NOORUL ISLAM COLLEGE OF ENGINEERING, KUMARACOIL
Department of Software Engineering
M. Sc. Software Engineering – Fifth Semester
XCS 355 – Design and Analysis of Algorithm

2 – Marks Question & Answer

1. **What is performance measurement?**

   Performance measurement is concerned with obtaining the space and the time requirements of a particular algorithm.

2. **What is an algorithm?**

   An algorithm is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:
   1) input
   2) Output
   3) Definiteness
   4) Finiteness
   5) Effectiveness.

3. **Define Program.**

   A program is the expression of an algorithm in a programming language. Sometimes works such as procedure, function and subroutine are used synonymously program.

4. **Write the For LOOP general format.**

   The general form of a for Loop is

```
For variable : = value 1 to value 2 step
                    Step do
    {
        <statement 1>
        <statement n >
    }
```

5. **What is recursive algorithm?**

   An algorithm is said to be recursive if the same algorithm is invoked in the body. An algorithm that calls itself is Direct recursive. Algorithm A is said to be indeed recursive if it calls another algorithm, which in turn calls A.

6. **What is space complexity?**

   The space complexity of an algorithm is the amount of memory it needs to run to completion.

7. **What is time complexity?**

   The time complexity of an algorithm is the amount of computer time it needs to run to completion.

8. Give the two major phases of performance evaluation

Performance evaluation can be loosely divided into two major phases:

(i) a prior estimates (performance analysis)

(ii) a Posterior testing(performance measurement)

9. Define input size.

The input size of any instance of a problem is defined to be the number of words(or the number of elements) needed to describe that instance.

10. Define best-case step count.

The best-case step count is the minimum number of steps that can be executed for the given parameters.

11. Define worst-case step count.

The worst-case step count is the maximum number of steps that can be executed for the given parameters.

12. Define average step count.

The average step count is the average number of steps executed an instances with the given parameters.

13. Define the asymptotic notation "Big on" (0)

The function f(n) = O(g(n)) iff there exist positive constants C and no such that $f(n) \leq C * g(n)$ for all n, $n \geq n_0$.

14. Define the asymptotic notation "Omega" (Ω).

The function f(n) =Ω (g(n)) iff there exist positive constant C and no such that f(n) C * g(n) for all n, $n \geq n_0$.

15. Define the asymptotic t\notation "theta" (θ)

The function f(n) =θ (g(n)) iff there exist positive constant C_1, C_2, and no such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all n, $n \geq n_0$.

16. Define Little "oh".

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

iff

$$\lim_{n - \propto} \frac{f(n)}{g(n)} = 0$$

17. Define Little Omega.

The function f(n) = ω (g(n))

iff

$$\lim_{n - \propto} \frac{f(n)}{g(n)} = 0$$

18. Write algorithm using iterative function to fine sum of n numbers.

```
Algorithm sum(a,n)
{       S : = 0.0
        For i=1 to n do
                S : - S + a[i];
Return S;
}
```

19. Write an algorithm using Recursive function to fine sum of n numbers,

```
Algorithm Rsum (a,n)
{
If(n≤ 0) then
        Return 0.0;
Else    Return Rsum(a, n- 1) + a(n);
```

20. Define the divide an conquer method.

Given a function to compute on ‘n’ inputs the divide-and-comquer strategy suggests splitting the inputs in to’k’ distinct susbsets, 1<k <n, yielding ‘k’ subproblems. The subproblems must be solved, and then a method must be found to combine subsolutions into a solution of the whole. If the subproblems are still relatively large, then the divide-and conquer strategy can possibly be reapplied.

21. Define control abstraction.

A control abstraction we mean a procedure whose flow of control is clear but whose primary operations are by other procedures whose precise meanings are left undefined.

22. Write the Control abstraction for Divide-and conquer.

```
Algorithm DAnd(ρ)
{
if small(p) then return S(ρ);
else
{
    divide P into smaller instance $ρ_1$, $ρ_2$, $ρ_k$, $k \geq 1$;
    Apply D and C to each of these subproblems
    Return combine (DAnd C($ρ_1$) DAnd C($ρ_2$),----, DAnd (($ρ_k$));
    }
}
```

23. What is the substitution method?

One of the methods for solving any such recurrence relation is called the substitution method.

24. What is the binary search?

If ‘q’ is always chosen such that ‘aq’ is the middle element(that is, q=[(n+1)/2), then the resulting search algorithm is known as binary search.

25. Give computing time for Bianry search?

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

θ(1) θ(logn) θ(Logn)

best average worst

unsuccessful searches

θ(logn)

best, average, worst

26. Define external path length?

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

27. Define internal path length.

The internal path length 'I' is the sum of the distances of all internal nodes from the root.

28. What is the maximum and minimum problem?

The problem is to find the maximum and minimum items in a set of 'n' elements. Though this problem may look so simple as to be contrived, it allows us to demonstrate divide-and-comquer in simple setting.

29. What is the Quick sort?

n quicksort, the division into subarrays is made so that the sorted subarrays do not need to be merged later.

30. Write the Anlysis for the Quick sot.

In analyzing QUICKSORT, we can only make the number of element comparisions c(n). It is easy to see that the frequency count of other operations is of the same order as C(n).

31. Is insertion sort better than the merge sort?

Insertion sort works exceedingly fast on arrays of less then 16 elements, though for large 'n' its computing time is $O(n^2)$.

32. Write a algorithm for straightforward maximum and minimum>

```
        algorithm straight MaxMin(a,n,max,min)
    //set max to the maximum and min to the minimum of a[1:n]
    {
        max := min: = a[i];
        for i = 2 to n do
    {
        if(a[i] >max) then max: = a[i];
        if(a[i] >min) then min: = a[i];


    }
}
```

33. Give the recurrence relation of divide-and-conquer?

The recurrence relation is

$$T(n) = \begin{cases} g(n) \\ T(n_1) + T(n_2) + \text{----} + T(n_k) + f(n) \end{cases}$$

34. Write the algorithm for Iterative binary search?

```
        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;
}
```

35. What are internal nodes?

The circular node is called the internal nodes.

36. Describe the recurrence relation ofr merge sort?

If the time for the merging operation is proportional to n, then the computing time of merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, \text{ a a constant} \\ 2T\,(n/2) + n & n > 1, \text{ c a constant} \end{cases}$$

37.What is meant by feasible solution?

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution.

38. Write any two characteristics of Greedy Algorithm?

* To solve a problem in an optimal way construct the solution from given set of candidates.

* As the algorithm proceeds, two other sets get accumulated among this one set contains the candidates that have been already considered and chosen while the other set contains the candidates that have been considered but rejected.

39. Define optimal solution?

A feasible solution either maximizes or minimizes the given objective function is called as optimal solution

40. What is Knapsack problem?

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

41. Define weighted tree.

A directed binary tree for which each edge is labeled with a real number (weight) is called as weighted tree.

42. What is the use of TVSP?

In places where the loss exceeds the tolerance level boosters have to the placed. Given a network and loss tolerance level the tree vertex splitting problems is to determine an optimal placement of boosters.

43. What is the Greedy choice property?

* The first component is greedy choice property (i.e.) a globally optimal solution can arrive at by making a locally optimal choice.

* The choice made by greedy algorithm depends on choices made so far but it cannot depend on any future choices or on solution to the sub problem.

* It progresses in top down fashion.

44. What is greedy method?

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can device an algorithm that works in stages considering one input at a time.

45. What are the steps required to develop a greedy algorithm?

* Determine the optimal substructure of the problem.
* Develop a recursive solution.
* Prove that at any stage of recursion one of the optimal choices is greedy choice.
Thus it is always safe to make greedy choice.
* Show that all but one of the sub problems induced by having made the greedy
choice are empty.
* Develop a recursive algorithm and convert into iterative algorithm.

46. What is activity selection problem?

The 'n' task will be given with starting time s_i and finishing time f_i. Feasible Solution is that the task should not overlap and optimal solution is that the task should be completed in minimum number of machine set.

47. Write the specification of TVSP

Let T= (V, E, W) be a weighted directed binary tree where

$V \rightarrow$ vertex set
$E \rightarrow$ edge set
$W \rightarrow$ weight function for the edge.

W is more commonly denoted as w (i,j) which is the weight of the edge $<i,j> \varepsilon$ E.

48. Define forest.

Collection of sub trees that are obtained when root node is eliminated is known as forest

49. .What does Job sequencing with deadlines mean?

* Given a set of 'n' jobs each job 'i' has a deadline d_i such that $d_i>=0$ and a profit p_i such that $p_i>=0$.
* For job 'i' profit p_i is earned iff it is completed within deadline.
* Processing the job on the machine is for 1unit of time. Only one machine is available.

50. Define post order traversal.

The order in which the TVSP visits the nodes of the tree is called the post order traversal.

51. Write the formula to calculate delay and also write the condition in which the node gets splitted?

To calculate delay

$$d(u)=\max_{v \in c(u)}\{d(v)+w(u, v)\}$$

The condition in which the node gets splitted are:

$d(u)+w(u ,v)>\delta$ and also $d(u)$ is set to zero.

52. What is meant by tolerance level?

The network can tolerate the losses only up to a certain limit tolerance limit.

53. When is a task said to be late in a schedule?

A task is said to be late in any schedule if it finishes after its deadline else a task is early in a schedule.

54. Define feasible solution for TVSP.

Given a weighted tree T(V,E,W) and a tolerance limit δ any subset X of V is a feasible solution if $d(T/X)<= \delta$.

55. Define optimal solution for TVSP.

An optimal solution is one in which the number of nodes in X is minimized.

56. Write the general algorithm for Greedy method control abstraction.

```
Algorithm Greedy (a, n)
{
  solution=0;
  for i=1 to n do
  {
    x= select(a);
    if feasible(solution ,x) then
        solution=Union(solution ,x);
  }
    return solution;
}
```

57. . Define optimal solution for Job sequencing with deadlines.

Feasible solution with maximum profit is optimal solution for Job sequencing with deadlines.

58.Write the difference between the Greedy method and Dynamic programming.

Greedy method	**Dynamic programming**
1.Only one sequence of decision is 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.

59.Define dynamic programming.

Dynamic programming is an algorithm design method that can be used when a solution to the problem is viewed as the result of sequence of decisions.

60.What are 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.

61.What are the drawbacks of dynamic programming?

- Time and space requirements are high, since storage is needed for all level.
- Optimality should be checked at all levels.

62.Write the general procedure of dynamic programming.

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

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

63.Define principle of optimality.

It states that an optimal sequence of decisions has the property that whenever the initial stage or decisions must constitute an optimal sequence with regard to stage resulting from the first decision.

64.Give an example of dynamic programming and explain.

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

65.Write about optimal merge pattern problem.

Two files x_1 and x_2 containing m & n records could be merged together to obtain one merged file. When more than 2 files are to be merged together. The merge can be accomplished by repeatedly merging the files in pairs. An optimal merge pattern tells which pair of files should be merged at each step. An optimal sequence is a least cost sequence.

66.Explain any one method of finding the shortest path.

One way of finding a shortest path from vertex i to j in a directed graph G is to decide which vertex should be the second, which is the third, which is the fourth, and so on, until vertex j is reached. An optimal sequence of decisions is one that results in a path of least length.

67.Define 0/1 knapsack problem.

The solution to the knapsack problem can be viewed as a result of sequence of decisions. We have to decide the value of $x_{i.}$ x_i is restricted to have the value 0 or 1 and by using the function knap(l, j, y) we can represent the problem as maximum $\Sigma p_i\ x_i$ subject to $\Sigma w_i\ x_i \leq y$ where l - iteration, j - number of objects, y – capacity.

68.What is the formula to calculate optimal solution in 0/1 knapsack problem?

The formula to calculate optimal solution is

$$g_0(m)=\max\{g_1, g_1(m-w_1)+p_1\}.$$

69.Write about traveling salesperson problem.

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.

70.Write some applications of traveling salesperson problem.

- Routing a postal van to pick up mail from boxes located at n different sites.
- Using a robot arm to tighten the nuts on some piece of machinery on an assembly line.
- Production environment in which several commodities are manufactured on the same set of machines.

71.Give the time complexity and space complexity of traveling salesperson problem.

- Time complexity is $O(n^2\ 2^n)$.
- Space complexity is $O(n\ 2^n)$.

72.Define flow shop scheduling.

The processing of jobs requires the performance of several distinct job. In flow shop we have n jobs each requiring n tasks i.e. T_{1i}, T2i,…...T_{mi}, $1 \leq i \leq n$.

73.What are the conditions of flow shop scheduling?

- Let T_{ji} denote jth task of the ith job. Task T_{ij} is to be performed on P_j number of processors where $1 \leq j \leq m$ i.e. number of processors will be equal to number of task
- Any task T_{ji} must be assigned to the processor P_j.
- No processor can have more than one task assigned to it at any time. For any job I processing the task for j>1 cannot be started until T(j-i),i has been completed.

74.Define nonpreemptive schedule.

A nonpreemptive schedule is a schedule in which the processing of a task on any processor is not terminated until the task is complete.

75. Define preemptive schedule.

A preemptive schedule is a schedule in which the processing of a task on any processor can be terminated before the task is completed.

76.Define finish time

The finish time f_i (S) of job i is the time at which all tasks of job i have been completed in schedule S.The finish time F(S) of schedule S is given by

$$F(S)=\max\{ f_i (S)\} \quad 1\le i\le n$$

77.Define mean flow time

The mean flow time MFT (S) is defined to be]

$$MFT (S) = \frac{1}{n} \sum_{1\le i\le n} fi(S)$$

78.Define optimal finish time.

Optimal finish time scheduling for a given set of tasks is a nonpreemptive schedule S for which F (S) is minimum over all nonpreemptive schedules S.

79.Define preemptive optimal finish time.

Preemptive optimal finish time scheduling for a given set of tasks is a preemptive schedule S for which F (S) is minimum over all preemptive schedules S.

80. What are the requirements that are needed for performing Backtracking?

To solve any problem using backtracking, it requires that all the solutions satisfy a complex set of constraints. They are:

i. Explicit constraints.
ii. Implicit constraints.

81.Define explicit constraint.

They are rules that restrict each x_i to take on values only from a give set. They depend on the particular instance I of the problem being solved. All tuples that satisfy the explicit constraints define a possible solution space.

82. Define implicit constraint.

They are rules that determine which of the tuples in the solution space of I satisfy the criteria function. It describes the way in which the x_i must relate to each other.

83.Define state space tree.

The tree organization of the solution space is referred to as state space tree.

84.Define state space of the problem.

All the paths from the root of the organization tree to all the nodes is called as state space of the problem

85.Define answer states.

Answer states are those solution states s for which the path from the root to s defines a tuple that is a member of the set of solutions of the problem.

86.What are static trees?

The tree organizations that are independent of the problem instance being solved are called as static tree.

87.What are dynamic trees?

The tree organizations those are independent of the problem instance being solved are called as static tree.

88.Define a live node.

A node which has been generated and all of whose children have not yet been generated is called as a live node.

89. Define a E – node.

E – node (or) node being expanded. Any live node whose children are currently being generated is called as a E – node.

90.Define a dead node.

Dead node is defined as a generated node, which is to be expanded further all of whose children have been generated.

91.,What are the factors that influence the efficiency of the backtracking algorithm?

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

i. The time needed to generate the next x_k
ii. The number of x_k satisfying the explicit constraints.
iii. The time for the bounding functions B_k
iv. The number of x_k satisfying the $B_{k.}$

92.Define Branch-and-Bound method.

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

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

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

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

94.State 8 – Queens problem.

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

95.State Sum of Subsets problem.

Given n distinct positive numbers usually called as weights , the problem calls for finding all the combinations of these numbers whose sums are m.

96. State m – colorability decision problem.

Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used.

97.Define chromatic number of the graph.

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

98. Define a planar graph.

A graph is said to be planar iff it can be drawn in such a way that no two edges cross each other.

99. What are NP- hard and Np-complete problems?

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

100. What is a decision problem?

Any problem for which the answer is either zero or one is called decision problem.

101. What is maxclique problem?

A maxclique problem is the optimization problrm that has to determine the size of a largest clique in Grapg G where clique is the maximal subgraph of a graph.

102. what is approximate solution?

A feasible solution with value close to the value of an optimal solution is called approximate solution.

Noorul Islam College of Engineering

Department of Software Engineering

XCS – 355 Designs and Analysis of Algorithm

16 marks Question & Answer

1. Explain algorithm specification

a. Pseudo code conventions

1. Comments begin with // and continue until the end of line.
1. Blocks are indicated and matching braces.: { and }.
2. An identifier begins with a letter
3. Assignment of values to variables is done using the assignment statement
   <variable> := <expression>
4. There are two Boolean values true and false
5. Elements of multi dimensional array are accessed using [and].
6. The following looping statements are employed: for, while and repeat – until.
7. A conditional statement has the following forms:
   If <condition> then <statement>;
8. Input and output are done using the instructions read and write.
9. There only and one type of procedure: Algorithm
   An algorithm consists of heading and body. The heading takes the form

Algorithm Name <parameter list>

b. Recursive algorithms

An algorithm is said to be recursive if the same algorithm is invoked in the body. An algorithm that calls itself is Direct recursive. Algorithm A is said to be indeed recursive if it calls another algorithm, which in turn calls A.

2. Explain all asymptotic notations

Big oh

The function $f(n) = O(g(n))$ iff there exist positive constants C and no such that $f(n) \leq C * g(n)$ for all n, $n \geq n_0$.

Omega

The function $f(n) = \Omega \quad (g(n))$ iff there exist positive constant C and no such that f(n) C * g(n) for all n, $n \geq n_0$.

Theta

The function $f(n) = \theta\ (g(n))$ iff there exist positive constant C_1, C_2, and no such that $C_1\ g(n) \leq f(n) \leq C_2\ g(n)$ for all n, $n \geq n_0$.

Little oh

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

$$\lim_{n - \propto} \frac{f(n)}{g(n)} = 0$$

Little Omega.

The function $f(n) = \omega\ (g(n))$
iff

$$\lim_{n - \propto} \frac{f(n)}{g(n)} = 0$$

3. Explain binary search method

If 'q' is always chosen such that 'aq' is the middle element(that is, q=[(n+1)/2), then the resulting search algorithm is known as binary search.

```
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;
}
```

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

$\theta(1)$ $\theta(\log n)$ $\theta(\log n)$

best average worst

unsuccessful searches

$\theta(\log n)$

best, average, worst

4. Explain the concepts of quick sort method and analyze its complexity

n quicksort, the division into subarrays is made so that the sorted subarrays do not need to be merged later.

Algorithm partition(a,m,p)

```
{
v:=a[m];
I:=m;
J:=p;
Repeat
{
repeat
I:=I+1;
Until(a[I]>=v);
Repeat
J:=j-1;
Until(a[j]<=v);
If(I<j) then interchange(a,I,j);
}until(I>=j);
a[m]:=a[j];
a[j]:=v;
return j;
}
```

Algorithm interchange(a,I,j)

```
{
        p:=a[I];
        a[I]:=a[j];
        a[j]:=p;
        }
```

Algorithm quichsort(p,q)

```
        {
        if(p<q) then
        {
        j:= partition(a,p,q+1);
        quicksort(p,j-1);
        quicksort(j+1,q);
        }
        }
```

In analyzing QUICKSORT, we can only make the number of element comparisions c(n). It is easy to see that the frequency count of other operations is of the same order as C(n).

5. Write the algorithm for finding the maximum and minimum and explain it.

The problem is to find the maximum and minimum items in a set of 'n' elements. Though this problem may look so simple as to be contrived, it allows us to demonstrate divide-and-comquer in simple setting.

```
Algorithm straight MaxMin(a,n,max,min)
  //set max to the maximum and min to the minimum of a[1:n]
  {
        max := min: = a[i];
        for i = 2 to n do
  {
        if(a[i] >max) then max: = a[i];
        if(a[i] >min) then min: = a[i];
  }
}
```

Algorithm maxmin(I,j,max,min)

```
{
if( I = j) then max := min:=a[I];
else if(I=j-1) then
{
if(a[I]<a[j]) then
{
max := a[j];min := a[I];
}
else
{
max:=a[I];
min := a[j];
}}
else
{
mid := [(I+j)/2];
maxmin(I,mid,max,min);
maxmin(mid+1,j,max1,min1);
if(max<max1) then max := max1;
if(min>min1) then min:=min1;
}}
```

6. Explain the concept of merge sort

In merge sort, the elements are to be sorted in non-decreasing order. Given a sequence of n elements i.e. a [1], a [2]…. a [n], the general idea is to imagine them split into 2 sets a [1], …a [(n/2)] and a [(n/2) +1], …. a [n].Each set is individually sorted, and the resulting sorted sequence are merge to produce a single sorted sequence of ‘n’elements. The time complexity is O (nlogn) for worst case.

Insertion sort works exceedingly fast on arrays of less then 16 elements, though for large ‘n’ its computing time is $O(n^2)$.

If the time for the merging operation is proportional to n, then the computing time of merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, \text{ a a constant} \\ 2T(n/2) + n & n > 1, \text{ c a constant} \end{cases}$$

Given a sequence of elements a[1], ..., a[n]. The general idea is to imagine them split in to 2 sets a[1], ..., a[n/2] and a[n/2 + 1], ..., a[n] each sets are individually sorted and the resulting sorted sequence are merged to produce a single sorted sequence of n elements.

```
Algorithm Mergesort(low, high)
{
        if (low<high) then
        {       mid := (low+high)/2;
                mergesort(low, mid);
                mergesort(mid+1, high);
                merge(low,mid,high)
        }
}

Merge Algorithm
Algorithm Merge(low, mid, high)
{       h=high, i=low, j=mid+1;
        while((h<=mid) and (j<=high)) do
        {       if (a[h]<=a[j]) then
                {       b[i]=a[h]
                        h = h + 1;
                }else{
                        b[i]=a[j], j=j+1;
                        }
                i=i+1;
        }
        if (h>mid) then
            for k=j to high do
            {       b[i]=a[k]; i=i+1; }
        else
            for k=hto mid do
            {       b[i]=a[k]; i=i+1;}
        for k=low to high do
            a[k] = b[k]
    }
```

7. Explain multi stage graphs

A multistage graph G = (V,E) is a directed graph in which the vertices are partitioned into k>=2 disjoint sets Vi. The multistage graph problem is to find a minimum cost path from s to t.

Graph:-

Using the forward approach we obtain

Cost(i,j) = min{c(j,l)+cost(i+1,l)}

```
Algorithm Fgraph(G,k,n,p)
{
    cost[n] := 0.0;
    for j := n-1 to 1 step –1 do
    {
        Let r be a vertex such that <j,r> is an edge of G and c[j,r] + cost[r] is minimum;
    Cost[j]:= c[j,r]+cost[r];
    d[j]:= r;
    }
    p[1]:= 1;
    p[k]:=n;
    for j:= 2 to k-1 do
    p[j]:=d[p[j-1]];
}
```

From the backward approach we obtain

Bcost(i,j) = min{bcost(i-1,l)+c(l,j)}

```
Algorithm Bgraph(G,k,n,p)
{
    bcost[1]:= 0.0;
    for j := 2 to n do
    {
        Let r be such that <r,j> is an edge of G and bcost[r]+c[r,j] is minimum;
        Bcost[j]:=bcost[r]+c[r,j];
        D[j]:=r;
    }
    p[1]:= 1;
    p[k]:= n;
    for j := k-1 to 2 do
    p[j]:= d[p[j+1]];
}
```

8. Explain the concepts of job sequencing with deadlines.

Let j be a set of k jobs and ∑ = i1, i2, ... ik a permutation of jobs in j such that $d_{i1} <= d_{i2} <$ ... d_{ik}. Then j is a feasible solution iff the jobs in j can be processed in the order without violating any deadlines.

* Given a set of 'n' jobs each job 'i' has a deadline d_i such that $d_i>=0$ and a profit p_i such that $p_i>=0$.
* For job 'i' profit p_i is earned iff it is completed within deadline.
* Processing the job on the machine is for 1unit of time. Only one machine is available.

Example:-

Let n = 4 (p1,p2,p3,p4) = (100,10,15,27) and (d1,d2,d3,d4) = (2,1,2,1). The feasible solutions and their values are:

	Feasible soln.	Processing sequence	value
1.	(1,2)	2,1	110
2.	(1,3)	3,1	115
3.	(1,4)	4,1	127
4.	(2,3)	3,2	25
5.	(3,4)	4,3	42
6.	(1)	1	100
7.	(2)	2	10
8.	(3)	3	15
9.	(4)	4	27

Solution 3 is optimal.

```
Algorithm JS(d,j,n)
{
    d[0]:=j[0]:=0;
    j[1]:=1;
    k:=1;
    for k:= 2 to n do
    {
        r:= k;
        while((d[j[r]]>=d[I]) and (d[j[r]]<>r)) do
        r:=r-1;
        if((d[j[r]]<=d[I])and (d[I]>r)) then
        {
            for q:=k to r+ 1 step –1 do
            j[q+1] := j[q];
            j[r+1]:=I;
            k:=k+1;
        }
    }
    return k;
```

}

For job sequence there are two possible parameters in terms of which its complexity can be measured. We can use n the number of jobs, and s, the number of jobs included in the solution j. If we consider the job set $p_i = d_i = n – I + 1, 1 <= I <= n$, the algorithm takes $\phi(n^2)$ time to determine j. Hence the worst case computing time for job sequence is $\phi(n^2)$

9. Insertion and Deletion of a Binary Search Tree.

To insert a new element x, we must first verify that its key is different from those of existing elements. To do this, a search is carried out. If the search is unsuccessful, then the element is inserted at the point the search terminated. To insert an element with key 80 into the tree. First we search the element 80 is present in the tree. This search terminates unsuccessfully, and the last node examined is the one with key 40. The new element is inserted as the right child of this node.

```
Algorithm Insert(x)
{
    found := false;
    p := tree;
    while ((p≠0) and not found) do
    {
        q := p;
        if (x = (p → data)) then found := true
        else if (x < (p → data)) then p := p → lchild;
        else p := p → rchild;
    }
    if (not found) then
    {
        p := new treenode;
        p → lchild :=; p → rchild := 0; p → data := x;
        if (tree ≠ 0) then
        {
            if (x < (q → data)) then q → lchild := p;
            else q → rchild := p;
        }
        else tree := p;
    }
}
```

b) Deletion from a binary tree.

To delete an element from the tree, the left child of its parent is set to 0 and the node disposed. To delete the 80 from this tree, the right child field of 40 is set to 0. Then the node containing 80 is disposed. The deletion of a nonleaf element that has only one child is also easy. The node containing the element to be deleted is disposed, and the single child takes the place of the disposed node. To delete another element from the tree, simply change the pointer from the parent node to the single child node.

10. Show that DFS and BFS visit all vertices in a connected graph G reachable from any one of vertices.

In the breadth first search we start at a vertex v and mark it as having been reached. The vertex v is at this time said to be unexplored. A vertex is said to have been explored by an algorithm when the algorithm has visited all vertices adjacent from it. All unvisited vertices adjacent from v are visited next. These are new unexplored vertices. Vertex v has now been explored. The newly visited vertices haven't been explored and are put onto the end of a list of unexplored vertices. The first vertex on this list is the next to be explored. Exploration continues until no unexplored vertex is left.

Breadth first search

We start at a vertex V and mark it as have been reached. The vertex v is at this time said to be unexplorted. All visited vertices adjacent from v are visited next.
If G is represented by its adjacent then the time is O(n2).

```
Algorithm BFS(v)

{

    u := v;

    visited[v] := 1;

    repeat

    {

        for all vertices w adjacent from u do

        {

            if (visited[w] = 0) then

            {

                add w to q;
```

```
                        visited[w] := 1;
                    }
                }
                if q is empty then return;
                delet u from q;
            } until (false)
        }
```

A depth first search of a graph differs from a breadth first search in that the exploration of a vertex v is suspended as soon as a new vertex is reached.
At this time the exploration of the new vertex u begins. When the new vertex has been explored, the exploration of v continues.

Depth first search
The exploration of a vertex V is suspended as soon as a new vertex is reached.
Algorithm DFS(v)

```
Algoithm DFS(v)
{
    visited[v]:=1;
    for each vertex q adjacent from v do
    {
    if (visited[w] =0 ) then DFS(w);
    }
}
```

11. Explain the concepts of traveling salesperson problem.

Let G = (V,E) be a directed graph with edge costs cij. The variable cij is defined such that cij > 0 for all I and j and cij $= \infty$ if <ij> $\notin$ E.Let |V| = n and assume n > 1. A 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 is to find a tour of minimum cost.

Let G=(V,E) be a directed graph defining an instance of the traveling salesperson problem. Let Cij equal the cost of edge (i,j) cij = 0 if (i,j)≠ E, and let |v| = n. The solution space s is given by s = {1, ∏, 1) is a permutation of (2,3,...,n)

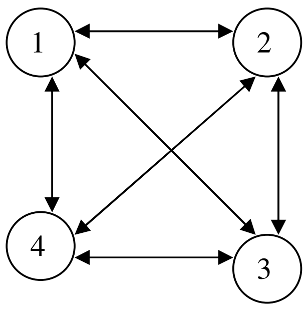
Let us now trace the progress of the LCBB algorithm on the problem instance. The portion of the state space tree that gets generated. Matrix is obtained from that and setting all entries in row 1 and column 3. All exercise examine the implementation considerations for the LCBB algorithm. A possible organization for the state space is a binary tree in which a left branch represents the exclution of that edge.

The principle of optimality is

G{1,V-{1}) = min{c1k+g(k,V-{1,k})}

Generalising

G(I,S) = min{cij+g(j,S-{j})}



$$\begin{pmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{pmatrix}$$

FIG: Directed graph and edge length matrix c

g(1,{2,3,4}) = min{c12+g(2,{3,4}),c13+g(3,{2,4}),c14+g(4,{2,3}))

= min{35,40,43}

= 35

12. Explain all the techniques for binary trees

When the search necessarily involves the examination of every vertex in the object being searched it is called a traversal. There are many operations that we want to perform on binary trees. If we let L,D and R stand for moving left printing the data and moving right when at a node then there are six possible combinations of traversal: LDR,LRD,DLR,DRL,RDL and RLD. If we adopt the convention that we traverse left before right then only three traversals remain: LDR,LRD and DLR. To these we assign the names inorder, postorder and preorder.

```
Algorithm inorder(t)
    {
        if t ≠ 0 then
        {
            inorder(t -> lchild);
            visit(t);
            inorder(t -> rchild);
        }
    }
```

```
Algorithm preorder(t)
    {
        if t ≠ 0 then
        {
            visit(t);
            preorder(t -> lchild);
            preorder(t -> rchild);
        }
    }
```

```
Algorithm postorder(t)
    {
        if t ≠ 0 then
        {
            postorder(t -> lchild);
            postorder(t -> rchild);
            visit(t);
        }
    }
```

13. Write the Bellman and ford algorithm, and explain its concepts (10)

The recurrence for dist is given

$dist^k[u] = \min\{dist^{k-1}[u], \min_i\{dist^{k-1}[i]+cost[i,u]\}\}$

Graph:-

```
Algorithm BellmanFord(v,cost,dist,n)
{
    for I:= 1 to n do
    dist[I] := cost[v,I];
    for k:= 2 to n-1 do
    for each u such that u ≠v and u has at least one incoming edge do
    for each <I,u> in the graph do
    if dist[u]> dist[I]+cost[I,u] then
    dist[u] := dist[I]+cost[I,u];
}
```

The overall complexity is $O(n^3)$ when adjacency matrices are used and O(ne) when adjacency lists are used.

14. Discuss the concepts of 8 – queens problem

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

One solution to the 8 queen problem

			Q				
					Q		
							Q
	Q						
						Q	
Q							
		Q					
				Q			

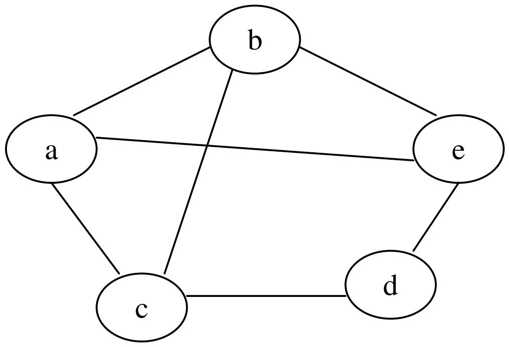
```
Algorithm place(k,I)
{
   for j := 1 to k-1 do
   if(x[j]=I) or(abs(x[j]-I)=abs(j-k))) then return false;
return true;
}
Algorithm Nqueens(k,n)
{
for I:= 1 to n do
{
if( place(k,I) then
{
x[k]:= I;
if(k=n) then write(x[1:n]);
```

else

Nqueens(k+1,n) } } }

15. Graph Coloring Techniques

Let G be a graph and m be a positive integer. The nodes of G can be colored in such a way that no two adjacent nodes have the same color yet only m colors are used. This is termed by m-color ability decision problem and given it is the degree of the queen graph, then it can be colored with j + 1 colors. The m – color ability optimization problem asks for the smallest integer m for which the graph G can be colored.



In the fig the graph can be colored with three colors 1, 2 and 3. The color of each node is indicated next to it. It can also be seen that three colors are needed to color this graph and hence this graphs chromatic number is 3.

Suppose we represent a graph by its adjascency matrix G[1:n, 1:n], where G[I, j] = 1; if [I, j] is an edge of G and G[I, j] = 0, otherwise the colors are represented by the integer 1, 2, 3, …., n and solution are given by the n tupile.
(x1, x2, …., xn) where xi is the color of node is 1.

```
Algorithm mcoloring(k)
{
    repeat
    {
        if (x[k] = 0) then return;
        if (k = n) then write (x[1: n]);
        else mcoloring(k+1);
    } until (false);
}
```

16. Write and explain the concepts of 0/1 knapsack problem.

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

The solution to the knapsack problem can be viewed as a result of sequence of decisions. We have to decide the value of $x_{i.}$ x_i is restricted to have the value 0 or 1 and by using the function knap(l, j, y) we can represent the problem as

maximum $\Sigma p_i x_i$
subject to $\Sigma w_i x_i \leq y$
where l - iteration, j - number of objects, y – capacity

The formula to calculate optimal solution is

$g_0(m)=\max\{g_1, g_1(m-w_1)+p_1\}$.

```
Algorithm BKnap(k, cp, cw)
{
    if (cw + w[k] <= m) then
    {
        y[k] := 1;
        if (k < n) then BKnap(k+1, cp + p[k], cw + w[k]);
        if ((cp + p[k] > fp) and (k = n)) then
        {
            fp := cp + p[k]; fw := cw + w[k];
            for j := 1 to k do
                x[j] := y[j];
        }
    }
    if (bound (cp, cw, k) >= fp) then
    {
        y[k] := 0;
        if (k<n) then BKnap(k+1, cp, cw)
        if ((cp> fp) and (k = n) then
        {
            fp := cp; fw := cw;
            for j := 1 to k do
                x[j] := y[j];
        }}}
```

17. Write and explain the techniques in branch and bound method

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

v. FIFO
vi. LIFO
vii. LC

viii. Heuristic search

Control Abstraction for LC – Search

In both LIFO and FIFO branch and bound the selection rule for the next E – node is rather rigid and in a sence blind. The selection rule for the next E – nod edoes not give any preference to a node that has a very good chance of getting the search to an answer not quickly.

The search for an answer node can often be speeded by using an intelligent ranking function for live node. The next E – node is selected on the basis of ranking function.

Let t be a state space tree and c() a cost function for the nodes in t. If x is a node in t, then c(x) is the minimum cost of any answer node in the subtree with root x. c(t) is the cost of a minimum – cost answer in tree t. As remarked earlier, it is usually not possible to find an easily computable function c(). The heuristic search should be easy to compute and generally has the property that if x is either an answer node or a leaf node, then c(x) = C(x).

```
Listnode = record{
                Listnode * next, *parent;
                Float coast;
```

```
Algorithm LCSearch(t)
{
    if *t is an answer node then output *t and return;
    E := t;
    Initialize the list of live nodes to be empty;
    Repeat
    {
        for each child x of E do
        {
            if x is an answer node than output the path from x to t and return
            add(x);
            x → parent := E;
        }
        if there are no more live nodes then
        {
            write ("No answer node");
            return;
        }
        E := Least();
    }until(false);
}
```

18. Non deterministic Algorithm

Algorithm has the property that the result of every operation whose outcome are not uniquely defined. The machine executing such operation is allowed to choose any one of these outcomes. To specify such algorithms, introduce three new functions.

- Choice (s) – Choose any one of the element of the set s.
- Failure () – Signals an unsuccessful completion
- Success () – Signals a successful completion.

The assignment statement x: = choice(1, n) could result in x being assigned any one of the integer in the range[1, n]. The failure() and Success() signals are used to define a computation of the algorithm. The set of choices that leads to a successful completion, then one such set of choice is always made and the algorithm terminates successfully.

Consider the problem of searching for an element x in a set of elements A[1, n] where n>= 1.

```
Step 1: J: = choice (1, n);
Step 2: If A[j] = x then
                Write (j);
                Success();
Step 3: (A[j] != x)
                Write(0);
                Failure();
```

19. NP – hard Graph Problems

1. Pick a problem L1 already known to be NP – hard.
2. Show how to obtain an instance I' of L2 from any Instance I of L1, Such that from the solution of I' we can determine the solution to instance I of L1.
3. Conclude from step 2 that L1 α L2.
4. From step1 and step 3, the transitivity of α that L2 is NP – hard

a) Clique Decision Problem (CDP)

A maximal complete sub graph of a graph G = (V, E) is a clique. The size of the clique is the number of vertices in it. The Max clique problem is an optimization problem that has to determine the size of a largest clique in G. The corresponding decision problem is to determine wheather G has a clique of size k for some k.

The Input to the max clique decision problem can be provided as a sequence of edged and an integer k. Each edge in E(G) is a pair of numbers(I, j). The size of the input for each edge(I, j) is log I 2 + log j 2 + 2. if a binary representation is assumed. The input size of any instance is

b) Node Cover Decision problem

A set V is a node cover for a graph G = (V, E) iff all edges in E are incident to atleast one vertex in s. The size [s] of the cover is the number of vertices in s.

c) Chromatic Number Decision Problem. (CNDP)

A coloring of a graph G = (V, E) is a function f:→ {1, 2, ….., k} defined for all x € v. If (V, E) € E, then f(u) ≠ f(v).

d) NP – hard scheduling problem

This problem requires us to decided whether a given multiset A = {a1, a2, a3, ….., an) of n positive integer has a partition p. such that

e) Job shop scheduling.

A job shop like a flow shop has m different processors. The n job to be scheduled require the completion of several task. The time of the j^{th} task for job ji. The task for any job ji are to be carried out in the order 1, 2, 3, ….. and so on

20. Approximation & ε - Approximation Algorithms

- A is an absolute approximation algorithm for problem p iff for every instance I of p. $\left|f^*(I) - \hat{f}(I)\right| \le k$ for some constant k.
- A is an f(n) approximate algorithm iff for every instance I of size n, $\left|f^*(I) - \hat{f}(I)\right| \le f(n)$ for p*(I) > 0.
- An ε - Approximation algorithm is an f(n) approximate algorithm for which f(n) <= ε for some constant e.
- A(e) is an approximation scheme iff for every given ε > 0 and problem instance I, A(e) generate α feasible solution such that $\frac{\left|f^*(I) - \hat{f}(I)\right|}{f^*(I)} \le e$ where f*(I) > 0.
- An approximation scheme is a polynomial time approximation scheme iff for every fixed e > 0 it has a computing time ie, a polynomial in the problem size.
- An approximation scheme whose computing time is a polynomial both in the problem size s in 1/e is a fully polynomial time approximation scheme.

Maximum program stored problem

Assume we have n programs and two storage device. let li be the amount of storage needed to store the i^{th} program. Let L be the storage capacity of each disk. Determine the maximum number of these n programs that can be stored on two desk is NP – hard

i) Partition of maximum program

ii) Let I be any instance of the maximum programs stored problem. Let f*(I) be the maximum number of programs that can be stored on two desks each of length L.

iii) Let $\left|\hat{f}(I)\right|$ be the number of programs stored using the function Pstore, then $\left|f^*(I) - \hat{f}(I)\right| \le 1$

```
Algorithm Pstore(l,n)
{
    I = 1;
    for j := 1 to 2 do
    {
        sum := 0;
        while (sum + l(i)) <= L do
        {
            write ("Store Program");
            sum := sum + l[i];
            i := i + 1;
            if i > n then return;
        }
    }
}
```

Simple heuristics are

- first fit (ff)
- Best fit (bf)
- First fit decreasing (ffd)
- Best fit decreasing (bfd)