DEPT OF COMPUTER SCIENCE AND ENGINEERING

DESIGN AND ANALYSIS OF ALGORITHM

2 Marks Qn and Answers

UNIT I

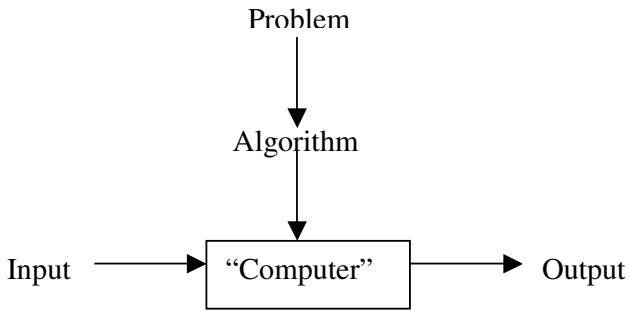
BASIC CONCEPTS OF ALGORITHMS

1. Why is the need of studying algorithms?
   From a practical standpoint, a standard set of algorithms from different areas of computing must be known, in addition to be able to design them and analyze their efficiencies. From a theoretical standpoint the study of algorithms in the cornerstone of computer science.

2. What is algorithmics?
   The study of algorithms is called algorithmics. It is more than a branch of computer science. It is the core of computer science and is said to be relevant to most of science, business and technology.

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

4. Give the diagram representation of Notion of algorithm.



5. What is the formula used in Euclid's algorithm for finding the greatest common divisor of two numbers?
   Euclid's algorithm is based on repeatedly applying the equality
   Gcd(m,n)=gcd(n,m mod n) until m mod n is equal to 0, since gcd(m,0)=m.

6. What are the three different algorithms used to find the gcd of two numbers?
   The three algorithms used to find the gcd of two numbers are
   - Euclid's algorithm
   - Consecutive integer checking algorithm
   - Middle school procedure

7. What are the fundamental steps involved in algorithmic problem solving?
   The fundamental steps are
   - Understanding the problem
   - Ascertain the capabilities of computational device
   - Choose between exact and approximate problem solving
   - Decide on appropriate data structures
   - Algorithm design techniques
   - Methods for specifying the algorithm
   - Proving an algorithms correctness
   - Analyzing an algorithm
   - Coding an algorithm

8. What is an algorithm design technique?
   An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.

9. What is pseudocode?
   A pseudocode is a mixture of a natural language and programming language constructs to specify an algorithm. A pseudocode is more precisethan a natural language and its usage often yields more concise algorithm descriptions.

10. What are the types of algorithm efficiencies?
    The two types of algorithm efficiencies are
    - Time efficiency: indicates how fast the algorithm runs
    - Space efficiency: indicates how much extra memory the algorithm needs

11. Mention some of the important problem types?
    Some of the important problem types are as follows
    - Sorting
    - Searching
    - String processing
    - Graph problems
    - Combinatorial problems
    - Geometric problems
    - Numerical problems

12. What are the classical geometric problems?
    The two classic geometric problems are
    - The closest pair problem: given n points in a plane find the closest pair among them
    - The convex hull problem: find the smallest convex polygon that would include all the points of a given set.

13. What are the steps involved in the analysis framework?
    The various steps are as follows
    - ❖ Measuring the input's size
    - ❖ Units for measuring running time
    - ❖ Orders of growth
    - ❖ Worst case, best case and average case efficiencies

14. What is the basic operation of an algorithm and how is it identified?
    The most important operation of the algorithm is called the basic operation of the algorithm, the operation that contributes the most to the total running time. It can be identified easily because it is usually the most time consuming operation in the algorithms innermost loop.

15. What is the running time of a program implementing the algorithm?
    The running time T(n) is given by the following formula

    $$T(n) \approx c_{op}C(n)$$

    c_{op} is the time of execution of an algorithm's basic operation on a particular computer and C(n) is the number of times this operation needs to be executed for the particular algorithm.

16. What are exponential growth functions?
    The functions 2^n and n! are exponential growth functions, because these two functions grow so fast that their values become astronomically large even for rather smaller values of n.

17. What is worst-case efficiency?
    The worst-case efficiency of an algorithm is its efficiency for the worst-case input of size n, which is an input or inputs of size n for which the algorithm runs the longest among all possible inputs of that size.

18. What is best-case efficiency?
    The best-case efficiency of an algorithm is its efficiency for the best-case input of size n, which is an input or inputs for which the algorithm runs the fastest among all possible inputs of that size.

19. What is average case efficiency?
    The average case efficiency of an algorithm is its efficiency for an average case input of size n. It provides information about an algorithm behavior on a "typical" or "random" input.

20. What is amortized efficiency?
    In some situations a single operation can be expensive, but the total time for the entire sequence of n such operations is always significantly better that the worst case efficiency of that single operation multiplied by n. this is called amortized efficiency.

21. Define O-notation?

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

$T(n) \leq cg(n)$ for all $n \geq n_0$

22. Define Ω-notation?

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

$T(n) \geq cg(n)$ for all $n \geq n_0$

23. Define Θ-notation?

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

$c_2g(n) \leq t(n) \leq c_1g(n)$ for all $n \geq n_0$

24. Mention the useful property, which can be applied to the asymptotic notations and its use?

If $t_1(n)$ ε $O(g_1(n))$ and $t_2(n)$ ε $O(g_2(n))$ then $t_1(n)+t_2(n)$ ε max $\{g_1(n),g_2(n)\}$ this property is also true for Ω and Θ notations. This property will be useful in analyzing algorithms that comprise of two consecutive executable parts.

25. What are the basic asymptotic efficiency classes?
The various basic efficiency classes are
- Constant : 1
- Logarithmic : log n
- Linear : n
- N-log-n : nlog n
- Quadratic : n^2
- Cubic : n^3
- Exponential : 2^n
- Factorial : n!

UNIT II
MATHEMATICAL ASPECTS AND ANALYSIS OF ALGORITHMS

26. Give an non-recursive algorithm to find out the largest element in a list of n numbers.

```
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
```

27. What are the two basic rules for sum manipulation?
    The two basic rules for sum manipulation are

$$\sum_{i=l}^{u} ca_i = c \sum_{i=l}^{u} a_i$$

$$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

28. Mention the two summation formulas?
    The two summation formulas are

$$\sum_{i=l}^{u} 1 = u-l+1 \text{ where } l \leq u \text{ are some lower and upper integer limits}$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1+2+3+\ldots+n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$$

29. Write the general plan for analyzing the efficiency for non-recursive algorithms.
    The various steps include
    - ❖ Decide on a parameter indicating input's size.
    - ❖ Identify the algorithms basic operation.
    - ❖ Check whether the number of times the basic operation is executed depends on size of input. If it depends on some additional property the worst, average and best-case efficiencies have to be investigated separately.
    - ❖ Set up a sum expressing the number of times the algorithm's basic operation is executed.
    - ❖ Using standard formulas and rules of manipulation, find a closed-form formula for the count or at least establish its order of growth.

30. Give a non-recursive algorithm for element uniqueness problem.
    ALGORITHM *UniqueElements*(A[0..n-1])

```
        //Checks whether all the elements in a given array are distinct
        //Input :An array A[0..n-1]
        //Output Returns 'true' if all elements in A are distinct and 'false'
        //otherwise
        for I ← to n-2 do
            for j ← I+1 to n-1 do
                if A[I] = A[j] return false
        return true
```

31. Mention the non-recursive algorithm for matrix multiplication?
    ALGORITHM *MatrixMultiplication*(A[0..n-1,0..n-1], B[0..n-1,0..n-1])

```
        //Multiplies two square matrices of order n by the definition based
        //algorithm
```

```
//Input : Two n-by-n matrices A and B
//Output : Matrix C = AB
for I ← 0 to n-1 do
    for j ← 0 to n-1 do
        C[I,j] ← 0.0
        for k ← 0 to n-1 do
            C[I,j] ← C[I,j] + A[I,k]*B[k,j]
return C
```

32. Write a non-recursive algorithm for finding the number of binary digits for a positive decimal integer.
    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

33. Write a recursive algorithm to find the n-th factorial number.
    ALGORITHM F(n)
    // Computes n! recursively
    // Input A non-negative integer n
    // Output The value of n!
    if n=0 return 1
    else return F(n-1) * n

34. What is the recurrence relation to find out the number of multiplications and the initial condition for finding the n-th factorial number?
    The recurrence relation and initial condition for the number of multiplications is
        M(n)=M(n-1)+1 for n>0
        M(0)=0

35. Write the general plan for analyzing the efficiency for recursive algorithms.
    The various steps include
    - ❖ Decide on a parameter indicating input's size.
    - ❖ Identify the algorithms basic operation.
    - ❖ Check whether the number of times the basic operation is executed depends on size of input. If it depends on some additional property the worst, average and best-case efficiencies have to be investigated separately.
    - ❖ Set up a recurrence relation with the appropriate initial condition , for the number of times the basic operation is executed.
    - ❖ Solve the recurrence or at least ascertain the orders of growth of its solution.

36. Write a recursive algorithm for solving Tower of Hanoi problem.
    ALGORITHM
    - ❖ To move n>1 disks from peg1 to peg3, with peg2 as auxiliary, first move recursively n-1 disks from peg1 to peg2 with peg3 as auxiliary.

- ❖ Then move the largest disk directly from peg1 to peg3
- ❖ Finally move recursively n-1 disks from peg2 to peg3 with peg1 as auxiliary
- ❖ If n=1 simply move the single disk from source peg to destination peg.

37. What is the basic operation in the Tower of Hanoi problem and give the recurrence relation for the number of moves?
    The moving of disks is considered the basic operation in the Tower of Hanoi problem and the recurrence relation for the number of moves is given as
    M(n)=2M(n)+1 for n>1
    M(1)=1

38. Write a recursive algorithm to find the number of binary digits in the binary representation of an integer.
    ALGORITHM BinRec(n)
    // Input A positive decimal integer n
    // Output The number of binary digits in n's binary representation
    if n=1 return 1
    else return BinRec(n/2)+1

39. Who introduced the Fibonacci numbers and how can it be defined by a simple recurrence?
    Leonardo Fibonacci introduced the fibonacci numbers in 1202 as a solution to a problem about the size of rabbit population. It can be defined by the simple recurrence
    F(n)=F(n-1)+F(n-2) for n>1
    And two initial conditions
    F(0)=0 and F(1)=1

40. What is the explicit formula for the n^{th} Fibonacci number?
    The formula for the n^{th} Fibonacci number is given by
    $F(n)= 1/\sqrt{5}\,(\Phi^n - \Phi^n)$
    Where
    Φ $=(1+\sqrt{5})/2$
    Φ $=(1-\sqrt{5})/2$

41. Write a recursive algorithm for computing the n^{th} fibonacci number?
    ALGORITHM F(n)
    // Computes the n^{th} Fibonacci number recursively by using the definition
    // Input A non-negative integer n
    // Output The n^{th} Fibonacci number
    if $n \leq 1$ return n
    else return F(n-1)+F(n-2)

42. Write a non-recursive algorithm for computing the n^{th} fibonacci number.
    ALGORITHM Fib(n)
    // Computes the n^{th} Fibonacci number iteratively by using its definition
    // Input A non-negative integer n
    // Output The n^{th} Fibonacci number
    F[0] ← 0;

```
F[1] ← 1
For I ←2 to n do
        F[I] ← F[I-1]+F[I-2]
return  F[n]
```

43. What is the Θ(log n) algorithm for computing the n^{th} fibonacci number based on?
    There exists a Θ(log n) algorithm for computing the n^{th} fibonacci number that manipulates only integers. It is based on the equality

$$\begin{bmatrix} F(n-1)F(n) \\ F(n)F(n+1) \end{bmatrix} = \begin{bmatrix} 01 \\ 11 \end{bmatrix}$$ for n ≥ 1 with an efficient way of computing matrix powers

44. What is the general plan for the Empirical Analysis of Algorithm Efficiency?
    The general plan is as follows
    - ❖ Understand the experiment's purpose.
    - ❖ Decide on the efficiency metric M to be measured & the measurement unit.
    - ❖ Decide on characteristics of the input sample
    - ❖ Prepare a program implementing the algorithm for the experimentation
    - ❖ Generate a sample of inputs
    - ❖ Run the algorithm on the sample input and record the data observed
    - ❖ Analyze the data obtained

45. Give the various system commands used for timing the program implementing the algorithm.
    The system commands are as follows

UNIX	*time* command
C & C++	function *clock*
Java	method *currentTimeMillis*() in the *System* class

46. Mention the linear congruential method for generating pseudorandom numbers.
    ALGORITHM Random(n,m,seed,a,b)
    //Generates a sequence of n pseudorandom numbers using linear congruential //method
    //Input A positive integer n and positive integer parameters m, seed, a, b
    //Output A sequence $r_1,\ldots..,r_n$ of n pseudorandom integers uniformly distributed //among integer values between 0 and m-1
    r_0 ← seed
    for I ← 1 to n do
        r_i ← $(a*r_{i-1}+b) \bmod m$

47. What are the guidelines in choosing the values of m, seed, a, b for linear congruential method
    The recommendations are as follows

seed	May be chosen arbitrarily and is often set to current date and time

m Should be large and may be conveniently taken as 2^w where w is the computer's word size

a Should be selected as an integer between 0.01m and 099m with no particular pattern in its digits but such that a mod 8=5

b The value can be chosen as 1

48. What is algorithm visualization?
    Algorithm visualization is a way to study algorithms. It is defined as the use of images to convey some useful information about algorithms. That information can be a visual illustration of algorithm's operation, of its performance on different kinds of inputs, or of its execution speed versus that of other algorithms for the same problem.

49. What are the two variations of algorithm visualization?
    The two principal variations of algorithm visualization"
    - *Static algorithm visualization*: It shows the algorithm's progress through a series of still images
    - *Dynamic algorithm visualization*: Algorithm animation shows a continuous movie like presentation of algorithms operations

50. What are the features that are desired in algorithm animation?
    Peter Gloor, who was the principal developer of Animated Algorithms suggested the following desirable features
    - Be consistent
    - Be interactive
    - Be clear and concise
    - Be forgiving to the user
    - Adapt to the knowledge level of the user
    - Emphasize the visual component
    - Keep the user interested
    - Incorporate both symbolic and iconic representations
    - Include algorithm's analysis & comparisons with other algorithms for the same problem
    - Include execution history

51. What are the applications of algorithm visualization?
    The two applications are
    - *Research*: Based on expectations that algorithm visualization may help uncover some unknown feature of the algorithm
    - *Education*: Seeks to help students learning algorithms

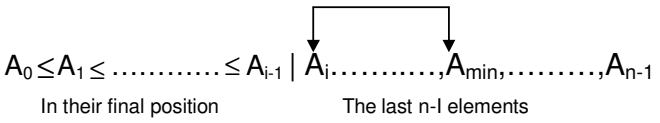
UNIT III
ANALYSIS OF SORTING AND SEARCHING ALGORITHMS

52. Define Brute force approach?
    Brute force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved. The brute force approach is one that is easiest to apply.

53. What are the advantages of brute force technique?
    The various advantages of brute force technique are

- ❖ Brute force applicable to a very wide variety of problems. It is used for many elementary but important algorithmic tasks
- ❖ For some important problems this approach yields reasonable algorithms of at least some practical value with no limitation on instance size
- ❖ The expense to design a more efficient algorithm may be unjustifiable if only a few instances of problems need to be solved and a brute force algorithm can solve those instances with acceptable speed
- ❖ Even if inefficient in general it can still be used for solving small-size instances of a problem
- ❖ It can serve as a yardstick with which to judge more efficient alternatives for solving a problem

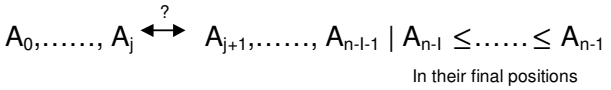
54. What is selection sort?
    Selection sort is started by scanning the entire list to find the smallest element and exchange it with the first element, putting the first element in the final position in the sorted list. Then the scan starts from the second element to find the smallest among n-1 elements and exchange it with the second element.



55. Mention the pseudocode for selection sort.

```
ALGORITHM SelectionSort(A[0..n-1])
//The algorithm sorts a given array by selection sort
//Input: An array A[0..n-1] of orderable elements
//Output: Array A[0..n-1] sorted in ascending order
for I ← 0 to n-2 do
        min ← I
        for j ← I+1 to n-1 do
                if A[j] < A[min] min ← j
        swap A[I] and A[min]
```

56. What is bubble sort?
    Another brute force approach to sort a problem is to compare adjacent elements of the list and exchange them if they are out of order, so we end up "bubbling up" the largest element to the last position in the list. The next pass bubbles up the second largest element, and so on until n-1 passes , the list is sorted. Pass I can be represented as follows



57. Give an algorithm for bubble sort?

```
ALGORITHM BubbleSort(A[0..n-1])
//The algorithm sorts array A[0..n-1] by bubble sort
//Input: An array A[0..n-1] of orderable elements
```

```
//Output: Arrar A[0..n-1] sorted in ascending order
for I ← 0 to n-2 do
        for j ← 0 to n-2-I do
                if A[j+1]<A[j] swap A[j] and A[j+1]
```

58. Give the benefit of application of brute force technique to solve a problem.
    With the application of brute force strategy, the first version of an algorithm obtained can often be improved with a modest amount of effort. So a first application of the brute force approach often results in an algorithm that can be improved with a modest amount of effort.

59. Explain about the enhanced version of sequential search.
    Sequential search simply compares successive elements of a list with a given search key until either a match is encountered or the list is exhausted without finding a match. The enhancement in this version is to append the search key to the end of the list , then the search for the key will have to be successful & so we can eliminate a check for the list's end on each iteration.

60. Give the algorithm for the enhanced version of sequential search.
    ALGORITHM *SequentialSearch2*(A[0..n-1],K)
    //The algorithm implements sequential search with the search key as sentinael
    //Input: An array A of n elements and a search key K
    //Output: The position of the first element in a[0..n-1] whose value is equal to K or //–1 if no such element is found
    A[n] ← K
    I ← 0
    while A[I] ≠ K do
            I ← I+1
    If I < n return I
    else return -1

61. What is the improvement that can be applied to sequential search if the list is sorted?
    The straightforward improvement that can be incorporated in sequential search if a given list is known to be sorted is that searching in the list can be stopped as soon as an element greater than or equal to the search key is encountered.

62. Define brute force string matching.
    The brute force string matching has a given string of n characters called the text and a string of m characters called the pattern, find a substring of the text that matches the pattern. And find the index I of the leftmost character of the first matching substring in the text.

63. Mention the algorithm for brute force string matching
    ALGORITHM *BruteForceStringMatching*(T[0..n-1],P[0..m-1])
    //The algorithm implements brute force string matching
    //Input: an array T[0..n-1] of n characters representing text

```
//An array P[0..m-1] of m characters representing pattern
//Output : The position of the first characters in the text that starts the first
//matching substring if search is successful and –1 otherwise
for I ← 0 to n-m do
        j ← 0
        while j < m and P[j] = T[I+j] do
                j ← j+1
        if j =m return I
return –1
```

64. Give the general plan for divide-and-conquer algorithms.
    The general plan is as follows
    - ❖ A problems instance is divided into several smaller instances of the same problem, ideally about the same size
    - ❖ The smaller instances are solved, typically recursively
    - ❖ If necessary the solutions obtained are combined to get the solution of the original problem

65. State the Master theorem and its use.
    If $f(n) \in \theta(n^d)$ where $d \geq 0$ in recurrence equation $T(n) = aT(n/b)+f(n)$, then

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

    The efficiency analysis of many divide-and-conquer algorithms are greatly simplified by the use of Master theorem.

66. What is the general divide-and-conquer recurrence relation?
    An instance of size 'n' can be divided into several instances of size n/b, with 'a' of them needing to be solved. Assuming that size 'n' is a power of 'b', to simplify the analysis, the following recurrence for the running time is obtained:
    $T(n) = aT(n/b)+f(n)$
    Where f(n) is a function that accounts for the time spent on dividing the problem into smaller ones and on combining their solutions.

67. Define mergesort.
    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.

68. Give the algorithm for mergesort.

```
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)

69. Give the 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]
```

70. What is the difference between quicksort and mergesort?
    Both quicksort and mergesort use the divide-and-conquer technique in which the given array is partitioned into subarrays and solved. The difference lies in the technique that the arrays are partitioned. For mergesort the arrays are partitioned according to their position and in quicksort they are partitioned according to the element values.

71. Give the algorithm for Quicksort.

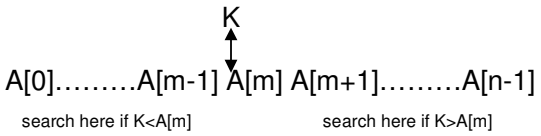
```
ALGORITHM Quicksort(A[l..r])
//Sorts a array by quicksort
//Input: A subarray A[l..r] of A[0..n-1], defined by the left and right indices l & r
//Output: the subarray A[l..r] sorted in nondecreasing order
if l < r
        s ← Partition(A[l..r])
        Quicksort(A[l..s-1])
        Quicksort(A[s+1..r])
```

72. Mention the algorithm used to partition an array for quicksort.

```
ALGORITHM Partition(A[l..r])
//Partitions a subarray using the first element as pivot
//Input: A subarray A[l..r] of A[o..n-1]
//Output: A partition of A[l..r] with split position returned as function's value
p ← A[l]
I ← l; j ← r+1
repeat
        repeat I ← I+1 until A[I] ≥ p
        repeat j ← j-1 until A[j] ≤ p
        swap(A[I],A[j])
until i ≥ j
swap(A[l],A[j])
swap(A[l],A[j])
```

return j

73. What is binary search?
    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].

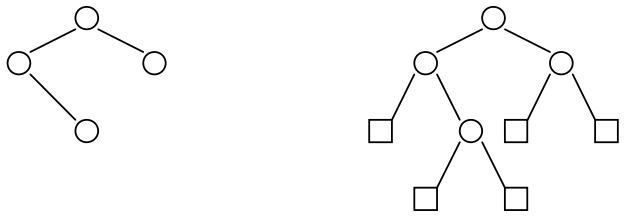


74. What is a binary tree extension and what is its use?
    The binary tree extension can be drawn by replacing the empty subtrees by special nodes in a binary tree. The extra nodes shown as little squares are called external & the original nodes shown as little circles called internal. The extension of a empty binary tree is a single external node. The binary tree extension helps in analysis of tree algorithms.

75. What are the classic traversals of a binary tree?
    The classic traversals are as follows
    - *Preorder traversal*: the root is visited before left & right subtrees
    - *Inorder traversal*: the root is visited after visiting left subtree and before visiting right subtree
    - *Postorder traversal*: the root is visited after visiting the left and right subtrees

76. Mention an algorithm to find out the height of a binary tree.
    ALGORITHM *Height*(T)
    //Compares recursively the height of a binary tree
    //Input: A binary tree T
    //Output: The height of T
    if T = Φ return –1
    else return max{Height(T_L), Height(T_R)}+1

77. Draw the extension tree of the given binary tree.



78. What is decrease and conquer approach and mention its variations?
    The decrease and conquer technique based on exploiting the relationship between a solution to a given instance of a problem and a

solution to a smaller instance of the same problem. The three major variations are

- ❖ Decrease by a constant
- ❖ Decrease by a constant-factor
- ❖ Variable size decrease

79. What is insertion sort?
Insertion sort in an application of decrease-by-one technique to sort an array A[0..n-1]. We assume that the smaller problem of sorting an array A[0..n-2] has already been solved to give us a sorted array of size n-1. Then an appropriate position for A[n-1] is found among the sorted element and then the element is inserted.

80. Give the algorithm for insertion sort.
//Sorts a given array by insertion sort
//Input: An array A[0..n-1] of n orderable elements
//Output: Array A[0..n-1] sorted in non-decreasing order
for I ← 1 to n-1 do
    v ← A[I]
    j ← I-1
    while j ≥ 0 and A[j] > v do
        A[j+1] ← A[j]
        j ← j – 1
        A[j+1] ← v

81. What is a tree edge and back edge?
In the depth first search forest, whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex from which it is being reached. Such an edge is called tree edge because the set of all such edges forms a forest. The algorithm encounters an edge leading to a previously visited vertex other than its immediate predecessor. Such an edge is called a back edge because it connects a vertex to its ancestor, other than the parent, in the depth first search forest.

82. What is a tree edge and cross edge?
In the breadth first search forest, whenever a new unvisited vertex is reached for the first time, it is attached as a child to the vertex from which it is being reached. Such an edge is called tree edge. If an edge is leading to a previously visited vertex other than its immediate predecessor, that edge is noted as cross edge.

UNIT IV ALGORITHMIC TECHNIQUES

83. What is transform and conquer technique?
The group of design techniques that are based on the idea of transformation is called transform and conquer technique because the

methods work as two stage procedures. First in the transformation stage, the problem's instance is modified to be more amenable (agreeable) to the solution. Then in the second or conquering stage, it is solved.

84. What are the three variations by which a given instance of a problem is transformed into?
    The three major variations are
    - ❖ Transformation to a simpler or more convenient instance of the same problem called instance simplification
    - ❖ Transformation to a different representation of the same instance called representation change
    - ❖ Transformation to an instance of a different problem for which the algorithm is already available called problem reduction.

85. What is presorting?
    Presorting is the idea of sorting a list so that problems can be solved more easier than in an unsorted list. The time efficiency of the algorithm that involve sorting before solving the problem depends on the sorting algorithm being used.

86. Give the algorithm for element uniqueness using presorting?
    ALGORITHM *PresortElementUniqueness*(A[0..n-1]0
    //Solves the element uniqueness problem by sorting the array first
    //Input An array A[0..n-1] of orderable elements
    //Output Returns "true" if A has no equal elements, "false" otherwise
    Sort the array A
    for I ← 0 to n-2 do
        If A[I] = A[I+1] return false
    return true

87. Compare the efficiency of solving the problem of element uniqueness using presorting and without sorting the elements of the array?
    The brute force algorithm compares pairs of the array's elements until either two equal elements were found or no more pairs were left. It's worst case efficiency was in $\theta(n^2)$.
    The running time of the presorted algorithm depends on the time spent on sorting and the time spent on checking consecutive elements. The worst case efficiency of the entire presorting based algorithm will be as follows
    $T(n) = T_{sort}(n)+T_{scan}(n) \in \theta(n \log n) + \theta(n) = \theta(n \log n)$

88. Give the algorithm for computing the mode using presorting.
    ALGORITHM *PresortMode*(A[0..n-1])
    //Computes the mode of an array by sorting it first
    //Input an array A[0..n-1] of orderable elements
    //Output The array's mode
    Sort the array
    i ← 0
    modefrequency ← 0
    while i ≤ n-1 do
        runlength ← 1; runvalue ← A[I]

```
        while i + runlength ≤ n-1 and A[i + runlength] = runvalue
                runlength ← runlength + 1
        if runlength > modefrequency
                modefrequency ← runlength; modevalue ← runvalue
        i ← i + runlength
return modevalue
```

89. Compare the efficiencies of the algorithms used to compute the mode before and after sorting the array of elements.

    The efficiency of computing the mode before sorting the array for the worst case is a list with no equal elements. For such a list the I^{th} element is compared with I-1 elements of the auxiliary list of distinct values seen so far before being added to the list with a frequency of 1. The worst case number of comparisons made by the algorithm in creating the frequency list is $\theta(n^2)$.

    The analysis of the presorted algorithm will be dominated by the time spent on sorting the list, since the remainder time to search the frequency list takes only linear time. So the efficiency class for the algorithm is $\theta(n \log n)$.

90. Define AVL trees and who was it invented by?

    An AVL tree is a binary search tree in which the balance factor of every node, which is defined as the difference between the heights of the node's left and right subtrees, is either 0 or +1 or –1. the height of an empty subtree is defined as –1. AVL trees were invented in 1962 by two Russian scientists, G.M.Adelson-Velsky and E.M.Landis, after whom the data struture is named.

91. What are binary search trees and what is it mainly used for?

    Binary search trees is one of the principal data structures for implementing dictionaries. It is a binary tree whose nodes contain elements of a set of orderable items, one element per node, so that all elements in the left subtree are smaller than the element in the subtree's root and all elements in the right subtree are greater than it.

92. What is a rotation in AVL tree used for?

    If an insertion of a new node makes an AVL tree unbalanced, the tree is transformed by a rotation. A rotation in an AVL tree is a local transformation of its subtree rooted at a node whose balance has become either +2 or –2; if there are several such nodes, then the tree rooted at the unbalanced node that is closest to the newly inserted leaf is rotated.

93. What are the types of rotation?

    There are four types of rotations, in which two of them are the mirror images of the other two rotations. The four rotations are

    - Single right rotation or R-rotation
    - Single left rotation or L-rotation
    - Double left-right rotation or LR-rotation
    - Double right-left rotation or RL-rotation

94. Write about the efficiency of AVL trees?

As with any search tree , the critical characteristic is the tree's height. The tree's height is bounded above and below by logarithmic functions. The height 'h' of any AVL tree with 'n' nodes satisfies the inequalities

$\log_2 n \leq h < 1.4405 \log_2(n+2) - 1.3277$

The inequalities imply that the operations of searching and insertion are $\theta(\log n)$ in the worst case. The operation of key deletion in an AVL tree is more difficult than insertion, but it turns out to have the same efficiency class as insertion i.e., logarithmic.

95. What are the drawbacks of AVL trees?

The drawbacks of AVL trees are

- Frequent rotations
- The need to maintain balances for the tree's nodes
- Overall complexity, especially of the deletion operation.

96. What are 2-3 trees and who invented them?

A 2-3 tree is a tree that can have nodes of two kinds:2-nodes and 3-nodes. A 2-node contains a single key K and has two children, the left child serves as the root of a subtree whose keys are less than K and the right child serves as the root of a subtree with keys greater than K.

A 3-node contains two ordered keys K1 & K2 (K1<K2). The leftmost child serves as the root of a subtree with keys less than K1, the middle child serves as the root of a subtree with keys between K1 & K2 and the rightmost child serves as the root of a subtree with keys greater than K2. The last requirement of 2-3 trees is that all its leaves must be on the same level, a 2-3 tree is always height balanced. 2-3 trees were introduced by John Hopcroft in 1970.

97. What is a heap?

A heap is a partially ordered data structure, and can be defined as a binary tree assigned to its nodes, one key per node, provided the following two conditions are met

- The tree's shape requirement-The binary tree is essentially complete, that is all the leaves are full except possibly the last level, where only some rightmost leaves will be missing.
- The parental dominance requirement-The key at each node is greater that or equal to the keys of its children

98. What is the main use of heap?

Heaps are especially suitable for implementing priority queues. Priority queue is a set of items with orderable characteristic called an item's priority, with the following operations

- Finding an item with the highest priority
- Deleting an item with highest priority
- Adding a new item to the set

99. Give three properties of heaps?

The properties of heap are

- There exists exactly one essentially complete binary tree with 'n' nodes. Its height is equal to $\log_2 n$
- The root of the heap is always the largest element

- A node of a heap considered with all its descendants is also a heap

100. Give the main property of a heap that is implemented as an array.

A heap can be implemented as an array by recording its elements in the top-down, left-to-right fashion. It is convenient to store the heap's elements in positions 1 through n of such an array. In such a representation

- The parental node keys will be in the first n/2 positions of the array, while the leaf keys will occupy the last n/2 positions
- The children of a key in the array's parental position 'i' ($1 \le i \le n/2$) will be in positions 2i and 2i+1and correspondingly, the parent of the key in position 'i' ($2 \le i \le n$) will be in position i/2.

101. What are the two alternatives that are used to construct a heap?

The two alternatives to construct a heap are

- Bottom-up heap construction
- Top-down heap construction

102. Give the pseudocode for Bottom-up heap construction.

```
ALGORITHM HeapBottomUp(H[1..n])
//Constructs a heap from the elements of the given array
//Input An array H[1..n] of orderable elements
//Output A heap H[1..n]
for I ← n/2 downto 1 do
        k ← I ; v ← H[k]
        heap ← false
        while not heap and 2*k ≤ n do
                j ← 2*k
                if j < n
                        if H[j] < H[j+1] j ←j+1
                if v ≥ H[j]
                        heap ← true
                else H[k] ← H[j]; k ← j
        H[k] ← v
```

103. What is the algorithm to delete the root's key from the heap?

ALGORITHM

- Exchange the root's key with the last key K of the heap
- Decrease the heap's size by one
- "Heapify" the smaller tree by sifting K down the tree exactly in the same way as bottom-up heap construction. Verify the parental dominance for K: if it holds stop the process, if not swap K with the larger of its children and repeat this operation until the parental dominance holds for K in its new position.

104. Who discovered heapsort and how does it work?

Heapsort was discovered by J.W.J. Williams. This is a two stage process that works as follows

- Stage 1 Heap construction: construct a heap for a given array.

- Stage 2 Maximum deletions: Apply the root deletion operation n-1 times to the remaining heap

105. What is dynamic programming and who discovered it?

Dynamic programming is a technique for solving problems with overlapping subproblems. These subproblems arise from a recurrence relating a solution to a given problem with solutions to its smaller subproblems only once and recording the results in a table from which the solution to the original problem is obtained. It was invented by a prominent U.S Mathematician, Richard Bellman in the 1950s.

106. Define transitive closure.

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 ith row ($1 \le i \le n$) and the jth column ($1 \le j \le n$) is 1 if there exists a non trivial directed path from the ith vertex to the jth vertex otherwise, t_{ij} is 0

107. What is the formula used by Warshall's algorithm?

The formula for generating the elements of matrix $R^{(k)}$ from the matrix $R^{(k-1)}$ is

$r_{ij}^{(k)} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$

This formula implies the following rule for generating elements of matrix $R^{(k)}$ from the elements of matrix $R^{(k-1)}$

- If an element r_{ij} is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$
- If an element r_{ij} is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row 'i' and column 'k' and the element in its row 'k' and column 'j' are both 1's in $R^{(k-1)}$

108. Give the Warshall's algorithm.

```
ALGORITHM Warshall(A[1..n,1..n])
//Implements Warshall's algorithm for computing the transitive closure
//Input The adjacency matrix A of a digraph with 'n' vertices
//Output The transitive closure of the digraph
R(0) ← A
for k ← 1 to n do
        for i ← 1 to n do
                for j ← 1 to n do
                        R(k)[I,j] ← R(k-1)[I,j] or R(k-1)[I,k] and R(k-1)[k,j]
return R(n)
```

109. 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 ← 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

110. How many binary search trees can be formed with 'n' keys?

The total number of binary search trees with 'n' keys is equal to the nth Catalan number

$$c(n) = \binom{2n}{n}\frac{1}{n+1} \text{ for } n>0,\ c(0) = 1,$$

which grows to infinity as fast as $4^n/n^{1.5}$.

111. Give the algorithm used to find the optimal binary search tree.

```
ALGORITHM OptimalBST(P[1..n])
//Finds an optimal binary search tree by dynamic programming
//Input An array P[1..n] of search probabilities for a sorted list of 'n' keys
//Output Average number of comparisons in successful searches in the optimal //BST and table R of subtrees' roots in the optimal BST
for I ← 1 to n do
        C[I,I-1] ← 0
        C[I,I] ← P[I]
        R[I,I] ← I
C[n+1,n] ← 0
for d ← 1 to n-1 do
        for i ← 1 to n-d do
                j ← i +d
                minval ← ∞
                for k ← I to j do
                        if C[I,k-1]+C[k+1,j] < minval
                                minval ← C[I,k-1]+C[k+1,j]; kmin ← k
                R[I,j] ← k
                Sum ←P[I]; for s ← I+1 to j do sum ← sum + P[s]
                C[I,j] ← minval+sum
Return C[1,n], R
```

112. What is greedy technique?

Greedy technique suggests a greedy grab of the best alternative available in the hope that a sequence of locally optimal choices will yield a globally optimal solution to the entire problem. The choice must be made as follows

- *Feasible* : It has to satisfy the problem's constraints
- *Locally optimal* : It has to be the best local choice among all feasible choices available on that step.
- *Irrevocable* : Once made, it cannot be changed on a subsequent step of the algorithm

113. Mention the algorithm for Prim's 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}
```

$E_T \leftarrow \Phi$
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 $V-V_T$
$V_T \leftarrow V_T \cup \{u^*\}$
$E_T \leftarrow E_T \cup \{e^*\}$
return E_T

114. What are the labels in Prim's algorithm used for?

Prim's algorithm makes it necessary to provide each vertex not in the current tree with the information about the shortest edge connecting the vertex to a tree vertex. The information is provided by attaching two labels to a vertex

- The name of the nearest tree vertex
- The length of the corresponding edge

115. How are the vertices not in the tree split into?

The vertices that are not in the tree are split into two sets

- Fringe : It contains the vertices that are not in the tree but are adjacent to atleast one tree vertex.
- Unseen : All other vertices of the graph are called unseen because they are yet to be affected by the algorithm.

116. What are the operations to be done after identifying a vertex u^* to be added to the tree?

After identifying a vertex u^* to be added to the tree, the following two operations need to be performed

- Move u^* from the set $V-V_T$ to the set of tree vertices V_T.
- For each remaining vertex u in $V-V_T$ that is connected to u^* by a shorter edge than the u's current distance label, update its labels by u^* and the weight of the edge between u^* and u, respectively.

117. What is a min-heap?

A min-heap is a mirror image of the heap structure. It is a complete binary tree in which every element is less than or equal to its children. So the root of the min-heap contains the smallest element.

118. What is the use of Kruskal's algorithm and who discovered it?

Kruskal's algorithm is one of the greedy techniques to solve the minimum spanning tree problem. It was discovered by Joseph Kruskal when he was a second-year graduate student.

119. Give the Kruskal's algorithm.

ALGORITHM *Kruskal*(G)
//Kruskal's algorithm for constructing a minimum spanning tree
//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 non decreasing order of the edge weights $w(e_{i1}) \leq \ldots\ldots\ldots \leq w(e_{i|E|})$
$E_T \leftarrow \Phi$
Ecounter ← 0

```
k ← 0
while ecounter < |V|-1
        k ← k+1
        if E_T ∪ {e_ik} is acyclic
                E_T ← E_T ∪ {e_ik}; ecounter ← ecounter + 1
return E_T
```

120. What is a subset's representative?

One element from each of the disjoint subsets in a collection is used as the subset's representative. Some implementations do not impose any specific constraints on such a representative, others do so by requiring the smallest element of each subset to be used as the subset's representative.

121. What is the use of Dijksra's algorithm?

Dijkstra's algorithm is used to solve the single-source shortest-paths problem: for a given vertex called the source in a weighted connected graph, find the shortest path to all its other vertices. The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may have edges in common.

122. What is encoding and mention its types?

Encoding is the process in which a text of 'n' characters from some alphabet is assigned with some sequence of bits called codewords. There are two types of encoding they are

- Fixed-length encoding
- Variable-length encoding

123. What is the problem faced by variable-length encoding and how can it be avoided?

Variable-length encoding which assigns codewords of different lengths to different characters introduces a problem of identifying how many bits of an encoded text represent the first character or generally the ith character. To avoid this prefix-free codes or prefix codes are used. In prefix codes, no codeword is a prefix of a codeword of another character.

124. Mention the Huffman's algorithm.

ALGOITHM *Huffman*

- Initialize n one-node trees and label them with the characters of the alphabet. Record the frequency of each character in its tree's root to indicate the tree's weight.
- Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight. Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.

A tree constructed by the above algorithm is called Huffman tree and it defines the Huffman code

UNIT V ALGORITHM DESIGN METHODS

125. What is backtracking?

Backtracking constructs solutions one component at a time and such partially constructed solutions are evaluated as follows

- If a partially constructed solution can be developed further without violating the problem's constraints, it is done by taking the first remaining legitimate option for the next component.
- If there is no legitimate option for the next component, no alternatives for the remaining component need to be considered. In this case, the algorithm backtracks to replace the last component of the partially constructed solution with its next option.

126. What is a state space tree?

The processing of backtracking is implemented by constructing a tree of choices being made. This is called the state-space tree. Its root represents a initial state before the search for a solution begins. The nodes of the first level in the tree represent the choices made for the first component of the solution, the nodes in the second level represent the choices for the second component and so on.

127. What is a promising node in the state-space tree?

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.

128. What is a non-promising node in the state-space tree?

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.

129. What do leaves in the state space tree represent?

Leaves in the state-space tree represent either non-promising dead ends or complete solutions found by the algorithm.

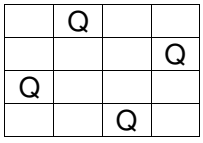
130. What is the manner in which the state-space tree for a backtracking algorithm is constructed?

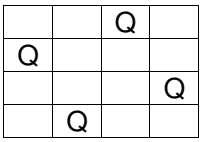
In the majority of cases, a state-space tree for backtracking algorithm is constructed in the manner of depth-first search. If the current node is promising, its child is generated by adding the first remaining legitimate option for the next component of a solution, and the processing moves to this child. If the current node turns out to be non-promising, the algorithm backtracks to the node's parent to consider the next possible solution to the problem, it either stops or backtracks to continue searching for other possible solutions.

131. What is n-queens problem?

The problem is to place 'n' queens on an n-by-n chessboard so that no two queens attack each other by being in the same row or in the column or in the same diagonal.

132. Draw the solution for the 4-queen problem.





133. Define the Hamiltonian circuit.

The Hamiltonian is defined as a cycle that passes through all the vertices of the graph exactly once. It is named after the Irish mathematician Sir William Rowan Hamilton (1805-1865).It is a sequence of n+1 adjacent vertices

$v_{i0}, v_{i1}, \ldots\ldots, v_{in-1}, v_{i0}$

where the first vertex of the sequence is same as the last one while all the other n-1 vertices are distinct.

134. What is the subset-sum problem?

Find a subset of a given set $S=\{s_1,\ldots\ldots\ldots,s_n\}$ of 'n' positive integers whose sum is equal to a given positive integer 'd'.

135. When can a node be terminated in the subset-sum problem?

The sum of the numbers included are added and given as the value for the root as s'. The node can be terminated as a non-promising node if either of the two equalities holds:

$s'+s_{i+1}>d$ (the sum s' is too large)

$s'+ \sum_{j=i+1}^{n} s_j<d$ (the sum s' is too small)

136. How can the output of a backtracking algorithm be thought of?

The output of a backtracking algorithm can be thought of as an n-tuple $(x_1, \ldots x_n)$ where each coordinate x_i is an element of some finite linearly ordered set S_i. If such a tuple $(x_1, \ldots x_i)$ is not a solution, the algorithm finds the next element in S_{i+1} that is consistent with the values of $(x_1, \ldots x_i)$ and the problem's constraints and adds it to the tuple as its (I+1)st coordinate. If such an element does not exist, the algorithm backtracks to consider the next value of x_i, and so on.

137. Give a template for a generic backtracking algorithm.

```
ALGORITHM Backtrack(X[1..i])
//Gives a template of a generic backtracking algorithm
//Input X[1..i] specifies the first I promising components of a solution
//Output All the tuples representing the problem's solution
if X[1..i] is a solution write X[1..i]
else
        for each element x∈ S_{i+1} consistent with X[1..i] and the constraints do
                X[i+1] ← x
                Backtrack(X[1..i+1])
```

138. What are the tricks used to reduce the size of the state-space tree?
The various tricks are
- ❖ Exploit the symmetry often present in combinatorial problems. So some solutions can be obtained by the reflection of others. This cuts the size of the tree by about half.
- ❖ Preassign values to one or more components of a solution
- ❖ Rearranging the data of a given instance.

139. What is the method used to find the solution in n-queen problem by symmetry?
The board of the n-queens problem has several symmetries so that some solutions can be obtained by other reflections. Placements in the last n/2 columns need not be considered, because any solution with the first queen in square (1,i), $n/2 \leq i \leq n$ can be obtained by reflection from a solution with the first queen in square (1,n-i+1)

140. What are the additional features required in branch-and-bound when compared to backtracking?
Compared to backtracking, branch-and-bound requires:
- ❖ A way to provide, for every node of a state space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partial solution represented by the node.
- ❖ The value of the best solution seen so far

141. What is a feasible solution and what is an optimal solution?
In optimization problems, a feasible solution is a point in the problem's search space that satisfies all the problem's constraints, while an optimal solution is a feasible solution with the best value of the objective function.

142. When can a search path be terminated in a branch-and-bound algorithm?
A search path at the current node in a state-space tree of a branch-and-bound algorithm can be terminated if
- ❖ 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 solution because the constraints of the problem are already violated.
- ❖ The subset of feasible solutions represented by the node consists of a single point in this case 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.

143. Compare backtracking and branch-and-bound.

Backtracking	Branch-and-bound
State-space tree is constructed using depth-first search	State-space tree is constructed using best-first search
Finds solutions for combinatorial non-optimization problems	Finds solutions for combinatorial optimization problems
No bounds are associated with the	Bounds are associated with the each

nodes in the state-space tree	and every node in the state-space tree

144. What is the assignment problem?

Assigning 'n' people to 'n' jobs so that the total cost of the assignment is as small as possible. The instance of the problem is specified as a n-by-n cost matrix C so that the problem can be stated as: select one element in each row of the matrix so that no two selected items are in the same column and the sum is the smallest possible.

145. What is best-first branch-and-bound?

It is sensible to consider a node with the best bound as the most promising, although this does not preclude the possibility that an optimal solution will ultimately belong to a different branch of the state-space tree. This strategy is called best-first branch-and-bound.

146. What is knapsack problem?

Given n items of known weights w_i and values v_i, i=1,2,…,n, and a knapsack of capacity W, find the most valuable subset of the items that fit the knapsack. It is convenient to order the items of a given instance in descending order by their value-to-weight ratios. Then the first item gives the best payoff per weight unit and the last one gives the worst payoff per weight unit.

147. Give the formula used to find the upper bound for knapsack problem.

A simple way to find the upper bound 'ub' is to add 'v', the total value of the items already selected, the product of the remaining capacity of the knapsack W-w and the best per unit payoff among the remaining items, which is v_{i+1}/w_{i+1}

$ub = v + (W-w)(v_{i+1}/w_{i+1})$

148. What is the traveling salesman problem?

The problem can be modeled as a weighted graph, with the graph's vertices representing the cities and the edge weights specifying the distances. Then the problem can be stated as finding the shortest Hamiltonian circuit of the graph, where the Hamiltonian is defined as a cycle that passes through all the vertices of the graph exactly once.

149. What are the strengths of backtracking and branch-and-bound?

The strengths are as follows

- It is typically applied to difficult combinatorial problems for which no efficient algorithm for finding exact solution possibly exist
- It holds hope for solving some instances of nontrivial sizes in an acceptable amount of time
- Even if it does not eliminate any elements of a problem's state space and ends up generating all its elements, it provides a specific technique for doing so, which can be of some value.