventions input express as _____

- a. input c. Read
- b. Write d. Return

Ans : Write

20. In pseudo-code conventions output express as _____

- a. input c. Read
- b. Write d. Return

Ans : Read

UNIT-IV

1. Tight bound is denoted as _____

- a. Ω c. Θ
- b. Ω d. O

Ans : Θ

2. Upper bound is denoted as _____

- a. Ω c. Θ
- b. ω d. O

Ans : O

3. lower bound is denoted as _____

- a. Ω c. Θ
- b. ω d. O

Ans : Ω

4. The function $f(n)=o(g(n))$ if and only if Limit $f(n)/g(n)=0$ as $n \rightarrow \infty$

- a. Little oh b. Little omega

b. Big oh d. Omega

Ans : Little oh

5. The function $f(n)=o(g(n))$ if and only if $\lim_{n \rightarrow \infty} g(n)/f(n)=0$

a. Little oh b. Little omega

b. Big oh d. Omega

Ans : Little omega

6. The general criteria of algorithm; zero or more quantities are externally supplied is _____

a. Output b. Finiteness

b. Effectiveness d. Input

Ans : Input

7. The general criteria of algorithm; at least one quantity is produced _____

a. Output b. Finiteness

b. Effectiveness d. Input

Ans : Output

8. The general criteria of algorithm; Each instruction is clear and unambiguous _____

a. Output b. Definiteness

b. Effectiveness d. Input

Ans : Definiteness

9. The general criteria of algorithm; algorithm must terminates after a finite number of steps _____

a. Output b. Finiteness

b. Effectiveness d. Input

Ans : Finiteness

10. Which is not a criteria of algorithm

a. Input b. Output

b. Time complexity d. Best case

Ans : Best case

11. Which is not in general criteria of algorithm

a. Input b. Output

b. Time complexity d. Effectiveness

Ans : Time complexity

12. Time complexity of given algorithm

Algorithm Display(A)

{

S:=0.0;

For i:=0 to n-1

{

S:=S+A[i];

Return S;

}

}

a. $4n+4$ c. $4n^2+4$

b. $2n^2+2n+2$ d. $4n+4$

Ans : 4n+4

13. Time complexity of given algorithm

AlgorithmSum(A,S)

{

for i:=1 to n-1

{

for j:=2 to n-1

```

{
S:=S+i+j;
return S;
}
}
}

a.  $6n^2 - 14n + 4$  c.  $4n^2 + 6n + 12$   

b.  $6n^2 + 14n + 10$  d.  $6n^2 - 14n + 10$ 

```

Ans : $6n^2 - 14n + 10$

14. Kruskal algorithm is based on _____ method
 a. Divide and conquer method b. Greedy method c. Dynamic programming d. Branch and bound

Ans. Greedy method

15. Prims algorithm is based on _____ method
 a. Divide and conquer method c. Dynamic programming
 b. Greedy method d. Branch and bound

Ans. Greedy Method

16. The output of Kruskal and Prims algorithm is _____
 a. Maximum spanning tree c. Spanning tree
 b. Minimum spanning tree d. None of these

UNIT-V

1. Job sequencing with deadline is based on _____ method
 a. greedy method c. branch and bound
 b. dynamic programming d. divide and conquer

Ans. Greedy method

2. Fractional knapsack is based on _____ method
 a. greedy method c. branch and bound
 b. dynamic programming d. divide and conquer

Ans. Greedy method

3. 0/1 knapsack is based on _____ method
 a. greedy method c. branch and bound
 b. dynamic programming d. divide and conquer

Ans. Dynamic programming

4. The files x_1, x_2, x_3 are 3 files of length 30, 20, 10 records each. What is the optimal merge pattern value?
 a. 110 c. 60
 b. 90 d. 50

Ans. 90

5. The optimal merge pattern is based on _____ method
 a. Greedy method b. Dynamic programming
 c. Knapsack method d. Branch and bound

Ans. Greedy method

6. Who invented the word Algorithm
 a. Abu Ja'far Mohammed ibn Musa c. Abu Mohammed Khan
 b. Abu Jafar Mohammed Kasim d. Abu Ja'far Mohammed Ali Khan

Ans. Abu Ja'far Mohammed ibn Musa

7. In Algorithm comments begin with _____
 a. /* c. /
 b. */ d. //

Ans : //

8. The _____ of an algorithm is the amount of memory it needs to run to completion.

- a. Space Complexity c. Best Case
- b. Time Complexity d. Worst Case

Ans : Space Complexity

9. _____ is the process of executing a correct program on data sets and measuring the time and space it takes to compute the results.

- a. Debugging c. Combining
- b. Profiling d. Conquer

Ans : Profiling

10. In Algorithm Specification the blocks are indicated with matching _____

- a. Braces c. Square Brackets
- b. Parenthesis d. Slashes

Ans : Braces

11. Huffmancode are the applications of _____ with minimal weighted external path length obtained by an optimal set.

- a. BST b. MST
- c. Binary tree d. Weighted Graph

Ans : Binary tree

12. From the following which is not return optimal solution

- a. Dynamic programming c. Backtracking
- b. Branch and bound d. Greedy method

Ans . Backtracking

13. _____ is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions

- a. Dynamic programming c. Backtracking
- b. Branch and bound d. Greedy method

Ans : Dynamic programming

14. The name backtrack was first coined by _____

- a. D.H.Lehmer c. L.Baumert
- b. R.J.Walker d. S. Golomb

Ans : D.H.Lehmer

15. The term _____ refers to all state space search methods in which all children of the – nodes are generated before any other live node can become the E-node.

- a. Backtacking c. Depth First Search
- b. Branch and Bound d. Breadth First Search

Ans ; Branch and Bound

16. A _____ is a round trip path along n edges of G that visits every vertex once and returns to its starting position.

- a. MST c. TSP
- b. Multistage Graph d. Hamiltonian Cycle

Ans :Hamiltonian Cycle

17. Graph Coloring is which type of algorithm design strategy

- a. Backtacking c. Greedy
- b. Branch and Bound d. Dynamic programming

Ans : Backtracking

18. Which of the following is not a limitation of binary search algorithm?

- a. must use a sorted array
- b. requirement of sorted array is expensive when a lot of insertion and deletions are needed

- c. there must be a mechanism to access middle element directly
- d. binary search algorithm is not efficient when the data elements are more than 1000.

Ans : binary search algorithm is not efficient when the data elements are more than 1000.

19. Binary Search Algorithm cannot be applied to
- a. Sorted linked list c. Sorted linear array
 - b. Sorted binary tree d. Pointer array

Ans :Sorted linked list

20. Two main measures for the efficiency of an algorithm are
- a. Processor and memory c. Time and space
 - b. Complexity and capacity d. Data and space

Ans : Time and Space

XII. GATE QUESTIONS:

1 The order of an internal node in a B+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

- A) 24 B) 25 C) 26 D) 27

Answer : (C)

2 The best data structure to check whether an arithmetic expression has balanced parentheses is a

- A) queue B) stack C) tree D) list

Answer : (B)

3. A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is given below: 10, 8,5,3,2 Two new elements 1 and 7 are inserted in the heap in that order. The level-order traversal of the heap after the insertion of the elements is

- A) 10,8,7,5,3,2,1 B) 10,8,7,2,3,1,5 C) 10,8,7,1,2,3,5 D) 10,8,7,3,2,1,5

Answer : (D)

4 The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

- A) 2 B) 3 C) 4 D) 6

Answer : (B)

5 The goal of structured programming is to

- A) have well indented programs B) be able to infer the flow of control from the compiled code C) be able to infer the flow of control from the program text D) avoid the use of GOTO statements

Answer : (C)

6 The tightest lower bound on the number of comparisons, in the worst case, for comparison-based sorting is of the order of

- A) n B) n^2 C) $n \log n$ D) $n \log^2 n$

Answer : (B)

7 Let G be a simple graph with 20 vertices and 100 edges. The size of the minimum vertex cover of G is 8. Then, the size of the maximum independent set of G is

- A) 12 B) 8 C) Less than 8 D) More than 12

Answer : (A)

8 Let A be a sequence of 8 distinct integers sorted in ascending order. How many distinct pairs of sequences, B and C are there such that (i) each is sorted in ascending order, (ii) B has 5 and C has 3 elements, and (iii) the result of merging B and C gives A ?

A) 2 B) 30 C) 56 D) 256

Answer : (D)

9 A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The level-order traversal of the heap is given below: 10, 8,5,3,2 Two new elements 1 and 7 are inserted in the heap in that order. The level-order traversal of the heap after the insertion of the elements is

A) 10,8,7,5,3,2,1 B) 10,8,7,2,3,1,5 C) 10,8,7,1,2,3,5 D) 10,8,7,3,2,1,5

Answer : (D)

10 The S-N curve for steel becomes asymptotic nearly at

A) 10^3 cycles B) 10^4 cycles C) 10^6 cycles D) 10^9 cycles

Answer : (C)

XIII. WEBSITES:

- http://www.ki.inf.tu-dresden.de/~hans/www-adr/alg_course.html --String Matching, Sorting, Linear Programming
- http://www.algorithmist.com/index.php/Main_Page it contains dynamic programming, greedy, ,Graph Theory, sorting, Data Structures

<http://www-2.cs.cmu.edu/~guyb/realworld.html>- it contains Data Compression, Indexing and Search engines, linear Programming, Pattern matching

XIV. EXPERT DETAILS:

Professor Sartaj Kumar Sahni is an Indian computer scientist, now based in the USA, and is one of the pioneers in the field of data structures. He is a distinguished professor in the Department of Computer and Information Science and Engineering at the University of Florida.

<http://www.cise.ufl.edu/~sahni/>

XV. JOURNALS:

1. Journal of Graph Algorithms and Applications
[url: http://www.emis.de/journals/JGAA/home.html](http://www.emis.de/journals/JGAA/home.html)
2. Algorithmica –A journal about the design of algorithms in many applied and fundamental areas <http://www.springerlink.com>

XVI. LIST OF TOPICS FOR STUDENT SEMINARS:

- Randomized Algorithms, Binary search, Connected components and spanning Trees
- Hamiltonian cycles, Single source shortest path problem, Non-deterministic algorithms

XVII. CASE STUDIES / SMALL PROJECTS:

1. Techniques for Algorithm Design and Analysis: Case Study of a Greedy Algorithm.

Six different implementations of a greedy dominating set algorithm are presented and analyzed. The implementations and analysis illustrate many of the important techniques in the design and analysis of algorithms, as well as some interesting graph theory.

2. Scheduling Two Salesmen in a Network.

The two-server problem is concerned with the movement of two servers to request points in a metric space. We consider an offline version of the problem in a graph in which the requests may be served in any order. A family of approximations algorithms is developed for this NP-complete problem.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, RAMAPURAM CAMPUS
COMPUTER SCIENCE AND ENGINEERING
QUESTION BANK
18CSC204J DESIGN AND ANALYSIS OF ALGORITHMS

UNIT 1

Introduction-Algorithm Design, Fundamentals of Algorithms, Correctness of algorithm, Time complexity analysis, Insertion sort-Line count, Operation count, Algorithm Design paradigms, Designing an algorithm, And its analysis-Best, Worst and Average case, Asymptotic notations Based on growth functions. O,O, Θ , ω , Ω Mathematical analysis, Induction, Recurrence relations , Solution of recurrence relations, Substitution method, Solution of recurrence relations, Recursion tree, Solution of recurrence relations, Examples

PART A

1. _____ is the first step in solving the problem

- A. Understanding the Problem
- B. Identify the Problem
- C. Evaluate the Solution
- D. Coding the Problem

Answer: - B

2. While solving the problem with computer the most difficult step is _____.

- A. describing the problem
- B. finding out the cost of the software
- C. writing the computer instructions
- D. testing the solution

Answer:- C

3. _____ solution requires reasoning built on knowledge and experience

- A. Algorithmic Solution
- B. Heuristic Solution
- C. Random Solution
- D. Brute force Solution

Answer: - B

4. The correctness and appropriateness of _____ solution can be checked very easily.

- A. algorithmic solution
- B. heuristic solution
- C. random solution
- D. Brute force Solution

Answer:- A

5. When determining the efficiency of algorithm, the space factor is measured by

- A. Counting the maximum memory needed by the algorithm
- B. Counting the minimum memory needed by the algorithm
- C. Counting the average memory needed by the algorithm
- D. Counting the maximum disk space needed by the algorithm

Answer: - A

6. The elements of an array are stored successively in memory cells because

- A. by this way computer can keep track only the address of the first element and the addresses of other elements can be calculated
- B. the architecture of computer memory does not allow arrays to store other than serially
- C. Either A or B
- D. Both A and B

Answer: - A

7. The hierarchy of operations is denoted as _____.

I. +, - II. Power III. *, / IV. \, MOD

- A. I, II, III, IV
- B. II, IV, III, I
- C. IV, I, III, II
- D. II, III, IV, I

Answer:- B

8. What is the time complexity of following code:

```
int a = 0, i = N;
while (i > 0)
{
    a += i;
    i /= 2;
}
```

- A. O(N)
- B. O(Sqrt(N))
- C. O(N / 2)
- D. O(log N)

Answer: - D

9. Two main measures for the efficiency of an algorithm are

- A. Processor and memory
- B. Complexity and capacity
- C. Time and space
- D. Data and space

Answer: - C

10. What does the algorithmic analysis count?

- A. The number of arithmetic and the operations that are required to run the program
- B. The number of lines required by the program
- C. The number of seconds required by the program to execute
- D. None of these

Answer:- A

11. An algorithm that indicates the amount of temporary storage required for running the algorithm, i.e., the amount of memory needed by the algorithm to run to completion is termed as_____.

- A. Big Theta $\Theta(f)$
- B. Space complexity
- C. Big Oh $O(f)$
- D. Time Complexity

Answer B

12. Consider a linked list of n elements. What is the time taken to insert an element after an element pointed by some pointer?

- A. (1)
- B. (n)
- C. $(\log_2 n)$
- D. $(n \log_2 n)$

Answer A

13. If the address of $A[1][1]$ and $A[2][1]$ are 1000 and 1010 respectively and each element occupies 2 bytes then the array has been stored in order.

- A. row major
- B. column major
- C. matrix major
- D. none of these

Answer A

14. The time factor when determining the efficiency of algorithm is measured by

- A. Counting microseconds
- B. Counting the number of key operations
- C. Counting the number of statements
- D. Counting the kilobytes of algorithm

Answer B

15. Time complexities of three algorithms are given. Which should execute the slowest for large values of N?

- A. $(n \log n)$
- B. $O(n)$
- C. $O(\log n)$
- D. $O(n^2)$

Answer B

16. Which one of the following is the tightest upper bound that represents the number of swaps required to sort n number using selection sort?

- A. $(\log n)$
- B. $O(n)$
- C. $(n \log n)$
- D. $O(n^2)$

Answer B

17. How many comparisons are needed for linear Search array when elements are in order in best case?

- A. 1
- B. n
- C. $n+1$
- D. $n-1$

Answer A

18. The complexity of Bubble sort algorithm is_____

- A. $O(n)$
- B. $O(\log n)$
- C. $O(n^2)$
- D. $O(n \log n)$

Answer : C

19. What is the time complexity of following code:

```
int a = 0, i = N;
while (i > 0)
{
    a += i;
    i /= 2;
}
```

- A. $O(N)$
- B. $O(\text{Sqrt}(N))$
- C. $O(N / 2)$
- D. $O(\log N)$

Answer D

20. Which of the given options provides the increasing order of asymptotic complexity of functions f1, f2, f3 and f4?

f1(n) = 2^n

f2(n) = n^(3/2)

- A. f3, f2, f1, f4
- B. f2, f3, f1, f4
- C. f2, f3, f4, f1
- D. f3, f2, f4, f1

Answer is: D

f3(n) = nLogn

f4(n) = n^(Logn)

21. How much number of comparisons is required in insertion sort to sort a file if the file is sorted in reverse order?

- A. N^2
- B. N

- C. N-1
- D. N/2

Answer A

22. The worst-case occur in linear search algorithm when

- A. Item is somewhere in the middle of the array
- B. Item is not in the array at all
- C. Item is the last element in the array
- D. Item is the last element in the array or item is not there at all

Answer D

23. What is the time complexity of fun()?

```
int fun(int n)
{
    int count = 0;
    for (int i = 0; i < n; i++)
        for (int j = i; j > 0; j--)
            count = count + 1;
    return count;
}
```

- A. Theta (n)

- C. Theta (n*Logn)

- B. Theta (n^2)

- D. Theta (nLognLogn)

Answer : B

24. The time complexity of the following C function is (assume n > 0)

```
(int recursive (mt n)
{
    if (n == 1)
        return (1);
    else
        return (recursive (n-1) + recursive (n-1));
```

}

- A. $O(n)$
- B. $O(n \log n)$
- C. $O(n^2)$
- D. $O(2^n)$

Answer D

25. **A function in which $f(n) = \Omega(g(n))$, if there exist positive values k and c such that $f(n) \geq c*g(n)$, for all $n \geq k$. This notation defines a lower bound for a function f(n):**

- A. Big Omega $\Omega(f)$
- B. Big Theta $\Theta(f)$
- C. Big Oh $O(f)$
- D. Big Alpha $\alpha(f)$

Answer A

26. **The concept of order Big O is important because _____**

- A. It can be used to decide the best algorithm that solves a given problem
- B. It determines the maximum size of a problem that can be solved in a given amount of time
- C. It is the lower bound of the growth rate of algorithm
- D. Both A and B

Answer A

27. **The upper bound on the time complexity of the nondeterministic sorting algorithm is**

- A. $O(n)$
- B. $O(n \log n)$
- C. $O(1)$
- D. $O(\log n)$

Answer: A

28. **In the analysis of algorithms, what plays an important role?**

- A. Text Analysis
- B. Growth factor
- C. Time
- D. Space

Answer: B

29. **Which one of the following correctly determines the solution of the recurrence relation given below with $T(1) = 1$ and $T(n) = 2T(n/4) + n^{1/2}$**

- A. $O(n^2)$
- B. $O(n)$
- C. $O(n^{1/2} \log n)$
- D. $O(\log n)$

Answer C

30. **What is the time complexity of recursive function given below:**

$$T(n) = 4T(n/2) + n^2$$

- | | |
|-----------------------|----------------------------|
| A. O(n ²) | C. O(n ² log n) |
| B. O(n) | D. O(n log n) |

Answer C

PART B

- 1 . What is an Algorithm?
- 2 . Give the notion of an algorithm.
- 3 . Design an algorithm for computing gcd(m,n) using Euclid's algorithm.
- 4 . Design an algorithm to compute the area and circumference of a circle.
- 5 . Differentiate Sequential and Parallel Algorithms.
- 6 . Write the process for design and analysis of algorithm.
- 7 . What are the fundamental steps for design and analysis of an algorithm?
- 8 . Compare Exact and Approximation algorithm.
- 9 . What is an Algorithm Design Technique?
- 10 . Define Pseudo code.
- 11 . Define Flowchart.
- 12 . Prove the correctness of an algorithm's.
- 13 . Define algorithm validation.
- 14 . What is validation and program verification?
- 15 . Define program proving and program verification.
- 16 . Write the characteristics of an algorithm.
- 17 . What is the Efficiency of algorithm?
- 18 . What is time and space complexity?
- 19 . What is generality of an algorithm?
- 20 . What is algorithm's Optimality?
- 21 . Write an algorithm to find the number of binary digits in the binary representation of a positive decimal integer.

- 22 . What are the types of problems in algorithm?
- 23 . How will you measure input size of algorithms?
- 24 . What is the average case complexity of linear search algorithm?
- 25 . Differentiate searching and sorting algorithm.
- 26 . What are combinatorial problems?
- 27 . Define a graph and its type.
- 28 . Define performance analysis.
- 29 . What do you mean by Worst case-Efficiency of an algorithm?
- 30 . What do you mean by Best case-Efficiency of an algorithm?
- 31 . Define the Average-case efficiency of an algorithm.
- 32 . What do you mean by Amortized efficiency?
- 33 . How to analyze an algorithm framework?
- 34 . How to measure the algorithm's efficiency?
- 35 . What is called the basic operation of an algorithm?

- 36 .How to measure an algorithm's running time?
- 37 .Define time and space complexity.
- 38 .Write an algorithm for adding 'n' natural numbers and find the time and space required by that algorithm.
- 39 .Define order of growth.
- 40 .What is meant by linear search?
- 41 .Compare the two functions 2^n and n^2 for various values of n. Determine when the second function will become the same, smaller and larger than the first function.
- 42 .What are the properties of big-Oh notation?
- 43 .Define Big oh notation.
- 44 .Define little Oh and Omega notations.
- 45 .Define Ω notation.
- 46 .Define Θ – notation.
- 47 .What is the use of Asymptotic Notations?
- 48 .What are the properties of asymptotic notations?
- 49 .Mention the general plan for analyzing time efficiency of Non recursive algorithms.
- 50 .Define recursive and non – recursive algorithm.
- 51 .What is recurrence equation?
- 52 .Define Recurrence relation with an example.
- 53 .Give the time complexity $1+3+5+7+\dots+999$.
- 54 .Compare order of growth $n(n-1)/2$ and n^2 .
- 55 .Find the order of growth of the following sums.

$$\sum_{i=1}^{n-1} (i^2 + 1)^2$$

- 56 .Solve the following recurrence relations.

$X(n)=x(n-1) + 5$ for $n>1$, $x(1) = 0$

- 57 .Consider the following algorithm

```

S=0
for =1 to n do
    S=S+i
return i
  
```

What does this algorithm compute? How many times is the basic operation executed?

- 58 .Design an algorithm to compute the area and Circumference of a circle.
- 59 .The $(\log n)$ th smallest number of n unsorted numbers can be determined in $O(n)$ average-case time.
- 60 .Write the recursive Fibonacci algorithm and its recurrence relation.

PART C

- 1 . Describe the steps in analyzing & coding an algorithm.
- 2 . Enumerate the problem types used in the design of algorithm.
- 3 . What are the steps that need to be followed while designing and analyzing algorithm?
- 4 . Explain the fundamental of algorithmic problem solving.
- 5 . Use the most appropriate notation to indicate the time efficiency class of sequential algorithm in the worst case, best case and the average case.
- 6 . Consider the following algorithm for the searching problem.

Algorithm:

```

Linear search (A[0,...n-1],key)
//Searches an array for a key value by linear search
//Input: Array A[0..n-1] of values and a key value to search
//Output: Returns index if search is successful
for i<-0 to n-1 do
    if(key==A[i])
        return i

```

- 7 . Explain some of the problem types used in the design of algorithm.
- 8 . Define time complexity and space complexity. Write an algorithm for adding 'n' natural numbers and find the time and space required by that algorithm.
- 9 . Explain the general framework for analyzing the efficiency of algorithm.
- 10 .Write the Insertion Sort algorithm and estimate its running time.
- 11 .What is space complexity? With an example, explain the components of fixed and Variable part in space complexity?
- 12 .Show how to implement a stack using two queues. Analyze the running time of stack operations.
- 13 .Discuss the properties of asymptotic notations.
- 14 .Explain the various asymptotic notations used in algorithm design. With an Example
- 15 .Give the definition and graphical representation of O notations.
- 16 .Define asymptotic notations. Distinguish between Asymptotic notation and conditional asymptotic notation.
- 17 .Prove that for any two functions $f(n)$ and $g(n)$, we have

- $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
- 18 .Write the linear search algorithm and analyze for its best worst and average case time
Complexity.
- 19 .Discuss about recursive and non-recursive algorithms with example.
- 20 .What is the general plan for time efficiency of recursive algorithm and find the number
of binary digits in the binary representation of positive decimal integer find recurrence relation and complexity.
- 21 .State the general plan for analyzing the time efficiency of non-recursive algorithms and
explain with an example.
- 22 .Compare the order of the growth of the following.
 i) $(1/2)n(n-1)$ and n^2
 ii) $\log_2 n$ and \sqrt{n}
 iii) $n!$ and 2^n
- 23 .Find the closest asymptotic tight bound by solving the recurrence equation
 $T(n)=8T(n/2)+n^2$ with $(T(1)=1)$ using Recursion tree method. [Assume $T(1) \in \Theta(1)$]
- 24 .Give an algorithm to check whether all the elements in a given array of n-elements are
distinct, find the worst case complexity of the same.
- 25 .Explain the towers of Hanoi problem and solve it using recursion
- 26 .Prove the time complexity of the matrix multiplication is $O(n^3)$
- 27 . Define recurrence equation and explain how solving recurrence equations are done.
- 28 . Solve the following recurrence relations.
 I. $x(n)=x(n-1)+5$ for $n>1$, $x(1)=0$
 II. $x(n)=3x(n-1)$ for $n>1$, $x(1)=4$
 III. $x(n)=x(n-1)+n$ for $n>0$, $x(0)=0$
 IV. $x(n)=x(n/2)+n$ for $n>1$, $x(1)=1$ (solve for $n=2^k$)
 V. $x(n)=x(n/3)+1$ for $n>1$, $x(1)=1$ (solve for $n=3^k$)
- 29 . Solve the following recurrence relations.
 I. $x(n)=x(n-1)+n$ for $n>0$, $x(0)=0$

- II. $x(n)=x(n/2)+n$ for $n>1$, $x(1)=1$ (solve for $n=2^k$)
III. $x(n)=3x(n-1)$ for $n>1, x(1)=4$
- 30 . Suppose W satisfies the following recurrence equation and base case (where c is constant) : $W(n)=c.n+W(n/2)$ and $W(1)=1$. What is the asymptotic order of $W(n)$. With a suitable example, explain the method of solving recurrence equations.
- 31 . Consider the following recursion algorithm $\text{Min1}(A[0 \dots n-1])$
- ```

If n=1 return A[0]
Else temp = Min1(A[0.....n-2]) If temp <= A[n-1] return temp
Else
 Return A[n-1]

```
- What does this algorithm compute?
- 32 . Consider the following algorithm.
- ```

Algorithm :
Sum(n)
// A non negative integer n
S <- 0
for i <- 1 to n do
    S<-S+i
Return S

```
- i. What does this algorithm compute?
 - ii. What is its basic operation?
 - iii. How many times is the basic operation executed?
 - iv. What is the efficiency class of this algorithm?
 - v. Suggest an improved algorithm and indicate its efficiency class. If you cannot do it, try to prove that it cannot be done.
- 33 . Setup a recurrence relation for the algorithms basic operation count and solve it.
- 34 . Derive the recurrence relation for Fibonacci series algorithm; also carry out the time complexity analysis.
- 35 . Give the non recursive algorithm for finding the value of the largest element in a list of n numbers.

UNIT II: DIVIDE AND CONQUER

Introduction-Divide and Conquer, Maximum Sub array Problem- Binary Search, Complexity of binary search- Merge sort, Time complexity analysis-Quick sort and its Time complexity analysis, Best case, Worst case, Average case analysis - Strassen's Matrix multiplication and its recurrence relation, Time complexity analysis of Merge sort - Largest sub-array sum, Time complexity analysis of Largest sub-array sum- Master Theorem Proof, Master theorem examples- Finding Maximum and Minimum in an array, Time complexity analysis- Examples- Algorithm for finding closest pair problem, Convex Hull problem

PART-A

1.) Partition and exchange sort is_____

- A. quick sort
- B. tree sort
- C. heap sort
- D. bubble sort

ANSWER: A

2) Which of the following is not the required condition for binary search algorithm?

- A. The list must be sorted
- B. There should be the direct access to the middle element in any sub list
- C. There must be mechanism to delete and/or insert elements in list.
- D. Number values should only be present

ANSWER: C

3) Which of the following sorting algorithm is of divide and conquer type?

- A. Bubble sort
- B. Insertion sort
- C. Merge sort

D. Selection sort

ANSWER: C

4) _____ order is the best possible for array sorting algorithm which sorts n item.

- A. $O(n \log n)$
- B. $O(n^2)$
- C. $O(n + \log n)$
- D. $O(\log n)$

ANSWER: C

5) The complexity of merge sort algorithm is _____

- A. $O(n)$
- B. $O(\log n)$
- C. $O(n^2)$
- D. $O(n \log n)$

ANSWER: D

6) Binary search algorithm cannot be applied to _____

- A. sorted linked list
- B. sorted binary trees
- C. sorted linear array
- D. pointer array

ANSWER: A

7) Which of the following is not a limitation of binary search algorithm?

- A. must use a sorted array
- B. requirement of sorted array is expensive when a lot of insertion and deletions are needed
- C. there must be a mechanism to access middle element directly
- D. binary search algorithm is not efficient when the data elements more than 1500.

ANSWER: D

8) Which of the following is an external sorting?

- A. Insertion Sort
- B. Bubble Sort
- C. Merge Sort
- D. Tree Sort

ANSWER: B

9) Merging k sorted tables into a single sorted table is called _____

- A. k way merging
- B. k th merge
- C. k+1 merge
- D. k-1 merge

ANSWER: A

10) The operation that combines the element is of A and B in a single sorted list C with $n=r+s$ element is called _____

- A. Inserting
- B. Mixing
- C. Merging
- D. Sharing

ANSWER: C

11) Which of the following is a stable sorting algorithm?

- a) Merge sort
- b) typical in-place quick sort
- c) Heap sort
- d) Selection sort

ANSWER: A

12) Which of the following is not an in-place sorting algorithm?

- a) Selection sort
- b) Heap sort

- c) Quick sort
- d) Merge sort

ANSWER: D

13)The time complexity of a quick sort algorithm which makes use of median, found by an $O(n)$ algorithm, as pivot element is

- a) $O(n^2)$
- b) $O(n \log n)$
- c) $O(n \log \log n)$
- d) $O(n)$

ANSWER: B

14) Which of the following algorithm design technique is used in the quick sort algorithm?

- a) Dynamic programming
- b) Backtracking
- c) Divide-and-conquer
- d) Greedy method

ANSWER: C

15) Merge sort uses

- a) Divide-and-conquer
- b) Backtracking
- c) Heuristic approach
- d) Greedy approach

ANSWER: A

16)For merging two sorted lists of size m and n into sorted list of size $m+n$, we require comparisons of

- a) $O(m)$
- b) $O(n)$
- c) $O(m+n)$
- d) $O(\log m + \log n)$

ANSWER: C

17) The running time of Strassen's algorithm for matrix multiplication is

- (A) $\Theta(n)$ (B) $\Theta(n^3)$ (C) $\Theta(n^2)$ (D) $\Theta(n^{2.81})$

ANSWER: D

18) The Stassen's algorithm's time complexity is

- (A) $O(n)$ (B) $O(n^2)$ (C) $O(n^{2.80})$ (D) $O(n^{2.81})$

ANSWER: C

19) Which algorithm is used for matrix multiplication?

- a. Simple algorithm
- b. Specific algorithm
- c. Strassen algorithm
- d. Addition algorithm

ANSWER: C

20) Which algorithm is a divided and conquer algorithm that is asymptotically faster:

- a. Simple algorithm
- b. Specific algorithm
- c. Strassen algorithm
- d. Addition algorithm

ANSWER: C

21) Which algorithm is named after Volker Strassen

- a. Strassen algorithm
- b. Matrix algorithm
- c. Both
- d. None of these

ANSWER: A

- 22) Which of the following algorithms is NOT a divide & conquer algorithm by nature?
- (A) Euclidean algorithm to compute the greatest common divisor
 - (B) Heap Sort
 - (C) Closest pair problem
 - (D) Quick Sort

Answer: B

23). what is the average case time complexity of merge sort?

- a) $O(n \log n)$
- b) $O(n^2)$
- c) $O(n^2 \log n)$
- d) $O(n \log n^2)$

ANSWER: A

24). which of the following method is used for sorting in merge sort?

- a) Merging
- b) Partitioning
- c) Selection
- d) Exchanging

ANSWER: A

25) Which of the following is not a stable sorting algorithm?

- a) Quick sort
- b) Cocktail sort
- c) Bubble sort
- d) Merge sort

ANSWER: A

26) What is the runtime efficiency of using brute force technique for the closest pair problem?

- a) $O(N)$
- b) $O(N \log N)$
- c) $O(N^2)$

d) $O(N^3 \log N)$

ANSWER: C

27) What is the basic operation of closest pair algorithm using brute force technique?

- a) Euclidean distance
- b) Radius
- c) Area
- d) Manhattan distance

ANSWER: A

28) _____ is a method of constructing a smallest polygon out of n given points.

- a) Closest pair problem
- b) Quick hull problem
- c) Path compression
- d) union-by-rank

ANSWER: B

29) Find the maximum sub-array sum for the given elements.

$\{-2, -1, -3, -4, -1, -2, -1, -5, -4\}$

- a) -3
- b) 5
- c) 3
- d) -1

ANSWER: D

30) Master's theorem is used for?

- a) Solving recurrences
- b) Solving iterative relations
- c) Analyzing loops
- d) Calculating the time complexity of any code

ANSWER: A

PART-B

1. How the large integers are multiplies using divide and conquer technique?
2. Give the recurrence relation for divide and conquer.
3. Define Master Theorem.
4. Find the order of growth for the following recurrence.
5. $T(n) = 4T(n/2) + n^2$, $T(1)=1$
6. Give the examples for divide and conquer method.
7. Write the control abstraction for divide and conquer technique.
8. What are the best case, worst case and average case complexity of Quick sort?
9. Solve the average case recurrence for quick sort.
10. How to search an element using binary search?
11. What are the merits of binary search?
12. What are the merits of divide and conquer technique?
13. What are the demerits of binary search?
14. Trace the operation of the binary search algorithm for the input $-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151$, if you are searching for the element 9.
15. What is the time complexity of binary search?
16. What is average case efficiency and worst case complexity of binary search?
17. Derive the complexity of Binary Search algorithm.
18. What are the best case, worst case and average case complexity of binary Search?
19. Prove the equality $a \log_b c = c \log_b a$.
20. Solve the average case recurrence for quick sort.
21. How the operations performed in Strassen's Matrix multiplication?
22. Compute 2101×1130 by applying the divide and conquer algorithm.
23. What is the time complexity of closest pair and quick hull problem?

PART C

- 1) Determine the efficiency of divide and conquer algorithms.
- 2) Explain and analyze the merge sort algorithm.
- 3) How quick sort can be improved?
- 4) Explain the binary searching algorithm in detail, with an example. Show the worst case efficiency of binary search is in $\Theta(\log n)$
- 5) Explain the divide and conquer strategy with examples.
- 6) Apply quick sort to sort the list E, X, A, M, P, L, E in alphabetical order. Draw the tree of the recursive calls made.
 - i. Write the best case input for the quick sort
 - ii. Find the best case time efficiency for the quick sort

iii. Are the arrays made up of all equal elements the worst case input, the best case input or neither?

- 7) Write a pseudo code for divide & conquer algorithm for finding the position of the largest element in an array of numbers.
- 8) Write a quick sort algorithm and derive the worst case and average case complexity class of this algorithm.
- 9) Discuss the efficiency of quick sort algorithm.
- 10) Explain how the merge sort can be viewed as a recursive application of the Divide and conquer methodology. Suggest a pseudo code for merge sort and analyze its complexities. Trace its application to the following data set 9, 4, 3,8,6,2,1,5,7.
- 11) Explain Strassen's matrix multiplication. Evaluate its efficiency.
- 12) Discuss about the Quick Hull Algorithm.
- 13) Explain in Detail about the Closest pair Algorithm
- 14) Explain in detail about min-max problem using divide and conquer and derive its time complexity.
- 15) Discuss the advantages and disadvantages of Divide and Conquer Algorithm?

UNIT III

Introduction-Greedy and Dynamic Programming, Examples of problems that can be solved by using greedy and dynamic approach Huffman coding using greedy approach, Comparison of brute force and Huffman method of encoding Knapsack problem using greedy approach, Complexity derivation of knapsack using greedy Tree traversals, Minimum spanning tree - Greedy, Kruskal's algorithm - greedy Minimum spanning tree - Prim's algorithm, Introduction to dynamic programming 0/1 knapsack problem, Complexity calculation of knapsack problem Matrix chain multiplication using dynamic programming, Complexity of matrix chain

**multiplication Longest common subsequence using dynamic programming,
Explanation of LCS with an example Optimal binary search tree (OBST)using
dynamic programming, Explanation of OBST with an example**

PART A

1. ----- is a Boolean-valued function that determines whether x can be included into the solution vector
 - a) Overlapping subproblems
 - b) Feasible solution**
 - c) Memoization
 - d) Greedy

Answer B

2. Trees with edge with weights are called -----
 - a) weighted tree**
 - b) unweighted tree
 - c) bruteforce
 - d) Greedy

Answer A

3. ----- is to determine an optimal placement of booster
 - a) Weighted tree
 - b) Vertex
 - c) Tree Vertex Splitting Problem (TVSP)**
 - d) Greedy

Answer C

4. The order in which TVS visits that computes the delay values of the nodes of the tree is called the-----.

- a)treeorder
- b) inorder
- c) preorder
- d) postorder**

Answer D

5. Algorithm TVS takes ----- time, where n is the number f nodes in the tree

- a) $O(N)$**
- b) $\Omega(n \log n)$
- c) $O(n^2 \log n)$
- d) $O(n \log n)$

Answer A

6. ----- is a greedy method to obtain a minimum-cost spanning tree builds this tree edge by edge

- a)Prim's algorithm**
- b)Dynamic algorithm
- c)Greedy algorithm
- d)Dynamic algorithm

Answer A

7. Kruskal's algorithm (choose best non-cycle edge) is better than Prim's
Choose best tree edge) when the graph has relatively few edges

- a)True**
- b)False

Answer A

8. Two sorted files containing n and m records respectively could be merged together to obtain one sorted file in time -----.

- a) $\Omega(n \log n)$
- b) **O(n+m)**
- c) $O(n^2 \log n)$
- d) $O(n \log n)$

Answer **B**

9. The two-way merge pattern can be represented by-----

- a) Weighted tree
- b) Vertex
- c) Binary merge tree**
- d) Greedy

Answer **C**

10. What algorithm technique is used in the implementation of Kruskal solution for the MST?

- a) greedy technique**
- b) divide-and-conquer technique
- c) dynamic programming technique
- d) the algorithm combines more than one of the above techniques

Answer **A**

11. The function Tree of Algorithm uses the ----- stated to obtain a two-way merge tree for n file

- a) divide-and-conquer technique
- b) greedy rule**
- c) dynamic programming technique
- d) the algorithm combines more than one of the above techniques

Answer **B**

12. A decode tree is a----- in which external nodes represent messages.

- a) minimum spanning tree
- b) B tree

c) **Binary tree**

d) AVL tree

Answer **B**

13. The -----in the code word for a message determine the branching needed at each level of the decode tree to reach the correct external node.

a) **binary bits**

b) decoder

c) encoder

d) binary bytes

Answer **A**

14. The cost of decoding a -----is proportional to the number of bits in the code

a) binary bits

b) code word

c) data

d) binary bytes

Answer **B**

15. What is the edges on the shortest paths from a vertex v to all remaining vertices in a connected undirected graph G form a spanning tree of G is called?

a) MST

b) shortest-path spanning tree

c) binary tree

d) AVL tree

Answer **B**

16. -----is an algorithm design method that can be used when the

solution to a problem can be viewed as the result of a sequence of decisions.

a) Dynamic Programming

- b) Greedy method
- c) Huffman coding
- d) Tree traversal

Answer A

17. What is another important feature of the dynamic programming approach that optimal solutions are retained so as to avoid re-computing their values.

a) Dynamic Programming

- b) Greedy method
- c) Huffman coding
- d) Tree traversal

Answer A

18. ----- often drastically reduces the amount of enumeration by avoiding the enumeration of some decision sequences that cannot possibly be optimal.

a) Dynamic Programming

- b) Greedy method
- c) Huffman coding
- d) Tree traversal

Answer A

19. In the -----only one decision sequence is ever generated.

- a) Dynamic Programming

b) Greedy method

- c) Huffman coding
- d) Tree traversal

Answer B

20. Dynamic programming algorithms solve the----- to obtain a solution to the given problem instance
- a) optimistic
 - b) Greedy method
 - c) Huffman coding
 - d) recurrence**

Answer D

21. A dynamic programming formulation for a k-stage graph problem is obtained by first noticing that every s to t path is the result of a sequence of ----- decision.
- a) k
 - b) k-1
 - c) k-2**
 - d) 2k

Answer C

22. Which of the following problems is NOT solved using dynamic programming?
- a) 0/1 knapsack problem
 - b) Matrix chain multiplication problem
 - c) Edit distance problem
 - d) Fractional knapsack problem**

Answer D

23. The problem of -----is to identify a minimum-cost sequence of edit operations that will transform X into Y.
- a) 0/1 knapsack problem
 - b) Matrix chain multiplication problem
 - c) Edit distance problem
 - d) string editing**

Answer D

24. In Knapsack problem, the best strategy to get the optimal solution, where P_i , W_i is the Profit, Weight associated with each of the X_i^{th} object respectively is to

- a) Arrange the values P_i/W_i in ascending order
- b) Arrange the values P_i/X_i in ascending order
- c) Arrange the values P_i/W_i in descending order
- d) Arrange the values P_i/X_i in descending order**

Answer D

25. Greedy job scheduling with deadlines algorithms' complexity

- is defined as
- a) $O(N)$
 - b) $\Omega(n \log n)$**
 - c) $O(n^2 \log n)$
 - d) $O(n \log n)$

Answer B

26. In Huffman coding, data in a tree always occur?

- a) roots
- b) leaves**
- c) left sub trees
- d) right sub trees

Answer B

27. The multistage graph problem can also be solved using the -----

- a) backward approach
- b) forward approach**
- c) brute force approach
- d) right sub trees

Answer B

28. The all-pairs -----problem is to determine a matrix A such that A(i,j)is the length of a shortest path from i to j.

- a) backward approach
- b) forward approach
- c) brute force approach
- d) shortest-path**

Answer **D**

29. Which of the following methods can be used to solve the Knapsack problem?

- a) Brute force algorithm
- b) Recursion
- c) Dynamic programming
- d) Brute force, Recursion and Dynamic Programming**

Answer **D**

30. The inorder and preorder traversals of a binary tree are d b e a f c g and a b d e c f g, respectively. The postorder traversal of the binary tree is:

- a) d e b f g c a**
- b) e d b g f c a
- c) e d b f g c a
- d) d e f g b c a

Answer **A**

31. Which of the following pairs of traversals is not sufficient to build a binary tree from the given traversals?

- a) Preorder and Inorder
- b) Preorder and Postorder
- c) Inorder and Postorder
- d) Inorder and levelorder**

Answer D

32. Consider the following C program segment

```
struct CellNode
{
    struct CelINode *leftchild;
    int element;
    struct CellNode *rightChild;
}

int Dosomething(struct CelINode *ptr)
{
    int value = 0;
    if (ptr != NULL)
    {
        if (ptr->leftChild != NULL)
            value = 1 + DoSomething(ptr->leftChild);
        if (ptr->rightChild != NULL)
            value = max(value, 1 + DoSomething(ptr->rightChild));
    }
    return (value);
}
```

The value returned by the function DoSomething when a pointer to the root of a non empty tree is passed as argument is

- a) The number of leaf nodes in the tree
- b) The number of nodes in the tree
- c) The number of internal nodes in the tree
- d) The height of the tree**

Answer D

33. Given items as {value, weight} pairs {{60, 20}, {50, 25}, {20, 5}}. The capacity of knapsack=40. Find the maximum value output assuming items to be divisible and nondivisible respectively.

- a) 100,80

- b) 110,70
- c) 130,110
- d) 110,80**

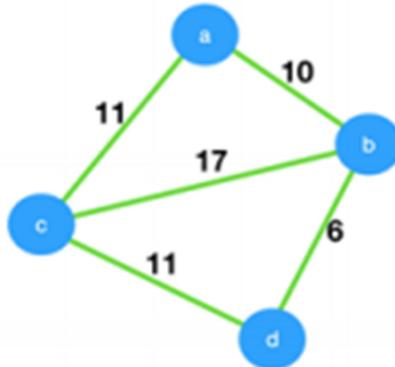
Answer **D**

34. Given items as {value, weight} pairs {{40, 20}, {30, 10}, {20, 5}}. The capacity of knapsack=20. Find the maximum value output assuming items to be divisible

- a) 60**
- b) 80
- c) 100
- d) 40

Answer **A**

35. Consider the given graph.



What is the weight of the minimum spanning tree using the Prim's algorithm, starting from vertex a?

- a) 23
- b) 28
- c) 27**
- d) 11

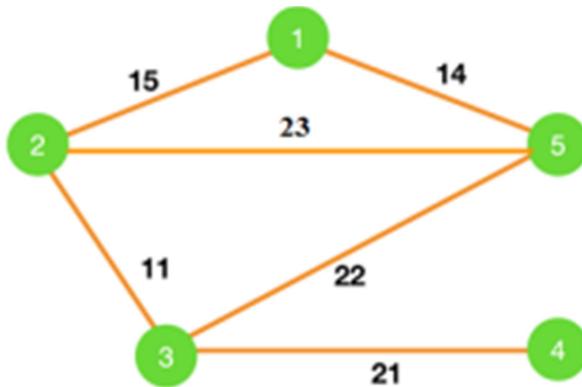
Answer **C**

36. Worst case is the worst case time complexity of Prim's algorithm if adjacency matrix is used?

- a) $O(\log V)$
- b) $O(V^2)$**
- c) $O(E^2)$
- d) $O(V \log E)$

Answer **B**

37. Consider the graph shown below.

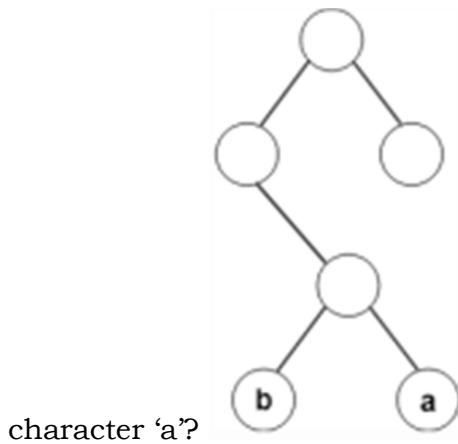


Which of the following edges form the MST of the given graph using Prim'a algorithm, starting from vertex 4.

- a) (4-3)(5-3)(2-3)(1-2)
- b) (4-3)(3-5)(5-1)(1-2)
- c) (4-3)(3-5)(5-2)(1-5)
- d) (4-3)(3-2)(2-1)(1-5)**

Answer **D**

38. From the following given tree, what is the code word for the



- a) **011**
- b) 010
- c) 100
- d) 101

Answer A

39. What will be the cost of the code if character c_i is at depth d_i and occurs at frequency f_i ?

- a) $c_i f_i$
- b) $\sum c_i f_i$
- c) $\sum f_i d_i$**
- d) $f_i d_i$

Answer C

40. What is the running time of the Huffman encoding algorithm?

- a) $O(C)$
- b) $O(\log C)$
- c) $O(C \log C)$**
- d) $O(N \log C)$

Answer C

41. The weighted array used in TVS problems for the following binary tree is

- a) [1,2,3,0,0,4,0,5,6]
- b) [1,2,3,0,0,4,0,5,0,0,0,0,6]**
- c) [1,2,3,4,5,6]
- d) [1,2,3,0,0,4,5,6]

Answer **B**

42. What is the time complexity of the brute force algorithm used to find the longest common subsequence?

- a) $O(n)$
- b) $O(n^2)$
- c) $O(n^3)$
- d) $O(2^n)$**

Answer **D**

43. Find the longest increasing subsequence for the given sequence:

{10, -10, 12, 9, 10, 15, 13, 14}

- a) {10, 12, 15}
- b) {10, 12, 13, 14}
- c) {-10, 12, 13, 14}
- d) {-10, 9, 10, 13, 14}**

Answer **D**

44. What is the space complexity of the following dynamic programming implementation used to find the length of the longest increasing subsequence?

```
#include<stdio.h>
int longest_inc_sub(int *arr, int len)
{
    int i, j, tmp_max;
    int LIS[len]; // array to store the lengths of the longest increasing
    subsequence LIS[0]=1;
    for(i = 1; i < len; i++)
```

```

{
    tmp_max = 0;
    for(j = 0; j < i; j++)
    {
        if(arr[j] < arr[i])
        {
            if(LIS[j] > tmp_max)
                tmp_max = LIS[j];
        }
    }
    LIS[i] = tmp_max + 1;
}
int max = LIS[0];
for(i = 0; i < len; i++)
{
    if(LIS[i] > max)
        max = LIS[i];
}
return max;
}
int main()
{
    int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;
    int ans = longest_inc_sub(arr, len);
    printf("%d",ans);
    return 0;
}
a) O(1)
b) O(n)
c) O(n2)
d) O(nlogn)

```

Answer **B**

45. The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following

graph is

- a) $O(1)$
- b) $O(n)$**
- c) $O(n^2)$
- d) $O(n \log n)$

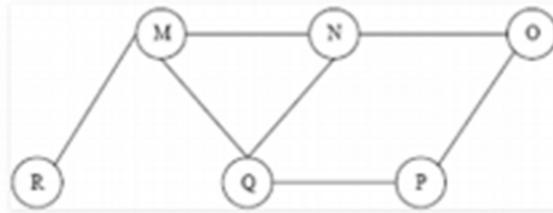
Answer **B**

46. Uniform-cost search expands the node n with the _____

- a) Lowest path cost**
- b) Heuristic cost
- c) Highest path cost
- d) Average path cost

Answer **A**

47. The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is

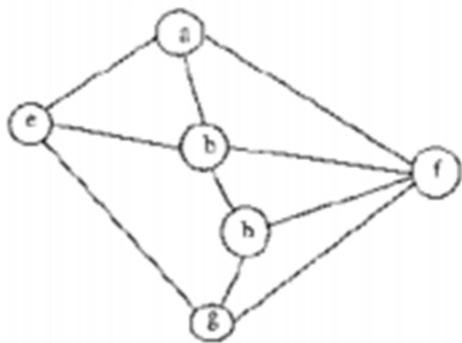


- a) MNOPQR
- b) NQMPOR
- c) QMNPRO**
- d) QMNPOR

Answer **C**

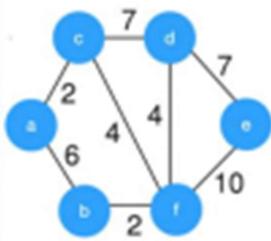
48. Consider the following graph,

Which are depth first traversals of the above graph?



- a) I,II and IV
- b) I and IV only
- c) II,III and IV only
- d) I,III and IV only

49. Consider the given graph.



What is the weight of the minimum spanning tree using the Kruskal's algorithm?

- a) 24
- b) 23
- c) 15
- d) 19**

Answer D

50. Which of the following is false about the Kruskal's algorithm?

- a) It is a greedy algorithm
- b) It constructs MST by selecting edges in increasing order of their weights
- c) It can accept cycles in the MST**

d) It uses union-find data structure

Answer **C**

PART B

- 1 Discuss the components of Greedy Algorithm.
- 2 Compare Greedy technique with dynamic programming and divide and compare.
3. Draw the Characteristics of a good software design
- 4 What is brute force algorithm? List the strength and weakness of brute force algorithm.
- 5 Give the general plan for divide-and-conquer algorithms.
- 6 What is the general divide-and-conquer recurrence relation?
- 7 List out Disadvantages of Divide and Conquer Algorithm
- 8 Define dynamic programming and its features
- 9 Write the difference between the Greedy method and Dynamic programming.
- 10 What are the steps required to develop a greedy algorithm?
- 11 What are the labels in Prim's algorithm used for?
- 12 What is minimum spanning tree.
- 13 How are the vertices not in the tree split into?
- 14 What are the operations to be done after identifying a vertex u^* to be added to the tree?
- 15 Explain Kruskal's algorithm of greedy method?
- 16 Explain the sum of subsets and with a suitable example?
- 17 Write backtracking knapsack Algorithm.
- 18 Compare brute force and Huffman method of greedy.
- 19 Write about Longest Common Subsequence using dynamic programming.
- 20 Explain about OBST with an example.

PART C

1. Explain in detail about greedy knapsack problem. Find an optimal solution to the knapsack instance
 $n=7, m=15, (P_1, P_2, P_3, P_4, P_5, P_6, P_7) = (10, 5, 15, 7, 6, 18, 3)$ and
 $(W_1, W_2, W_3, W_4, W_5, W_6, W_7) = (2, 3, 5, 7, 1, 4, 1)$
2. Write dynamic programming solution for the travelling salesperson problem for the network with the cost adjacency matrix

$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

3. Explain in detail about Huffman code algorithm. Let $A=\{a/5, d/5, c/12, d/13, e/16, f/45\}$ be the letters and its frequency distribution in a text file. Compute a suitable Huffman coding to compress the data effectively and also compute optimal cost.
4. Write an algorithm to determine the sum of subsets for a given sum and a set of numbers. Draw the tree representation to solve the subset sum problem given the number set as $\{5, 10, 15, 20, 25\}$ with the sum=30. Draw all the subsets.
5. Consider the travelling salesman instance defined by the cost matrix

$$\begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 18 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

Find the optimal cost using branch and bound technique

6. Explain Divide And Conquer Method
7. Explain in detail about knapsack problem.
8. Explain Kruskal's Algorithm and Prim's Algorithm
9. Explain Memory Function algorithm for the Knapsack problem
10. Explain in detail about Huffman tree.

UNIT IV

Introduction to backtracking - branch and bound, N queen's problem – backtracking, Sum of subsets using backtracking, Complexity calculation of sum of subsets, Graph introduction, Hamiltonian circuit – backtracking, Branch and bound - Knapsack problem, Example and complexity calculation. Differentiate with dynamic and greedy, Travelling salesman problem using branch and bound, Travelling salesman problem using branch and bound example, Travelling salesman problem using branch and bound example, Time complexity calculation with an example, Graph algorithms, Depth first search and Breadth first search, Shortest path introduction, Floyd-Warshall Introduction, Floyd-Warshall with sample graph, Floyd-Warshall complexity

PART A

1. Which of the following is not a backtracking algorithm?

- (A) Knight tour problem
- (B) N queen problem
- (C) Tower of hanoi
- (D) M coloring problem

Answer: - C

2. Backtracking algorithm is implemented by constructing a tree of choices called as?

- A) State-space tree
- B) State-chart tree
- C) Node tree
- D) Backtracking tree

Answer: - A

3. What happens when the backtracking algorithm reaches a complete solution?

- A) It backtracks to the root
- B) It continues searching for other possible solutions
- C) It traverses from a different route

D) Recursively traverses through the same route

Answer: - B

4. In what manner is a state-space tree for a backtracking algorithm constructed?

- A) Depth-first search
- B) Breadth-first search
- C) Twice around the tree
- D) Nearest neighbour first

Answer: - A

5. In general, backtracking can be used to solve?

- A) Numerical problems
- B) Exhaustive search
- C) Combinatorial problems
- D) Graph coloring problems

Answer: - C

6. Which one of the following is an application of the backtracking algorithm?

- A) Finding the shortest path
- B) Finding the efficient quantity to shop
- C) Ludo
- D) Crossword

Answer: - D

7. Who coined the term ‘backtracking’?

- A) Lehmer
- B) Donald
- C) Ross
- D) Ford

Answer: - A

8. The problem of finding a subset of positive integers whose sum is equal to a given positive integer is called as?

- A) n- queen problem
- B) Subset sum problem
- C) Knapsack problem

D) Hamiltonian circuit problem

Answer: - B

9. The problem of placing n queens in a chessboard such that no two queens attack each other is called as?

- A) n-queen problem
- B) eight queens puzzle
- C) four queens puzzle
- D) 1-queen problem

Answer: - A

10. In how many directions do queens attack each other?

- A) 1
- B) 2
- C) 3
- D) 4

Answer: - C

11. Placing n-queens so that no two queens attack each other is called?

- A) n-queen's problem
- B) 8-queen's problem
- C) Hamiltonian circuit problem
- D) subset sum problem

Answer: - A

12. Where is the n-queens problem implemented?

- A) carom
- B) chess
- C) ludo
- D) cards

Answer: - B

13. Not more than 2 queens can occur in an n-queens problem.

- A) true
- B) false

Answer: - B

14. In n-queen problem, how many values of n does not provide an

optimal solution?

- A) 1
- B) 2
- C) 3
- D) 4

Answer: - B

15. Which of the following methods can be used to solve n-queen's problem?

- A) greedy algorithm
- B) divide and conquer
- C) iterative improvement
- D) backtracking

Answer: - D

16. Of the following given options, which one of the following is a correct option that provides an optimal solution for 4-queens problem?

- A) (3,1,4,2)
- B) (2,3,1,4)
- C) (4,3,2,1)
- D) (4,2,3,1).

Answer: - A

17. How many possible solutions exist for an 8-queen problem?

- A) 100
- B) 98
- C) 92
- D) 88

Answer: - C

18. How many possible solutions occur for a 10-queen problem?

- A) 850
- B) 742
- C) 842
- D) 724.

Answer: - D

19. The Knapsack problem is an example of _____

- A) Greedy algorithm
- B) 2D dynamic programming
- C) 1D dynamic programming
- D) Divide and conquer

Answer: - B

20. Which of the following methods can be used to solve the Knapsack problem?

- A) Brute force algorithm
- B) Recursion
- C) Dynamic programming
- D) Brute force, Recursion and Dynamic Programming

Answer: - D

21. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?

- A) 160
- B) 200
- C) 170
- D) 90

Answer: - A

22. Which of the following problems is equivalent to the 0-1 Knapsack problem? A) You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of $\{w_1, w_2, w_3, \dots, w_n\}$ and a value of $\{v_1, v_2, v_3, \dots, v_n\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value
B) You are studying for an exam and you have to study N questions. The questions take $\{t_1, t_2, t_3, \dots, t_n\}$ time(in hours) and carry $\{m_1, m_2, m_3, \dots, m_n\}$ marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized
C) You are given infinite coins of denominations $\{v_1, v_2, v_3, \dots, v_n\}$ and a

sum S. You have to find the minimum number of coins required to get the sum S

D) You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of {10, 20, 15, 40} and a value of {1, 2, 3, 4}. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

Answer: - B

23. What is the time complexity of the brute force algorithm used to solve the Knapsack problem?

- A) $O(n)$
- B) $O(n!)$
- C) $O(2^n)$
- D) $O(n^3)$

Answer: - C

24. Which of the following is/are property/properties of a dynamic programming problem?

- A) Optimal substructure
- B) Overlapping subproblems
- C) Greedy approach
- D) Both optimal substructure and overlapping subproblems

Answer: - D

25. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses _____ property.

- A) Overlapping subproblems
- B) Optimal substructure
- C) Memoization
- D) Greedy

Answer: - B

26. If a problem can be broken into subproblems which are reused several times, the problem possesses _____ property.

- A) Overlapping subproblems
- B) Optimal substructure

C) Memoization

D) Greedy

Answer: - A

27. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called _____

A) Dynamic programming

B) Greedy

C) Divide and conquer

D) Recursion

Answer: - C

28. In dynamic programming, the technique of storing the previously calculated values is called _____

A) Saving value property

B) Storing value property

C) Memoization

D) Mapping

Answer: - C

29. When a top-down approach of dynamic programming is applied to a problem, it usually _____

A) Decreases both, the time complexity and the space complexity

B) Decreases the time complexity and increases the space complexity

C) Increases the time complexity and decreases the space complexity

D) Increases both, the time complexity and the space complexity

Answer: - B

30. Which of the following problems is NOT solved using dynamic programming?

A) 0/1 knapsack problem

B) Matrix chain multiplication problem

C) Edit distance problem

D) Fractional knapsack problem

Answer: - D

31. Which of the following problems should be solved using dynamic programming?

- A) Mergesort
- B) Binary search
- C) Longest common subsequence
- D) Quicksort

Answer: - C

32. Time Complexity of Breadth First Search is? (V - number of vertices, E - number of edges)

- A) $O(V+E)$
- B) $O(V)$
- C) $O(E)$
- D) $O(VE)$

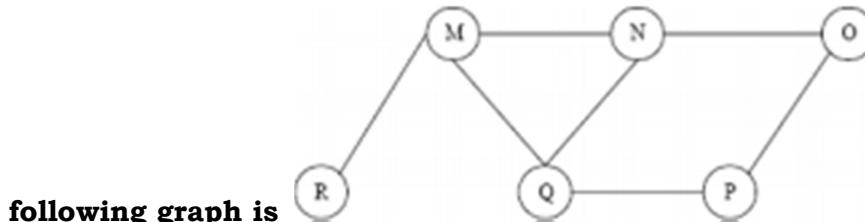
Answer: - A

33. The spanning tree of connected graph with 10 vertices contains

- A) 9 edges
- B) 11 edges
- C) 10 edges
- D) 8 edges

Answer: - A

34. The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the



following graph is

- A) MNOPQR
- B) NQMPOR
- C) QMNPRO
- D) QMNPOR

Answer: - C

35. What is the maximum height of queue (To keep track of un-explored nodes) required to process a connected Graph G1 which contains 'N' node using BFS algorithm?

- A) $(N/2)-1$
- B) $(N/2)/2$
- C) $N-1$
- D) N

Answer: - C

PART B

1. What is meant by knapsack problem?
2. Define fractional knapsack problem.
3. Write the running time of 0/1 knapsack problem.
4. Write recurrence relation for 0/1 knapsack problem
5. What is meant by travelling salesperson problem?
6. What is the running time of dynamic programming TSP?
7. State if backtracking always produces optimal solution.
8. Define backtracking.
9. What are the two types of constraints used in backtracking?
10. What is meant by optimization problem?
11. Define Hamiltonian circuit problem.
12. What is Hamiltonian cycle in an undirected graph?
13. Define 8queens problem. 8. List out the application of backtracking.
14. Define promising node and non-promising node.
15. Give the explicit and implicit constraint for 8-queen problem.
16. How can we represent the solution for 8-queen problem?
17. Give the categories of the problem in backtracking.
18. Differentiate backtracking and over exhaustive search.
19. Find optimal solution for the knapsack instance $n = 3, w = [20, 15, 15], P = [40, 25, 25]$ and $C = 30$
20. What is travelling salesperson problem?
21. What is the formula used to find upper bound for knapsack problem?
22. Differentiate between back tracking and branch and bound.
23. List out the application of branch and bound technique.
24. Analyze the time complexity for Warshall's and Floyd's algorithm.
25. Test the 0/1 knapsack problem.

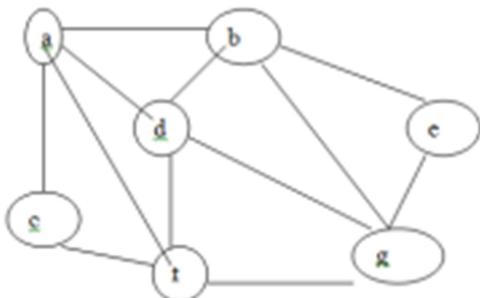
26. Summarize Warshall's algorithm
27. Compare feasible and optimal solution.
28. Differentiate between DFS and BFS.
29. What is efficiency of DFS based algorithm for topological sorting
30. What are the different applications of DFS and BFS?

PART C

1. Describe the travelling salesman problem and discuss how to solve it using dynamic programming?
2. Find the optimal solution for the given knapsack problem

Item	1	2	3	4
weight	2	1	3	2
Value	\$12	\$10	\$20	\$15

3. Apply backtracking technique to solve the following instance of the subset sum problem $S = \{1, 3, 4, 5\}$ and $d=11, 16$
4. Explain subset-sum problem and discuss the possible solution strategies using backtracking.
5. Explain N-queens problem with an algorithm.
6. Explain why backtracking is defined as a default procedure of last resort for solving problems.
7. Explain the subset-sum problem in detail by justifying it using backtracking algorithm.
8. Apply backtracking to the problem of finding a Hamiltonian circuit for the following graph.



9. What is backtracking? Explain in detail.
10. Solve the following instance of the knapsack problem by the branch and bound algorithm.

Item	Weight	Value
1	4	\$40
2	7	#42
3	5	\$25
4	3	\$12
The Knapsack's capacity W=10		

11. Discuss the solution for knapsack problem using branch and bound technique.
12. What is branch and bound technique? Explain how knapsack problem could be solved using branch and bound technique. Solve the following instance of the knapsack problem by branch and bound algorithm for W=16

Item	Weight	Value in Rs.
1	10	100
2	7	63
3	8	56
4	4	12

13. What is branch and bound? Explain in detail.
14. Consider the below matrix for assignment problem involving persons and jobs. Explain in detail how branch and bound technique is useful in solving assignment problems.

	Job1	Job2	Job3	Job4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

15. Discuss Warshall's algorithm with suitable example.

UNIT V

Introduction to randomization and approximation algorithm - Randomized hiring problem - Randomized quick sort, Complexity analysis - String matching algorithm, Examples - Rabin Karp algorithm for string matching, Example discussion - Approximation algorithm, Vertex covering - Introduction Complexity classes, P type problems - Introduction to NP type problems, Hamiltonian cycle problem - NP complete problem introduction, Satisfiability problem - NP hard problems - Examples

PART A

1. What is a Rabin and Karp Algorithm?

- (A) String Matching Algorithm
- (B) Shortest Path Algorithm
- (C) Minimum spanning tree Algorithm
- (D) Approximation Algorithm

Answer: - A

2. What is the pre-processing time of Rabin and Karp Algorithm?

- A) Theta(m^2)
- B) Theta($m\log n$)
- C) Theta(m)
- D) Big-Oh(n)

Answer: - C

3. Rabin Karp Algorithm makes use of elementary number theoretic notions.

- A) True
- B) FALSE

Answer: - A

4. Given a pattern of length- 5 window, find the spurious hit in the given text string.

Pattern: 3 1 4 1 5

Modulus: 13

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Text: 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1 3 9

- A) 6-10

- B) 12-16
- C) 3-7
- D) 13-17

Answer: - D

5. What is the basic principle in Rabin Karp algorithm?

- A) Hashing
- B) Sorting
- C) Augmenting
- D) Dynamic Programming

Answer: - A

6. The worst-case efficiency of solving a problem in polynomial time is?

- A) $O(p(n))$
- B) $O(p(n\log n))$
- C) $O(p(n^2))$
- D) $O(p(m \log n))$

Answer: - A

7. Problems that can be solved in polynomial time are known as?

- A) Intractable
- B) Tractable
- C) Decision
- D) Complete

Answer: - B

8. _____ is the class of decision problems that can be solved by non-deterministic polynomial algorithms.

- A) NP
- B) P
- C) Hard
- D) Complete

Answer: - A

9. The Euler's circuit problem can be solved in?

- A) $O(N)$
- B) $O(N \log N)$
- C) $O(\log N)$
- D) $O(N^2)$

Answer: - D

10. To which of the following class does a CNF-satisfiability problem belong?

- A) NP class
- B) P class
- C) NP complete
- D) NP hard

Answer: - C

11. Quick sort uses which of the following algorithm to implement sorting?

- A) backtracking
- B) greedy algorithm
- C) divide and conquer
- D) dynamic programming

Answer: - C

12. What is the worst case time complexity of randomized quicksort?

- A) $O(n)$
- B) $O(n \log n)$
- C) $O(n^2)$
- D) $O(n^2 \log n)$

Answer: - C

13. What is the purpose of using randomized quick sort over standard quick sort?

- A) so as to avoid worst case time complexity
- B) so as to avoid worst case space complexity
- C) to improve accuracy of output
- D) to improve average case time complexity

Answer: - A

14. Which of the following is incorrect about randomized quicksort?

- A) it has the same time complexity as standard quick sort
- B) it has the same space complexity as standard quick sort
- C) it is an in-place sorting algorithm
- D) it cannot have a time complexity of $O(n^2)$ in any case.

Answer: - D

15. Which of the following is the fastest algorithm in string matching field?

- A) Boyer-Moore's algorithm
- B) String matching algorithm
- C) Quick search algorithm
- D) Linear search algorithm

Answer: - C

16. What is vertex coloring of a graph?

- A) A condition where any two vertices having a common edge should not have same color
- B) A condition where any two vertices having a common edge should always have same color

- C) A condition where all vertices should have a different color
- D) A condition where all vertices should have same color

Answer: - A

17. How many edges will a tree consisting of N nodes have?

- A) $\log(N)$
- B) N
- C) $N-1$
- D) $N+1$

Answer: - C

18. Minimum number of unique colors required for vertex coloring of a graph is called?

- A) vertex matching
- B) chromatic index
- C) chromatic number
- D) color number.

Answer: - C

19. How many unique colors will be required for proper vertex coloring of an empty graph having n vertices?

- A) 0
- B) 1
- C) n
- D) $n!$

Answer: - C

20. What will be the chromatic number of the following graph?



- A) 1
- B) 2
- C) 3
- D) 4

Answer: - B

21. Assuming P != NP, which of the following is true ?

- A) NP-complete = NP
- B) NP-complete \cap P = \Phi
- C) NP-hard = NP
- D) P = NP-complete

Answer: - B

22. Let X be a problem that belongs to the class NP. Then which one of the

following is TRUE?

- A) There is no polynomial time algorithm for X.
- B) If X can be solved deterministically in polynomial time, then P = NP.
- C) If X is NP-hard, then it is NP-complete.
- D) X may be undecidable.

NP Complete

Answer: - C

23. Which of the following statements are TRUE?

- 1. The problem of determining whether there exists a cycle in an undirected graph is in P.
 - 2. The problem of determining whether there exists a cycle in an undirected graph is in NP.
 - 3. If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.
- A) 1, 2 and 3
 - B) 1 and 2 only
 - C) 2 and 3 only
 - D) 1 and 3 only

Answer: - A

24. Consider the following two problems on undirected graphs

a : Given G(V, E), does G have an independent set of size | V | - 4?

β : Given G(V, E), does G have an independent set of size 5?

Which one of the following is TRUE?

- A) a is in P and β is NP-complete
- B) a is NP-complete and β is in P
- C) Both a and β are NP-complete
- D) Both a and β are in P

Answer: - C

25. Which of the following algorithm can be used to solve the Hamiltonian path problem

efficiently?

- A) branch and bound
- B) iterative improvement
- C) divide and conquer
- D) greedy algorithm

Answer: - A

26. Hamiltonian path problem is _____

- A) NP problem

- B) N class problem
- C) P class problem
- D) NP complete problem

Answer: - D

27. There is no existing relationship between a Hamiltonian path problem and Hamiltonian circuit problem.

- A) true
- B) false

Answer: - B

28. Which of the following problems is similar to that of a Hamiltonian path problem?

- A) knapsack problem
- B) closest pair problem
- C) travelling salesman problem
- D) assignment problem

Answer: - C

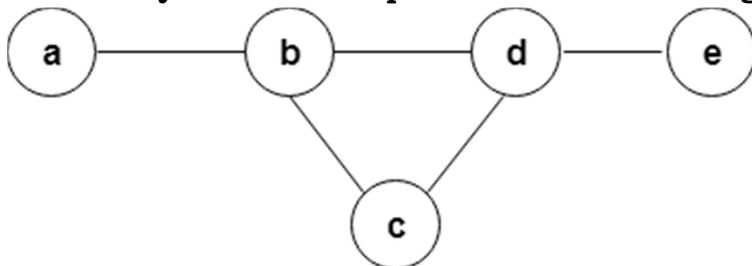
29. In what time can the Hamiltonian path problem can be solved using dynamic

programming?

- A) $O(N)$
- B) $O(N \log N)$
- C) $O(N^2)$
- D) $O(N^2 2^N)$

Answer: - D

30. How many Hamiltonian paths does the following graph have?



- A) 1
- B) 2
- C) 3
- D) 4

Answer: - A

31. A node is said to be _____ if it has a possibility of reaching a complete solution.

- A) Non-promising
- B) Promising
- C) Succeeding
- D) Preceding

Answer: - B

32. Minimum number of unique colors required for vertex coloring of a graph is called?

- A) vertex matching
- B) chromatic index
- C) chromatic number
- D) color number

Answer: C

PART B

1. Define NP hard and NP completeness.
2. Compare NP hard and NP completeness.
3. Write Short notes on “the class P and NP problem”.
4. How NP Hard problems are different from NP Complete?
5. Whether class P solves a problem in polynomial time? Justify.
6. An NP hard problem can be solved in deterministic polynomial time, how?
7. Give examples for NP Complete problems
8. State the property of NP complete problem.
9. Define adversary method.
10. Define lower bound.
11. What type of output yields trivial lower bound?
12. What is information theoretic lower bound?
13. Define complexity theory.
14. What is halting problem?
15. What is CNFs satisfiability problem?
16. Define Matching.
17. Define a bipartite graph.
18. How will you check the stability?
19. What is stable marriage problem?
20. Define the term stable pair
21. What do you mean by perfect match in bipartite graph?
22. Write Rabin Karp string matching algorithm
23. Describe Hamiltonian cycle problem

PART C

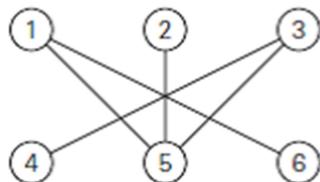
1. Describe in detail about P and NP Problems
2. Write short notes on NP Complete Problem
3. Write short notes on the following using approximation Algorithm

- i) Nearest -neighbor algorithm with example
 - ii) Multi fragment heuristic algorithm with example
4. Describe in detail about Twice around the tree algorithm with example
5. Explain local search heuristic with example
6. Explain Approximation Algorithms for the Travelling Salesman Problem
7. Explain the Assignment problem in Branch and bound with Example.
8. Suggest an approximation algorithm for TSP. Assume that the cost function satisfies the triangle inequality.
9. Using an example prove that, satisfiability of Boolean formula in 3-Conjunctive Normal Form is NP – complete.
10. State the relationships among the complexity class algorithms with the help of neat diagrams
11. Explain the algorithm for stable marriage problem and prove the theorem with Example
12. Consider an instance of the stable marriage problem given by the ranking matrix

	A	B	C
a	1,3	2,2	3,1
b	3,1	1,3	2,2
c	2,2	3,1	1,3

For each of its marriage matching's, indicate whether it is stable or not

13. Apply the maximum matching algorithm to the following bipartite graphs



14. Write the algorithm for maximum matching in Bipartite Graphs and prove the theorem With example

15. Explain the algorithm:

- i. Blocking pair
- ii. Stable marriage problem
- iii. Man optimal
- iv. Women optimal

16. Explain briefly on minimum weight perfect matching algorithm.

17. Explain briefly on reducing bipartite graph to net flow

18. Explain local search heuristic with example

19. Consider the following minimization problem:

DEGREE BOUNDED SPANNING TREE:

Instance: Graph G = (V,E)

Solution:: A spanning tree T of G

Value: Maximum degree of T

Goal: Find a solution with minimum value.

20. Consider the following scheduling problem. You are given n jobs where job i is specified by an earliest start time s_i and a processing time p_i . In homework 1, we considered a preemptive version of this problem and gave a greedy algorithm to give an optimal preemptive schedule. In this problem we consider the non-preemptive version of this scheduling problem. Here a job CANNOT be suspended but rather must be performed in a contiguous time interval. Consider the following heuristic for the non-preemptive problem: schedule the jobs in the order in which they complete in an optimal preemptive schedule starting each job as soon as the one before it completes. You are to prove that this algorithm is a 2-approximation algorithm.

21. Using Rabin karp string matching algorithm match the given pattern P with given string S.

P = 745

S = 745727457

22. Using KMP string matching algorithm, find the occurrence of the given pattern P in the given text T.

T § ABABACAB

P § ABAB

UNIT IV

INTRODUCTION TO BACKTRACKING- BRANCH AND BOUND

Multiple Choice Questions with Answers:

Multiple Choice Questions with Answers:

1. Which of the problems cannot be solved by backtracking method? **CLO-4 Pg No- 486**

 - a) n-queen problem
 - b) Subset sum problem
 - c) Hamiltonian circuit problem
 - d) Travelling salesman problem**

2. What happens when the backtracking algorithm reaches a complete solution? **CLO-4 Pg No- 518**

 - a) It backtracks to the root
 - b) It continues searching for other possible solutions**
 - c) It traverses from a different route
 - d) Recursively traverses through the same route

3. In general, backtracking can be used to solve? **CLO-4 Pg No-517**

 - a) Numerical problems
 - b) Exhaustive search
 - c) Combinatorial problems**
 - d) Graph coloring problems

4. How many solutions are there for 8 queens on 8*8 board? **CLO-4 Pg No-522**

 - a) 12
 - b) 91
 - c) 92**
 - d) 93

5. In how many directions do queens attack each other? **CLO-4 Pg No- 523**

 - a) 1
 - b) 2
 - c) 3**
 - d) 4

6. Which of the following methods can be used to solve n-queen's problem? **CLO-4 Pg No-523**

 - a) Greedy algorithm
 - b) Divide and conquer
 - c) Iterative improvement
 - d) Backtracking**

7. What is the domination number for 8-queen's problem? **CLO-4 Pg No-523**

 - a) 8
 - b) 7
 - c) 6
 - d) 5**

8. Which of the following is true about the time complexity of the recursive solution of the subset sum problem? **CLO-4 Pg No-525**

 - a) It has an exponential time complexity**
 - b) It has a linear time complexity
 - c) It has a logarithmic time complexity
 - d) it has a time complexity of $O(n^2)$

9. What is the worst case time complexity of dynamic programming solution of the subset sum problem(sum=given subset sum)? **CLO-3 Pg No-525**

- a) $O(n)$
- b) $O(\text{sum})$
- c) $O(n^2)$
- d) $O(\text{sum} * n)$**

10. Which of the following is not true about subset sum problem? **CLO-3 Net Source**

- a) the recursive solution has a time complexity of $O(2n)$
- b) there is no known solution that takes polynomial time
- c) the recursive solution is slower than dynamic programming solution
- d) the dynamic programming solution has a time complexity of $O(n \log n)$**

11. Which of the following algorithm can be used to solve the Hamiltonian path problem efficiently? **CLO-3 Pg No-531**

- | | |
|----------------------------|--------------------------|
| a) Branch and Bound | b) Iterative improvement |
| c) Greedy algorithm | d) Divide and Conquer |

12. The problem of finding a path in a graph that visits every vertex exactly once is called? **CLO-3 Pg No-466**

- | | |
|------------------------------------|------------------------------|
| a) Hamiltonian path problem | b) Hamiltonian cycle problem |
| c) Subset sum problem | d) NP problem |

13. In what time can the Hamiltonian path problem be solved using dynamic programming? **CLO-3 Pg No-531**

- | | |
|-------------|-----------------------------------|
| a) $O(N)$ | b) $O(n \log n)$ |
| c) $O(N^2)$ | d) $O(N^2 2^N)$ |

14. Which of the following problems is similar to that of a Hamiltonian path problem? **CLO-3 Net Source**

- | | |
|---------------------------------------|-------------------------|
| a) knapsack problem | b) closest pair problem |
| c) travelling salesman problem | d) assignment problem |

15. What is the time complexity for finding a Hamiltonian path for a graph having N vertices (using permutation)? **CLO-3 Pg No- 531**

- | | |
|----------------|----------------------------------|
| a) $O(N!)$ | b) $O(N! * N)$ |
| c) $O(\log N)$ | d) $O(N)$ |

16. What is the time complexity of the brute force algorithm used to solve the Knapsack problem? **CLO-3 Pg No- 496**

- | | |
|-------------------------------|-------------|
| a) $O(n)$ | b) $O(n!)$ |
| c) $O(2^n)$ | d) $O(n^3)$ |

17. The Knapsack problem is an example of _____ **CLO-3 Pg No-497**

- | | |
|---------------------------|----------------------------------|
| a) Greedy algorithm | b) 2D dynamic programming |
| c) 1D dynamic programming | d) Divide and conquer |

18. Which of the following methods can be used to solve the Knapsack problem? **CLO-3 Pg No- 496**

- a) Brute force algorithm

- b) Recursion
 - c) Dynamic programming
 - d) Brute force, Recursion and Dynamic Programming**

19. Travelling salesman problem is an example of **CLO-4 Pg No- 486**
a.Dynamic Algorithm
c.Recursive Approach
b.Greedy Algorithm
d.Divide & Conquer

20. A graph in which all nodes are of equal degree is called **CLO-3 Net Source**
a) Multi graph
c) **Regular graph**
b) Non regular graph
d) Complete graph

21. Which of the following algorithms can be used to most efficiently determine the presence of a cycle in a given graph ? **CLO-3 Pg No-468**
a) Depth First Search
b) Breadth First Search
c) Prim's Minimum Spanning Tree Algorithm
d) Kruskal' Minimum Spanning Tree Algorithm

22. Traversal of a graph is different from tree because **CLO-4 Pg No-469**
a) There can be a loop in graph so we must maintain a visited flag for every vertex
b) DFS of a graph uses stack, but in-order traversal of a tree is recursive
c) BFS of a graph uses queue, but a time efficient BFS of a tree is recursive
d) All of the above

23. Given two vertices in a graph s and t, which of the two traversals (BFS and DFS) can be used to find if there is path from **s to t?** **CLO-4 Net Source**
a) Only BFS
c) Both BFS and DFS
b) Only DFS
d) Neither BFS nor DFS

24. A complete graph can have **CLO-3 Pg No-466**
a. n^2 spanning trees
c. n^{n+1} spanning trees
b. $n^{(n-2)}$ spanning trees
d. n^n spanning trees

25. Graphs are represented using **CLO-3 Net source**
a.Adjacency tree
c.Adjacency graph
b.Adjacency linked list
d.Adjacency queue

26. The spanning tree of connected graph with 10 vertices contains **CLO-3 Net source**
a.9 edges
c.10 edges
d. 9 vertices

27. Which of the following algorithms solves the all-pair shortest path problem? **CLO-4 Pg No-478**
a.Floyd's algorithm
c.Dijkstra's algorithm
b.Prim's algorithm
d.Warshall's algorithm

28. The minimum number of colors needed to color a graph having n (>3) vertices and 2 edges is **CLO-3 Net Source**

- a.1
- b.2**
- c.3
- d.4

29. Which of the following is useful in traversing a given graph by breadth first search?

CLO-3 Net Source

- a.set
- b.List**
- c.stacks
- d.Queue**

30. The minimum number of edges in a connected cyclic graph on n vertices is **CLO-3 Pg No- 466**

- a.n**
- b. $n+1$
- c. $n-1$
- d.none of the above

31. Floyd Warshall's Algorithm can be applied on _____ **CLO-3 Pg No-478**

- a) Undirected and unweighted graphs
- b) Undirected graphs
- c) Directed graphs**
- d) Acyclic graphs

32. What is the running time of the Floyd Warshall Algorithm? **CLO-3 Pg No-479**

- a) Big-oh(V)
- b) Theta(V^2)
- c) Big-Oh(VE)**
- d) Theta(V^3)

34. What procedure is being followed in Floyd Warshall Algorithm? **CLO-3 Pg No-479**

- a) Top down
- b) Bottom up**
- c) Big bang
- d) Sandwich

35. Floyd Warshall Algorithm can be used for finding _____ **CLO-3 Pg No-480**

- a) Single source shortest path
- b) Topological sort
- c) Minimum spanning tree
- d) Transitive closure**

36. What approach is being followed in Floyd Warshall Algorithm? **CLO-3 Pg No-480**

- a) Greedy technique
- b) Dynamic Programming**
- c) Linear Programming
- d) Backtracking

37. What happens when the value of k is 0 in the Floyd Warshall Algorithm? **Pg No-481**

- a) 1 intermediate vertex
- b) 0 intermediate vertex**
- c) N intermediate vertices
- d) N-1 intermediate vertices

38. The time required to find shortest path in a graph with n vertices and e edges is

CLO-3 Net Source

- a. $O(e)$
- b. $O(n)$**
- c. $O(n^2)$**
- d. $O(e^2)$

39. For 0/1 KNAPSACK problem, the algorithm takes _____ amount of time for memory table, and _____ time to determine the optimal load, for N objects and W as the capacity of KNAPSACK **CLO-4 Pg No-497**

- a. O(NW), O(N+W)
- b. O(N), O(NW)
- c. O(N+W), O(NW)
- d. O(NW), O(N)

40. Given a directed graph where weight of every edge is same, we can efficiently find shortest path from a given source to destination using? **CLO-3 Pg No-478**

(A) Breadth First Traversal

(B) Dijkstra's Shortest Path Algorithm

(C) Neither Breadth First Traversal nor Dijkstra's algorithm can be used

(D) Depth First Search

41. What can be the applications of Breadth First Search? **CLO-3 Net Source**

A. Finding shortest path between two nodes

B. Finding bipartiteness of a graph

C. GPS navigation system

D. All of the mentioned

42. What can be the applications of Depth First Search? **CLO-3 Net Source**

A. For generating topological sort of a graph

B. For generating Strongly Connected Components of a directed graph

C. Detecting cycles in the graph

D. All of the mentioned

43. When the Depth First Search of a graph is unique? **CLO-3 Net Source**

A. When the graph is a Binary Tree

B. When the graph is a Linked List

C. When the graph is a n-ary Tree

D. None of the mentioned

44. For a directed graph with edge lengths, the floyd warshall algorithm can compute the between each pair of nodes in $O(n^3)$. **CLO-3 Pg No- 480**

A) Transitive Hull

B) Minimax Distance

C) Max Min Distance

D) Safest Path

45. Dijkstra algorithm is also called the shortest path problem. **CLO-3 Pg No- 470**

A) multiple source

B) single source

C) single destination

D) multiple destination

46. The floyd-warshall all pairs shortest path algorithm computes the shortest paths between each pair of nodes in **CLO-3 Pg No- 479**

A) $O(\log n)$

B) $O(n^2)$

C) $O(mn)$

D) $O(n^3)$

PART B (4 Marks)

1. What is Backtracking?
2. Explain the back tracking Explicit and implicit constraints.
3. Write an algorithm for Backtracking with an example.
4. Which are the problems solved using backtracking method
5. Define sum of subsets
6. Give an example of sum of subset.
7. How to generate state space tree ?
8. Explain Sum of subset Algorithm:
9. Explain Hamiltonian cycles:
10. What are the steps used in Hamiltonian Procedure:
11. Write an Algorithm for Finding all Hamiltonian cycle
12. Write short notes about 4-Queens problem.
13. What are the steps to generate the 8 Queen solution.
14. Write an algorithm for N queens problem.
15. Draw any Two possible solutions for 4 Queen problem.
16. Illustrate graph coloring problem.
17. What are the general Steps to color the Graph in graph coloring problem.
19. Write short notes on graph coloring Algorithm.
20. Define Knapsack Problem using Backtracking method.
21. Write about backtracking knapsack Algorithm
22. Give short notes on Floyds- Warshall.
23. What is the difference between BFS and DFS.
24. Give an example for Floyds – Warshall algorithm.
25. Write short notes on BFS and DFS.
26. What is shortest path algorithm? Give an example.
27. Explain the time complexity calculation of Travelling salesman Problem.
28. Explain the time complexity calculation of Floyds- warshall algorithm.

PART C (12 Marks)

1. Briefly explain about backtracking method with suitable example.
2. Explain about 8 queens problem with an algorithm.
3. Explain Sum of subsets with an example.
4. What is Graph coloring problem and explain its algorithm using backtracking method.
5. How to find all Hamiltonian cycle using backtracking method? Explain with an algorithm.
6. Explain about Travelling Salesman problem using backtracking.
7. Explain about Floyds- Warshall algorithm with suitable examples.
8. Explain about Travelling Salesman problem using branch and bound algorithm.
9. Explain Depth First Search with an example.
10. What is BFS? Give an explanation with example.
11. Explain about Knapsack problem with suitable examples.

Reg. No. _____

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
CYCLE TEST – III – APRIL- 2020**

Fourth Semester – Computer Science and Engineering

18CSC204J- Design and Analysis of Algorithms

Duration: 90 Minutes

Max. Marks: 50

PART – A (10 X 1 = 10 Marks)

Answer ALL Questions

1. Backtracking algorithm is implemented by constructing a tree of choices called as?
 - a) State-space tree
 - b) State-chart tree
 - c) Node tree
 - d) Backtracking tree
2. Which data structure is most suitable for implementing best first branch and bound strategy?
 - a) stack
 - b) queue
 - c) priority queue
 - d) linked list
3. In what time can the Hamiltonian path problem can be solved using dynamic programming?
 - a) $O(n)$
 - b) $O(n \log n)$
 - c) $O(n^2)$
 - d) $O(n^2 2^n)$
4. In what manner is a state-space tree for a backtracking algorithm constructed?
 - a) Depth-first search
 - b) Breadth-first search
 - c) Twice around the tree
 - d) Nearest neighbour first
5. What is the worst-case running time of Rabin Karp Algorithm?
 - a) $\theta(n)$
 - b) $\theta(n - m)$
 - c) $\theta((n - m + 1)m)$
 - d) $\theta(n \log m)$
6. What approach is being followed in Floyd Warshall Algorithm?
 - a) Greedy technique
 - b) Dynamic Programming
 - c) Linear Programming
 - d) Backtracking
7. Which of the following options match the given statement:
Statement: The algorithms that use the random input to reduce the expected running time or memory usage, but always terminate with a correct result in a bounded amount of time.
 - a) Las Vegas Algorithm
 - b) Monte Carlo Algorithm
 - c) Atlantic City Algorithm
 - d) None of the mentioned
8. What is vertex coloring of a graph?
 - a) A condition where any two vertices having a common edge should not have same color
 - b) A condition where any two vertices having a common edge should always have same color
 - c) A condition where all vertices should have a different color
 - d) A condition where all vertices should have same color

9. To which of the following class does a CNF-satisfiability problem belong?
 - a) NP class
 - b) P class
 - c) NP complete
 - d) NP hard
10. The time complexity of the normal quick sort, randomized quick sort algorithms in the worst case is
 - a) $O(n^2), O(n \log n)$
 - b) $O(n^2), O(n^2)$
 - c) $O(n \log n), O(n^2)$
 - d) $O(n \log n), O(n \log n)$

PART – B (4 X 4 = 16 Marks)

Answer ANY FOUR questions

11. Solve the following 0/1 Knapsack problem using Branch and Bound.
(W(capacity of knapsack= 15)

Profit	10	10	12	18
Weight	2	4	6	9

12. What do you mean by Indicator Random Variable? What is its significance in randomized algorithms?
13. What do you understand by string matching algorithms? List any three such algorithms.
14. Discuss the travelling salesman problem and show how it is an instance of the Hamiltonian Circuit problem
15. Differentiate Backtracking and Branch and Bound.

PART – C (2 X 12 = 24 Marks)

Answer ALL questions

16. a) What is backtracking? What kind of search technique is used for backtracking? Show how the technique can be applied to a 4x4 board using state space tree with algorithm.
(OR)
b) For given set w = {5, 10, 12, 13, 15, 18} and target sum=30, find all possible combination of subsets using state space tree with algorithm.
17. a) What do you understand by spurious hits? For string matching, how many spurious hits does the Rabin-Karp matcher encounters in Text T = 31415926535..... and Pattern P = 26. Calculate hash function as (P modulo (length of text string))
(OR)
b) Differentiate between deterministic algorithms and non-deterministic algorithms and accordingly explain in detail NP, Np Hard and NP Complete.

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, RAMAPURAM
CAMPUS**
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
QUESTION BANK
18CSC204J DESIGN AND ANALYSIS OF ALGORITHMS

UNIT 5

Introduction to randomization and approximation algorithm - Randomized hiring problem - Randomized quick sort, Complexity analysis - String matching algorithm, Examples - Rabin Karp algorithm for string matching, Example discussion - Approximation algorithm, Vertex covering - Introduction Complexity classes, P type problems - Introduction to NP type problems, Hamiltonian cycle problem - NP complete problem introduction, Satisfiability problem - NP hard problems - Examples

PART A

1. What is a Rabin and Karp Algorithm?

- (A) String Matching Algorithm
- (B) Shortest Path Algorithm
- (C) Minimum spanning tree Algorithm
- (D) Approximation Algorithm

Answer: - A

2. What is the pre-processing time of Rabin and Karp Algorithm?

- A) Theta(m^2)
- B) Theta($m\log n$)
- C) Theta(m)
- D) Big-Oh(n)

Answer: - C

3. Rabin Karp Algorithm makes use of elementary number theoretic notions.

- A) True
- B) FALSE

Answer: - A

- 4. Given a pattern of length- 5 window, find the spurious hit in the given text string.**

Pattern: 3 1 4 1 5

Modulus: 13

Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Text: 2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1 3 9

- A) 6-10
- B) 12-16
- C) 3-7
- D) 13-17

Answer: - D

- 5. What is the basic principle in Rabin Karp algorithm?**

- A) Hashing
- B) Sorting
- C) Augmenting
- D) Dynamic Programming

Answer: - A

- 6. The worst-case efficiency of solving a problem in polynomial time is?**

- A) $O(p(n))$
- B) $O(p(n\log n))$
- C) $O(p(n^2))$
- D) $O(p(m \log n))$

Answer: - A

- 7. Problems that can be solved in polynomial time are known as?**

- A) Intractable
- B) Tractable
- C) Decision
- D) Complete

Answer: - B

- 8. _____ is the class of decision problems that can be solved by non-deterministic polynomial algorithms.**

- A) NP
- B) P
- C) Hard
- D) Complete

Answer: - A

9. The Euler's circuit problem can be solved in?

- A) O(N)
- B) O(N log N)
- C) O(logN)
- D) O(N^2)

Answer: - D

10. To which of the following class does a CNF-satisfiability problem belong?

- A) NP class
- B) P class
- C) NP complete
- D) NP hard

Answer: - C

11. Quick sort uses which of the following algorithm to implement sorting?

- A) backtracking
- B) greedy algorithm
- C) divide and conquer
- D) dynamic programming

Answer: - C

12. What is the worst case time complexity of randomized quicksort?

- A) O(n)
- B) O(n log n)
- C) O(n^2)
- D) O($n^2 \log n$)

Answer: - C

13. What is the purpose of using randomized quick sort over standard quick sort?

- A) so as to avoid worst case time complexity
- B) so as to avoid worst case space complexity
- C) to improve accuracy of output
- D) to improve average case time complexity

Answer: - A

14. Which of the following is incorrect about randomized quicksort?

- A) it has the same time complexity as standard quick sort
- B) it has the same space complexity as standard quick sort
- C) it is an in-place sorting algorithm
- D) it cannot have a time complexity of O(n^2) in any case.

Answer: - D

15. Which of the following is the fastest algorithm in string matching field?

- A) Boyer-Moore's algorithm
- B) String matching algorithm
- C) Quick search algorithm
- D) Linear search algorithm

Answer: - C

16. What is vertex coloring of a graph?

- A) A condition where any two vertices having a common edge should not have same color
- B) A condition where any two vertices having a common edge should always have same color
- C) A condition where all vertices should have a different color
- D) A condition where all vertices should have same color

Answer: - A

17. How many edges will a tree consisting of N nodes have?

- A) $\log(N)$
- B) N
- C) $N-1$
- D) $N+1$

Answer: - C

18. Minimum number of unique colors required for vertex coloring of a graph is called?

- A) vertex matching
- B) chromatic index
- C) chromatic number
- D) color number.

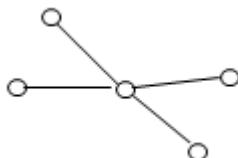
Answer: - C

19. How many unique colors will be required for proper vertex coloring of an empty graph having n vertices?

- A) 0
- B) 1
- C) n
- D) $n!$

Answer: - C

20. What will be the chromatic number of the following graph?



- A) 1
- B) 2
- C) 3

D) 4

Answer: - B

21. Assuming $P \neq NP$, which of the following is true ?

- A) NP-complete = NP
- B) NP-complete $\cap P = \emptyset$
- C) NP-hard = NP
- D) $P = NP$ -complete

Answer: - B

22. Let X be a problem that belongs to the class NP. Then which one of the following is TRUE?

- A) There is no polynomial time algorithm for X.
- B) If X can be solved deterministically in polynomial time, then $P = NP$.
- C) If X is NP-hard, then it is NP-complete.
- D) X may be undecidable.

NP Complete

Answer: - C

23. Which of the following statements are TRUE?

- 1. The problem of determining whether there exists a cycle in an undirected graph is in P.
- 2. The problem of determining whether there exists a cycle in an undirected graph is in NP.
- 3. If a problem A is NP-Complete, there exists a non-deterministic polynomial time algorithm to solve A.

A) 1, 2 and 3

B) 1 and 2 only

C) 2 and 3 only

D) 1 and 3 only

Answer: - A

24. Consider the following two problems on undirected graphs

α : Given $G(V, E)$, does G have an independent set of size $|V| - 4$?

β : Given $G(V, E)$, does G have an independent set of size 5?

Which one of the following is TRUE?

- A) α is in P and β is NP-complete
- B) α is NP-complete and β is in P
- C) Both α and β are NP-complete
- D) Both α and β are in P

Answer: - C

25. Which of the following algorithm can be used to solve the Hamiltonian path problem efficiently?

- A) branch and bound

- B) iterative improvement
- C) divide and conquer
- D) greedy algorithm

Answer: - A

26. Hamiltonian path problem is _____

- A) NP problem
- B) N class problem
- C) P class problem
- D) NP complete problem

Answer: - D

27. There is no existing relationship between a Hamiltonian path problem and Hamiltonian circuit problem.

- A) true
- B) false

Answer: - B

28. Which of the following problems is similar to that of a Hamiltonian path problem?

- A) knapsack problem
- B) closest pair problem
- C) travelling salesman problem
- D) assignment problem

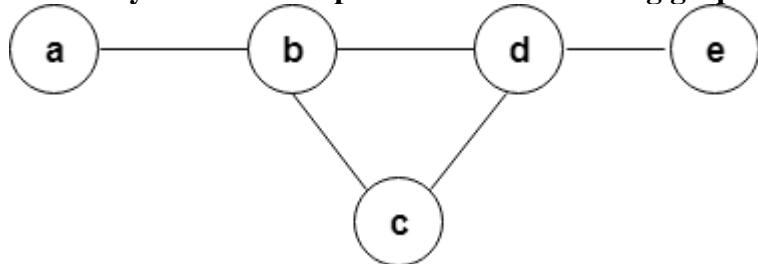
Answer: - C

29. In what time can the Hamiltonian path problem can be solved using dynamic programming?

- A) $O(N)$
- B) $O(N \log N)$
- C) $O(N^2)$
- D) $O(N^2 2^N)$

Answer: - D

30. How many Hamiltonian paths does the following graph have?



- A) 1
- B) 2
- C) 3
- D) 4

Answer: - A

31. A node is said to be _____ if it has a possibility of reaching a complete solution.

- A) Non-promising
- B) Promising
- C) Succeeding
- D) Preceding

Answer: - B

32. Minimum number of unique colors required for vertex coloring of a graph is called?

- A) vertex matching
- B) chromatic index
- C) chromatic number
- D) color number

Answer: C

PART B

1. Define NP hard and NP completeness.
2. Compare NP hard and NP completeness.
3. Write Short notes on “the class P and NP problem”.
4. How NP Hard problems are different from NP Complete?
5. Whether class P solves a problem in polynomial time? Justify.
6. An NP hard problem can be solved in deterministic polynomial time, how?
7. Give examples for NP Complete problems
8. State the property of NP complete problem.
9. Define adversary method.
10. Define lower bound.
11. What type of output yields trivial lower bound?
12. What is information theoretic lower bound?
13. Define complexity theory.
14. What is halting problem?

15. What is CNFs satisfiability problem?
16. Define Matching.
17. Define a bipartite graph.
18. How will you check the stability?
19. What is stable marriage problem?
20. Define the term stable pair
21. What do you mean by perfect match in bipartite graph?
22. Write Rabin Karp string matching algorithm
23. Describe Hamiltonian cycle problem

PART C

1. Describe in detail about P and NP Problems
2. Write short notes on NP Complete Problem
3. Write short notes on the following using approximation Algorithm
 - i) Nearest –neighbor algorithm with example
 - ii) Multi fragment heuristic algorithm with example
4. Describe in detail about Twice around the tree algorithm with example
5. Explain local search heuristic with example
6. Explain Approximation Algorithms for the Travelling Salesman Problem
7. Explain the Assignment problem in Branch and bound with Example.
8. Suggest an approximation algorithm for TSP. Assume that the cost function satisfies the triangle inequality.
9. Using an example prove that, satisfiability of Boolean formula in 3- Conjunctive Normal Form is NP – complete.
10. State the relationships among the complexity class algorithms with the help of

neat diagrams

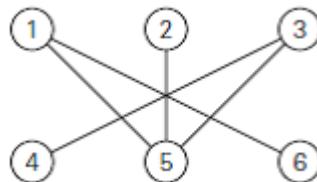
11. Explain the algorithm for stable marriage problem and prove the theorem with Example

12. Consider an instance of the stable marriage problem given by the ranking matrix

	A	B	C
a	1,3	2,2	3,1
b	3,1	1,3	2,2
c	2,2	3,1	1,3

For each of its marriage matching's, indicate whether it is stable or not

13. Apply the maximum matching algorithm to the following bipartite graphs



14. Write the algorithm for maximum matching in Bipartite Graphs and prove the theorem With example

15. Explain the algorithm:

- i. Blocking pair
- ii. Stable marriage problem
- iii. Man optimal
- iv. Women optimal

16. Explain briefly on minimum weight perfect matching algorithm.

17. Explain briefly on reducing bipartite graph to net flow

18. Explain local search heuristic with example

19. Consider the following minimization problem:

DEGREE BOUNDED SPANNING TREE:

Instance: Graph $G = (V, E)$

Solution:: A spanning tree T of G

Value: Maximum degree of T

Goal: Find a solution with minimum value.

20. Consider the following scheduling problem. You are given n jobs where job i is specified by an earliest start time s_i and a processing time p_i . In homework 1, we considered a preemptive version of this problem and gave a greedy algorithm to give an optimal preemptive schedule. In this problem we consider the non-preemptive version of this scheduling problem. Here a job CANNOT be suspended but rather must be performed in a contiguous time interval. Consider the following heuristic for the non-preemptive problem: schedule the jobs in the order in which they complete in an optimal preemptive schedule starting each job as soon as the one before it completes. You are to prove that this algorithm is a 2-approximation algorithm.
21. Using Rabin karp string matching algorithm match the given pattern P with given string S .
 $P = 745$
 $S = 745727457$
22. Using KMP string matching algorithm, find the occurrence of the given pattern P in the given text T .
 $T \triangleq ABABACAB$
 $P \triangleq ABAB$

UNIT IV

INTRODUCTION TO BACKTRACKING- BRANCH AND BOUND

Multiple Choice Questions with Answers:

Multiple Choice Questions with Answers:

1. Which of the problems cannot be solved by backtracking method? **CLO-4 Pg No- 486**

 - a) n-queen problem
 - b) Subset sum problem
 - c) Hamiltonian circuit problem
 - d) **Travelling salesman problem**

2. What happens when the backtracking algorithm reaches a complete solution? **CLO-4 Pg No- 518**

 - a) It backtracks to the root
 - b) It continues searching for other possible solutions**
 - c) It traverses from a different route
 - d) Recursively traverses through the same route

3. In general, backtracking can be used to solve? **CLO-4 Pg No-517**

 - a) Numerical problems
 - b) Exhaustive search
 - c) Combinatorial problems**
 - d) Graph coloring problems

4. How many solutions are there for 8 queens on 8*8 board? **CLO-4 Pg No-522**

 - a) 12
 - b) 91
 - c) 92**
 - d) 93

5. In how many directions do queens attack each other? **CLO-4 Pg No- 523**

 - a) 1
 - b) 2
 - c) 3**
 - d) 4

6. Which of the following methods can be used to solve n-queen's problem? **CLO-4 Pg No-523**

 - a) Greedy algorithm
 - b) Divide and conquer
 - c) Iterative improvement
 - d) Backtracking**

7. What is the domination number for 8-queen's problem? **CLO-4 Pg No-523**

 - a) 8
 - b) 7
 - c) 6**
 - d) 5

8. Which of the following is true about the time complexity of the recursive solution of the subset sum problem? **CLO-4 Pg No-525**

 - a) It has an exponential time complexity**
 - b) It has a linear time complexity
 - c) It has a logarithmic time complexity
 - d) it has a time complexity of $O(n^2)$

9. What is the worst case time complexity of dynamic programming solution of the subset sum problem(sum=given subset sum)? **CLO-3 Pg No-525**

- a) $O(n)$
- b) $O(\text{sum})$
- c) $O(n^2)$
- d) $O(\text{sum} * n)$**

10. Which of the following is not true about subset sum problem? **CLO-3 Net Source**

- a) the recursive solution has a time complexity of $O(2n)$
- b) there is no known solution that takes polynomial time
- c) the recursive solution is slower than dynamic programming solution
- d) the dynamic programming solution has a time complexity of $O(n \log n)$**

11. Which of the following algorithm can be used to solve the Hamiltonian path problem efficiently? **CLO-3 Pg No-531**

- | | |
|----------------------------|--------------------------|
| a) Branch and Bound | b) Iterative improvement |
| c) Greedy algorithm | d) Divide and Conquer |

12. The problem of finding a path in a graph that visits every vertex exactly once is called? **CLO-3 Pg No-466**

- | | |
|------------------------------------|------------------------------|
| a) Hamiltonian path problem | b) Hamiltonian cycle problem |
| c) Subset sum problem | d) NP problem |

13. In what time can the Hamiltonian path problem be solved using dynamic programming? **CLO-3 Pg No-531**

- | | |
|-------------|-----------------------------------|
| a) $O(N)$ | b) $O(n \log n)$ |
| c) $O(N^2)$ | d) $O(N^2 2^N)$ |

14. Which of the following problems is similar to that of a Hamiltonian path problem? **CLO-3 Net Source**

- | | |
|---------------------------------------|-------------------------|
| a) knapsack problem | b) closest pair problem |
| c) travelling salesman problem | d) assignment problem |

15. What is the time complexity for finding a Hamiltonian path for a graph having N vertices (using permutation)? **CLO-3 Pg No- 531**

- | | |
|----------------|----------------------------------|
| a) $O(N!)$ | b) $O(N! * N)$ |
| c) $O(\log N)$ | d) $O(N)$ |

16. What is the time complexity of the brute force algorithm used to solve the Knapsack problem? **CLO-3 Pg No- 496**

- | | |
|-------------------------------|-------------|
| a) $O(n)$ | b) $O(n!)$ |
| c) $O(2^n)$ | d) $O(n^3)$ |

17. The Knapsack problem is an example of _____ **CLO-3 Pg No-497**

- | | |
|---------------------------|----------------------------------|
| a) Greedy algorithm | b) 2D dynamic programming |
| c) 1D dynamic programming | d) Divide and conquer |

18. Which of the following methods can be used to solve the Knapsack problem? **CLO-3 Pg No- 496**

- a) Brute force algorithm

- b) Recursion
 - c) Dynamic programming
 - d) Brute force, Recursion and Dynamic Programming**

19. Travelling salesman problem is an example of **CLO-4 Pg No- 486**
a.Dynamic Algorithm
c.Recursive Approach
b.Greedy Algorithm
d.Divide & Conquer

20. A graph in which all nodes are of equal degree is called **CLO-3 Net Source**
a) Multi graph
c) **Regular graph**
b) Non regular graph
d) Complete graph

21. Which of the following algorithms can be used to most efficiently determine the presence of a cycle in a given graph ? **CLO-3 Pg No-468**
a) Depth First Search
b) Breadth First Search
c) Prim's Minimum Spanning Tree Algorithm
d) Kruskal' Minimum Spanning Tree Algorithm

22. Traversal of a graph is different from tree because **CLO-4 Pg No-469**
a) There can be a loop in graph so we must maintain a visited flag for every vertex
b) DFS of a graph uses stack, but in-order traversal of a tree is recursive
c) BFS of a graph uses queue, but a time efficient BFS of a tree is recursive
d) All of the above

23. Given two vertices in a graph s and t, which of the two traversals (BFS and DFS) can be used to find if there is path from **s to t?** **CLO-4 Net Source**
a) Only BFS
c) **Both BFS and DFS**
b) Only DFS
d) Neither BFS nor DFS

24. A complete graph can have **CLO-3 Pg No-466**
a. n^2 spanning trees
c. n^{n+1} spanning trees
b. $n^{(n-2)}$ spanning trees
d. n^n spanning trees

25. Graphs are represented using **CLO-3 Net source**
a.Adjacency tree
c.Adjacency graph
b.Adjacency linked list
d.Adjacency queue

26. The spanning tree of connected graph with 10 vertices contains **CLO-3 Net source**
a.9 edges
c.10 edges
b.11 edges
d. **9 vertices**

27. Which of the following algorithms solves the all-pair shortest path problem? **CLO-4 Pg No-478**
a.Floyd's algorithm
c.Dijkstra's algorithm
b.Prim's algorithm
d.Warshall's algorithm

28. The minimum number of colors needed to color a graph having n (>3) vertices and 2 edges is **CLO-3 Net Source**

- a.1
- b.2**
- c.3
- d.4

29. Which of the following is useful in traversing a given graph by breadth first search?

CLO-3 Net Source

- a.set
- b.List**
- c.stacks
- d.Queue**

30. The minimum number of edges in a connected cyclic graph on n vertices is **CLO-3 Pg No- 466**

- a.n**
- b. $n+1$
- c. $n-1$
- d.none of the above

31. Floyd Warshall's Algorithm can be applied on _____ **CLO-3 Pg No-478**

- a) Undirected and unweighted graphs
- b) Undirected graphs
- c) Directed graphs**
- d) Acyclic graphs

32. What is the running time of the Floyd Warshall Algorithm? **CLO-3 Pg No-479**

- a) Big-oh(V)
- b) Theta(V^2)
- c) Big-Oh(VE)**
- d) Theta(V^3)

34. What procedure is being followed in Floyd Warshall Algorithm? **CLO-3 Pg No-479**

- a) Top down
- b) Bottom up**
- c) Big bang
- d) Sandwich

35. Floyd Warshall Algorithm can be used for finding _____ **CLO-3 Pg No-480**

- a) Single source shortest path
- b) Topological sort
- c) Minimum spanning tree
- d) Transitive closure**

36. What approach is being followed in Floyd Warshall Algorithm? **CLO-3 Pg No-480**

- a) Greedy technique
- b) Dynamic Programming**
- c) Linear Programming
- d) Backtracking

37. What happens when the value of k is 0 in the Floyd Warshall Algorithm? **Pg No-481**

- a) 1 intermediate vertex
- b) 0 intermediate vertex**
- c) N intermediate vertices
- d) N-1 intermediate vertices

38. The time required to find shortest path in a graph with n vertices and e edges is **CLO-3 Net Source**

- a. O (e)
- b. O (n)**
- c. O (n²)**
- d. O (e²)

39. For 0/1 KNAPSACK problem, the algorithm takes _____ amount of time for memory table, and _____ time to determine the optimal load, for N objects and W as the capacity of KNAPSACK **CLO-4 Pg No-497**

- a. O(NW), O(N+W)
- b. O(N), O(NW)
- c. O(N+W), O(NW)
- d. O(NW), O(N)

40. Given a directed graph where weight of every edge is same, we can efficiently find shortest path from a given source to destination using? **CLO-3 Pg No-478**

(A) Breadth First Traversal

(B) Dijkstra's Shortest Path Algorithm

(C) Neither Breadth First Traversal nor Dijkstra's algorithm can be used

(D) Depth First Search

41. What can be the applications of Breadth First Search? **CLO-3 Net Source**

A. Finding shortest path between two nodes

B. Finding bipartiteness of a graph

C. GPS navigation system

D. All of the mentioned

42. What can be the applications of Depth First Search? **CLO-3 Net Source**

A. For generating topological sort of a graph

B. For generating Strongly Connected Components of a directed graph

C. Detecting cycles in the graph

D. All of the mentioned

43. When the Depth First Search of a graph is unique? **CLO-3 Net Source**

A. When the graph is a Binary Tree

B. When the graph is a Linked List

C. When the graph is a n-ary Tree

D. None of the mentioned

44. For a directed graph with edge lengths, the floyd warshall algorithm can compute the between each pair of nodes in $O(n^3)$. **CLO-3 Pg No- 480**

A) Transitive Hull

B) Minimax Distance

C) Max Min Distance

D) Safest Path

45. Dijkstra algorithm is also called the shortest path problem. **CLO-3 Pg No- 470**

A) multiple source

B) single source

C) single destination

D) multiple destination

46. The floyd-warshall all pairs shortest path algorithm computes the shortest paths between each pair of nodes in **CLO-3 Pg No- 479**

A) $O(\log n)$

B) $O(n^2)$

C) $O(mn)$

D) $O(n^3)$

PART B (4 Marks)

1. What is Backtracking?
2. Explain the back tracking Explicit and implicit constraints.
3. Write an algorithm for Backtracking with an example.
4. Which are the problems solved using backtracking method
5. Define sum of subsets
6. Give an example of sum of subset.
7. How to generate state space tree ?
8. Explain Sum of subset Algorithm:
9. Explain Hamiltonian cycles:
10. What are the steps used in Hamiltonian Procedure:
11. Write an Algorithm for Finding all Hamiltonian cycle
12. Write short notes about 4-Queens problem.
13. What are the steps to generate the 8 Queen solution.
14. Write an algorithm for N queens problem.
15. Draw any Two possible solutions for 4 Queen problem.
16. Illustrate graph coloring problem.
17. What are the general Steps to color the Graph in graph coloring problem.
19. Write short notes on graph coloring Algorithm.
20. Define Knapsack Problem using Backtracking method.
21. Write about backtracking knapsack Algorithm
22. Give short notes on Floyds- Warshall.
23. What is the difference between BFS and DFS.
24. Give an example for Floyds – Warshall algorithm.
25. Write short notes on BFS and DFS.
26. What is shortest path algorithm? Give an example.
27. Explain the time complexity calculation of Travelling salesman Problem.
28. Explain the time complexity calculation of Floyds- warshall algorithm.

PART C (12 Marks)

1. Briefly explain about backtracking method with suitable example.
2. Explain about 8 queens problem with an algorithm.
3. Explain Sum of subsets with an example.
4. What is Graph coloring problem and explain its algorithm using backtracking method.
5. How to find all Hamiltonian cycle using backtracking method? Explain with an algorithm.
6. Explain about Travelling Salesman problem using backtracking.
7. Explain about Floyds- Warshall algorithm with suitable examples.
8. Explain about Travelling Salesman problem using branch and bound algorithm.
9. Explain Depth First Search with an example.
10. What is BFS? Give an explanation with example.
11. Explain about Knapsack problem with suitable examples.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY, RAMAPURAM
CAMPUS COMPUTER SCIENCE AND ENGINEERING
QUESTION BANK
18CSC204J DESIGN AND ANALYSIS OF ALGORITHMS

UNIT 4

Introduction to backtracking - branch and bound, N queen's problem – backtracking, Sum of subsets using backtracking, Complexity calculation of sum of subsets, Graph introduction, Hamiltonian circuit – backtracking, Branch and bound - Knapsack problem, Example and complexity calculation. Differentiate with dynamic and greedy, Travelling salesman problem using branch and bound, Travelling salesman problem using branch and bound example, Travelling salesman problem using branch and bound example, Time complexity calculation with an example, Graph algorithms, Depth first search and Breadth first search, Shortest path introduction, Floyd-Warshall Introduction, Floyd-Warshall with sample graph, Floyd-Warshall complexity

PART A

1. Which of the following is not a backtracking algorithm?

- (A) Knight tour problem
- (B) N queen problem
- (C) Tower of hanoi
- (D) M coloring problem

Answer: - C

2. Backtracking algorithm is implemented by constructing a tree of choices called as?

- A) State-space tree
- B) State-chart tree
- C) Node tree
- D) Backtracking tree

Answer: - A

3. What happens when the backtracking algorithm reaches a complete solution?

- A) It backtracks to the root

- B) It continues searching for other possible solutions
- C) It traverses from a different route
- D) Recursively traverses through the same route

Answer: - B

4. In what manner is a state-space tree for a backtracking algorithm constructed?

- A) Depth-first search
- B) Breadth-first search
- C) Twice around the tree
- D) Nearest neighbour first

Answer: - A

5. In general, backtracking can be used to solve?

- A) Numerical problems
- B) Exhaustive search
- C) Combinatorial problems
- D) Graph coloring problems

Answer: - C

6. Which one of the following is an application of the backtracking algorithm?

- A) Finding the shortest path
- B) Finding the efficient quantity to shop
- C) Ludo
- D) Crossword

Answer: - D

7. Who coined the term ‘backtracking’?

- A) Lehmer
- B) Donald
- C) Ross
- D) Ford

Answer: - A

8. The problem of finding a subset of positive integers whose sum is equal to a given positive integer is called as?

- A) n- queen problem
- B) subset sum problem
- C) knapsack problem
- D) hamiltonian circuit problem

Answer: - B

9. The problem of placing n queens in a chessboard such that no two queens attack each other is called as?

- A) n-queen problem
- B) eight queens puzzle
- C) four queens puzzle

D) 1-queen problem

Answer: - A

10. In how many directions do queens attack each other?

A) 1

B) 2

C) 3

D) 4

Answer: - C

11. Placing n-queens so that no two queens attack each other is called?

A) n-queen's problem

B) 8-queen's problem

C) Hamiltonian circuit problem

D) subset sum problem

Answer: - A

12. Where is the n-queens problem implemented?

A) carom

B) chess

C) ludo

D) cards

Answer: - B

13. Not more than 2 queens can occur in an n-queens problem.

A) true

B) false

Answer: - B

14. In n-queen problem, how many values of n does not provide an optimal solution? A) 1

B) 2

C) 3

D) 4

Answer: - B

15. Which of the following methods can be used to solve n-queen's problem? A) greedy algorithm

B) divide and conquer

C) iterative improvement

D) backtracking

Answer: - D

16. Of the following given options, which one of the following is a correct option that provides an optimal solution for 4-queens problem?

- A) (3,1,4,2)
- B) (2,3,1,4)
- C) (4,3,2,1)
- D) (4,2,3,1).

Answer: - A

17. How many possible solutions exist for an 8-queen problem?

- A) 100
- B) 98
- C) 92
- D) 88

Answer: - C

18. How many possible solutions occur for a 10-queen problem?

- A) 850
- B) 742
- C) 842
- D) 724.

Answer: - D

19. The Knapsack problem is an example of _____

- A) Greedy algorithm
- B) 2D dynamic programming
- C) 1D dynamic programming
- D) Divide and conquer

Answer: - B

20. Which of the following methods can be used to solve the Knapsack problem? A) Brute force algorithm

- B) Recursion
- C) Dynamic programming
- D) Brute force, Recursion and Dynamic Programming

Answer: - D

21. You are given a knapsack that can carry a maximum weight of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. What is the maximum value of the items you can carry using the knapsack?

- A) 160
- B) 200
- C) 170
- D) 90

Answer: - A

22. Which of the following problems is equivalent to the 0-1 Knapsack problem? A)

You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of $\{w_1, w_2, w_3, \dots, w_n\}$ and a value of $\{v_1, v_2, v_3, \dots, v_n\}$. You

can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

B) You are studying for an exam and you have to study N questions. The questions take $\{t_1, t_2, t_3, \dots, t_n\}$ time(in hours) and carry $\{m_1, m_2, m_3, \dots, m_n\}$ marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized

C) You are given infinite coins of denominations $\{v_1, v_2, v_3, \dots, v_n\}$ and a sum S. You have to find the minimum number of coins required to get the sum S

D) You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of $\{10, 20, 15, 40\}$ and a value of $\{1, 2, 3, 4\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value

Answer: - B

23. What is the time complexity of the brute force algorithm used to solve the Knapsack problem?

- A) $O(n)$
- B) $O(n!)$
- C) $O(2^n)$
- D) $O(n^3)$

Answer: - C

24. Which of the following is/are property/properties of a dynamic programming problem?

- A) Optimal substructure
- B) Overlapping subproblems
- C) Greedy approach
- D) Both optimal substructure and overlapping subproblems

Answer: - D

25. If an optimal solution can be created for a problem by constructing optimal solutions for its subproblems, the problem possesses _____ property. A) Overlapping subproblems

- B) Optimal substructure
- C) Memoization
- D) Greedy

Answer: - B

26. If a problem can be broken into subproblems which are reused several times, the problem possesses _____ property.

- A) Overlapping subproblems
- B) Optimal substructure
- C) Memoization
- D) Greedy

Answer: - A

27. If a problem can be solved by combining optimal solutions to non-overlapping problems, the strategy is called _____

- A) Dynamic programming
 - B) Greedy
 - C) Divide and conquer
 - D) Recursion
- Answer: - C

28. In dynamic programming, the technique of storing the previously calculated values is called _____

- A) Saving value property
- B) Storing value property
- C) Memoization
- D) Mapping

Answer: - C

29. When a top-down approach of dynamic programming is applied to a problem, it usually _____

- A) Decreases both, the time complexity and the space complexity
- B) Decreases the time complexity and increases the space complexity
- C) Increases the time complexity and decreases the space complexity
- D) Increases both, the time complexity and the space complexity

Answer: - B

30. Which of the following problems is NOT solved using dynamic programming? A) 0/1 knapsack problem

- B) Matrix chain multiplication problem
- C) Edit distance problem
- D) Fractional knapsack problem

Answer: - D

31. Which of the following problems should be solved using dynamic programming? A) Mergesort

- B) Binary search
- C) Longest common subsequence
- D) Quicksort

Answer: - C

32. Time Complexity of Breadth First Search is? (V - number of vertices, E - number of edges)

- A) $O(V+E)$
- B) $O(V)$
- C) $O(E)$
- D) $O(VE)$

Answer: - A

33. The spanning tree of connected graph with 10 vertices contains

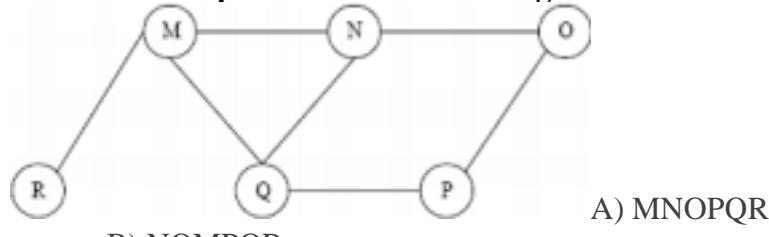
- A) 9 edges
- B) 11 edges

C) 10 edges

D) 8 edges

Answer: - A

34. The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is



A) MNOPQR

B) NQMPOR

C) QMNPRO

D) QMNPOR

Answer: - C

35. What is the maximum height of queue (To keep track of un-explored nodes) required to process a connected Graph G1 which contains 'N' node using BFS algorithm? A) $(N/2)-1$

B) $(N/2)/2$

C) N-1

D) N

Answer: - C

PART B

1. What is meant by knapsack problem?
2. Define fractional knapsack problem.
3. Write the running time of 0/1 knapsack problem.
4. Write recurrence relation for 0/1 knapsack problem
5. What is meant by travelling salesperson problem?
6. What is the running time of dynamic programming TSP?
7. State if backtracking always produces optimal solution.
8. Define backtracking.
9. What are the two types of constraints used in backtracking?
10. What is meant by optimization problem?
11. Define Hamiltonian circuit problem.
12. What is Hamiltonian cycle in an undirected graph?
13. Define 8queens problem. 8. List out the application of backtracking.
14. Define promising node and non-promising node.
15. Give the explicit and implicit constraint for 8-queen problem.
16. How can we represent the solution for 8-queen problem?
17. Give the categories of the problem in backtracking.

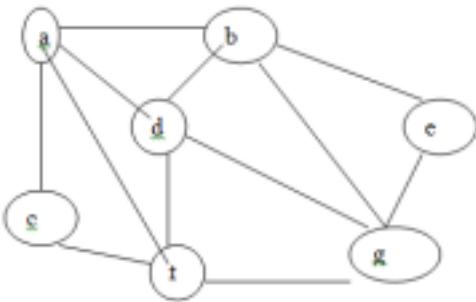
18. Differentiate backtracking and over exhaustive search.
19. Find optimal solution for the knapsack instance $n = 3, w = [20, 15, 15], P = [40, 25, 25]$ and $C = 30$
20. What is travelling salesperson problem?
21. What is the formula used to find upper bound for knapsack problem?
22. Differentiate between back tracking and branch and bound.
23. List out the application of branch and bound technique.
24. Analyze the time complexity for Warshall's and Floyd's algorithm.
25. Test the 0/1 knapsack problem.
26. Summarize Warshall's algorithm
27. Compare feasible and optimal solution.
28. Differentiate between DFS and BFS.
29. What is efficiency of DFS based algorithm for topological sorting
30. What are the different applications of DFS and BFS?

PART C

1. Describe the travelling salesman problem and discuss how to solve it using dynamic programming?
2. Find the optimal solution for the given knapsack problem

Item	1	2	3	4
weight	2	1	3	2
Value	\$12	\$10	\$20	\$15

3. Apply backtracking technique to solve the following instance of the subset sum problem $S = \{1, 3, 4, 5\}$ and $d = 11, 16$
4. Explain subset-sum problem and discuss the possible solution strategies using backtracking.
5. Explain N-queens problem with an algorithm.
6. Explain why backtracking is defined as a default procedure of last resort for solving problems.
7. Explain the subset-sum problem in detail by justifying it using backtracking algorithm.
8. Apply backtracking to the problem of finding a Hamiltonian circuit for the following graph.



9. What is backtracking? Explain in detail.
 10. Solve the following instance of the knapsack problem by the branch and bound algorithm.

Item	Weight	Value
1	4	\$40
2	7	#42
3	5	\$25
4	3	\$12
The Knapsack's capacity W=10		

11. Discuss the solution for knapsack problem using branch and bound technique. 12. What is branch and bound technique? Explain how knapsack problem could be solved using branch and bound technique. Solve the following instance of the knapsack problem by branch and bound algorithm for W=16

Item	Weight	Value in Rs.
1	10	100
2	7	63
3	8	56
4	4	12

13. What is branch and bound? Explain in detail.
 14. Consider the below matrix for assignment problem involving persons and jobs. Explain in detail how branch and bound technique is useful in solving assignment problems.

	Job1	Job2	Job3	Job4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

15. Discuss Warshall's algorithm with suitable example.

1. In analysis of algorithm, approximate relationship between the size of the job and the amount of work required to do is expressed by using _____
(a) Central tendency (b) Differential equation (c) Order of execution (d) Order of magnitude
(e) Order of Storage.

Ans :Order of execution

2. Worst case efficiency of binary search is

- (a) $\log_2 n + 1$ (b) n (c) N^2 (d) $2n$ (e) $\log n$.

Ans : $\log_2 n + 1$

3. For analyzing an algorithm, which is better computing time?

- (a) $O(100 \log N)$ (b) $O(N)$ (c) $O(2N)$ (d) $O(N \log N)$ (e) $O(N^2)$.

Ans : $O(100 \log N)$

4. Consider the usual algorithm for determining whether a sequence of parentheses is balanced. What is the maximum number of parentheses that will appear on the stack AT ANY ONE TIME when the algorithm analyzes: $((())((())(())$)

- (a) 1 (b) 2 (c) 3 (d) 4

Ans :3

5. Breadth first search _____

- (a) Scans each incident node along with its children. (b) Scans all incident edges before moving to other node. (c) Is same as backtracking (d) Scans all the nodes in random order.

Ans : Scans all incident edges before moving to other node.

6. Which method of traversal does not use stack to hold nodes that are waiting to be processed?

- (a) Depth First (b) D-search (c) Breadth first (d) Back-tracking

Ans : Breadth first

7. The Knapsack problem where the objective function is to minimize the profit is _____

- (a) Greedy (b) Dynamic 0 / 1 (c) Back tracking (d) Branch & Bound 0/1

Ans : Branch & Bound 0/1

8. Choose the correct answer for the following statements:

- I. The theory of NP-completeness provides a method of obtaining a polynomial time for

NP algorithms.

II. All NP-complete problem are NP-Hard.

- (a) I is FALSE and II is TRUE (b) I is TRUE and II is FALSE (c) Both are TRUE (d) Both are FALSE

Ans :I is FALSE and II is TRUE

9. If all $c(i, j)$'s and $r(i, j)$'s are calculated, then OBST algorithm in worst case takes one of the following time.

- (a) $O(n \log n)$ (b) $O(n^3)$ (c) $O(n^2)$ (d) $O(\log n)$ (e) $O(n^4)$.

Ans : $O(n^3)$

10. The upper bound on the time complexity of the nondeterministic sorting algorithm is

- (a) $O(n)$ (b) $O(n \log n)$ (c) $O(1)$ (d) $O(\log n)$ (e) $O(n^2)$.

Ans: $O(n)$

11. The worst case time complexity of the nondeterministic dynamic knapsack algorithm is

- (a) $O(n \log n)$ (b) $O(\log n)$ (c) $O(n^2)$ (d) $O(n)$ (e) $O(1)$.

Ans : $O(n)$

12. Recursive algorithms are based on

- (a) Divide and conquer approach (b) Top-down approach (c) Bottom-up approach (d) Hierarchical approach

Ans :Bottom-up approach

13. What do you call the selected keys in the quick sort method?

- (a) Outer key (b) Inner Key (c) Partition key (d) Pivot key (e) Recombine key.

Ans :c

14. How do you determine the cost of a spanning tree?

- (a) By the sum of the costs of the edges of the tree (b) By the sum of the costs of the edges and vertices of the tree
(c) By the sum of the costs of the vertices of the tree (d) By the sum of the costs of the edges of the graph
(e) By the sum of the costs of the edges and vertices of the graph.

Ans :By the sum of the costs of the edges of the tree8.

15. The time complexity of the normal quick sort, randomized quick sort algorithms in the worst case is

- (a) $O(n^2)$, $O(n \log n)$ (b) $O(n^2)$, $O(n^2)$ (c) $O(n \log n)$, $O(n^2)$ (d) $O(n \log n)$, $O(n \log n)$ (e) $O(n \log n)$, $O(n^2 \log n)$.

Ans : $O(n^2)$, $O(n^2)$

16. Let there be an array of length 'N', and the selection sort algorithm is used to sort it, how many times a swap function is called to complete the execution?

- (a) $N \log N$ times (b) $\log N$ times (c) N^2 times (d) $N-1$ times (e) N times.

Ans : $N-1$ times

17. The Sorting method which is used for external sort is

- (a) Bubble sort (b) Quick sort (c) Merge sort (d) Radix sort (e) Selection sort.

Ans :Radix sort

18. The graph colouring algorithm's time can be bounded by _____

- (a) $O(mnm)$ (b) $O(nm)$ (c) $O(nm \cdot 2n)$ (d) $O(nmn)$.

Ans : $O(nmn)$.

19. Sorting is not possible by using which of the following methods?

- (a) Insertion (b) Selection (c) Deletion (d) Exchange

Ans :Deletion

20. What is the type of the algorithm used in solving the 8 Queens problem?

- (a) Backtracking (b) Dynamic (c) Branch and Bound (d) D and C

Ans :Backtracking

1. Name the node which has been generated but none of its children nodes have been generated in state space tree of backtracking method.

- (a) Dead node (b) Live node (c) E-Node (d) State Node

Ans: Livenode

2. How many nodes are there in a full state space tree with $n = 6$?

- (a) 65 (b) 64 (c) 63 (d) 32

Ans : 63

3. This algorithm scans the list by swapping the entries whenever pair of adjacent keys are

out of desired order.

- (a) Insertion sort. (b) Bubble sort. (c) Shell sort. (d) Quick sort.

Ans: Bubble sort.

5. From the following chose the one which belongs to the algorithm paradigm other than to which others from the following belongs to.

- (a) Minimum & Maximum problem. (b) Knapsack problem. (c) Selection problem.(d) Merge sort.

Ans: Knapsack problem.

6. To calculate $c(i, j)$'s, $w(i, j)$'s and $r(i, j)$'s; the OBST algorithm in worst case takes the following time.

- (a) $O(\log n)$ (b) $O(n^4)$ (c) $O(n^3)$ (d) $O(n \log n)$

Ans: $O(n^3)$

7. What is the type of the algorithm used in solving the 4 Queens problem?

- (a) Greedy (b) Dynamic (c) Branch and Bound (d) Backtracking.

Ans: Backtracking.

8. In Knapsack problem, the best strategy to get the optimal solution, where P_i , W_i is the Profit, Weight associated with each of the X_i object respectively is to

- (a) Arrange the values P_i/W_i in ascending order (b) Arrange the values P_i/X_i in ascending order
(c) Arrange the values P_i/W_i in descending order (d) Arrange the values P_i/X_i in descending order

Ans: Arrange the values P_i/X_i in descending order

9. Greedy job scheduling with deadlines algorithms' complexity is defined as

- (a) $O(N)$ (b) $\Omega(n \log n)$ (c) $O(n^2 \log n)$ (d) $O(n \log n)$

Ans: $O(N)$

12 From the following choose the one which belongs to the algorithm paradigm other than to which others from the following belongs to.

- (a) Minimum & Maximum problem (b) Knapsack problem(c) Selection problem (d) Merge sort

Ans : Knapsack problem

14. Identify the name of the sorting in which time is not proportional to n^2 .

- (a) Selection sort (b) Bubble sort (c) Quicik sort (d) Insertion sort.

Ans : Insertion sort

15. The optimal solution to a problem is a combination of optimal solutions to its subproblems. This is known as

- (a) Principleof Duality (b) Principle of Feasibility (c) Principle of Optimality (d)

Principle of Dynamicity.

Ans : Principle of Optimality

16. Which of the following versions of merge sort algorithm does uses space efficiently?

- (a) Contiguous version (b) Array version (c) Linked version (d) Structure version (e) Heap version.

Ans : Linked version

17. Identify the correct problem for multistage graph from the list given below.

- (a) Resource allocation problem (b) Traveling salesperson problem
- (c) Producer consumer problem (d) Barber's problem

Ans : Resource allocation problem

18. How many edges are there in a Hamiltonian cycle if the edge cost is 'c' and the cost of cycle is ' cn '

- (a)c (b) cn (c) n (d) $2c$

Ans : n .

19. A problem L is NP-complete iff L is NP-hard and

- (a) $L \approx NP$ (b) $L \propto NP$ (c) $L \in NP$ (d) $L = NP$

Ans : $L \in NP$

20. What would be the cost value for any answering node of a sub tree with root 'r' using branch-bound algorithm?

- (a) Maximum (b) Minimum (c) Optimal (d) Average

Ans: Minimum

1. From the following pick the one which does not belongs to the same paradigm to which others belongs to.

- (a) Minimum & Maximum problem
- (b) Knapsack problem
- (c) Selection problem
- (d) Merge sort

Ans:Knapsack problem

2. Primsalgorithm is based on _____ method

- a. Divide and conquer method
- c. Dynamic programming
- b. Greedy method
- d. Branch and bound

Ans. Greedy Method

3. The amount of memory needs to run to completion is known as_____

- a. Space complexity
- c. Worst case
- b. Time complexity
- d. Best case

Ans: Space complexity

4. The amount of time needs to run to completion is known as_____

- a. Space complexity
- c. Worst case
- b. Time complexity
- d. Best case

Ans: Time complexity

5. _____ is the minimum number of steps that can executed for the given parameters

- a. Average case
- c. Worst case
- b. Time complexity
- d. Best case

Ans: Best case

6. _____ is the maximum number of steps that can executed for the given parameters

- a. Average case
- c. Worst case
- b. Time complexity
- d. Best case

Ans:Worst case

7. _____ is the average number of steps that can executed for the given parameters

- a. Average case
- c. Worst case
- b. Time complexity
- d. Best case

Ans: Average Case

8. Testing of a program consists of 2 phases which are _____ and _____

- a. Average case & Worst case b. Time complexity & Space complexity
- c. Validation and checking errors d. Debugging and profiling

Ans: Debugging and profiling

9. Worst case time complexity of binary search is _____

- a. $O(n)$ b. $O(\log n)$ c. $\Theta(n \log n)$ d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

10. Best case time complexity of binary search is _____

- a. $O(n)$ c. $\Theta(n \log n)$
- b. $O(\log n)$ d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

11. Average case time complexity of binary search is _____

- a. $O(n)$ c. $\Theta(n \log n)$
- b. $O(\log n)$ d. $\Theta(\log n)$

Ans: $\Theta(\log n)$

12. Merge sort invented by _____

- a. CARHOARE c. HAMILTON
- b. JOHN VON NEUMANN d. STRASSEN

Ans : JOHN VON NEUMANN

13. Quick sort invented by _____

- a. CARHOARE c. HAMILTON
- b. JOHN VON NEUMANN d. STRASSEN

Ans : CARHOARE

14. Worst case time complexity of Quick sort is _____

- a. $O(n^2 \log 7)$ c. $O(n \log n)$
- b. $O(n^2)$ d. $O(\log n)$

Ans : $O(n^2)$

15. Best case time complexity of Quick sort is _____

a. $O(n^2 \log n)$ c. $O(n \log n)$

b. $O(\log n)$ d. $O(\log n^2)$

Ans : $O(n \log n)$

16. Average case time complexity of Quick sort is _____

a. $\Theta(n \log n)$ b. $O(\log n)$ c. $O(n \log n)$ d. $\Theta(\log n)$ Ans : $O(n \log n)$

17. Which design strategy stops the execution when it finds the solution otherwise starts the problem from top

- a. Back tracking c. Divide and conquer
- b. Branch and Bound d. Dynamic programming

Ans: Back Tracking

18. Graphical representation of algorithm is _____

- a. Pseudo-code c. Graph Coloring
- b. Flow Chart d. Dynamic programming

Ans: Flow Chart

19. In pseudo-code conventions input express as _____

- a. input c. Read
- b. Write d. Return

Ans : Write

20. In pseudo-code conventions output express as _____

- a. input c. Read
- b. Write d. Return

Ans : Read

1. Tight bound is denoted as _____

- a. Ω c. Θ
- b. Ω d. O

Ans : Θ

2. Upper bound is denoted as _____

- a. Ω c. Θ
- b. ω d. O

Ans : O

3. lower bound is denoted as _____

- a. Ω
- c. Θ
- b. ω
- d. O

Ans : Ω

4. The function $f(n)=o(g(n))$ if and only if $\lim f(n)/g(n)=0$ as $n \rightarrow \infty$

- a. Little oh
- b. Little omega
- b. Big oh
- d. Omega

Ans : Little oh

5. The function $f(n)=o(g(n))$ if and only if $\lim g(n)/f(n)=0$ as $n \rightarrow \infty$

- a. Little oh
- b. Little omega
- b. Big oh
- d. Omega

Ans : Little omega

6. The general criteria of algorithm; zero or more quantities are externally supplied is _____

- a. Output
- b. Finiteness
- b. Effectiveness
- d. Input

Ans : Input

7. The general criteria of algorithm; at least one quantity is produced _____

- a. Output
- b. Finiteness
- b. Effectiveness
- d. Input

Ans : Output

8. The general criteria of algorithm; Each instruction is clear and unambiguous _____

- a. Output
- b. Definiteness
- b. Effectiveness
- d. Input

Ans : Definiteness

9. The general criteria of algorithm; algorithm must terminates after a finite number of steps _____

- a. Output
- b. Finiteness
- b. Effectiveness
- d. Input

Ans : Finiteness

10. Which is not a criteria of algorithm

- a. Input b. Output
- c. Time complexity d. Best case

Ans : Best case

11. Which is not in general criteria of algorithm

- a. Input b. Output
- c. Time complexity d. Effectiveness

Ans : Time complexity

12. Time complexity of given algorithm

Algorithm Display(A)

{

S:=0.0;

For i:=0 to n-1

{

S:=S+A[i];

Return S;

}

}

a. $4n+4$ c. $4n^2+4$

b. $2n^2+2n+2$ d. $4n+4$

Ans : $4n+4$

13. Time complexity of given algorithm

AlgorithmSum(A,S)

{

for i:=1 to n-1

{

for j:=2 to n-1

{

```
S:=S+i+j;  
return S;  
}  
}  
}
```

- a. $6n^2 - 14n + 4$
- c. $4n^2 + 6n + 12$
- b. $6n^2 + 14n + 10$
- d. $6n^2 - 14n + 10$

Ans : $6n^2 - 14n + 10$

14. Kruskal algorithm is based on _____ method

- a. Divide and conquer method
- b. Greedy method
- c. Dynamic programming
- d. Branch and bound

Ans. Greedy method

15. Prims algorithm is based on _____ method

- a. Divide and conquer method
- c. Dynamic programming
- b. Greedy method
- d. Branch and bound

Ans. Greedy Method

16. The output of Kruskal and Prims algorithm is _____

- a. Maximum spanning tree
- c. Spanning tree
- b. Minimum spanning tree
- d. None of these

1. Job sequencing with deadline is based on _____ method

- a. greedy method
- c. branch and bound
- b. dynamic programming
- d. divide and conquer

Ans. Greedy method

2. Fractional knapsack is based on _____ method

- a. greedy method
- c. branch and bound

3. 0/1 knapsack is based on _____ method

- a. greedy method c. branch and bound
- b. dynamic programming d. divide and conquer

Ans. Dynamic programming

4. The files x₁,x₂,x₃ are 3 files of length 30,20,10 records each. What is the optimal merge pattern value?

- a. 110 c. 60
- b. 90 d. 50

Ans. 90

5. The optimal merge pattern is based on _____ method

- a. Greedy method b. Dynamic programming
- c. Knapsack method d. Branch and bound

Ans. Greedy method

6. Who invented the word Algorithm

- a. Abu Ja'far Mohammed ibn Musa c. Abu Mohammed Khan
- b. Abu Jafar Mohammed Kasim d. Abu Ja'far Mohammed Ali Khan

Ans. Abu Ja'far Mohammed ibn Musa

7. In Algorithm comments begin with_____

- a. /* c. /
- b. */ d. //

Ans : //

8. The _____ of an algorithm is the amount of memory it needs to run to completion.

- a. Space Complexity c. Best Case
- b. Time Complexity d. Worst Case

Ans : Space Complexity

9. _____ is the process of executing a correct program on data sets and measuring the time and space it takes to compute the results.

- a. Debugging c. Combining
- b. Profiling d. Conquer

Ans : Profiling

10. In Algorithm Specification the blocks are indicated with matching _____

- a. Braces c. Square Brackets
- b. Parenthesis d. Slashes

Ans : Braces

11. Huffmancodes are the applications of _____ with minimal weighted external path length obtained by an optimal set.

- a. BST b. MST
- c. Binary tree d. Weighted Graph

Ans : Binary tree

12. From the following which is not return optimal solution

- a. Dynamic programming c. Backtracking
- b. Branch and bound d. Greedy method

Ans. Backtracking

13. _____ is an algorithm design method that can be used when the solution to a problem can be viewed as the result of a sequence of decisions

- a. Dynamic programming c. Backtracking
- b. Branch and bound d. Greedy method

Ans : Dynamic programming

14. The name backtrack was first coined by _____

- a. D.H.Lehmer c. L.Baumert
- b. R.J.Walker d. S. Golomb

Ans : D.H.Lehmer

15. The term _____ refers to all state space search methods in which all children of the – nodes are generated before any other live node can become the E-node.

- a. Backtacking c. Depth First Search
- b. Branch and Bound d. Breadth First Search

Ans ; Branch and Bound

16. A _____ is a round trip path along n edges of G that visits every vertex once

and returns to its starting position.

- a. MST c. TSP
- b. Multistage Graph d. Hamiltonian Cycle

Ans :Hamiltonian Cycle

17. Graph Coloring is which type of algorithm design strategy

- a. Backtacking c. Greedy
- b. Branch and Bound d. Dynamic programming

Ans : Backtracking

18. Which of the following is not a limitation of binary search algorithm?

- a. must use a sorted array
- b. requirement of sorted array is expensive when a lot of insertion and deletions are needed
- c. there must be a mechanism to access middle element directly
- d. binary search algorithm is not efficient when the data elements are more than 1000.

Ans : binary search algorithm is not efficient when the data elements are more than 1000.

19. Binary Search Algorithm cannot be applied to

- a. Sorted linked list c. Sorted linear array
- b. Sorted binary tree d. Pointer array

Ans :Sorted linked list

20. Two main measures for the efficiency of an algorithm are

- a. Processor and memory c. Time and space
- b. Complexity and capacity d. Data and space

Ans : Time and Space

1 The order of an internal node in a B+ tree index is the maximum number of children it can have. Suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes, and the block size is 512 bytes. What is the order of the internal node?

- A) 24 B) 25 C) 26 D) 27

Answer : (C)

2 The best data structure to check whether an arithmetic expression has balanced parentheses is a

- A) queue B) stack C) tree D) list

Answer : (B)

3. A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The levelorder traversal of the heap is given below: 10, 8,5,3,2 Two new elements 1 and 7 are inserted in the heap in that order. The level-order traversal of the heap after the insertion of the elements is

- A) 10,8,7,5,3,2,1 B) 10,8,7,2,3,1,5 C) 10,8,7,1,2,3,5 D) 10,8,7,3,2,1,5

Answer : (D)

4 The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

- A) 2 B) 3 C) 4 D) 6

Answer : (B)

5 The goal of structured programming is to

- A) have well indented programs B) be able to infer the flow of control from the compiled code C) be able to infer the flow of control from the program text D) avoid the use of GOTO statements

Answer : (C)

6 The tightest lower bound on the number of comparisons, in the worst case, for comparisonbased sorting is of the order of

- A) n B) n^2 C) $n \log n$ D) $n \log 2$

n

Answer : (B)

7 Let G be a simple graph with 20 vertices and 100 edges. The size of the minimum vertex cover of G is 8. Then, the size of the maximum independent set of G is

- A) 12 B) 8 C) Less than 8 D) More than 12

Answer : (A)

8 Let A be a sequence of 8 distinct integers sorted in ascending order. How many distinct pairs of sequences, B and C are there such that (i) each is sorted in ascending order, (ii) B has 5 and C has 3 elements, and (iii) the result of merging B and C gives A?

- A) 2 B) 30 C) 56 D) 256

Answer : (D)

9 A Priority-Queue is implemented as a Max-Heap. Initially, it has 5 elements. The levelorder traversal of the heap is given below: 10, 8,5,3,2 Two new elements 1 and 7 are inserted in the heap in that order. The level-order traversal of the heap after the insertion of the elements is

- A) 10,8,7,5,3,2,1 B) 10,8,7,2,3,1,5 C) 10,8,7,1,2,3,5 D) 10,8,7,3,2,1,5

Answer : (D)

10 The S-N curve for steel becomes asymptotic nearly at

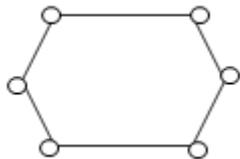
- A) 10³
cycles B) 10⁴
cycles C) 10⁶
cycles D) 10⁹
cycles

Answer : (C)

UNIT 4 – Backtracking and Branch & Bound MCQs

1. What will be the chromatic number of the following graph?

- a) 1 b) 2 c) 3 d) 4



2. What is the condition for proper coloring of a graph?

a) two vertices having a common edge should not have same color

b) two vertices having a common edge should always have same color

c) all vertices should have a different color

d) all vertices should have same color

3. What is a chromatic number?

a) The maximum number of colors required for proper edge coloring of graph

b) The maximum number of colors required for proper vertex coloring of graph

c) The minimum number of colors required for proper vertex coloring of graph

d) The minimum number of colors required for proper edge coloring of graph

4. The Data structure used in standard implementation of Breadth First Search is?

a) Stack

b) Queue

c) Linked List

d) Tree

5. The Data structure used in standard implementation of Depth First Search is?

a) Stack

b) Queue

c) Linked List

d) Tree

6. Backtracking algorithm is implemented by constructing a tree of choices called as?

a) State-space tree b) State-chart tree c) Node tree d) Backtracking tree

7. A node is said to be _____ if it has a possibility of reaching a complete solution.

a) Non-promising b) **Promising** c) Succeeding d) Preceding

8. In what manner is a state-space tree for a backtracking algorithm constructed?

a) Depth-first search b) Breadth-first search c) Twice around the tree d) Nearest neighbour first

9. _____ enumerates a list of promising nodes that could be computed to give the possible solutions of a given problem.

a) Exhaustive search b) Brute force c) **Backtracking** d) Divide and conquer

10. A _____ is a round trip path along n edges of G that visits every vertex once and return to its starting position

a) MST b) TSP c) Multistage Graph d) **Hamiltonian Cycle**

11. In general, backtracking can be used to solve?

a) Numerical problems b) Exhaustive search c) **Combinatorial problems** d) Graph coloring problems

12. Which of the following is not a branch and bound strategy to generate branches?

a) LIFO branch and bound b) FIFO branch and bound
c) Lowest cost branch and bound d) **Highest cost branch and bound**

13. Which of the following can traverse the state space tree only in DFS manner?

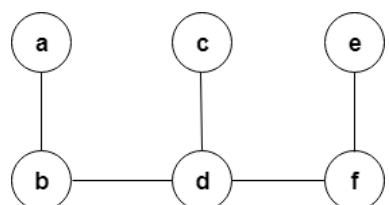
a) branch and bound b) dynamic programming c) greedy algorithm d) **backtracking**

14. which of the following problems is similar to that of a Hamiltonian path problem?

a) knapsack problem b) closest pair problem
c) **travelling salesman problem** d) assignment problem

15. How many Hamiltonian paths does the following graph have?

a) 1 b) 2 c) 3 d) 4



Reg. No.	
----------	--

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

CYCLE TEST – III – APRIL- 2020

Fourth Semester – Computer Science and Engineering

18CSC204J- Design and Analysis of Algorithms

Duration: 90 Minutes

Max. Marks: 50

PART – A (10 X 1 = 10 Marks)

Answer **ALL** Questions

1. What is the type of the algorithm used in solving the 8 Queens problem?
a)Greedy method b)Dynamic programming
c)Branch and Bound d)**Backtracking**.
2. The Hamiltonian cycles problem uses the following line of code to generate a next vertex, provided $x[]$ is a global array and k th vertex is under consideration:
a) $x[k] \leftarrow (x[k] + 1) \bmod n$ b) $x[k] \leftarrow (x[k]) \bmod (n)$
c) **$x[k] \leftarrow (x[k] + 1) \bmod (n+1)$** d) $x[k] \leftarrow x[k+1] \bmod n$
3. A node whose child node have been generated in state space tree is
a) E-node b) Live node c)**current node** d)state node
4. Let G be a graph with ‘ n ’ nodes and let ‘ m ’ be the chromatic number of the graph. Then the time taken by the backtracking algorithm to color it is
a) $O(nm)$ b) $O(n+m)$ c) $O(n^m)$ d) **$O(nm^n)$** .
5. How many edges are there in a Hamiltonian cycle if the edge cost is ‘ c ’ and the cost of cycle is ‘ cn ’
a) c b) C_n c) $2c$ d) **n**.
6. Let X be a problem that belongs to the class NP. Then which one of the following is TRUE?
a) There is no polynomial time algorithm for X .
b) If X can be solved deterministically in polynomial time, then $P = NP$.
c) **If X is NP-hard, then it is NP-complete.**
d) X may be undecidable.
7. The time complexity of the normal quick sort, randomized quick sort algorithms in the worst case is
a) **$O(n^2)$, $O(n \log n)$** b) $O(n^2)$, $O(n^2)$
c) $O(n \log n)$, $O(n^2)$ d) $O(n \log n)$, $O(n \log n)$

8. Randomized Algorithm also called as
a) Approximation Algorithm b)**probabilistic algorithm**
c) Optimal algorithm d) logarithmic algorithm
9. Which of following class of decision problems that can be solved by non-deterministic polynomial algorithms?
a)**NP** b) P c) Hard d) Complete.
10. Which of the following is true
a) **P is subset of NP** b) NP is subset of P
c) P and NP are equal d) NP is subset of NP hard

PART – B (4 X 4 = 16 Marks)

Answer **ANY FOUR** questions

11. Define Following term : E-node, Dead Node, Live Node
12. Define the Sum of subset problem.
13. Compare backtracking with branch and bound.
14. How NP Hard differ from NP Complete
15. Differentiate Randomized algorithm and Deterministic Algorithm.

PART – C (2 X 12 = 24 Marks)

Answer **ALL** questions

16. a) Solve N Queen Problem 8x8 using Backtracking and state space tree with algorithm
(OR)
b. For given set $w = \{5, 10, 15, 20, 25\}$ and target sum=30, find all possible combination of subsets using state space tree with algorithm.
17. a Define Randomized Algorithm. Explain in detail randomized quick sort algorithm with analysis.
(OR)
b. Explain in detail NP problem , Np Hard and NP Complete



MATURE

PDF

18CSC204J / DESIGN AND ANALYSIS OF ALGORITHMS

UNIT-4

Prepared by,
S.Sridhar, AP/CSE,
SRMIST-VDP Campus

Backtracking and Branch- and- Bound

- Usually for problems with high complexity
- Exhaustive Search is too time consuming
- Cut down on some search using special methods
- Idea: Construct partial solutions and extend
- Smart method to extend partial solutions can lead to faster solutions
- If the current partial solution cannot lead to a full solution, prune
- If the current solution is worse than some earlier known solution, prune
- This approach makes it possible to sometimes solve large problems in reasonable time
- Worst case is still too much time

Difference between backtracking & Branch and Bound

- **Backtracking**
 - [1] It is used to find all possible solutions available to the problem.
 - [2] It traverse tree by DFS(Depth First Search).
 - [3] It realizes that it has made a bad choice & undoes the last choice by backing up.
 - [4] It search the state space tree until it found a solution.
 - [5] It involves feasibility function.
- **Branch-and-Bound (BB)**
 - [1] It is used to solve optimization problem.
 - [2] It may traverse the tree in any manner, DFS or BFS.
 - [3] It realizes that it already has a better optimal solution than the pre-solution leads to so it abandons that pre-solution.
 - [4] It completely searches the state space tree to get optimal solution.
 - [5] It involves bounding function

Backtracking Method

- A given **problem** has a set of constraints and possibly an objective function
- The **solution** optimizes an objective function, and/or is feasible.
- We can represent the **solution space** for the problem using a **state space tree**
 - The *root* of the tree represents **0 choices**,
 - Nodes at depth 1 represent **first choice**
 - Nodes at depth 2 represent the **second choice**, etc.
 - In this tree *a path* from a root to a leaf represents a **candidate solution**

Note:

- The backtracking algorithm has the ability to yield the same answer as brute force algorithm with far fewer than m-trials.

Backtracking Terminology

EXPLICIT CONSTRAINTS are rules which restrict the values of x_i .

Examples $x_i \geq 0$ or $x_1 = 0$ or 1 or $l_i \leq x_i \leq u_i$.

IMPLICIT CONSTRAINTS describe the way in which the x_i must relate to each other .

- For E.g In 0/1 Knapsack Problem the *explicit constraints refer to the placement of item in sack and should be either 0 or 1 (i.e) $X_i = 0$ or 1 where i represents the Item number*
- *Implicit constraints represents that placement of items should not exceed the sack capacity*

Backtracking Terminology

Tuples that satisfy the explicit constraints define a **solution space**.

The solution space can be organized into a tree.

Each node in the tree defines a **problem state**.

All paths from the root to other nodes define the **state-space** of the problem.

Solution states are those states leading to a tuple in the solution space.

Answer nodes are those solution states leading to an answer-tuple(i.e. tuples which satisfy implicit constraints).

Backtracking Terminology

LIVE NODE A node which has been generated and all of whose children are not yet been generated .

E-NODE (Node being expanded) - The live node whose children are currently being generated .

DEAD NODE - A node that is either not to be expanded further, or for which all of its children have been generated

DEPTH FIRST NODE GENERATION- In this, as soon as a new child C of the current E-node R is generated, C will become the new E-node.

Backtracking Terminology

BOUNDING FUNCTION - will be used to kill live nodes without generating all their children.

BACTRACKING-is depth – first node generation with bounding functions.

BRANCH-and-BOUND is a method in which E-node remains E-node until it is dead.

Backtracking Terminology

BREADTH-FIRST-SEARCH : Branch-and Bound with each new node placed in a queue .The front of the queue becomes the new E-node.

DEPTH-SEARCH (D-Search) : New nodes are placed in to a stack.The last node added is the first to be explored.

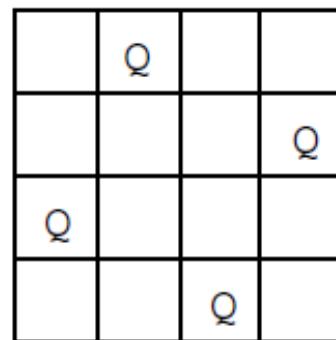
For Video Lecture : <https://tinyurl.com/yb5deykj>

Certain problems which are solved using backtracking method are,

- 1. Sum of subsets.**
- 2. Graph coloring.**
- 3. Hamiltonian cycle.**
- 4. N-Queens problem**

N-Queens problem

- Place n queens on an n -by- n chess board so that no two of them ‘attack’ each other, that is, no two of them are in the same row, same column or same ‘diagonal’.
- Can consider Chess Board as $n \times n$ matrix. So, if two queens are at (i, j) and (i', j') , then we want $i \neq i'$, $j \neq j'$ and $i - i' \neq j - j'$.
- Ex.: $n = 4$:



The conditions to test whether two queens are on the same diagonal

Observe that

- i) For the elements in the upper left to lower Right diagonal, the row - column values are same or $\text{row} - \text{column} = 0$, e.g. $1-1=2-2=3-3=4-4=0$

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

- ii) For the elements in the upper right to the lower left diagonal, row + column value is the same e.g. $1+4=2+3=3+2=4+1=5$

Thus two queens are placed at positions (i, j) and (k, l) , then they are on the same diagonal only if

- $i - j = k - l$ or
- $i + j = k + l$

Two queens lie on the same diagonal if and only if

- $|j - l| = |i - k|$

N Queen Algorithm

Algorithm: Queen-Place(k,i)

- Where k=queen k and i is column number in which queen k is placed.
- Queen-Place(k,i) returns true if a queen can be placed in the kth row and ith column otherwise returns false.
- ABS(r)returns the absolute value of r.

Steps:

```
1. For j←1 to K-1 do  
    if x[j]=i or ABS (x[j]-i)= ABS(j-k)then  
        return false  
2. Return true
```

Algorithm: N-Queen(k,n)

- Where x[] is a global array whose first k-1 values have been set.

Steps:

```
1. For i ← 1 to n do  
    if Queen-Place(k,i) then  
        x[k] ← i  
        if k=n then  
            write (x[1....n])  
        else  
            N- Queen(k+1,n)  
2. return
```

Analysis: As a result in worst case time complexity of the algorithm is $O(n^n)$.

For Video Lecture : <https://tinyurl.com/tarrx2v>

GRAPH COLOURING PROBLEM

Let G be a graph and m be a positive integer .

The problem is to color the vertices of G using only m colors in such a way that no two adjacent nodes / vertices have the same color.

It is necessary to find the smallest integer m . m is referred to as the chromatic number of G .

GRAPH COLOURING PROBLEM (Contd..)

A map can be transformed into a graph by representing each region of map into a node and if two regions are adjacent, then the corresponding nodes are joined by an edge.

For many years it was known that 5 colors are required to color any map.

After several hundred years, mathematicians with the help of a computer showed that 4 colours are sufficient.

Solving the Graph Colouring Problems

The graph is represented by its adjacency matrix Graph (1:n,1:n) where GRAPH (i,j) = true if $\langle i,j \rangle$ is an edge and Graph (i,j) = false otherwise.

The colours will be represented by the integers 1,2,...,m and the solution with n-tuple $(X(1),\dots,X(n))$, where $X(i)$ is the colour of node i.

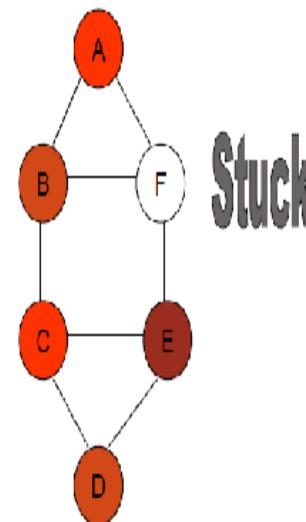
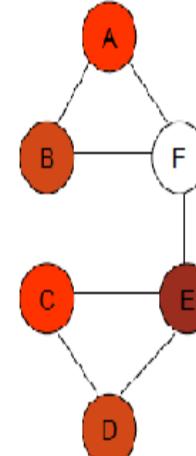
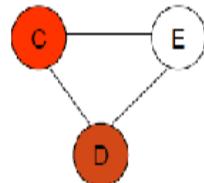
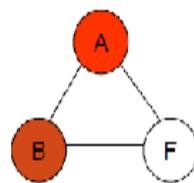
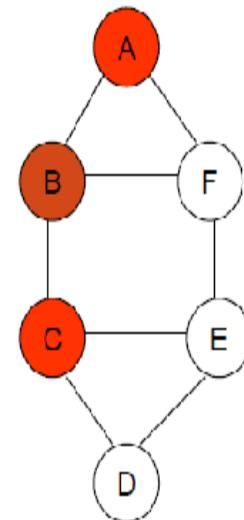
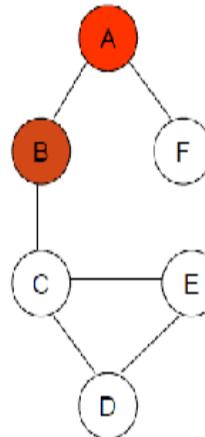
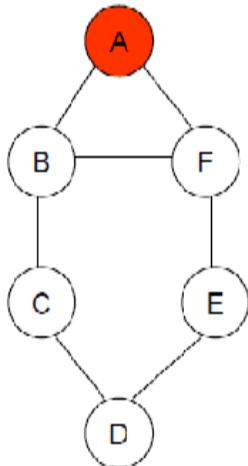
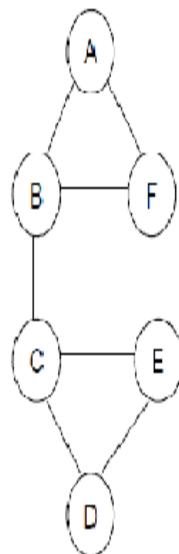
The solution can be represented as a state space tree.

Each node at level i has m children corresponding to m possible assignments to $X(i)$
 $1 \leq i \leq m$.

Nodes at level $n+1$, are leaf nodes. The tree has degree m with height $n+1$.

Graph Coloring

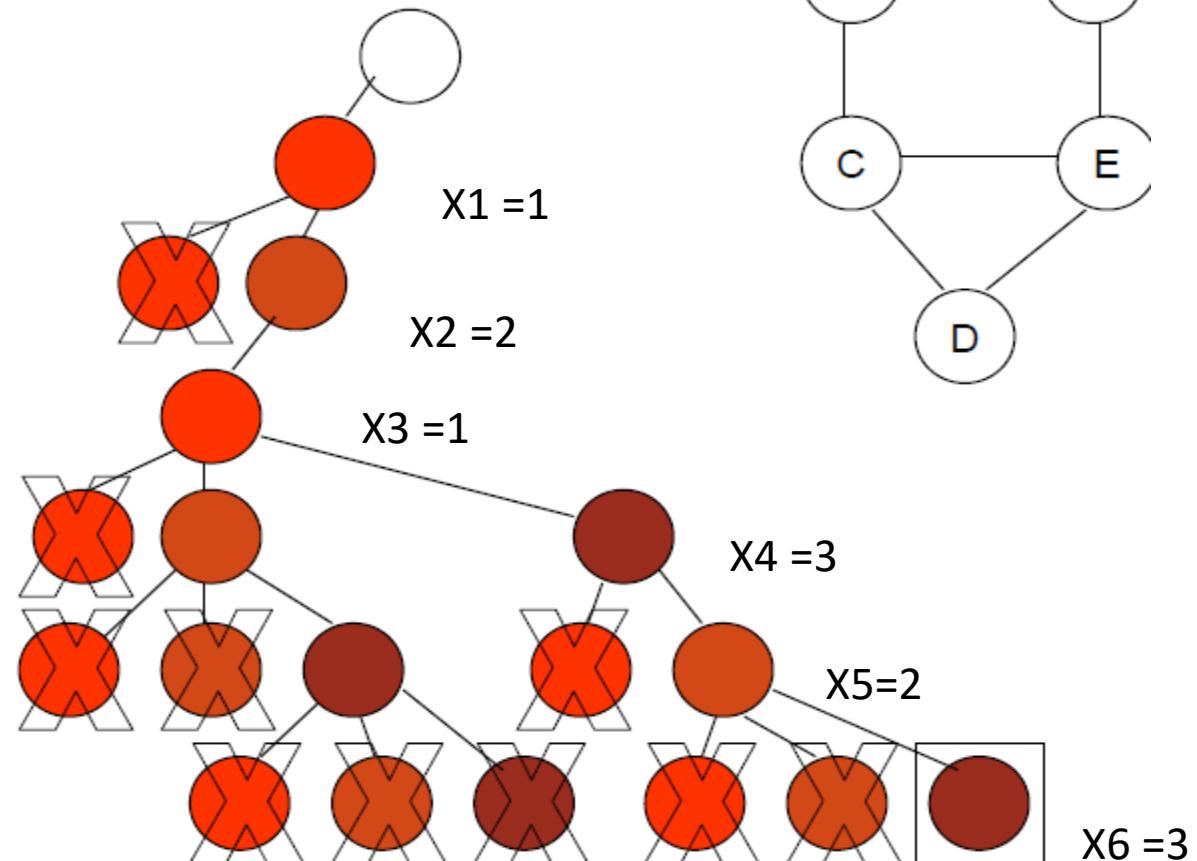
- As an example:
 - The vertices are enumerated in order A-F
 - The colors are given in order: R, G, B



STATE SPACE TREE

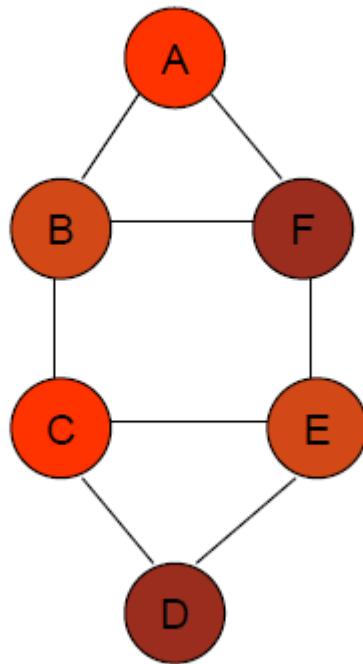
Graph Coloring

A
B
C
D
E
F



X1,x2,x3,...,x6 are nodes in Graph and 1,2,3 represents color

Solution



Time Complexity = $O(nm^n)$ //where n=no. of vertex, m=no. of color used

For Video Lecture : <https://tinyurl.com/twwt3je>

Algorithm mColoring(k)

```
// the graph is represented by its Boolean adjacency matrix G[1:n,1:n] .  
//All assignments //of 1,2,.....,m to the vertices of the graph such // th  
adjacent vertices are assigned //distinct integers are printed.  
//'k' is the index of the next vertex to color.  
{  
  
repeat  
{  
  
// generate all legal assignment for X[k].  
  
Nextvalue(k); // Assign to X[k] a legal color.  
  
If (X[k]=0) then return; // No new color possible.  
  
If (k=n) then // Almost 'm' colors have been used to color the 'n'  
vertices  
  
Write(x[1:n]);  
  
Else mcoloring(k+1);  
  
}until(false);  
}
```

Algorithm Nextvalue(k)

// $X[1], \dots, X[k-1]$ have been assigned integer values in the range $[1, m]$
such that //adjacent values have distinct integers. A value for $X[k]$ is
determined in the //range $[0, m]$. $X[k]$ is assigned the next highest
numbers color while maintaining //distinctness from the adjacent
vertices of vertex K. If no such color exists, then $X[k]$ is 0.

{

repeat

{

$X[k] = (X[k] + 1) \bmod (m + 1)$; // next highest color.

If ($X[k] = 0$) then return; //All colors have been used.

For $j=1$ to n do

{

// Check if this color is distinct from adjacent color.

If (($G[k, j] \neq 0$) and ($X[k] = X[j]$))

// If (k, j) is an edge and if adjacent vertices have the same color.

Then break;

}

if ($j=n+1$) then return; //new color found.

} until (false); //otherwise try to find another color.

}

- The time spent by Nextvalue to determine the children is $O(mn)$
- Total time is = $O(m^n n)$

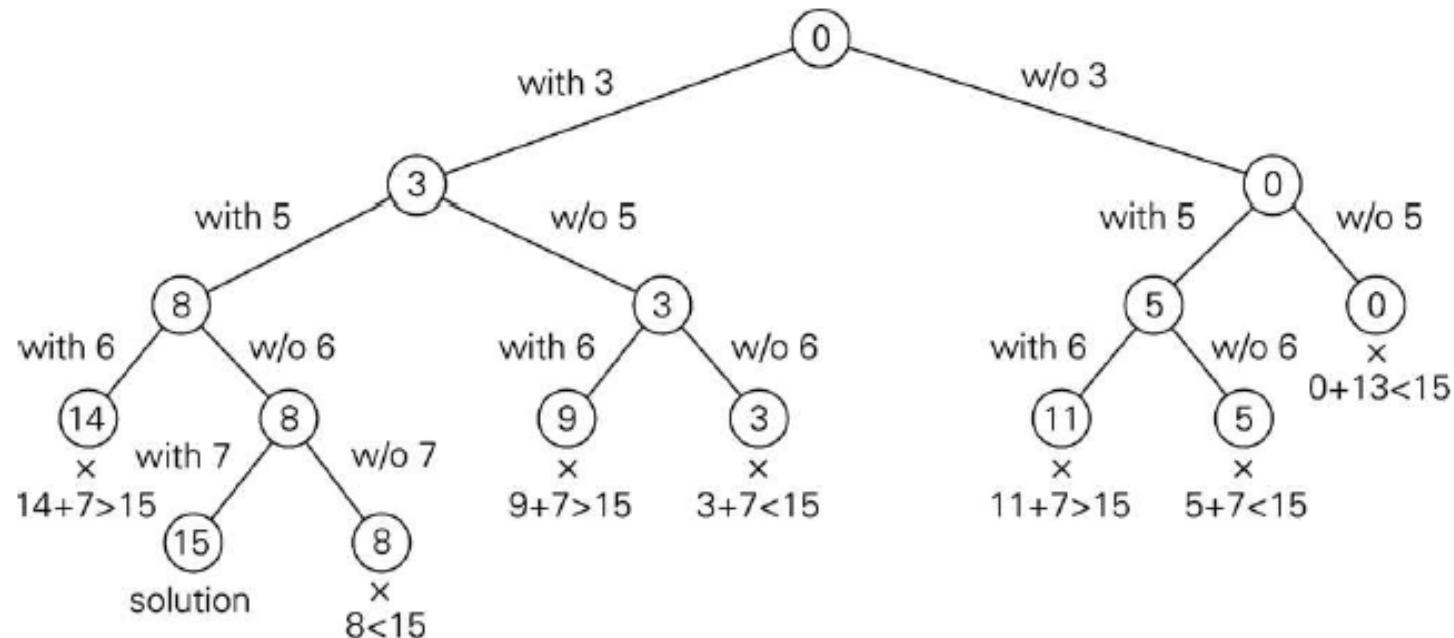
Subset Sum problem

- Problem: Given a set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers. Find a subset of S whose sum is d . Example: $S = \{1, 5, 7, 4, 3\}$, $d = 12$, then answer could be $\{5, 7\}$ or $\{4, 3, 5\}$.
- Root can be empty subset.
- Nodes at the ℓ -th level decide whether to add/not add s_i to the sum.
- If current sum $> d$, then the sum is too high, and any further search in this branch can be pruned.
- If current sum plus sum of remaining numbers is $< d$, then sum is too small, and any further search in this branch can be pruned.
- If $sum = d$, then done.

For Video Lecture : <https://tinyurl.com/wx2gplu>

Subset-Sum Problem

- Example: $S = \{3, 5, 6, 7\}$, $d = 15$.



Sum of Subsets Algorithm

```
void SumOfSub(float s, int k, float r)
```

```
{
```

```
    // Generate left child.
```

```
    x[k] = 1;
```

```
    if (s+w[k] == m)
```

```
    {   for (int j=1; j<=k; j++)
```

```
        Print (x[j] )
```

```
}
```

```
else if (s+w[k]+w[k+1] <= m)
```

```
    SumOfSub(s+w[k], k+1, r-w[k]);
```

```
// Generate right child and evaluate
```

```
if ((s+r-w[k] >= m) && (s+w[k+1] <= m)) {
```

```
    x[k] = 0;
```

```
    SumOfSub(s, k+1, r-w[k]);
```

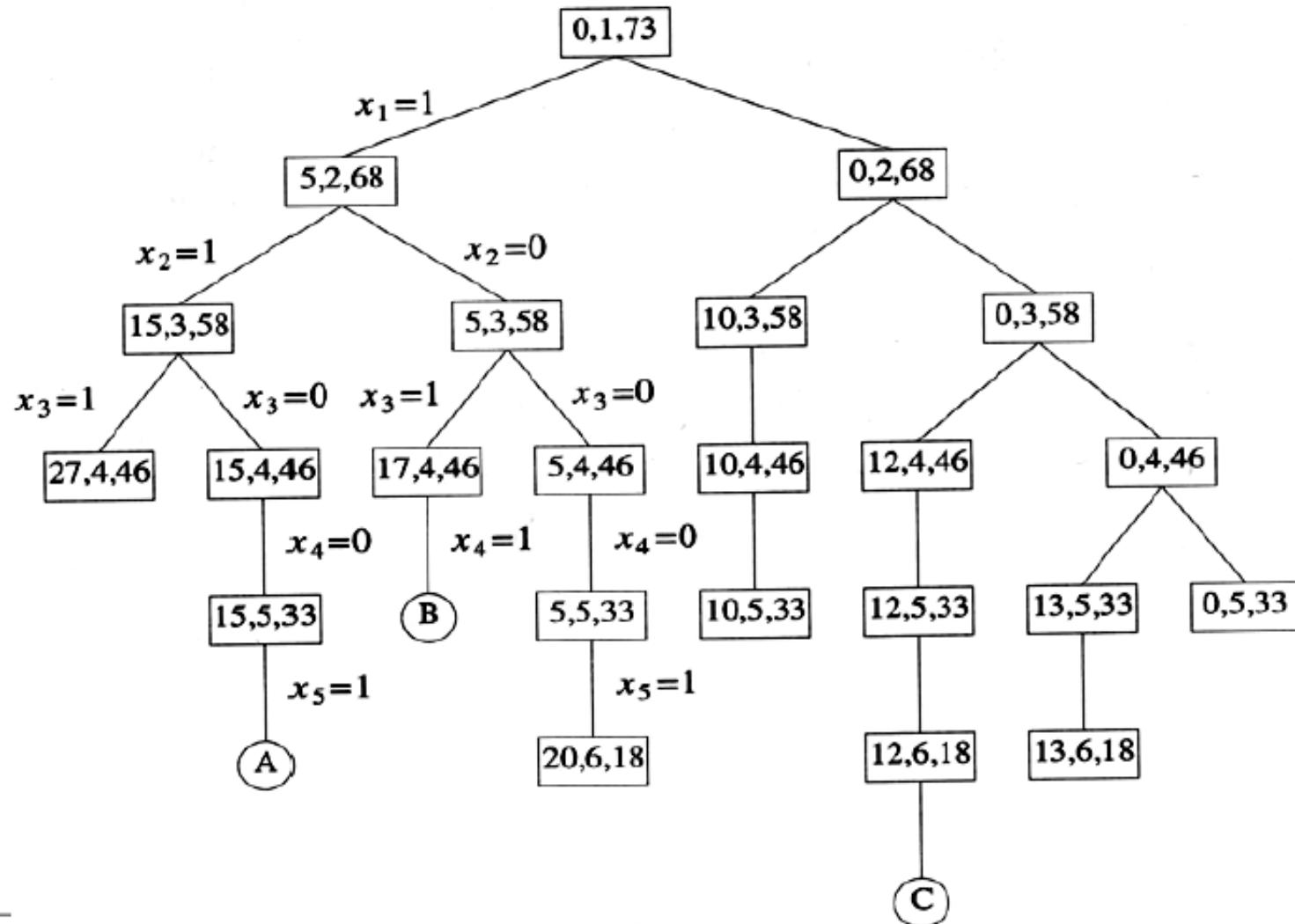
```
}
```

```
}
```

Time Complexity = O (2ⁿ)

Sum of Subsets State Space Tree

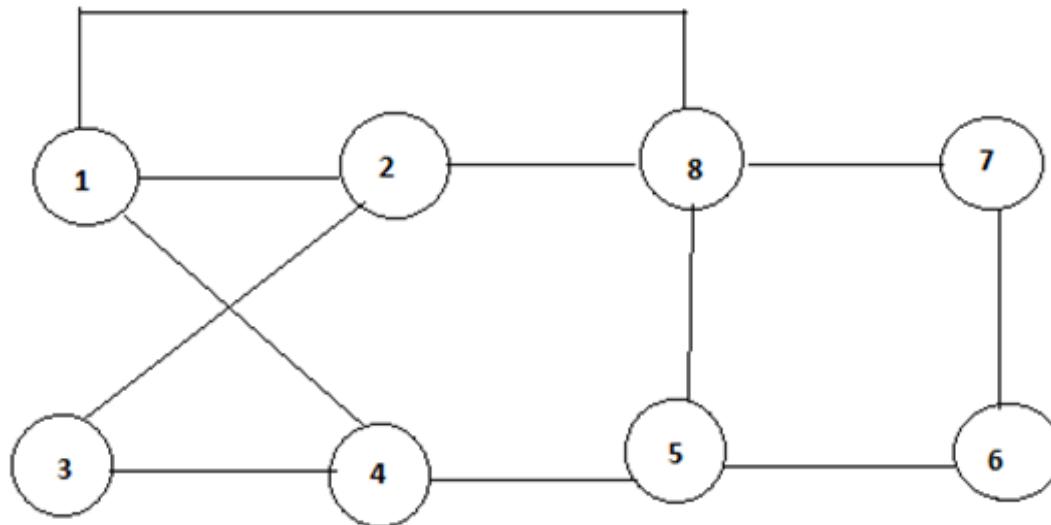
- Example n=6, w[1:6]={5,10,12,13,15,18}, m=30



Hamiltonian Cycle

- A **Hamiltonian** graph is the directed or undirected graph containing a Hamiltonian cycle.
- The **Hamiltonian cycle** is the cycle that traverses all the vertices of the given graph G exactly once and then ends at the starting vertex.
- The input for the Hamiltonian graph problem can be the directed or undirected graph. The Hamiltonian problem involves checking if the Hamiltonian cycle is present in a graph **G** or not

The following graph illustrates the presence of the Hamiltonian cycle in a graph **G**.



hamiltonian path is = 1 -- 2 -- 3 -- 4 -- 5 -- 6 -- 7 -- 8 -- 1

While generating the state space tree following bounding functions are to be considered, which are as follows:

The i^{th} vertex in the path must be adjacent to the $(i-1)^{th}$ vertex in any path.

The starting vertex and the $(n-1)^{th}$ vertex should be adjacent.

The i^{th} vertex cannot be one of the first $(i-1)^{th}$ vertex in the path.

It will take $(n-1)!$ Steps to find the solution roughly equals $n!$ and we can say $n! = O(n^n)$

So, Time Complexity = $O(n^n)$

Algorithm Hamiltonian (k)

```
1.  {
2.  Repeat
3.  {
4.  Next value (k);
5.  If (x[k] == 0) then return;
6.  If ( x[k] == n) then write x[1:n];
7.  Else Hamiltonian (k+1)
8.  }
9.  Until (false);
```

For Video Lecture : <https://tinyurl.com/u3xttbo>

Algorithm Next value (k)

```
1.  {
2.  Repeat
3.  {
4.  X [k] = (x[k] + 1 mod (n+1));
5.  If (x[k] == 0) then return;
6.  If (G (x[k-1]), x[k] != 0) then
7.  {
8.  For j=1 to k-1
9.  Do if ( x[ j ] == x[ k ]) then break;
10. If ( j == k) true
11. If ( k<n) or ( k = n ) and G ( x[ n ], x[1] != 0)
12. }
13. Then return;
14. }
15. Until (false);
16. }
17. }
```

0/1 knapsack problem using Branch and Bound

The 0/1 knapsack problem

- Positive integer P_1, P_2, \dots, P_n (profit)
 W_1, W_2, \dots, W_n (weight)
 M (capacity)

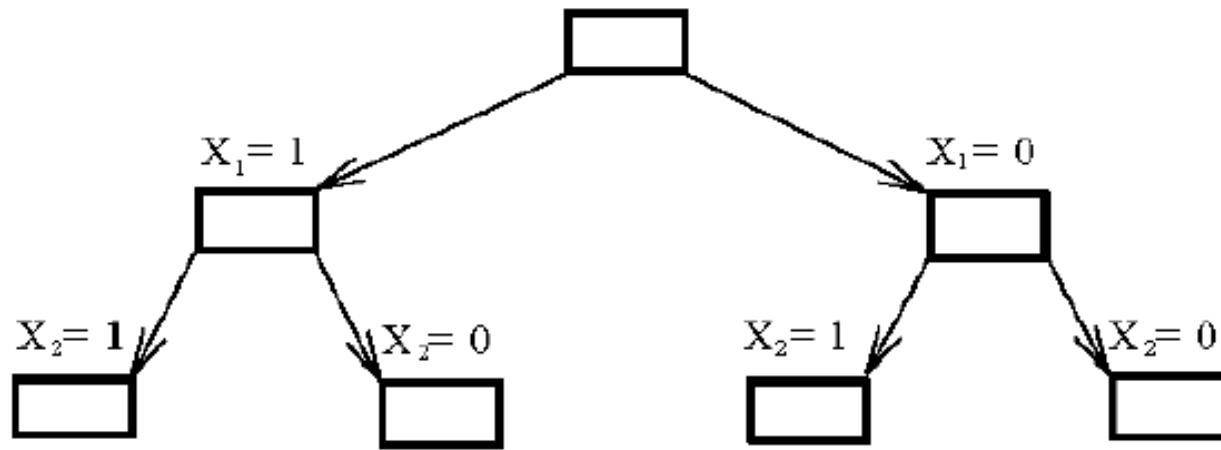
$$\text{maximize } \sum_{i=1}^n P_i X_i$$

$$\text{subject to } \sum_{i=1}^n W_i X_i \leq M \quad X_i = 0 \text{ or } 1, i = 1, \dots, n.$$

The problem is modified:

$$\text{minimize } - \sum_{i=1}^n P_i X_i$$

The 0/1 knapsack problem



The Branching Mechanism in the Branch-and-Bound Strategy to Solve 0/1 Knapsack Problem.

How to find the upper bound?

- Ans: by quickly finding a feasible solution in a **greedy manner**: starting from the smallest available i , scanning towards the largest i 's until M is exceeded. The upper bound can be calculated.

How to find the ranking Function

- Ans: by relaxing our restriction from $X_i = 0$ or 1 to $0 \leq X_i \leq 1$ (knapsack problem)

Let $- \sum_{i=1}^n P_i X_i$ be an optimal solution for 0/1

knapsack problem and $- \sum_{i=1}^n P_i X'_i$ be an optimal solution for **fractional knapsack problem**. Let

$$Y = - \sum_{i=1}^n P_i X_i, Y' = - \sum_{i=1}^n P_i X'_i.$$
$$\Rightarrow Y' \leq Y$$

How to expand the tree?

- By the best-first search scheme
- That is, by expanding the node with the best lower bound. If two nodes have the same lower bounds, expand the node with the lower upper bound.

procedure UBOUND (p , w , k , M)

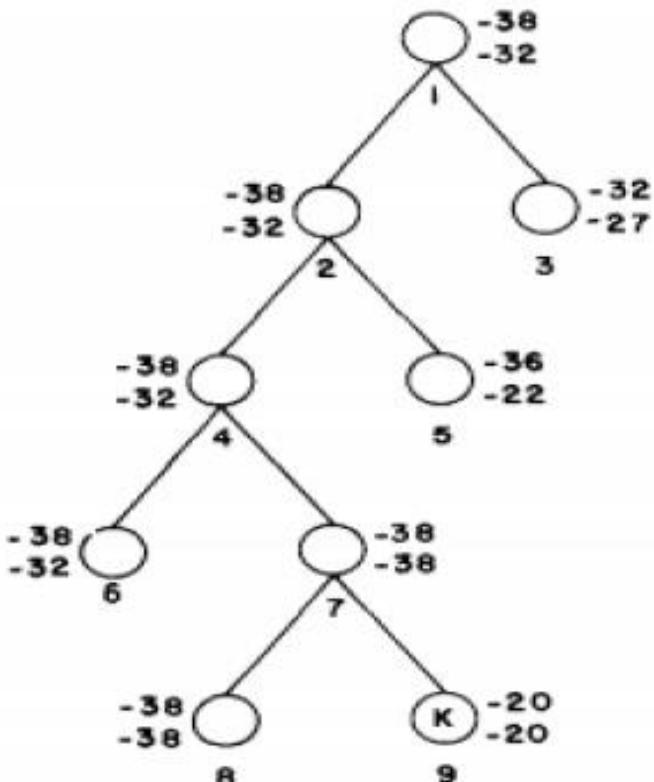
```
//  $W(i)$  and  $P(i)$  are respectively the weight and profit of the  $i$ th object//  
global  $W(1:n)$ ,  $P(1:n)$ ; integer  $i$ ,  $k$ ,  $n$   
 $b \leftarrow p$ ;  $c \leftarrow w$   
for  $i \leftarrow k + 1$  to  $n$  do  
    if  $c + W(i) \leq M$  then  $c \leftarrow c + W(i)$ ;  $b \leftarrow b - P(i)$  endif  
repeat  
    return ( $b$ )  
end UBOUND
```

0/1 Knapsack Example using LCBB (Least Cost)

- Example (LCBB)
- Consider the knapsack instance:
- $n = 4$;
- $(p_1, p_2, p_3, p_4) = (10, 10, 12, 18)$;
- $(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$ and
- $M = 15$.

For Video Lecture : <https://tinyurl.com/qmbmcfz>

0/1 Knapsack State Space tree of Example using LCBB



Time Complexity = O (2ⁿ)

Upper number = c
Lower number = u

Traveling Salesman problem (TSP) using Branch and Bound

The traveling salesperson problem

- Given a graph, the TSP Optimization problem is to find a tour, starting from any vertex, visiting every other vertex and returning to the starting vertex, with **minimal** cost.

Time Complexity = $O(n^2 2^n)$ using Dynamic Programming algorithms.

The worst case of branch and bound algorithms will not be better than $O(n^2 2^n)$.

*But use of **good bound functions** enables branch and bound algorithms to solve same problem in much less time than required by Dynamic Programming algorithms.*

Example- TSP

- Example with Cost Matrix(a) and its Reduced Cost Matrix (b)
- Reduced matrix means every row and column of matrix should contain at least one Zero and all other entries should be non negative.

∞	20	30	10	11
15	∞	16	4	2
3	5	∞	2	4
19	6	18	∞	3
16	4	7	16	∞

(a) Cost Matrix

∞	10	17	0	1
12	∞	11	2	0
0	3	∞	0	2
15	3	12	∞	0
11	0	0	12	∞

(b) Reduced Cost Matrix
 $L = 25$

For Video Lecture : <https://tinyurl.com/u4yteyf>

Reduced Matrix for node 2,3...10 of State Space tree using LC Method

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 12 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

a) path 1,2; node 2

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & 2 & 0 \\ \infty & 3 & \infty & 0 & 2 \\ 4 & 3 & \infty & \infty & 0 \\ 0 & 0 & \infty & 12 & \infty \end{bmatrix}$$

b) path 1,3; node 3

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & 12 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

c) path 1,4;node 4

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 12 & 0 & 9 & \infty & \infty \\ \infty & 0 & 0 & 12 & \infty \end{bmatrix}$$

d) path 1,5; node 5

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

e) path 1,4,2; node 6

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & 1 & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & 0 & \infty & \infty & \infty \end{bmatrix}$$

f) path 1,4,3; node 7

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty \end{bmatrix}$$

g) path 1,4,5; node 8

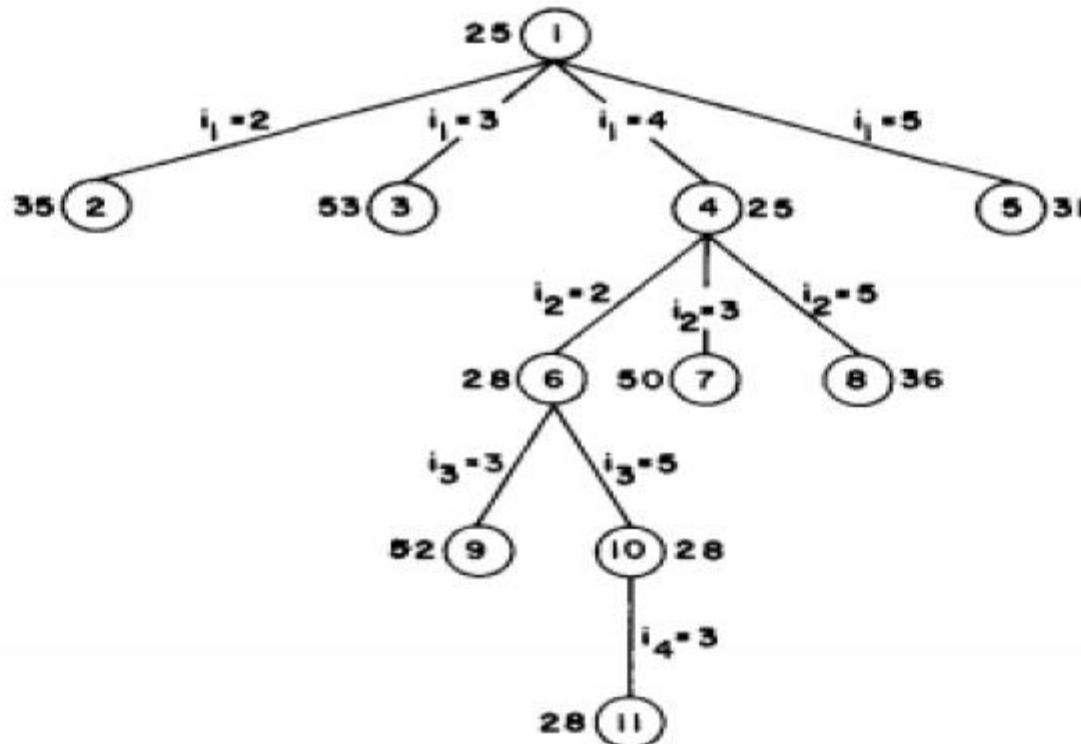
$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

h) path 1,4,2,3; node 9

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \end{bmatrix}$$

i) path 1,4,2,5; node 10

State Space tree of Example using LC Method



Numbers outside the node are C values

A COMPARISON

- A [**Greedy approach**](#) is to pick the items in decreasing order of value per unit weight. The Greedy approach works only for [fractional knapsack](#) problem and may not produce correct result for [0/1 knapsack](#).
- We can use [**Dynamic Programming \(DP\)**](#) for [**0/1 Knapsack problem**](#). In DP, we use a 2D table of size $n \times W$. The **DP Solution doesn't work if item weights are not integers**.
- Since DP solution doesn't always work, a solution is to use **Brute Force**. With n items, there are 2^n solutions to be generated, check each to see if they satisfy the constraint, save maximum solution that satisfies constraint. This solution can be expressed as **tree**.
- We can use **Backtracking** to optimize the Brute Force solution. In the tree representation, we can do DFS of tree. If we reach a point where a solution no longer is feasible, there is no need to continue exploring.

Shortest Path Algorithm

- The shortest path problem is about finding a path between 2 vertices in a graph such that the total sum of the edges weights is minimum.
- This problem could be solved easily using **(BFS)** if all edge weights were 1), but in practical applications weights can take any value.

Floyd–Warshall's Algorithm

- Floyd–Warshall's Algorithm is used to find the shortest paths between all pairs of vertices in a graph, where each edge in the graph has a weight which is positive or negative. The biggest advantage of using this algorithm is that all the shortest distances between any 2 vertices could be calculated in $O(V^3)$, where V is the number of vertices in a graph.

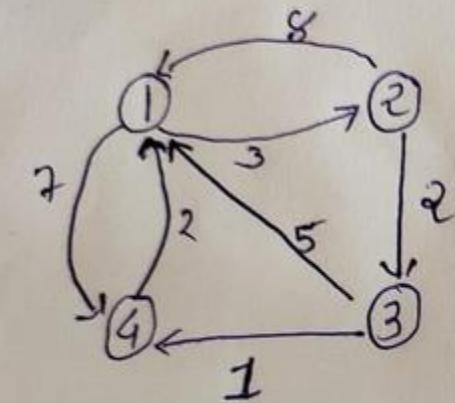
The Algorithm Steps:

- For a graph with N vertices:
- Initialize the shortest paths between any 2 vertices with Infinity.
- Find all pair shortest paths that use 0 intermediate vertices, then find the shortest paths that use 1 intermediate vertex and so on.. until using all N vertices as intermediate nodes.
- Minimize the shortest paths between any 2 pairs in the previous operation.
- For any 2 vertices (i, j) , one should actually minimize the distances between this pair using the first K nodes, so the shortest path will be:
 $\min (\text{dist}[i][k] + \text{dist}[k][j], \text{dist}[i][j])$.

`dist[i][k]` represents the shortest path that only uses the first `K` vertices, `dist[k][j]` represents the shortest path between the pair `k, j`. As the shortest path will be a concatenation of the shortest path from `i` to `k`, then from `k` to `j`.

```
for(int k = 1; k <= n; k++) {
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            dist[i][j] = min( dist[i][j], dist[i][k] + dist[k][j] );
        }
    }
}
```

Time Complexity of Floyd–Warshall's Algorithm is $O(V^3)$, where V is the number of vertices in a graph.



$$A^o = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix}$$

$$A' = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & \infty & 0 \end{bmatrix}$$

$$\rightarrow A^o[2,3] \quad A^o[2,1] + A^o[1,3] \quad \left| \begin{array}{l} \rightarrow A^o[2,4] \cdot A^o[2,1] + A^o[1,4] \\ \infty \rightarrow \frac{8+7}{15} \\ \text{include in } A^o[2,4] \end{array} \right.$$

$$A^2 = \begin{matrix} 1 & 2 & 3 & 4 \\ 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & 7 & 0 \end{matrix}$$

$$A^3 = \begin{matrix} 1 & 2 & 3 & 4 \\ 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{matrix}$$

$$A^4 = \begin{matrix} 1 & 2 & 3 & 4 \\ 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{matrix}$$

For Video Lecture : <https://tinyurl.com/vsp4ulb>

After including Second, Third and Four node one by one we get final matrix A^4 which show possible minimum distance between any two nodes.

Question Bank

1. What is m-colorability optimization?
2. Define sum of subsets problem.
3. What is chromatic numbers?
4. Define Backtracking.
5. What are the applications of backtracking?
6. Define n-queens problem.
7. Define Hamiltonian Circuit problem in an undirected Graph.
8. What is state space tree?
9. State the principle of Backtracking.
10. Define depth first searching techniques.
11. Define all-pair shortest path problem.
12. Compare Backtracking & Branch and Bound techniques with an example.

PART-B

1. What is Backtracking? Explain in detail.
2. Explain Subset-sum Problem & Discuss the possible solution strategies using backtracking.
3. Discuss the use of greedy method in solving knapsack problem and subset sum problem.
4. Write short notes on
 - (a) Graph coloring
 - (b) 8-Queens problem
5. Apply Backtracking technique to solve the following instance of the subset sum problems. $s=(1,3,4,5)$ & $d=11$
6. Explain subset-sum problem and discuss the possible solution strategies using backtracking.
7. Using Backtracking enumerate how can you solve the following problems
 - (a) 8-queens problem
 - (b) Hamiltonian circuit problem

8. Solve the all-pairs shortest path problem for the digraph with the weight matrix given below.

	A	B	C	D
A	0	∞	∞	3
B	2	0	∞	∞
C	∞	7	0	1
D	6	∞	∞	0

9. What is branch and bound technique? Explain how knapsack problem could be solved using branch and bound technique. Solve the following instance of the knapsack problem by branch and bound algorithm for W=16

Item	Weight	Value in Rs.
1	10	100
2	7	63
3	8	56
4	4	12

10. Give a suitable example and explain the birth first search and depth first search algorithm.
11. Solve the following TSP problem by using Branch and Bound Method

addresses	1	2	3	4	5
1	oo	600	1000	1900	1100
2	600	oo	1900	1900	1500
3	1000	1900	oo	1700	1200
4	1900	1900	1700	oo	1900
5	1100	1500	1200	1900	oo

Randomized Quick Sort

Quick Sort

Divide: Partition the array into two sub-arrays

$A[p \dots q-1]$ and $A[q+1 \dots r]$ such that each element of

$A[p \dots q-1]$ is less than or equal to $A[q]$, which in turn

less than or equal to each element of $A[q+1 \dots r]$

Quick Sort

Conquer: Sort the two sub-arrays $A[p \dots q-1]$ and

$A[q+1 \dots r]$ by recursive calls to quick sort.

Combine: Since the sub-arrays are sorted in place, no

work is needed to combine them.

Quick Sort

Worst-case partitioning:

The recurrence for the running time

$$T(n) = T(n-1) + T(0) + \theta(n)$$

$$= T(n-1) + \theta(n)$$

$$= \dots \theta(n^2)$$

The $\theta(n^2)$ running time occurs when the input array is already completely sorted

A Randomized Version of Quick Sort

In the randomized version of Quick sort we impose a distribution on input by picking the pivot element randomly.

Randomized Quick Sort works well even when the array is sorted/reversely sorted and the complexity is more towards $O(n \log n)$.

(Yet, there is still a possibility that the randomly picked element is always an extreme.)

Algorithm RANDOMIZEDQUICKSORT

Input: An array $A[1..n]$ of n elements.

Output: The elements in A sorted in nondecreasing order.

1. $rquicksort(1, n)$

Procedure $rquicksort(low, high)$

1. if $low < high$ then
2. $v \leftarrow \text{random}(low, high)$
3. interchange $A[low]$ and $A[v]$
4. SPLIT($A[low..high]$, w) $\{w \text{ is the new position of the pivot}\}$
5. $rquicksort(low, w - 1)$
6. $rquicksort(w + 1, high)$
7. end if

If the pivot is selected uniformly at random, the expected number of comparisons of this randomized version of Quicksort is bounded by $O(n \log n)$.

To analyze the *total number of comparisons let us take the total number of comparisons as a sum of simpler random variables, and then just analyze the simpler ones.*

Let $a = (a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_n)$ denote the list of elements we want to sort, *in sorted order*. Note that, *for the analysis* we may assume that we know the order. The input is any permutation of a . For all $1 \leq i < j \leq n$, let $X_{i,j}$ denote the random variable indicating whether Quicksort compared a_i and a_j .

$$X_{i,j} = \begin{cases} 1 & \text{if Quicksort compares } a_i \text{ and } a_j \\ 0 & \text{otherwise} \end{cases}$$

Any two elements get compared at most once. The expected number of comparisons is thus

$$E \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{i,j} \right] = E \left[\sum_{1 \leq i < j \leq n} X_{i,j} \right] = \sum_{1 \leq i < j \leq n} E[X_{i,j}]$$

In other words, for a given element i , it is compared to element $i + 1$ with probability 1, to element $i + 2$ with probability $2/3$, to element $i + 3$ with probability $2/4$, to element $i + 4$ with probability $2/5$ and so on

Quicksort compares a_i and a_j if and only if either a_i or a_j is selected as pivot *before* any of the elements between a_i and a_j . Selecting an element between a_i and a_j as pivot will partition the list into two parts such that a_i and a_j lie in different parts and do not get compared. The number of elements between a_i and a_j is $j - i - 1$. Since in each step the pivot element is selected uniformly at random, the probability that either a_i or a_j is selected as pivot before any of the $j - i - 1$ elements between them is $2/(j - i + 1)$.

We thus have

$$E[X_{i,j}] = \frac{2}{j-i+1}.$$

We derive

$$\begin{aligned} E\left[\sum_{i=1}^n \sum_{j=i+1}^n X_{i,j}\right] &= \sum_{i=1}^n \sum_{j=i+1}^n E[X_{i,j}] \\ &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &\leq 2 \sum_{i=1}^n \sum_{k=1}^n \frac{1}{k} \\ &= 2n \sum_{k=1}^n \frac{1}{k} \\ &= O(n \ln n). \end{aligned}$$

LINEARITY OF EXPECTATION basically says that the expected value of a sum of random variables is equal to the sum of the individual expectations.

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n \mathbb{E}[X_i].$$

STRING MATCHING ALGORITHMS

The fundamental string searching (matching) problem is defined as follows: given two strings – a text and a pattern, determine whether the pattern appears in the text.

The “Naive” Method

Its idea is straightforward — for every position in the text, consider it a starting position of the pattern and see if you get a match.

The “naive” approach is easy to understand and implement but it can be too slow in some cases. If the length of the text is n and the length of the pattern m , in the worst case it may take as much as **($n * m$)** iterations to complete the task.

Naïve Algorithm: An Example

Pattern: abaa; searched string: ababbaabaaaab

ababbaabaaaab		
abaa.....	step 1	ABA#	mismatch: 4th letter
_abaa.....	step 2	_#...	mismatch: 1st letter
__abaa.....	step 3	__AB#.	mismatch: 3rd letter
___abaa.....	step 4	___#...	mismatch: 1st letter
____abaa....	step 5	____#...	mismatch: 1st letter
_____abaa...	step 6	_____A#...	mismatch: 2nd letter
_____abaa_...	step 7	_____ABAA	success
_____abaa_...	step 8	_____#...	mismatch: 1st letter
_____abaa	step 9	_____.#..	mismatch: 2nd letter

Rabin-Karp algorithm

Rabin-Karp algorithm is an algorithm used for searching/matching patterns in the text using a hash function.

Unlike Naive string matching algorithm, it does not travel through every character in the initial phase rather it filters the characters that do not match and then performs the comparison

A hash function is a tool to map a larger input value to a smaller output value. This output value is called the hash value.

How Rabin-Karp Algorithm Works?

A sequence of characters is taken and checked for the possibility of the presence of the required string. If the possibility is found then, character matching is performed.

Let us understand the algorithm with the following steps:

1. Let the text be:

A | B | C | C | D | D | A | E | F | G

And the string to be searched in the above text be:

C | D | D

2. Let us assign a numerical value(v)/weight for the characters we will be using in the problem. Here, we have taken first ten alphabets only (i.e. A to J).

A	B	C	D	E	F	G	H	I	J
1	2	3	4	5	6	7	8	9	10

3. m be the length of the pattern and n be the length of the text. Here,

$$m = 10 \text{ and } n = 3.$$

Let d be the number of characters in the input set. Here, we have taken input set {A, B, C, ..., J}. So, $d = 10$. You can assume any suitable value for d.

4. Let us calculate the hash value of the pattern.



hash value = 6

$$\begin{aligned}\text{hash value for pattern}(p) &= \sum(v * d^{m-1}) \bmod 13 \\ &= ((3 * 10^2) + (4 * 10^1) + (4 * 10^0)) \bmod 13 \\ &= 344 \bmod 13 \\ &= 6\end{aligned}$$

In the calculation above, choose a prime number (here, 13) in such a way that we can perform all the calculations with single-precision arithmetic.

5. Calculate the hash value for the text-window of size m .

For the first window ABC,

$$\begin{aligned}\text{hash value for text}(t) &= \sum(v * d^{n-1}) \bmod 13 \\ &= ((1 * 10^2) + (2 * 10^1) + (3 * 10^0)) \bmod 13 \\ &= 123 \bmod 13 \\ &= 6\end{aligned}$$

6. Compare the hash value of the pattern with the hash value of the text. If they match then, character-matching is performed.

In the above examples, the hash value of the first window (i.e. t) matches with p so, go for character matching between ABC and CDD. Since they do not match so, go for the next window.

7. We calculate the hash value of the next window by subtracting the first term and adding the next term as shown below.

$$\begin{aligned}t &= (123 - (1 * 10^2)) * 10 + (3 * 10^0) \bmod 13 \\&= 233 \bmod 13 \\&= 12\end{aligned}$$

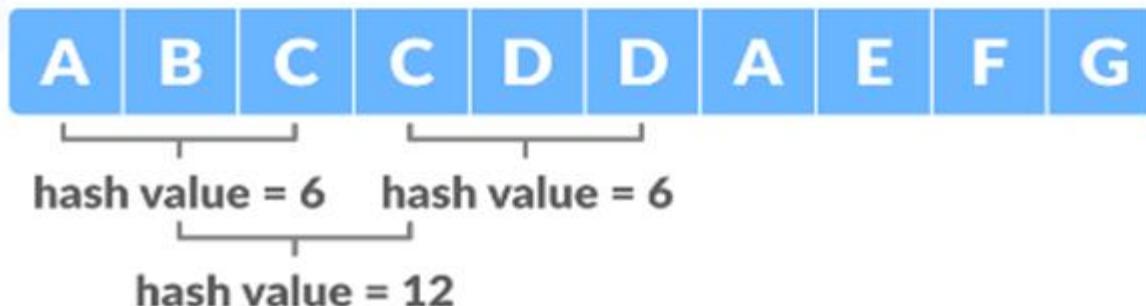
OR

$$\begin{aligned}t &= ((d * (t - v[\text{character to be removed}] * h) + v[\text{character to be added}]) \bmod 13 \\&= ((10 * (6 - 1 * 9) + 3) \bmod 13 \\&= 12\end{aligned}$$

$$\text{Where, } h = d^{m-1} \bmod 13 = 10^{3-1} \bmod 13 = 100 \bmod 13$$

8. For BCC, $t = 12 \neq 6$. Therefore, go for the next window.

After a few searches, we will get the match for the window **CDD** in the text.



Algorithm

```
n = t.length
m = p.length
h = dm-1 mod q
p = 0
t0 = 0
for i = 1 to m
    p = (dp + p[i]) mod q
    t0 = (dt0 + t[i]) mod q
for s = 0 to n - m
    if p = ts
        if p[1.....m] = t[s + 1..... s + m]
            print "pattern found at position" s
    If s < n-m
        ts + 1 = (d (ts - t[s + 1]h) + t[s + m + 1]) mod q
```

Rabin-Karp Algorithm Complexity

The average case and best case complexity of Rabin-Karp algorithm is $O(m + n)$ and the worst case complexity is $O(mn)$.

The worst-case complexity occurs when spurious hits occur a number for all the windows.

Rabin-Karp Algorithm Applications

- For pattern matching
 - For searching string in a bigger text
-

Other Algorithms:

Knuth-Morris-Pratt (over binary alphabet) KMP algorithm.

- Use knowledge of how search pattern repeats itself.
- Build DFA from pattern.
- Run DFA on text.

Boyer-Moore Boyer-Moore algorithm (1974).

Right-to-left scanning.

- find offset i in text by moving left to right.
- compare pattern to text by moving right to left.

Standard Complexity Classes :

Tractable vs Intractable Problems :

- We can distinguish problems between two distinct classes. Problems which can be solved by a polynomial time algorithm and problems for which no polynomial time algorithm is known.
- An algorithm for a given problem is said to be a polynomial time algorithm if its worst case time complexity is $O(n^k)$, where k is a fixed integer and n is size of a problem. For example, Sequential search : $O(n)$, Binary search : $O(\log n)$, Insertion sort : $O(n^2)$, product of two matrices : $O(n^3)$, Quick sort : $O(n \log n)$ etc.

Standard Complexity Classes :

Tractable vs Intractable Problems contd... :

- The set of all problems that can be solved in polynomial amount of time are called “**Tractable Problems**”. These problems can be solved in a reasonable amount of time for even very large amount of input data. Their worst case time complexity is $O(n^k)$.
- The set of all problems that can not be solved in polynomial amount of time are called “**Intractable Problems**”. Their worst case time complexity is $O(k^n)$. These problems require huge amount of time for even modest input sizes. For example, 0-1 Knapsack problem : $O(2^n)$, Traveling Salesperson Problem : $O(n^2 2^n)$

Standard Complexity Classes :

Decision Problem & Optimization Problem / Algorithm :

- **Decision Problem** : Any problem for which answer is either 0 or 1 is called a decision problem and the corresponding algorithm is referred as a decision algorithm. For example : To search a given number
- **Optimization Problem** : Any problem that involves the identification of an optimal (either min. or max.) value of a given cost function is known as a optimization problem and the corresponding algorithm is referred as an optimization algorithm. For example, Knapsack problem, Minimum cost spanning tree.

Standard Complexity Classes :

P vs NP class problems :

- **P class** : The class of decision problems that can be solved in polynomial time using deterministic algorithms is called the P class or Polynomial problems.
- **NP class** : The class of decision problems that can be solved in polynomial time using non-deterministic algorithms is called the NP class or Non-deterministic Polynomial problems.
- Any P class problem can be solved using NP class algorithm. Therefore P is contained in NP class ($P \subseteq NP$)

There are a lot of problems in NP that we do not know how to solve in polynomial time. Why?

Because they really don't have polynomial algorithms? Or,

because we are not yet clever enough to have found a polynomial algorithm for them?

CONCLUSION:

- ✓ All problems in NP are decidable.
- ✓ That means there is an algorithm.
- ✓ And, the algorithm is no worse than $O(2^n)$.

At the moment, no one knows.

Some believe all problems in NP have polynomial algorithms. Many do not (believe that).

The fundamental question in theoretical computer science is:

Does $P = NP$?

There is an award of one million dollars for a proof.

– Either way, True or False.

The "Key" to Complexity Theory 'Reductions,'

"Problem A *reduces* to problem B" or, simply, "A $\not\supseteq$ B"

Theorem. If A $\not\supseteq$ B and problem B is polynomial, then problem A is polynomial.

Corollary. If A $\not\supseteq$ B and problem A is exponential, then problem B is exponential.

A $\not\supseteq$ B means:

'Problem A is *no harder than* problem B,'
and, equivalently,
'Problem B is *as hard as* problem A.'

Standard Complexity Classes :

NP-Complete problems :

- A decision problem D is said to be NP-Complete if
 1. It belongs to NP class
 2. Every problem in NP class is polynomially reducible to D
- If one instance of such problem can be solved using a polynomial algorithm, the complete class of problems can be solved using a polynomial algorithm

Polynomial Transformations enforce an equivalence relationship on all decision problems, particularly, those in the Class NP. Class P is one of those classes and is considered to be the "easiest" class of problems in NP.

Is there a class in NP that is the "hardest" class in NP?

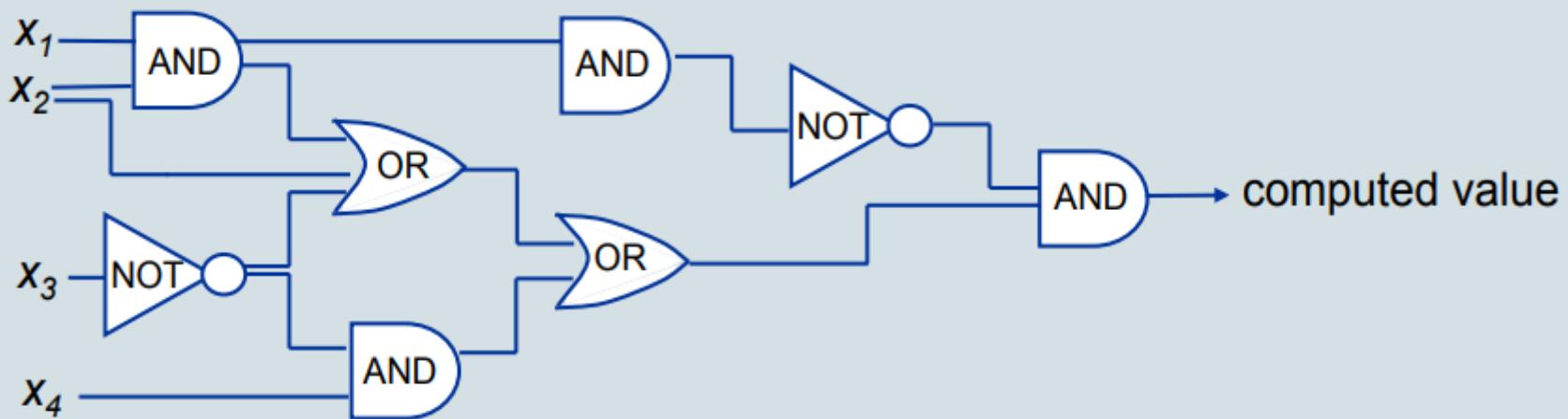
That is, a problem B in NP such that

$A \not\leq_P B$ – for every A in NP.

In 1971, Stephen Cook proved there was. Specifically, a problem called

***Satisfiability*(or, SAT).**

A first NP-complete problem: Circuit-SAT



Circuit-SAT

Input: Boolean combinational circuit

Question: Can variables be set such that circuit evaluates to true ?

Can we assign Boolean values to the variables in U so that every clause is TRUE?

There is no known polynomial algorithm!!

Cooks Theorem:

- 1) SAT is in NP
- 2) For every problem A in NP,

$$A \not\rightarrow_P SAT$$

Thus, SAT is as hard as every problem in NP.

Since SAT is itself in NP, that means SAT is a hardest problem in NP (there can be more than one).

A hardest problem in a class is called the "completion" of that class.

Therefore, SAT is NP–Complete.

Definition: Problem B is NP-Hard if there is a Turing reduction $A \rightarrow B$ for some problem A in NP-Complete.

This implies NP-Hard problems are at least as hard as NP-Complete problems.

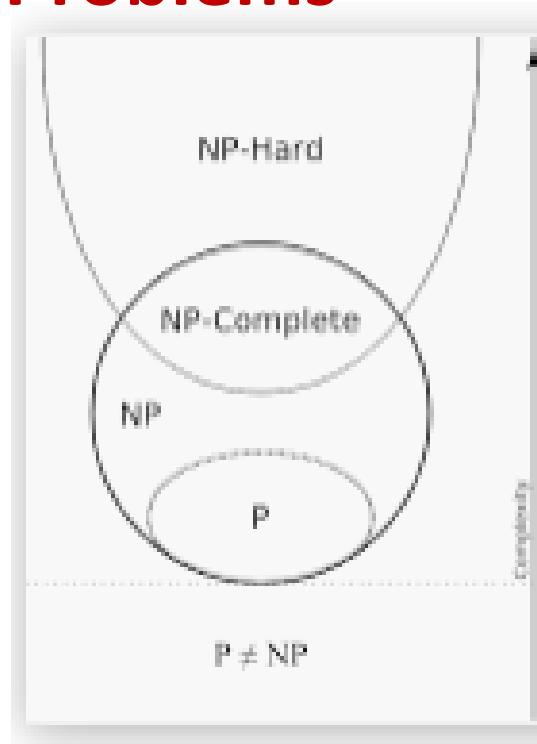
- NP hard problems are basically the optimization versions of the problems in NP complete class
- NP hard problems are not mere yes / no problems. They are problems wherein we need to find the optimal solution
- A problem L is NP complete if and only if L is NP hard and $L \in NP$

Thus, NP-Complete is a subset of NP-Hard.

Relationship between P, NP, NPC and NP HARD Problems

Examples for NP Complete problems as:

3-SAT
3DM
Vertex Cover,
Independent Set,
Knapsack,
Multiprocessor Scheduling, and Partition.



Special Example for NP Hard problems:

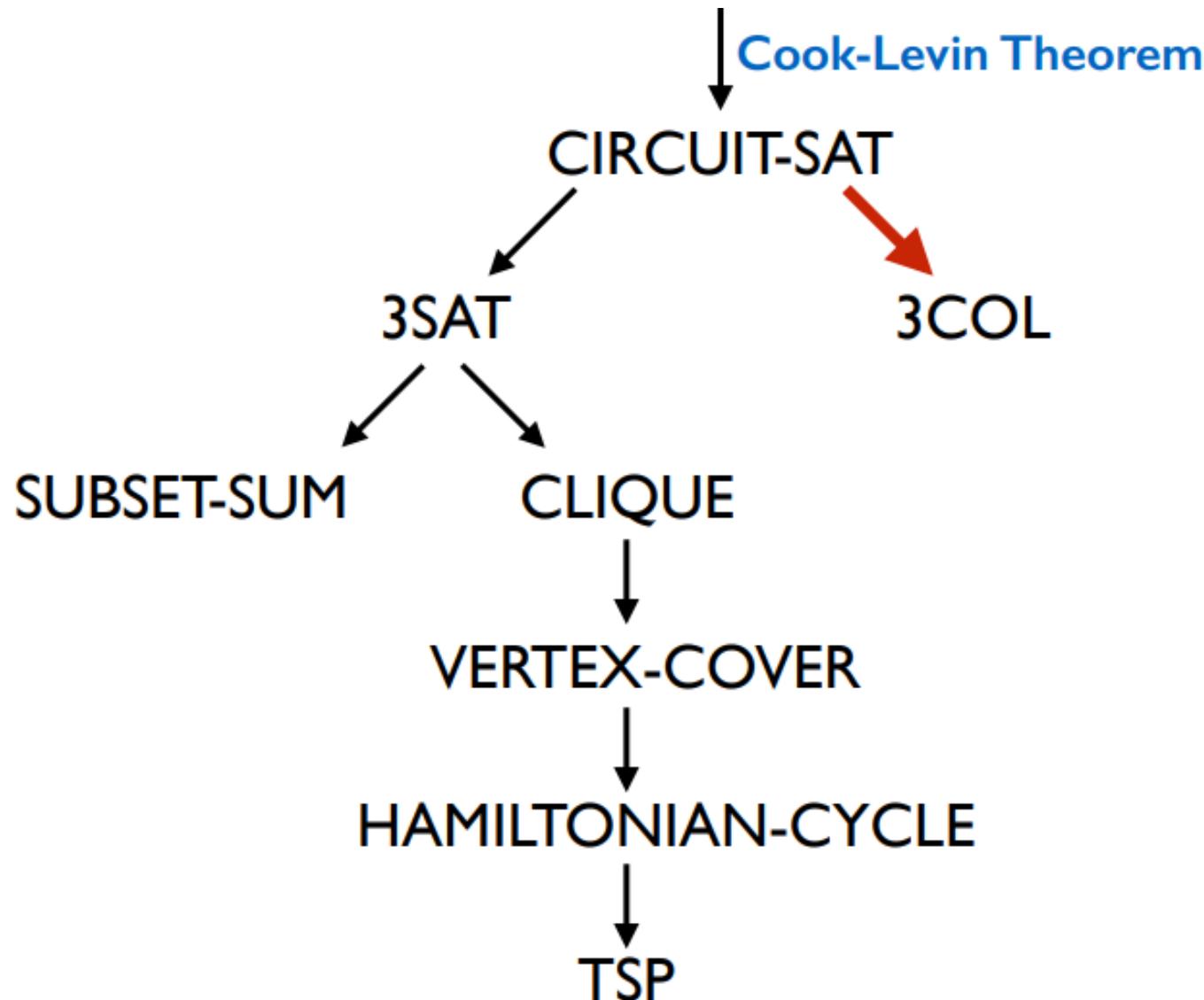
Halting Problem(Undecidable)

NP-COMPLETE AND NP-HARD

- NP-complete problems: the most difficult problems in NP
- if you can solve any one of the NP-complete problems in polynomial time, then you can solve every problem in NP in polynomial time
- because algorithm for NP-complete problem can be used to solve any other problem in NP, after suitable preprocessing (reduction)

Problem A is NP-hard if problem B \leq_P problem A for all problems B in NP.

Problem A is NP-complete if (i) problem A is NP-hard, (ii) problem A is in NP.



Implications of Reductions

Reductions also tell us about the relative difficulty of problems. If we have a way of quickly reducing instances of A into instances of B , then solving B is theoretically at least as difficult as solving A . After all, if you can solve B , then you can solve A by using your reduction and the B solver. I believe this is why we notate the fact that A reduces to B with the notation $A < B$ or $A \leq B$ or $A \leq_P B$. This notion is important in complexity theory. If we know that A is NP-complete, and we have a polynomial-time reduction from A to B , then we can conclude that B is also NP-complete.

SAT: Given some boolean formula in CNF, does it have a satisfying assignment?

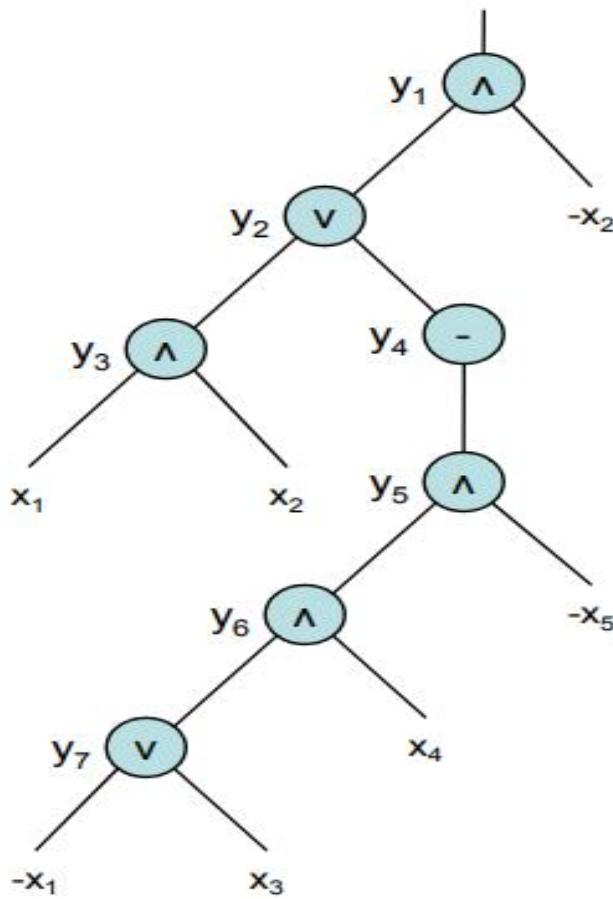
3-SAT: Given some boolean formula in CNF in which each clause contains 3 literals, does it have a satisfying assignment?

SAT appears to be a generalization of 3-SAT and is intuitively more difficult. However, it turns out we can reduce SAT to 3-SAT, so 3-SAT is just as hard as SAT.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_5) \wedge (x_1 \vee x_3 \vee x_4)$$

Problem: Reduce SAT to 3-SAT.

Solution: We first create a parse tree of the given boolean formula. For example, suppose we are given $\phi = (x_1 \wedge x_2) \vee \neg((\neg x_1 \vee x_3) \wedge x_4 \wedge \neg x_5) \wedge \neg x_2$. The parse tree for this ϕ looks like the tree in figure



After obtaining this parse tree we associate a new variable with each internal node, as shown in the same figure. Using these new variables we can write a new boolean formula

$$\begin{aligned}\phi' = & y_1 \wedge (y_1 \iff (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \iff (y_3 \vee y_4)) \\ & \wedge (y_3 \iff (x_1 \wedge x_2)) \\ & \wedge (y_4 \iff \neg y_5) \\ & \wedge (y_5 \iff (y_6 \wedge \neg x_5)) \\ & \wedge (y_6 \iff (y_7 \wedge x_4)) \\ & \wedge (y_7 \iff (\neg x_1 \vee x_3))\end{aligned}$$

We now have a bunch of clauses that each have at most 3 literals. We need to convert these clauses into CNF. To do this, for each clause we will write a truth table to see when the clause is true. For instance, if clause $C_1 = (y_1 \iff (y_2 \wedge \neg x_2))$, then the truth table for C_1 is:

y_1	y_2	$\neg x_2$	C_1
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

With this truth table we will examine each row where the value of C_1 is **false**, or where $\neg C_1$ is true. In this example we see that

$$\begin{aligned}\neg C_1 = & (\neg y_1 \wedge y_2 \wedge \neg x_2) \vee \\& (y_1 \wedge \neg y_2 \wedge x_2) \vee \\& (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee \\& (y_1 \wedge y_2 \wedge x_2)\end{aligned}$$

We can apply DeMorgan's laws to this formula to obtain a version of this clause in CNF:

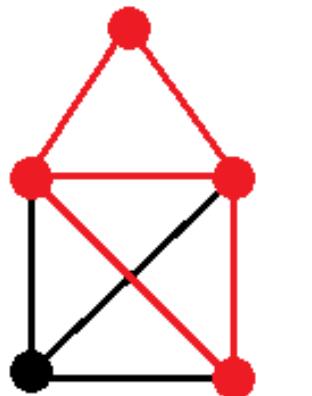
$$\begin{aligned}C_1 = \neg(\neg C_1) = & (y_1 \vee \neg y_2 \vee x_2) \wedge \\& (\neg y_1 \vee y_2 \vee \neg x_2) \wedge \\& (\neg y_1 \vee \neg y_2 \vee x_2) \wedge \\& (\neg y_1 \vee \neg y_2 \vee \neg x_2)\end{aligned}$$

Thus we can show that $\text{SAT} \not\leq_p \text{3SAT}$

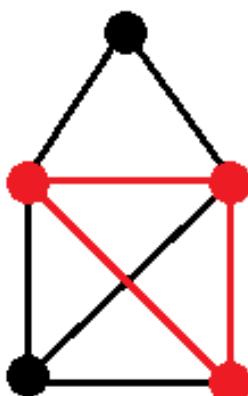
CLIQUE

A Clique is a subgraph of graph such that all vertcies in subgraph are completely connected with each other.

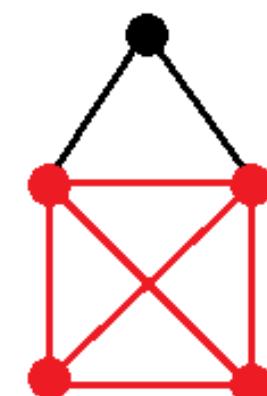
EXAMPLES



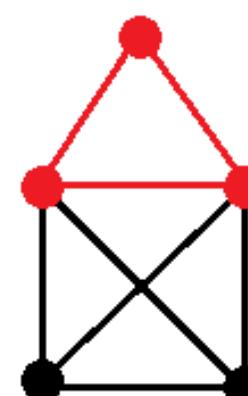
not a clique



non-maximal clique



maximal clique



maximal clique

CLIQUE is NP-complete (3)

Let us try to reduce 3SAT to CLIQUE:

Let F be a 3cnf-formula.

Let C_1, C_2, \dots, C_k be the clauses in F .

Let $x_{j,1}, x_{j,2}, x_{j,3}$ be the literals of C_j .

Hint: Construct a graph G such that

F is satisfiable $\Leftrightarrow G$ has a k -clique

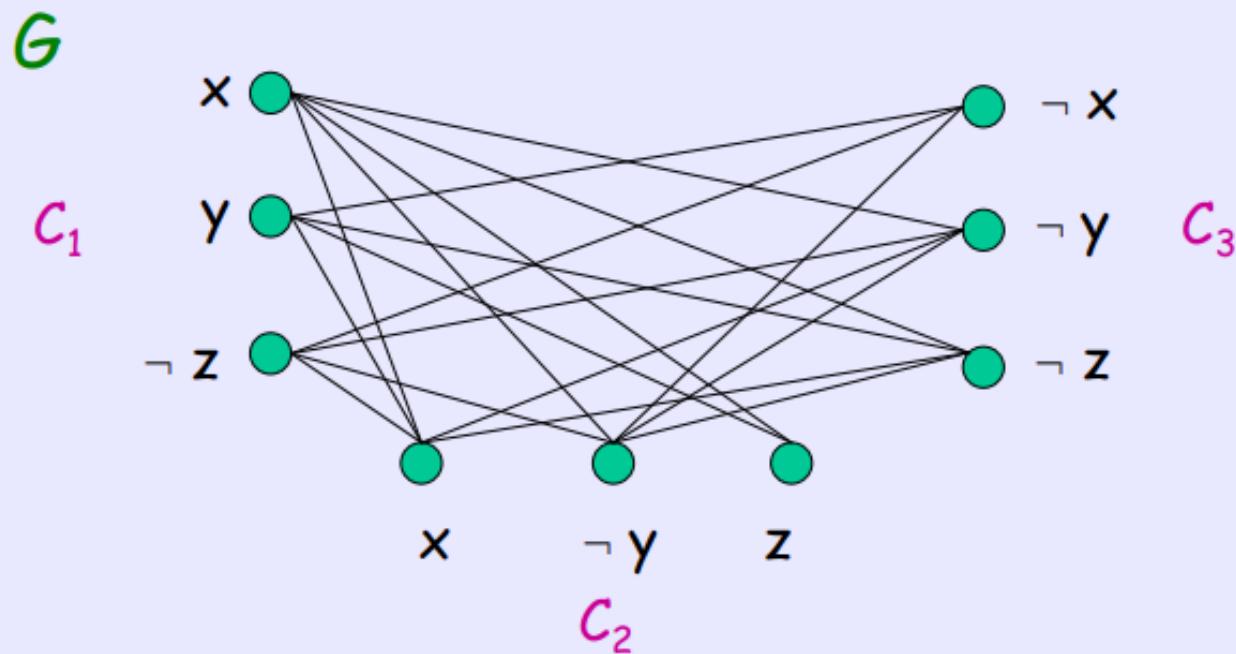
Proof (cont.):

We construct a graph G as follows:

1. For each literal $x_{j,q}$, we create a distinct vertex in G representing it
2. G contains all edges, except those
 - (i) joining two vertices in same clause,
 - (ii) joining two vertices whose literals is the negation of the others

Constructing G from F

$$F = (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge \\ (\neg x \vee \neg y \vee \neg z)$$



Proof (cont.): We now show that
 G has a k -clique $\Leftrightarrow F$ is satisfiable

(\Rightarrow) If G has a k -clique,

1. the k -clique must a vertex from each clause
2. also, no vertex will be the negation of the others in the clique

Thus, by setting the corresponding literal (not variable) to TRUE, F will be satisfied

(\Leftarrow) If F is satisfiable, at least a literal in each clause is set to TRUE in the satisfying assignment

So, the corresponding vertices forms a clique . Thus, G has a k -clique

Finally, since G can be constructed from F in polynomial time, so we have a polynomial time reduction from 3SAT to CLIQUE

Thus, CLIQUE is NP-complete

Hamiltonian Cycle and Traveling Salesman problems

For a given graph $G = (V, E)$, the Hamiltonian Cycle problem is to find whether G contains a Hamiltonian Cycle that is, a cycle that passes through all the vertices of the graph exactly once.

For a given weighted graph $G' = (V', E')$, with non-negative weights, and integer k' , the Traveling Salesman problem is to find whether G' contains a simple cycle of length $\leq k'$ that passes through all the vertices. [Length of a cycle is the sum of weights of all the edges in the cycle].

Reduction of Hamiltonian Cycle Problem to Traveling Salesman Problem

To reduce the Hamiltonian Cycle Problem to the Traveling Salesman problem for a given graph $G = (V, E)$, complete the graph G , by adding edges between all pairs of vertices that were not connected in G .

Let the new graph be $G' = (V', E')$ where $V' = V$ and $E' = \{(u, v)\}$ for any $u, v \in V'$.

For edges in G' that were also present in G , we assign a weight 0.

For other edges we assign weight 1

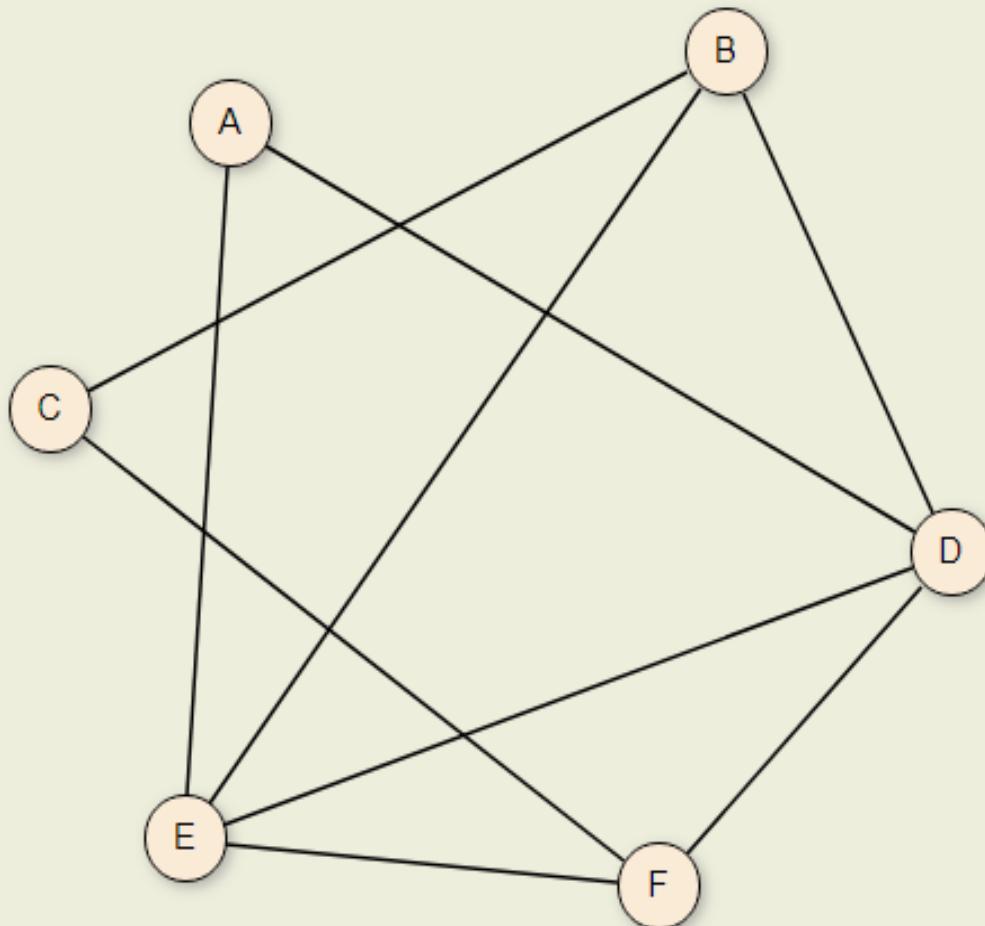
that is, $\forall e = (u, v) \in E'$,

$$W(e) = 0, \text{ if } (u, v) \in E$$

$$W(e) = 1, \text{ if } (u, v) \notin E$$

Example graph

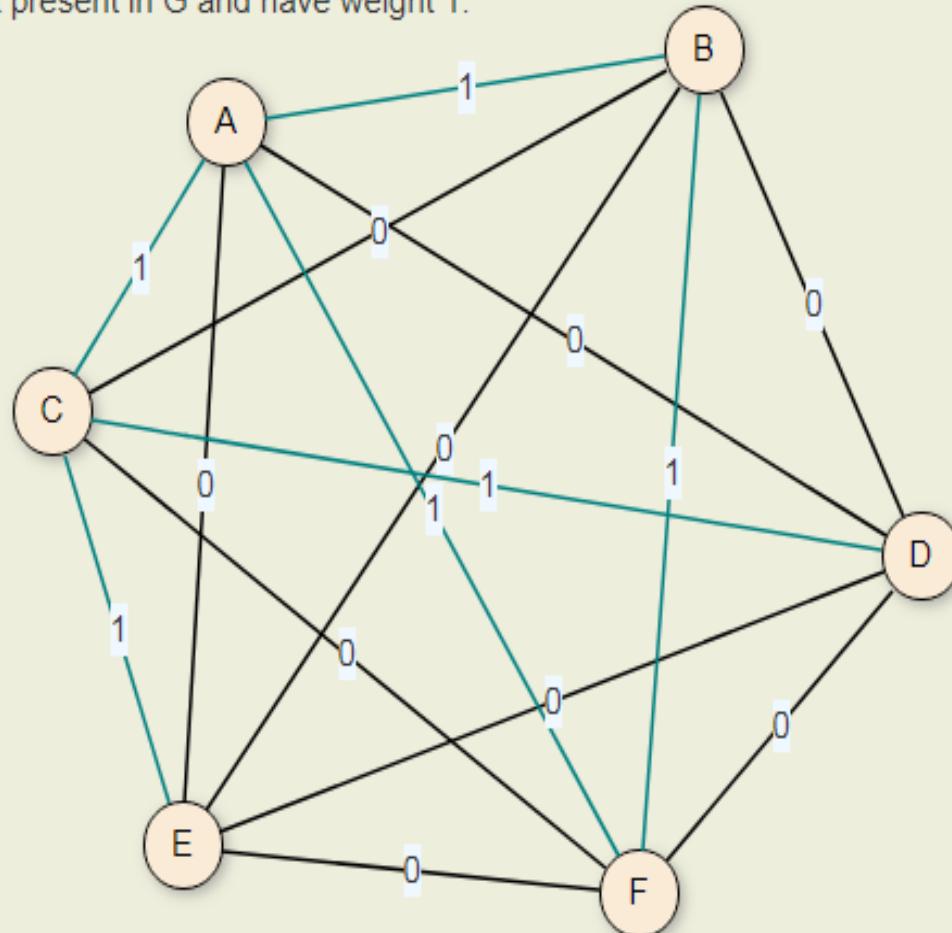
Let this graph G be an input to the Hamiltonian Cycle problem



Example graph

The constructed graph G' is as below.

The blue edges were not present in G and have weight 1.



Hamiltonian Cycle problem reduced to an instance of Traveling Salesman Problem

The graph G has a Hamiltonian Cycle if and only if there exists a cycle in G' passing through all vertices exactly once, and that has a length ≤ 0 (i.e. has a solution for the instance of Traveling Salesman Problem where $k = 0$)

1. If there is a cycle that passes through all vertex exactly once, and has length ≤ 0 in graph G' , the cycle contains only edges that were originally present in graph G . (The new edges in G' have weight 1 and hence can not be part of a cycle of length ≤ 0 .)

Hence there exist a Hamiltonian cycle in G

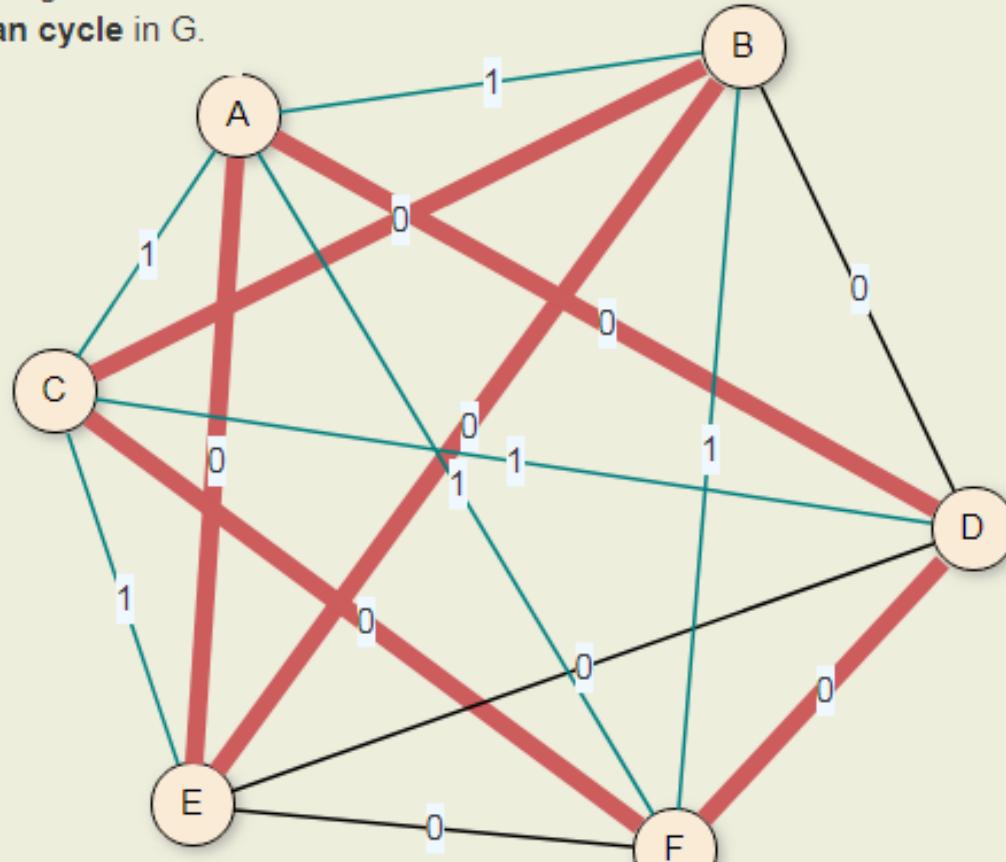
2. If there exists a Hamiltonian Cycle in the graph G , it forms a cycle in G' with length = 0, since a weights of all the edges is 0.

Hence there exists a solution for Traveling Salesman Problem in G' with length ≤ 0

Example:

G' has a cycle passing through all vertices exactly once with length ≤ 0

This cycle is a **Hamiltonian cycle** in G .



UNIT 5

INTRODUCTION TO RANDOMIZATION AND APPROXIMATION ALGORITHMS

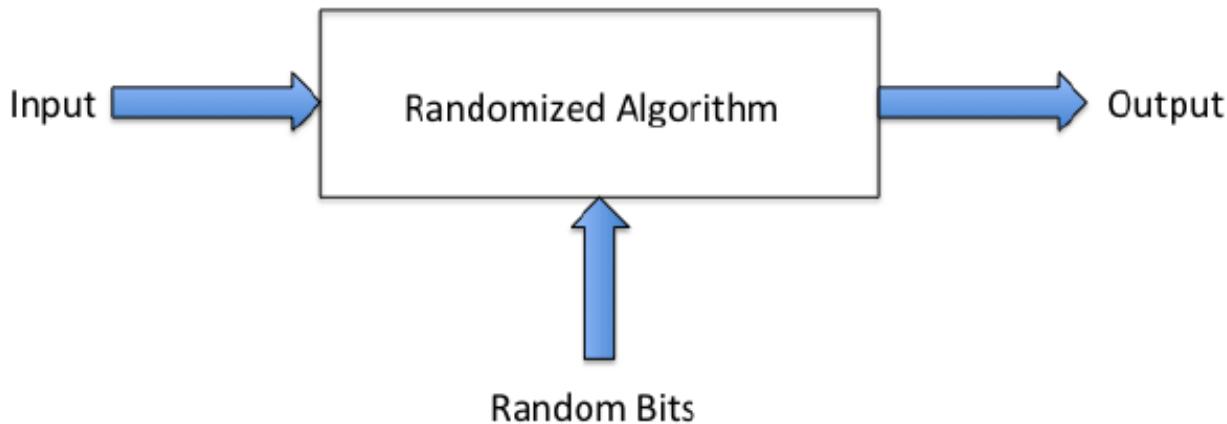
S.Sridhar, AP/CSE,
SRMIST, VADAPALANI CAMPUS.

Deterministic Algorithms



- ▶ Goal: To solve a computational problem correctly and efficiently.
- ▶ Behaviour of the algorithm is determined completely by the input.
- ▶ Upon reruns, the algorithm executes in exactly the same manner.

Randomized Algorithms



- ▶ In addition to the input, the algorithm execution depends on some random bits as well.
- ▶ Behaviour of the algorithm is not determined completely by the input.
- ▶ Upon reruns, the algorithm can execute in a different manner, even with the same input.

Randomized vs. Deterministic

Deterministic algorithms

- ▶ Always correct answer.
- ▶ Always runs within the worst case running time.

Randomized Algorithms

- ▶ Gives the right answer.
- ▶ In good running time.
- ▶ Not necessarily always, but with good probability.

EXAMPLE

Input: An array of $n \geq 2$ elements, having 'a's and b's in it

Output: Find an 'a' in the array.

Las Vegas algorithm:

```
findingA_LV(array A, n)
begin
    repeat
        Randomly select one element out of n elements.
    until 'a' is found
end
```

Broadly two types

Las Vegas Algorithms

- ▶ Correctness is guaranteed
- ▶ May not be fast always.
- ▶ Probability of worst case running time is small.
- ▶ Expected running time < Worst case running time

Monte Carlo Algorithms

- ▶ Running time is fixed.
- ▶ Correctness of the algorithm need not be assured.
- ▶ Probability of an incorrect output is small.

A simple Monte Carlo Method

Monte Carlo algorithm:

```
findingA_MC(array A, n, k)
begin
    i=0
    repeat
        Randomly select one element out of n elements.
        i = i + 1
    until i=k or 'a' is found
end
```

Advantages and Disadvantages

Advantages

- ▶ Simplicity
- ▶ In some cases, faster than deterministic
- ▶ In some cases, no deterministic algorithm exists.
- ▶ Adversary cannot choose a bad input.

Disadvantages

- ▶ Randomness is a resource.
- ▶ With some probability, we can get an incorrect output.
- ▶ With some probability, can perform in worst case time.

Probabilistic Analysis

- Algorithm output/performance can vary depending on random bits.
- Analysis will yield probabilistic statements.
- For a random variable X , $\Pr(X = x)$ denotes the probability with which X takes the value x .
- $E(X)$ denotes the expectation of the random variable X .

The hiring problem

Algorithm *Hire-Assistant*

```

1. CurrentAssistant  $\leftarrow$  nil
2. for  $i \leftarrow 1$  to  $n$ 
3.   do Interview candidate  $i$ 
4.   if candidate  $i$  is better than CurrentAssistant
5.     then CurrentAssistant  $\leftarrow$  candidate  $i$ 

```

If m people hired, **total cost $O(n c_i + m c_h)$** (If c_i is small, can focus on $(m c_h)$)

Worst-case?

- Hire each person interviewed (they come in increasing order of quality); then **total cost = $n c_h$**
 - But reasonable to expect this will not happen. Yet we don't know the order and we don't have a control over that
What do we expect to happen in an average case?

Algorithm *Hire-Assistant-Randomized*

1. Compute a random permutation of the candidates.
2. $CurrentAssistant \leftarrow \text{nil}$
3. **for** $i \leftarrow 1$ **to** n
4. **do** Interview candidate i in the random permutation
5. **if** candidate i is better than $CurrentAssistant$
6. **then** $CurrentAssistant \leftarrow$ candidate i

the expected total fee you have to pay

$$\begin{aligned} E[\text{cost}] &= E\left[\sum_{i=1}^n (\text{cost to be paid for } i\text{-th candidate})\right] \\ &= \sum_{i=1}^n E[(\text{cost to be paid for } i\text{-th candidate})] \text{ (by linearity of expectation)} \end{aligned}$$

Consider the event “ i -th candidate is better than candidates $1, \dots, i-1$ ”, and introduce *indicator random variable* X_i for it:

$$X_i = \begin{cases} 1 & \text{if } i\text{-th candidate is better than candidates } 1, \dots, i-1 \\ 0 & \text{otherwise} \end{cases}$$

(An indicator random variable for an event is a variable that is 1 if the event takes place and 0 otherwise.)

Let the first i candidates in the random order.
and so the i -th candidate is the best candidate with probability $1/i$. Hence,

$$\mathbb{E}[X_i] = \Pr[X_i = 1] = 1/i,$$

and we get

$$\begin{aligned}\mathbb{E}[\text{total cost}] &= \sum_{i=1}^n \mathbb{E}[\text{(fee to be paid for } i\text{-th candidate)}] \\ &= \sum_{i=1}^n C \cdot \mathbb{E}[X_i] \\ &= C \cdot \sum_{i=1}^n (1/i) \\ &\approx C \ln n\end{aligned}$$

INTRODUCTION TO APPROXIMATION ALGORITHMS

- There is a strong evidence to support that no NP-Hard problem can be solved in polynomial time.
- Yet many NP-Hard problems have great practical importance and it is desirable to solve large instances of these problems in a reasonable amount of time.
- The best known algorithms for NP-Hard problems have a worst case complexity that is exponential.
- The examples of NP-Hard problems are :
 - 0/1 Knapsack Problem
 - Sum of subsets Problem
 - Maximum Clique Problem
 - Minimum Node Cover Problem
 - Maximum independent set Problem

- Backtracking & Branch & Bound algorithmic strategies (use of heuristic approach) enable to quickly solve a large instance of a problem provided heuristic works on that instance.
- However this heuristic approach does not work equally effective on all problem instances. NP-Hard problems, even coupled with heuristics, still show exponential behavior on some set of inputs (instances).
- The discovery of a sub-exponential algorithm (Approximate) for NP-Hard problem increases the maximum problem size that can be solved.
- However for large problem instances, we need an algorithm of low polynomial time complexity (say $O(n)$ or $O(n^2)$).

DEFINITION

- Given an optimization problem P , an algorithm A is said to be an approximation algorithm for P , if for any given instance I , it returns an approximate solution, that is a feasible solution.

A COMPARISON

- ▶ Brute-force algorithms
 1. Develop clever enumeration strategies.
 2. Guaranteed to find optimal solution.
 3. No guarantees on running time.
- ▶ Heuristics
 1. Develop intuitive algorithms.
 2. Guaranteed to run in polynomial time.
 3. No guarantees on quality of solution.
- ▶ Approximation algorithms
 1. Guaranteed to run in polynomial time.
 2. Guaranteed to get a solution which is close to the optimal solution (a.k.a near optimal).
 3. *Obstacle : need to prove a solution's value is close to optimum value, without even knowing what the optimum value is !*