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) = Ω(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

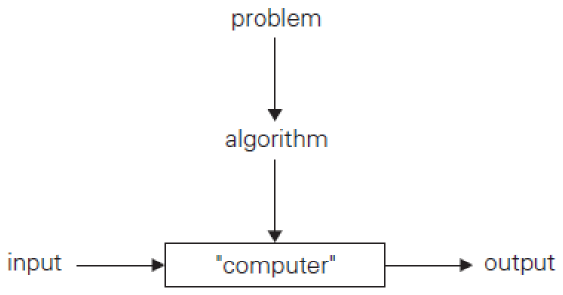
4. **Prove that f(n)= O(g(n)) and g(n)=O(f(g(n)) then f(n)=Θ(g(n)) [Apr/May 2019]**
   Prove by contradiction. Assume $f(n) = O(g(n))$, by the definition, there exist constants c, n0 > 0 such that $0 \le f(n) \le cg(n)$ or $0 \le n \le cn1+\sin n$ for all $n \ge n0$. It implies (0.6) $0 \le 1 \le cn\sin n$ for all $n \ge n0$. Can it be true? To show that the answer is No, it suffices to show: For any n0 > 0, we can always pick an $n \ge n0$ such that $cn\sin n < 1$.

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 thetanotation

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: gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12.

12. Write an algorithm to find the number of binary digits in the binary representation of apositive 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←[n/2]
        return count
```

13. Define Little "oh". [April/May2014]

The function f(n) =
0(g(n)) iff Lim f(n)
=0

n →∞ g(n)

14. Define Little Omega. [April/May2014]

The function f(n) =
ω (g(n)) iff Lim f(n)
=0

n →∞ g(n)

15. Define Big Theta Notations [Nov/Dec2014]

A function t(n) is said to be in Θ(g(n)) , denoted t(n) Є Θ (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 c1 and c2 and some nonnegative integer n0 such that c1

g(n) ≤t(n) ≤ c2g(n) for all n ≥ n0

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

A function t(n) is said to be in Ω(g(n)) , denoted t(n) Є Ω((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 n0 such that

t(n) ≥cg(n) for all for all n ≥ n0

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 no such that

t(n) ≤ cg(n) for all n≥ n0

18. Give the two major phases of performanceevaluation.

Performance evaluation can be loosely divided into two major phases:

- a prior estimates (performanceanalysis)
- a Posterior testing (performancemeasurement)

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: logn
- Linear:n
- N-log-n: n logn
- Quadratic:n2
- Cubic: n3
- Exponential:2n
- 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
    maxvalïƒŸa[0] for I ïƒŸ 1 to n-1 do
     if A[I] >maxval
    return maxvalïƒŸ
    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 defines 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 mod 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 Pm and Pn times in m and n, respectively, it should be repeated in min{Pm, Pn} 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 <a1, a2, …, an>
Output: A reordering <a´1, a´2, …, a´n> of the input sequence such that a´1≤a´2 ≤… ≤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 atext.
- ✓ 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-base, 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) of 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 <u>*no faster*</u> 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., <u>if there exist some positive constant c and</u> <u>some nonnegative integer n_0 such that</u> t(n) <= cg(n) for all n >= n_0

Ω(*g*(*n*)): class of functions *f*(*n*) that grow <u>*at least as fast*</u> as *g*(*n*)
A function *t(n)* is said to be in *Ω(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 $n \geq n_0$

Θ (g(n)): class of functions f(n) that grow at same rate as g(n)

A function *t(n)* is said to be in *(g(n))*, denoted *t(n)* *(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 $c_2\, g(n) <= t(n) <= c_1\, g(n)$ for all $n >= n_0$

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
  the solution F(n) = 1 if n = 0
  n * (n-1) * (n-2)… 3 * 2 * 1 if n > 0
- ✓ Recursive definition
  F(n) = 1 if n = 0
  n * F(n-1) if n > 0

```
    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) = F(n – 1) * n for every n > 0

F(0) = 1

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

M(0) = 0

M(n) ∈ Θ (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 ← 0
    while i < n and A[i] ≠ K do
        i ← 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 *top* (0 <=*p*<-=1) and

(b) the probability of the first match occurring in the ith 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

$$C_{avg}(n) = [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + n \cdot (1-p)$$

$$= \frac{p}{n}[1 + 2 + \cdots + i + \cdots + n] + n(1-p)$$

$$= \frac{p}{n}\frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p).$$

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 x1 and y1. x and y are updated using the below expressions.

x = y1 - ⌊b/a⌋ * x1

y = x1

Time complexity is log2(max(a,b)) and in good case, if a | b or b|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"
Time Complexity= O(m)
```

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

Time Complexity	**Space Complexity**
Best Case: Θ(n log n)	Best Case: Θ(n log n)
Average Case: Θ(n log n)	Average Case: n log n
Worst Case: Θ(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 consist of systematically enumeration 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 tsatisfies
   T (n) = aT (n/b)
   + *f (n) T (1)* = c
   Where *a* ≥1,*b*≥ 2,*c* > 0. *If f(n)* ∈*Θ(n^d) where d* ≥ 0, then

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

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 equalsize.
- The subproblems are solved (typically recursively, though sometimes a different algorithm is employed, especially when subproblems become smallenough).
- If necessary, the solutions to the subproblems are combined to get a solution to theoriginal 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].....A[m-1]A[m]A[m+1] A[n-1]
search here if KA[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 wi and profit pi. Fraction of object is considered as xi (i.e) 0<=xi<=1 .If fraction is 1 then entire object is put into sack. When we place this fraction into the sack, we get wi xi and pi xi.

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 ≥ 2) points p1(x1, y1), . . . ,pn(xn, yn)
//Output: The distance between the closest pair of points
    d←∞
    for i←1 to n − 1 do
        for j ←i+ 1 to n do
            d ←min(d, sqrt((xi − xj )^2 + (yi− yj )^2)) //sqrt is square root
    return d
```

17. Design a brute-force algorithm for computing the value of apolynomial
$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x \ldots a_0$ **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 ← P[0]; power ← 1
    for i ← 1 to n do
        power ← power ∗ x
        p ← p + P[i] ∗ power
    return p
```

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

The recurrence relation is

$T(n) = g(n)$

$T(n_1) + T(n_2) + \quad + T(n_{BTL}) + f(n)$

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 performanceissues

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

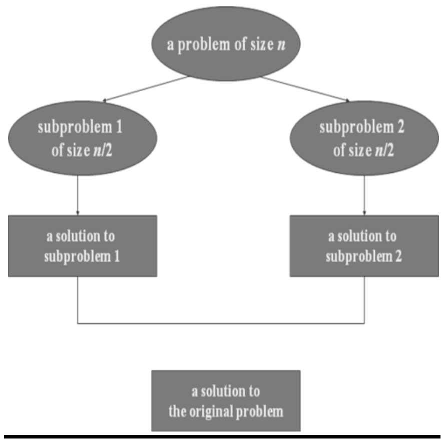
It is a general algorithm design techniques that solved a problem's instance by dividing it into several smaller instance, solving each of them recursively, and then combining their solutions to the original instance of the problem.

<u>PART B & C</u>

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.


a problem of size n
subproblem 1 of size n/2
subproblem 2 of size n/2
a solution to subproblem 1
a solution to subproblem 2
a solution to 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

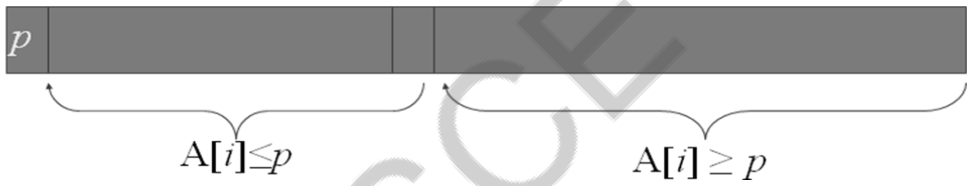
Quick sort is an algorithm of choice in many situations because it is not difficult to implement, it is a good \"general purpose\" sort and it consumes relatively fewer resources during execution.

<u>Quick Sort and divide and conquer</u>

- 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 ≤A[s] and each element of the second array is ≥ 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.

✓ <u>Steps in Quicksort</u>

- 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 ≤ sublist (a sublist whose elements are all smaller than or equal to the pivot) and a ≥ 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., ≤ 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  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 A[l]
i l; j  r + 1;
Repeat
    repeat i  i + 1 until A[i]>=p //left-
    right scan repeat j j – 1 until A[j]
    <= 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., n2) 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: Θ((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 ... w_n

values: v_1 v_2 ... 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 (Logn)^2) algortihm.

Input: An array of n points P[]

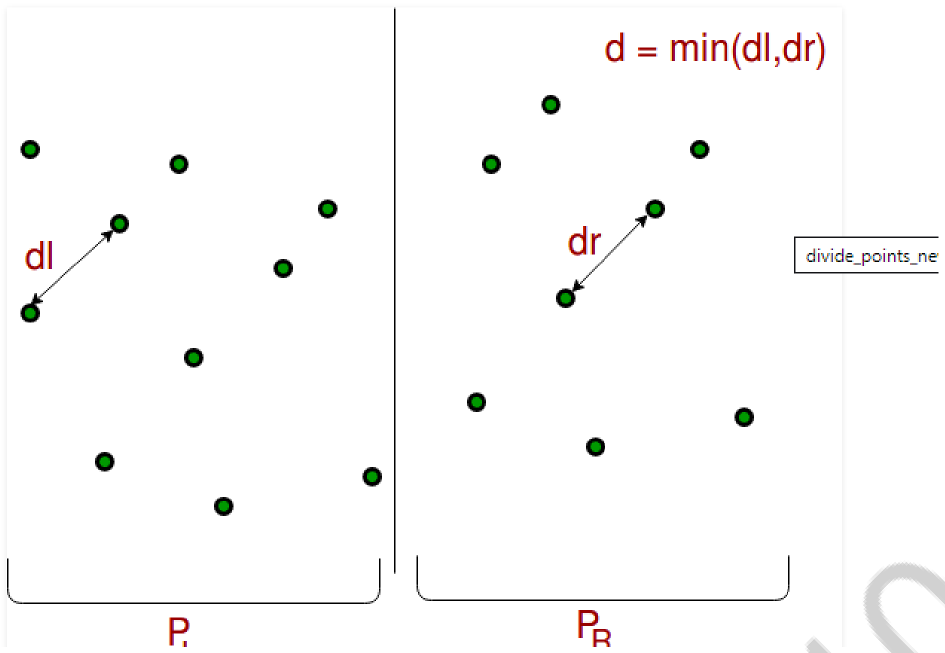
Output: The smallest distance between two points in the given array.

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

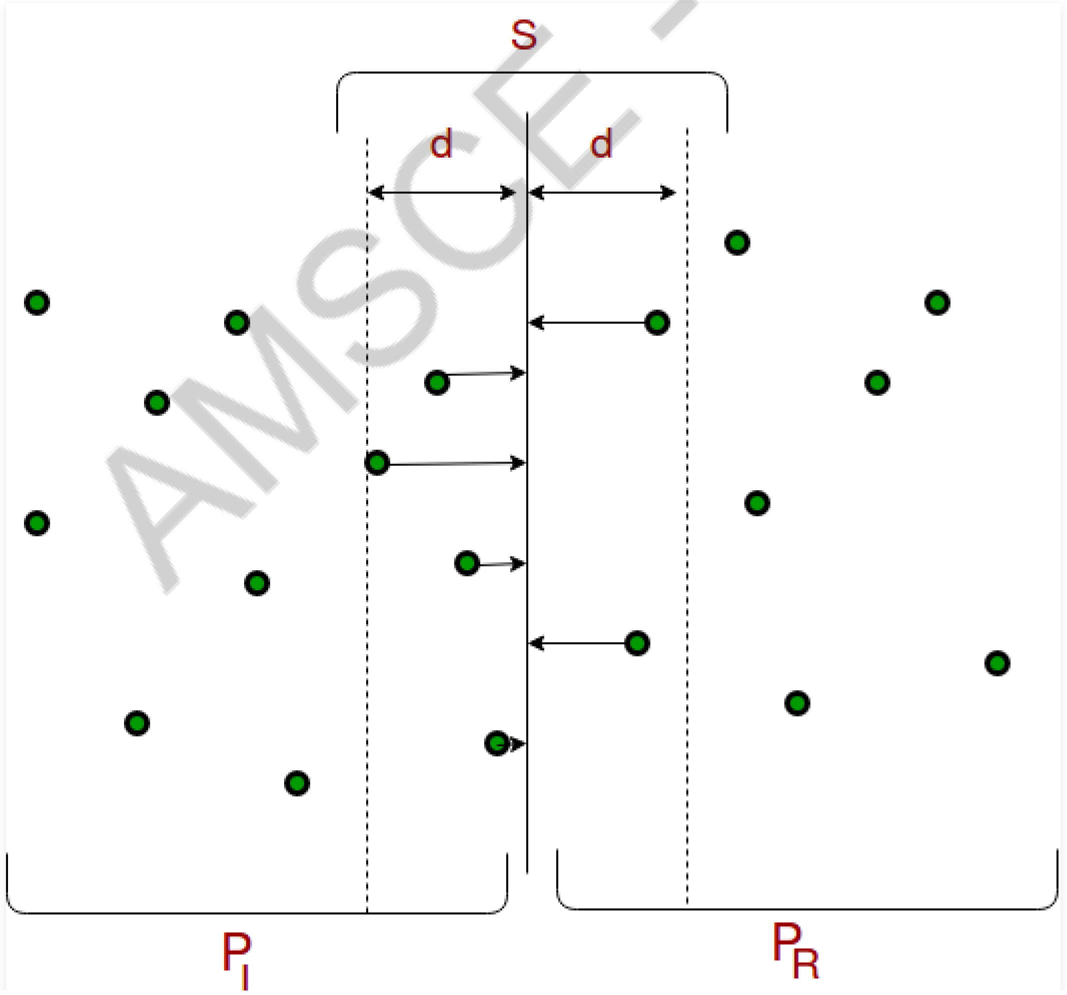
1) Find the middle point in the sorted array, we can take P[n/2] as middle point.

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 strip[] of all such points.



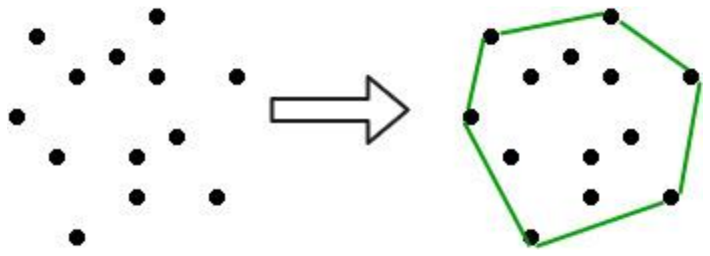
5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.

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 xi for $1<i\leq n$. First we make a decision on x1 and then on x2 and so on. An optimal sequence of decisions maximizes the object function $\Sigma p_i x_i$.

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={tij}, in which the elements in the ith row (1 i n) and the jth column (1 j n) is 1 if there exists a non trivial directed path from the ith vertex to the jth vertex otherwise, tij 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 negativelength.

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 oneperson.

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 othervertices.
Example:Dijikstras 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 isversatility
- 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)an + ... + C(n, k)an-kbk + ... + C(n, n)bn The recursive relation is defined by the prior power
C(n, k) = $C(n-1, k-1) + C(n-1, k)$ for $n > k > 0$
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 Kruscal(G)
//Input: A weighted connected graph $G = <V, E>$

//Output: $E_T$, the set of edges composing a minimum spanning tree of G.
Sort E in nondecreasing order of the edge weights

 $w(e_{i1}) <= ... <= w(e_{i|E|})$

$E_T \longleftarrow \longleftarrow$; ecounter $\longleftarrow 0$//initialize the set of tree edges and its size

$k \longleftarrow 0$

while encounter < $|V| - 1$ do

$k \longleftarrow k + 1$

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

$E_T \longleftarrow E_T \cup \{e_{ik}\}$ ; ecounter  ecounter + 1
```

return *E*

2. Discuss Prim's Algorithm in detail.

- ✓ Minimum Spanning Tree (MST)
    - o Spanning tree of a connected graph G: a connected acyclic subgraph (tree) of G that includes all of G‘s vertices.
    - o 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, … $T_{n-1.}$ .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 (Ti) 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 ← {v0}
E_T ← F
for i ← 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 ← V_T U {u*}
E_T ← E_T U {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.
    - o The choice made at each step must be:
    - o Feasible
        - ▪ Satisfy the problem‘s constraints
    - o locally optimal
        - ▪ Be the best local choice among all feasible choices
    - o Irrevocable
        - ▪ Once made, the choice can‘t be changed on subsequent steps.
- ✓ Applications of the Greedy Strategy
    - o 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:

a. solve several smaller (overlapping) subproblems
b. record solutions in a table so that each subproblem is only solved once
c. 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), …, 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 keys[0.. n-1] of search keys and an array freq[0.. n-1] of frequency counts, where freq[i] is the number of searches to 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: keys[] = {10, 12}, freq[] = {34, 50}

There can be following two possible BSTs

```
   10          12
    \         /
     12      10
    I         II
```

Frequency of searches of 10 and 12 are 34 and 50 respectively.

The cost of tree I is 34*1 + 50*2 = 134

The cost of tree II is 50*1 + 34*2 = 118

Input: keys[] = {10, 12, 20}, freq[] = {34, 8, 50}

There can be following possible BSTs

```
 10        12         20     10       20
  \       /  \       /        \      /
   12    10   20    12         20   10
    \              /          /      \
     20           10         12       12
 I        II        III      IV       V
```

Among all possible BSTs, cost of the fifth BST is minimum.

Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142

1) Optimal Substructure:

The optimal cost for freq[i..j] can be recursively calculated using following formula.

optcost(i, j) = sum_{k=i}^{j} freq \begin{bmatrix}k {bmatrix} + min_{r=i}^{j} {bmatrix} optcost(i, r-1) + optcost(r+1, j) \end{bmatrix}

We need to calculate optCost(0, n-1) to find the result.

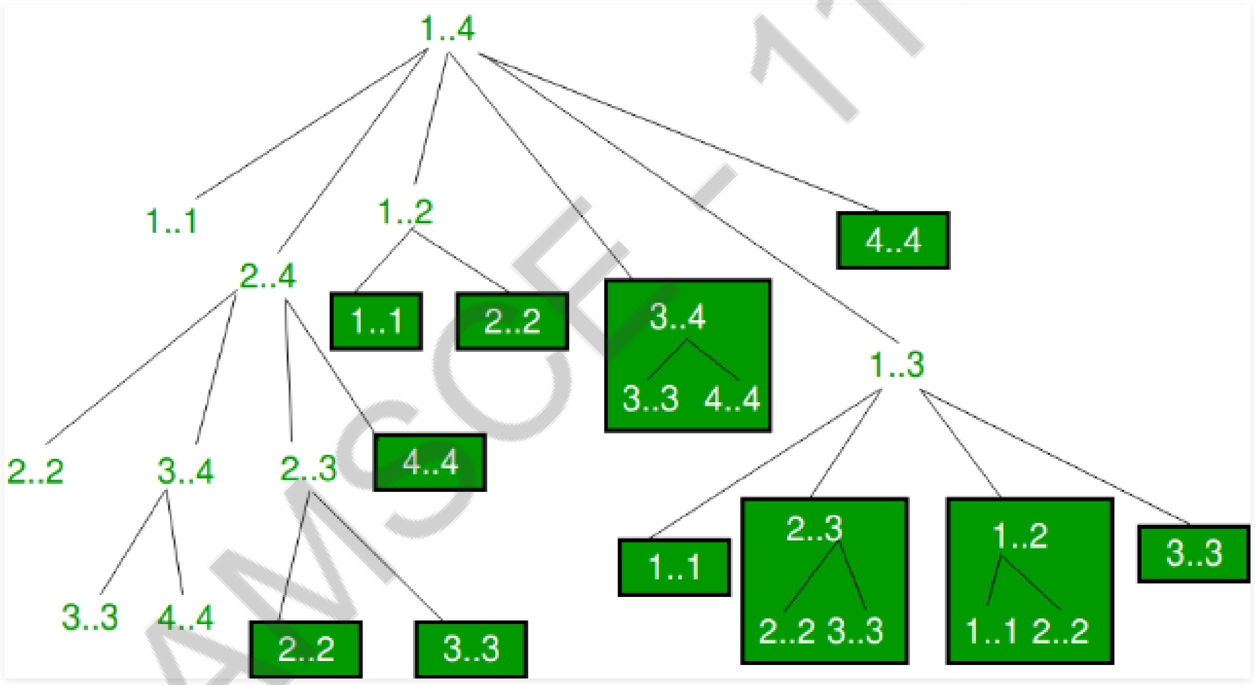
The idea of above formula is simple, we one by one try all nodes as root (r varies from i to j in second term). When we make rth node as root, we recursively calculate optimal cost from i to r-1 and r+1 to j.

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 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 thechoice 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 ongraphswith 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 tosink

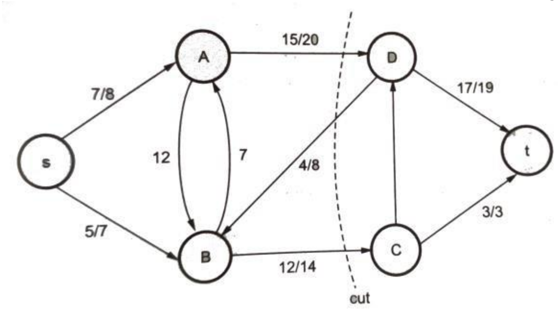
9. **What is maximum cardinality matching? [Nov/Dec 2016]**
   A **maximum matching** (also known as **maximum-cardinality matching**) is a **matching** that contains the largest possible number of edges. There may be many **maximum matchings**. The

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

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 theedge.

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 € 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

```
        ALGORITHMFloyd(
    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 ⇓ W
    for k ⇓ 1 to n do
        for i ⇓ 1 to n do
            for j ⇓ 1 to n do
                D[I,j] ⇓ min{D[I,j], D[I,k] + D[k,j]}
    return D
```

13. Define Ford – FulkersonMethod.

1. Start with the zero flow (*xij*= 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 besent
3. If a flow-augmenting path is found, adjust the flow along the edges of this path to get a flow ofincreased value and tryagain
4. If no flow-augmenting path is found, the current flow ismaximum
5. How to find flow augmenting path in Network flowproblem

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 <=4

x>=0, y>=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.

maximize $z = 3x + 5y + 0u + 0v$
subject to $x + y + u = 4$
$x + 3y + v = 6$
$x \geq 0, y \geq 0, u \geq 0, v \geq 0$ basic feasible solution
(0, 0, 4, 6)

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.

maximize $z = 3x + 5y + 0u + 0v$
subject to $x + y + u = 4$
$x + 3y + v = 6$
$x \geq 0, y \geq 0, u \geq 0, v \geq 0$

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**
  $0 \leq x_{ij} \leq u_{ij}$ for every edge (i,j) □ 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

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

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 $r = \min\{r_{ij}$ on all forward edges, x_{ji} 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, 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.

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) and c(X,X)
- 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 m1 and m2 and two women w1 and w2.
Let m1‘s list of preferences be {w1, w2}
Let m2‘s list of preferences be {w1, w2}
Let w1‘s list of preferences be {m1, m2}
Let w2‘s list of preferences be {m1, m2}

The matching { {m1, w2}, {w1, m2} } is not stable because m1 and w1 would prefer each other over their assigned partners. The matching {m1, w1} and {m2, w2} 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(n2).

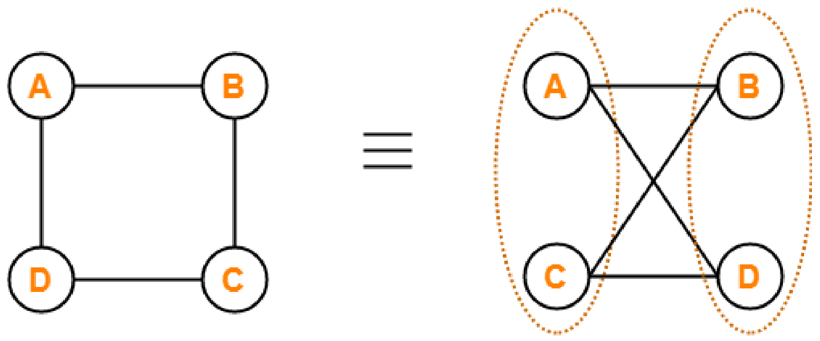
```
Initialize all men and women to free
while there exist a free man m who still has a woman w to propose to
{
   w = m's highest ranked such woman to whom he has not yet proposed
   if w is free
      (m, w) become engaged
   else some pair (m', w) already exists
      if w prefers m to m'
       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

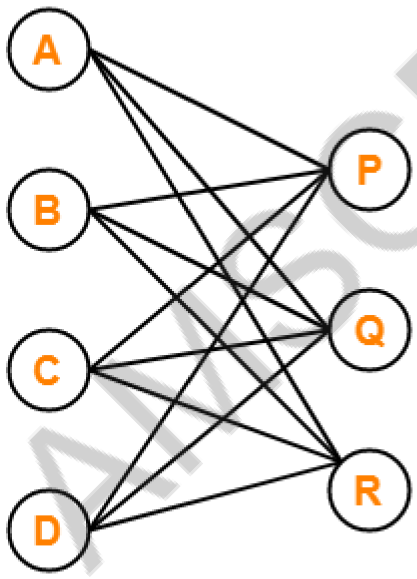
- It consists of two sets of vertices X and Y.
- The vertices of set X join only with the vertices of set Y.
- The vertices within the same set do not join.



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 $x_1+x_2 \le 20$
$x_1 \le 12$
$x_2 \le 16$
$x_1, x_2 \ge 0$

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), isa 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 polynomialtime.

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={S1,S2,S3,…….Sn} 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 otherfeasible **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]

Problemsthat are solvable by polynomial time algorithms asbeing **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. Heuristicsearch

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, columnor 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 mfor 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, $|A*(I) - A(I)| \leq k$. I 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, $\max\{A*(I)\ A(I), A(I)\ A*(I)\} \leq k$. I 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 canbecome 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 withfar fewer than m trials.

20.Listthe 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 nextx_{BTL}
- The number of x_k satisfying the explicitconstraints.
- 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 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.

			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 = {s1, s2, ….., sn } 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 ith 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.

<u>Definition: I</u>

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 ploynomially 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 ploynomially 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]<=d[i,k]+d[k,j] for any triple of cities

- Symmetry:

  d [i,j]=d [j,i] for any pair of

cities I and j The accuracy ratio for any such

instance with n>=2 cities.

f (Sa)/f(S*)<=1/2   [log2 n] +1
where f(Sa) 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 a 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 prims or kruskal‘s algorithm.

- length of the tour Sa
- Optimal tour S*, i.e. Sa twice of S*
- f (Sa)<=2 f(S*) (Hamiltonian circuit)

- Removing a edge from S* yields spanning tree T of weight w (T) w (T)>w (T*)
- f(S*)>w (T)/Tree >=w (T*)/Tree 2 /f(S*)>2 w (T*)
- The Length of the walk>=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.