Biyani's Think Tank

Concept based notes

Analysis and Design Algorithm

MCA

Ms Rashmi Sharma
Deptt. of IT
Biyani Girls College, Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

Edition : 2012

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the “Teach Yourself” style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this endeavour. They played an active role in coordinating the various stages of this endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus
MCA Semester 5

Introduction:- algorithm definition and specification – Design of Algorithms, and Complexity of Algorithms, Asymptotic Notations, Growth of function, Recurrences, Performance analysis – Elementary Data structures:- stacks and queues – trees – dictionaries – priority queues –sets and disjoint set union – graphs – basic traversal and search techniques. Divide – and – conquer:- General method – binary search – merge sort – Quick sort –

The Greedy method:-General method – knapsack problem – minimum cost spanning tree – single source shortest path.

Dynamic Programming – general method – multistage graphs – all pair shortest path – optimal binary search trees – 0/1 Knapsack – traveling salesman problem – flow shop scheduling.

Backtracking:- general method – 8-Queens problem – sum of subsets – graph coloring – Hamiltonian cycles – knapsack problem – Branch and bound:- The Method – 0/1 Knapsack problem – traveling salesperson.

Parallel models:-Basic concepts, performance Measures, Parallel Algorithms: Parallel complexity, Analysis of Parallel Addition, Parallel Multiplication and division, parallel

Evaluation of General Arithmetic Expressions, First-Order Linear recurrence

.

Unit 1
Introduction of Algorithm

Q.1 Agorithm is a combination of sequence of finite steps to solve problems.

Ans. Characterstics of Algorithm:

1. takes finite time.

2.produce at least one output.

3. should take 0 or more input

4. definate or deterministic.

5.It should be effective.

Q.2 What is the specification of algorithm?

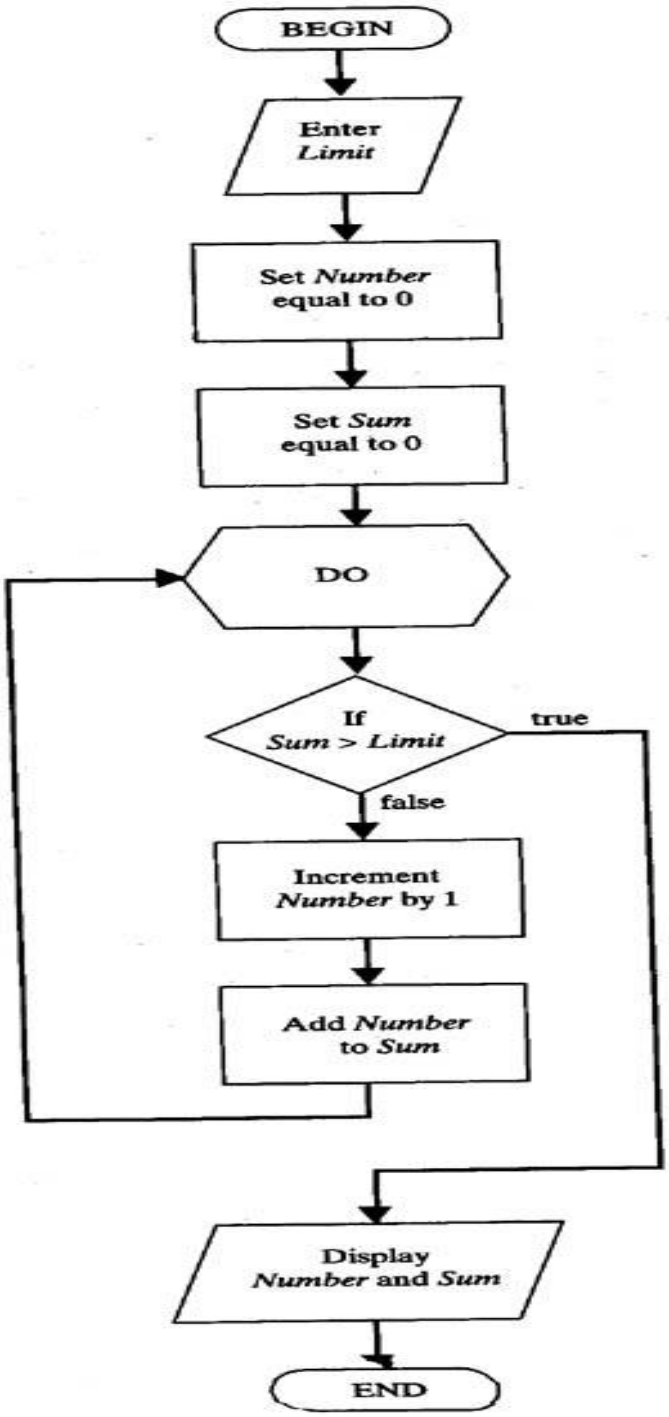
Ans. We can specify an algorithm by 3 types:

1. Using Natural language
2. Pseudocode;
3. And flowchart.

Natural language is general way to solve any problem
For e.g to perform addition of two numbers we write in natural language
Step1: read the first number (assume a)
Step2:read the second number (assume b)
Step3:add 2 numbers(a and b)and store the result in a variable c
Step4:display the result
Pseudocode is combination of natural language and programming language construct For e.g.to perform addition of two numbers we write in pseudo-code

```
//this algo perform addition of two integers
//input two integers a and b
// output of two integers
C=a+b
Write c
```

Another way to represent an algo is Flow chart:



Q.3 What are the steps needed to be followed while design and analysis an algorithm?

Ans. Steps are needed to be followed:
1.understand the problem

2.Decission making on capability of computational device,select appropriate method,Data structure, Algo Strategies.

3.Specificification of algorithm.4. Algo verification

5.analysis of algo

6. Implementation or coding of algo.

Q.4 What is analysis of an algorithm?

Ans. Algorithm analysis is an important part of a computational complexity theory.

There two types of analysis:

1.priori

2.posteriory

Priori Analysis is the theoretical estimation of resources required.On the otherhand posteriory Analysis done after implement the algorithm on a target machine.

Q.5 What is the complexity of an algorithm?explain time and space complexity?

Ans. Complexity (or efficiency) of an algorithm is a function that describes the efficiency of an algorithm in terms of time and space.
There are two types of complexity:
1.Time complexity:
Time complexity means the amount of time required by an algorithm to run. It expressed as an order of magnitude
2.space complexity:

Space complexity means the amount of space required by an algorithm to run.It expressed as an order of magnitude.There are two important factors to compute the space complexity:constant and instant characterstic.
e.g

```
factorial(n)
if n=0 then
  return 1;
else
return*factorial(n-1);
endif
```

space complexity of program = constant size + variable size

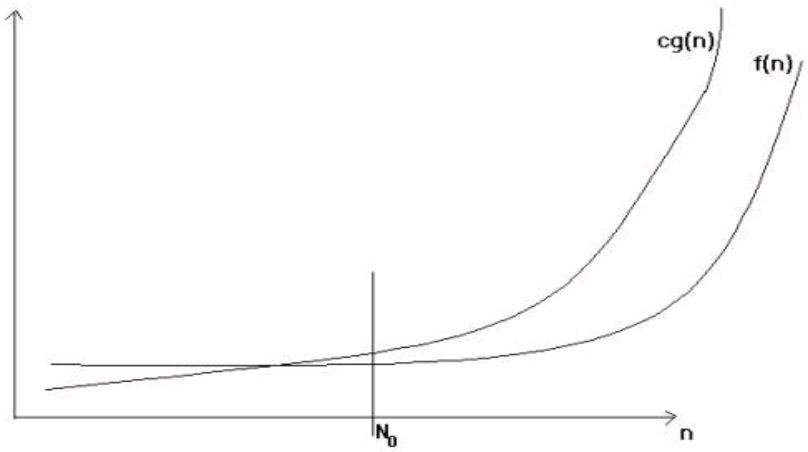
Q.6 What is performance analysis?

Ans. Performance analysis is the criteria for judging the algorithm.It have direct relationship to performance.When we solve a problem ,there may be more then one algorithm to solve a problem,through analysis we find the run time of an algorithms and we choose the best algorithm which takes leser run time.

Q.7 Explain Asymptotic notations?

Ans. Asymptotic notation is a way to represent the time complexity.
Types of Asympotatic notation:

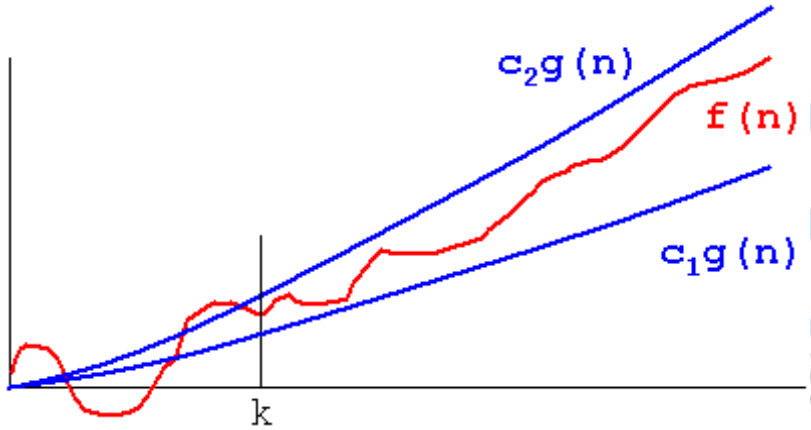
1.Big-O Notation



"The function f(n)=O(g(n)) if there exist positive constants c and n0 such that f(n)< c*g(n) for all n where n>n0" i.e, f(n)=O(g(n))

It is useful when we analyze the efficiency of an algorithm.

2.Theta Notations(Θ)



The function g(n)=f(n) iff there exists positive constants c1,c2 and n0 such that c1 g(n)≤f(n)≤c2 g(n) for all n≥n0

3.Omega notation(Ω)

The function (f(n))=Ω(g(n)) iff there exists positive constant c and n0 such that 0*c(g(n)≤ f(for all n≥n0)
Here is an example n^3 belongs to Ω(n^2)
N^3≥n^2 for all n≥0
i.e., we can select c=1 and n0=0
It shows the best case complexity of any algorithm

Q.8 Explain Recurrence Relation?How to analyze time efficiency of Recursive Algorithms?

Ans. Reccurence is a function that call itself many times to solve a particular problem .Recursive Relation are recursive definitation of

functions(mathematical).when an algorithm contain a recursive call to itself,its running time can be described by a recurrence.
Recurrence Relation is an equation that describes a function in the terms of its value on smaller inputs.There are three ways to solve Recurrence relation:
Iteration method
Substition method
Master Theorem method

Q.9 Genaral plan for time efficiency of Recursive Algorithms :

Ans. 1.Decide the parameters indicating the input`s size.
2.Identify the basic operations of an algorithm.
3.check how many number of times the basic operation is executed can vary on different inputs of the same size;If it can, the worst case,average case and best case must be calculate separately.
4.Solve the recurrence or at least determine the order of growth of its solution.

Q.10 Write a recursive procedure to compute the factorial of a number.

Ans.
```
Int Factorial(*int n)
{
If(n=1)then
Return 1;
Else
Result=n*Factorial(n-1);
}
```

Unit II
Elementary Data Structure

Q.1 What is a stack?Discuss its implementation.

Ans. A stack is a data structure for storing items which are to be accessed in Last In First Order.Data can be added or removed from only top.
Example: The tennis balls in thes container. One cannot remove 2 balls at the same time.
Data is added to the stack using the PUSH operation and rsemoved using POP operation.
To implement a stack using a one dimension array :
When we use one dimention array to implement Stack,the data is simply stored in the array. Top is an integer value that contain the array indec for the top of the stack. Each time data is added or removed, top is incremented and decremented accordingly.
Stack implemented using array :
are useful when a fixed amount of data is to be used.Using regular recursion, each recursive call pushes another entry onto the call stack. When the recursion is completed, the stack then has to pop each entry off all the way back down.

Q.2 What is a queue?

Ans. Aqueue is a data structure The bullet in a machine gun.(we cannot fire 2 bullets at the same time)
Basic operations:
Put (EnQueue): store a data item at the end of the queue
Get (DeQueue): retrieve a data item from the beginning of the queue

Q.4 Explain Tree, Binary tree and its Analysis?

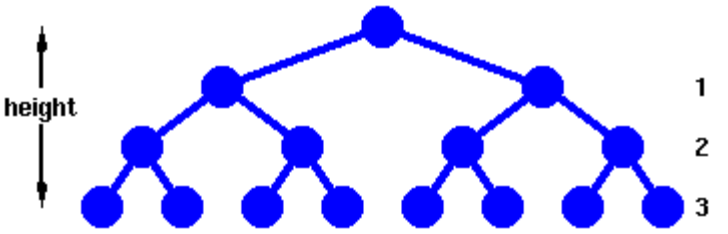
Ans. A tree is an undirected graph with no cycles and a vertex chosen to be the root of the tree.
Binary Trees
The simplest form of tree is a **binary tree**. A binary tree consists of
a. a root node and

b. left and right sub-trees.
Both the sub-trees are themselves binary trees.

Analysis

Complete Trees

each leaf has the same 'distance' from the root.



A complete tree

First, we need to work out how many nodes, ***n***, we have in such a tree of height, ***h***.

Now,

$n = 1 + 2^1 + 2^2 + ... + 2^h$
From which we have,
$n = 2^{h+1} - 1$
and
$h = \mathbf{floor(}\ \log_2 n\ \mathbf{)}$

In the worst case, $h+1$ or **ceiling(** $\log_2 n$ **)** comparisons are needed to find an item. This is the same as for binary search.

However, Add also requires **ceiling(** $\log_2 n$ **)** comparisons to determine where to add an item. adding the item takes a constant number of operations, so we say that a binary tree requires **O(log*n*)** operations for *both* adding and finding an item.

Deletion is also an **O(log*n*)** operation.

General binary trees

generaly addition of items to an ordered tree will not produce a complete tree. The worst case occurs if we add an ordered list of items to a tree.

Unit: III
Divide and Conquer

Q.1 What is divide and conquer algorithm?

Ans. Divide and conquer approach is a technique of designing algorithms,in which each problem divided into sub problems.If possible divide each sub problem in smaller problem,till a subproblem has a direct solution or can not sub divided further.It is a recursive approach.
e.g Merge sort(best example of divide conquer Algorithm).

Q.2 Analyze binary search algorithm?

Ans. Binary search is searching algorithm by which we can efficiently search the desired element.

```
int bSearch(int[] array, int value, int left, int right) {
 if (left > right)
return -1;
  int middle = (left + right) / 2;
if (array[middle] == value)
return middle;
else if (array[middle] > value)
return binarySearch(array, value, left, middle - 1);
 else
return binarySearch(array, value, middle + 1, right);
}
```

Complexity
Since each comparison binary search uses halves the search space, so binary search takes $O(\log N)$ comparisons to find the target value.

Q.3 Explain merge sort and its algorithm and analysis.

Ans. Merge sort is based on the divide-and-conquer approach. It is three step process:

1. Divide: divide the list of items into halves
2. Conquer: Merge the Partitioned element/object
3. Combine: Combine the elements back by merging the two sorted subarrays into a sorted sequence.

Merge Sort algorithm

Input:An array of n elements.
Output:sorted array in non decreasing order.
mergesort(A,1,n)
Prodecure m_sort(Low,High)
If Low < High then
Mid=(Low+High)/2
m_sort(A,Low,Mid)
m_sort(A,Mid+1,High)
m_(A,Low,Mid,High)
Endif

Merge Sort :Analysis

Mergesort goes through the same steps - independent of the data.

Best-case = Worst-case = Average-case.

T(n)= running time on a list on n elements.

T(1) = 1

T(n)= 2 times running time on a list of n/2 elements + linear merge

T(n)= 2T(n/2)+ n

and there are k=log n equations to get to T(1):

T(n) = nT(1)+nlog n = nlog n + n = O( )

Complexity : theta(n log n) for Best, Worst and Average Case

Q.4 Explain Quick sort algorithm and its analysis.

Ans. Quick sorting is a sorting algorithm that uses divide and conquer approach.

Algorithm:

Quick(A[0…n-1],low,high)
If(low<high) then
M=partition(A(low…high))
Quick(A[low….m-1])
Quick(A[mid+1…high])
In above algorithm partition performs arrangements of algorithms in asending order.
Analysis:
1. T(N) = T(i) + T(N - i -1) + cN
The time to sort the file is equal to
- the time to sort the left partition with i elements, plus
- the time to sort the right partition with N-i-1 elements, plus
- the time to build the partitions

worst-case: T(N) = O(N²)
best-case: T(N) = O(NlogN)
average-case: T(N) = O(NlogN)

Unit: IV
Graph Algorithm

Q.1 Explain Graph Traversal and Its techniques?

Ans. **Graph traversal(search)** means visiting all the nodes in a graph in a particular manner. Tree traversal is a special case of graph traversal. In graph traversal, each node may have to be visited more than once, and a root-like node that connects to all other nodes might not exist**.**
There are two widely used graph traversal techniques:
Depth-First Search (DFS)
Breadth-First Search (BFS)

Q.2 Explain DFS algorithm and its analysis?

Ans. A Depth First Search is a technique for traversing a finite undirected graph. DFS visits the child nodes before visiting the sibling nodes, that is, it traverses the depth of the tree before the breadth. Usually a stack is used in the search process.

```
DFS(v)
if (v is unvisited)
        mark v as "visited"
        for each neighbor n of v do
                DFS(n)
        end for

// traverse entire graph, connected or not
// assume instance menthod in Graph class
DFS_all(G)
mark all vertices as unvisited
 for each vertex v in V(G)
DFS(v)
 end for
```

Time Complexity:

- Every node is visited once.
- Therefore, the time of DFS is O(n+|E|).
- If the graph is connected, the time is O(|E|) because the graph has at least n-1 edges, and so n+|E| <= 2|E| -1, implying that n+|E) is O(|E|).

Application of DFS:
Minimum spanning tree

Q.3 Explain BFS algorithm and its analysis?

Ans. A BFS(Breadth First Search) is another technique for traversing a finite undirected graph. BFS visits the sibling nodes before visiting the child nodes. Usually a queue is used in the search process.

```
BFS(v_src)

if (v_src is unvisited)

        create an empty queue

        add v_src into the queue

        while queue is not empty
                v = first vertex in the queue (dequeue)
                for each neighbor n of v
                        add n to the queue
                end for
        end while

// to traverse entire graph, connected or not
// instance method for Graph class
BFS_all()
mark all vertices "univisited"
for each vertex v in the graph
BFS(v)

[ new version from Wed-Apr-06 ]

// do the checking (unvisited) before BFS is called on a vertex, and
```

```
//                  before a vertex is added to the queue

//                  (at time of addition, vertex should be unvisited)

BFS(v_src)
create an emppty queue q

visit v_src

add v_src to q

while q is not empty
        v = q.removeFirst

        for each unvisited neighbor n of v
                visit n
                add n to q

BFS_all()
unvisit all vertices

for each unvisited vertex v
BFS(v)
```

Time Complexity:

- Every node is visited once.
- Therefore, the time of BFS is O(n+|E|).

Applications of BFS:

Connectivity and spanning trees are easily done with BFS much as with DFS.

Unit V
Greedy Algorithms

Q.1 Explain Greedy Algorithm.

Greedy algorithm:

Ans. It is simple,quite efficient algorithm to solve optization problem.
According greedy algorithm candidate select the highest feasible or optimal solution that becomes the final solution and partial solution updated accordingly
Various applications of greedy algorithm:

- Knapsack problem,Prim`algorithm for minimum spanning tree,
- Kruskal`s algorithm for minimum spanning tree,
- finding shortest path etc.

Greedy (partialsolution p,subproblem s)

1. Generate all candidate choices as list ls for current subproblem
2. While(ls is not empty or other finish condition not met)
3. Compute the fesible value of each choice in ls
4. Modify p and s by taking choice in ls
5. Update ls according to p and s
6. Endwhile
7. Return the resulting complete solution

Q.2 Explain the Fractional Knapsack problem and its algorithm and analysis.

Ans. Knapsack problem also called racksack problem. In knapsack problem the object with minimum weight is choosen ,so that maximum profit can be made by such selection.
In Fractional Knapsack problem The setup is same, but the thief can take fractions of items, meaning that the items can be broken into smaller pieces so that thief may decide to carry only a fraction of x_i of item i, where $0 \leq x_i \leq 1$.
algorithm FractionalKnapsack(S,W):
Input: Set S of items i with weight wi and benefit pi all positive.
Knapsack capacity W>0.
Output: Amount xi of i that maximizes the total benefit without exceeding the capacity.

```
for each i in S do
 xi ← 0       { for items not chosen in next phase }
vi ← pi/wi    { the value of item i "per pound" }
w ← W          { remaining capacity in knapsack }

 while w > 0 do
remove from S an item of maximal value   { greedy choice }
 xi ← min(wi,w)  { can't carry more than w more }
w ← w-xi
```

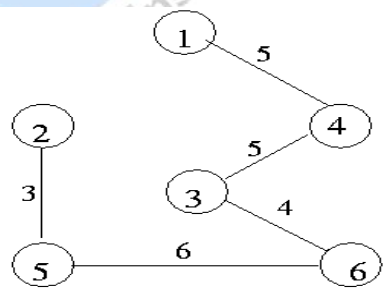
Fractional Knapsack time complexity is O(NlogN) where N is the number of items in S.

Q.3 Explain Minimum cost tree spanning problem?

Ans. A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.An MST of graph is a spanning tree of graph having a minimum cost.
MST properity :Suppose$G = (V, E)$is a connected graph with costs defined on all $e \in E$. L**et** U be some proper subset of V.

If (u, v) is an edge of lowest cost such that $u \in U$and $v \in V - U$, then there exists an MST that includes (u, v) as an edge.

e.g



A spanning tree with cost = 23

Q.4 Explain Dijkstra`s algorithm to solve Single source shortest path problem.

Ans. Dijkstra's algorithm, is a greedy algorithm that solves the single-source shortest path problem for a directed graph with non negative edge weights. For example, if the vertices of the graph represent cities and edge weights represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between two cities.

Complexity

n represents the number of nodes and *m* represents the numbers of edges.

complexity for a Dijkstra's algorithm O((n+m) log n

if we have a strongly (complete) connected graph. That is, a graph with n^2 edges, then the complexity of Dijkstra would be O(n^2 log n)

Unit VI
Dynamic Programming

Q.1 What is Dynamic programming?

Ans. Dynamic programming is efficient algorithm that is also applied to optimization problems.It uses principle of optamility. It considers all possible sequences to obtain the optimum solution.

Applications of Dynamic programming

- Multistage graph
- Travelling salesperson problem
- 0/1 knapsack problem
- Finding shortest path
- Optimal Binary search Tree.

Q.2 Explain 0/1 knapsack problem?

Ans. 0/1 knapsack problem is similar to fractional knapsack problem .In 0/1 knapsack problem we not take a fraction of an object. Suppose 'N' objects are given with weight Wi and profits Pi,where I varies from 1 to N and also knapsack have capacity 'M'.We have to fill the knapsack with the 'N' objects and the resulting profit has to be maximum.

Q.3 Explain travelling sales person problem?

Ans. Travelling sales person problem:

I f there are n number of cities and explorer wants to travel from city A to any other city B is given, then he have to obtain the cheapest round trip such that each city visited exactly once and at last he returns to the starting city A.

Unit VII
Backtracking

Q.1 What is Backtracking?

Ans. Backtracking: It is the fastest method of finding all (or some) solutions of computational problems. In this technique we search for optimal solution which satisfies some constraints and the search is refined at each stage by eliminating th possible non- generating solutions.We use backtracing to solve 8-Queen`s problem, Sum of subset problem,Hamiltonian cycle problem. etc.

Q.2 Explain 8-Queen`s problem?

Ans. The problems is to place 8 queens on the chessboard so that no two queens can be placed on the same row ,colum and diagonal. If we want to know all possible solutions then only way to solve the all problem is backtracking. For 8-queen we have 92 distinct solution.

1	2	3	4	5	6	7	8
					*		
			*				
						*	
*							
							*
	*						
				*			
		*					

Q.3 Explain sum of subset problem?

Ans. Sum of subset problem is similar to the knapsack problem but in sum-of-subcet we not worry about cost and we want to find all subsets with total weights the equal to the weight limit.
An instance of the Subset Sum problem is a pair (S, t), where S = {x1, x2, ..., xn} is a set of
positive integers and t (target) is a positive integer. The decision problem asks for a subset
of S whose sum is larger(as large as possible), but not larger than t.
This problem is NP-complete.

Q.4 Explain graph coloring problem?

Ans. Graph coloring is a way of colour different vertices one by one so that no to vertices share the same color.
example: let G(V,E) where V={1,2,3,4} and E ={(1,2),(1,3),(2,3),(2,4),(3,4)} and suppose that k=3(Red,Green,Blue).
Valid coloring c of g is:c(1)=R, c(2)=G; c(3)=B,c(4)=R.

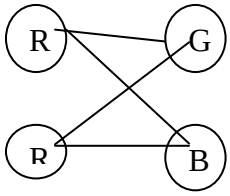


Figure: Graph coloring

Q.5 Explain Hamiltonian cycle?

Ans. Hamiltonian circuit of a graph is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.

Unit VIII

Parallel Models

Q.1 What is the Parallel Processing

Ans. It is the ability to perform multiple operation or task simultaneously by dividing the task among the executers. This activity is done in computer by CPUs or Cores using the *shared / distributed memory* to make programs run faster. Each Core / Process is equally capable / responsible for managing the flow of work through the system.

There are several parallel programming models in common use few of them are :

1. **Shared Memory**

>> In the Shared memory all processors share a single view of data and the communication between processors can be as fast as memory accesses to a same location hence in this model all the parallel task shared a common address space which they read and write to asynchronously. Because sharing the common space may be error-prone so locks / semaphores may be used to control access to the shared memory.

1. **Threads**

>> In this model a single process further n number of concurrent paths, these concurrent path are called threads

Ex- Main.exe -- Single process
Call proc1() ----- Thread 1 (T1)
Call Proc2() ----- Thread 2 (T2)
Do i..n ----- Thread 3 (T3)
A(i) = mul(1,2)
B(j) = div(4,2)
End do
Call proc3() ----- Thread 4 (T4)
Call proc4()----- Thread 5 (T5)

1. **Distributed Memory / Message Passing**

>> Each computer is having its own physical memory, In this model one computer is using the other's memory and the tasks exchange data by sending and receiving messages and a send operation must have a matching receive operation

Computer 1 NETWORK Computer 2

Task0 Task 1
Data ----------→ Data

Send

Task 2 Task3
Data ←---------- Data
Receive

1. Data Parallel

>> Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure. Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".

Do i=1,25 Do i=26,50 Do i=51,75 ---- Do i=m,n
K(i) = J(i)*m K(i) = J(i)*m K(i) = J(i)*m ---- K(i) = J(i)*m
End do End do End do ----
End do

TASK1 TASK2 TASK3 ----
TASK n

Q.2 How to measure performance of Parallel Processing.

Ans. We can measure the performance of Parallel Processing in following way -
Suppose :- Number of Processers are – P
Time to solve of a problem on P processors – Tp
Time to solve of a problem on Single processor (sequential time) – T1

How much faster execution on P processors than 1 processor --
Speedup(p) = T1/Tp
Efficiency(p) = Speedup(p)/P

Let T*(n) be the time complexity of a sequential algorithm to solve a problem P of input size sn

Let T_p (n) be the time complexity of a parallel algorithm to solves P on a parallel computer with p processors

Speedup

S_p (n) = T*(n) / T_p (n)

S_p (n) <= p

Best possible, S_p (n) = p

when T_p (n) = T*(n)/p

Efficiency -- E_p (n) = T_1 (n) / (p T_p (n))

where T_1 (n) is when the parallel algorithm run in 1 processor

Best possible, E_p (n) = 1

Speedup and efficiency are the most common ways to report the performance of a parallel algorithm

Q.3 Describe the complexity of Parallel Algorithms.

Ans. Complexity of an algorithm is determined by how fast and efficiently it's executed and it is a important measure of performance. In the case of parallel algorithms the complexity is express in terms of how fast (Time complexity) and how much resources it's uses when it runs. These criteria can be expressed as :-

- Run Time – The run time of a parallel algorithm is the length of the time period

between the time the first processor to begin execution starts and the time the last processor to finish execution terminates. the run time is usually obtained by counting the number of steps in the algorithm.

- Number of processors the algorithm uses to solve a problem.
- Cost – It is basically the total number of steps executed collectively by all processors. If the cost of an algorithm is *C,* the algorithm can be converted into a sequential one that runs in O(C) time on one processor. A parallel algorithm is said to be *cost optimal if its* cost matches the lower bound on the number of the sequential operations to solve a given problem within a constant factor. It follows that a parallel algorithm is not cost optimal if there exists a sequential algorithm whose run time is smaller than the cost of the parallel algorithm.

Q.4 What are the models used for Parallel Computation?

Ans. The Sequential algorithms uses the Random-Access-Machine model in which single processor connected to a memory system and each basic CPU operation, including arithmetic operations, logical operations, and memory accesses, requires one time step. Whereas in Multiprocessor module there are more then one processors and it is classified in three broad category

- **Local memory machine models**

It is consists of a set of n processors each with its own local memory and these processors are attached to a common communication network.

- **Modular memory machine models**

It is consists of m memory modules and n processors all attached to a common network

- **Parallel random-access machine (PRAM) models**

Ii is consists of a set of n processors all connected to a common shared memory and a processor can access any word of memory in a single step, however accesses can occur in parallel, in a single step, every processor can access the shared memory.

When multiple processors simultaneously make a request to read or write to the same resource such as a processor, memory module, or memory location—there are several possible ways

>> A PRAM model which allow exclusive-read or exclusive-write to each memory
location called (EREW) PRAM model.

>> A PRAM model which allow concurrent-read and concurrent-write to each memory
location called (CRCW) PRAM model

>> A PRAM model which allow concurrent but only exclusive writes to each memory
location called (CREW) PRAM model

Q.5 Describe the Divide-and-conquer parallel algorithmic technique

Ans. This is the technique of splitting of a problem into small independent subproblems and solving them in parallel and merges the solution to the

subproblems to construct a solution to the original problem. This improve program modularity and efficiency of algorithm.

Mention the Arithmetic computation like addition / multiplication / Division on parallel algorithm?

Addition :-

Lets consider an Algorithm for addition of elements of an Array - R

ADD(R)

if |R| =1 then return R[0]
else return ADD({R[2i]+R[2i+1] : i is the element of [0 …. |R|/2)})

Each + node adds the two values that arrive on its two incoming arcs, and places the result on its outgoing arc. The sum of all of the inputs to the circuit is returned on the single output arc at the bottom.

the total number of small-small independent process (work)

$$W(i) = W(i/2) + O(i)$$

and total number of Steps involved for final solution (Depth)

$$D(i) = D(i/2) + O(1)$$

hence the solution $W(i) = O(i)$
and $D(i) = O(\log I)$

Multiplication :-

Lest Consider Two Matrix X and Y and their result in Z
hence Z=X x Y

Using the method Divide-and-conquer, first it would be divide directly the computation of Z in
as many parts as processors can be used. Matrices X and Y are accessed only for reading their elements and that matrix Z is only accessed for writing its elements and all the data stored in Shared Memory.

Algo.

```
MATRIX-MULTIPLY(A, B, C, s)

-- X, Y: matrices to multiply
-- Z: result matrix
-- k: matrices size
{
MATRIX-MULTIPLY(X00, Y00, Z000, k/2); -- (1)
MATRIX-MULTIPLY(X01, Y10, Z100, k/2); -- (2)
MATRIX-MULTIPLY(X00, Y01, Z001, k/2); -- (3)
MATRIX-MULTIPLY(X01, Y11, Z101, k/2); -- (4)
MATRIX-MULTIPLY(X10, Y00, Z010, k/2); -- (5)
...
...
MATRIX-MULTIPLY(Xnn, Ynn, ZNnn, k/2); -- (n)
}
Z00 = Z000 + Z100;
Z01 = Z001 + Z101;
Z10 = Z010 + Z110;
...
...
Znn = ZN-1nn + ZNnn;
}
```

Now if the dimension of Matrix X,Y,Z are same then it have $O(n^3)$ work and O(log n), due to the depth of the summation

Division :-

```
DIVI(op1 by Op2)
(
Op3 = op1/Op2
}
```

the total number of small-small independent process (work)

W(i) = W(1)

and total number of Steps involved for final solution (Depth)

D(i) = O(1)

hence the solution W(1) = O(1)

and D(i) = O(1)

Q.6 Write a short note on Parallel Evaluation of General Arithmetic Expressions

Ans. All arithmetic expressions with n>= 1 (where n is natural number) variables and constants, arithmetic operations "+","*","/" can be evaluated in $4\log_2 n - 1 + 10(n-1)/p$ time having p>= 1 processors which can independently perform arithmetic operations in unit time.

Let E is the arithmetic expression with n distinct atoms and F and G are expression and suppose that n>=3 , for otherwise the result is trivial.

Hence F and G can be evaluated in :-
$[4\log_2(n-1)] - 2 < 4\log_2 n - 1$ with less then 10(n-1) operations
Now we see that F and G operation can be performed in $4\log_2(n-1)] - 2$ time
now 10(n-1) operations can be performed on p processors
$E = 4\log_2 n - 1 + 10(n-1)/p$ time.

Q.7 What is First-Order Linear recurrences ?

Ans. The continued functions and closely related to linear recurrence relations and can be defined using the composition of linear fractional transformations. A recurrence equation is one
that tells us how to compute the *n*th term of a sequence from the $(n-1)$st term or some or all the preceding terms.

If *n*-th term of a sequence can be expressed as a function
$x_n = k_n x_{n-1} + Z_n$ (Where n=1,2,3....)
then this equation is called First-Order recurrence relation. The values *k*1, *k*2, ... must be explicitely
given. They are called initial conditions. Recurrences can be solved by iteration (also called *telescoping*)

Key Terms

Algorithm Terminology

Order of growth is only a basic measure of the resources required. A process which requires n steps and another which requires 1000n steps have both the same order of growth O(n).

Backtracking is a strategy for finding solutions to constraint satisfaction problems. Cstraint satisfaction problems are problems with a complete solution, where the order of elements does not matter. The problems consist of a set of variables each of which must be assigned a value, subject to the particular constraints of the problem. Backtracking attempts to try all the combinations in order to obtain a solution. Its strength is that it can be implemented to avoid trying many partial combinations, thus speeding up the running-time. Backtracking is closely related to combinatorial search.

Data structures is a way of storing data in a computer so that it can be used efficiently. Often a carefully chosen data structure will allow a more efficient algorithm to be used. The choice of the data structure often begins from the choice of an abstract data structure. A well-designed data structure allows a variety of critical operations to be performed, using as little resources, both execution time and memory space, as possible.

Debugging is a methodical process of finding and reducing the number of bugs, or defects, in a computer program or a piece of electronic hardware thus making it behave as expected.

Divide-and-conquer is an important algorithm design paradigm. It works by recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

Dynamic programming is a method for reducing the runtime of algorithms exhibiting the properties of overlapping subproblems and optimal substructure.

Greedy search follow the heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum. For instance, applying the greedy strategy to the traveling salesman problem yields the following algorithm: At each stage visit the unvisited city nearest to the current city.

Balanced Binary Tree
Binary tree in which each leaf is the same distance from the root.

Iterations is the repetition of a process, typically within a computer program.

Shortest path
It is the path from the source node to the destination node such that the sum of the labels on each path is minimum. For example, the label may be simply the physical distance between nodes.

Optimality requirements
A candidate solution being considered should have possibility of being an optimal solution. This constraint on the possible candidate solutions is called optimality requirements.

Problem definitionis a description of software or hardware that may be used to develop an implementation. It describes what the system should do, not (necessarily) how the system should do it.
Upper boundof a subset S of some partially ordered set is an element which is greater than or equal to every element of S.

Valid solutionmeets all the requirements of a problem specification.

Root Node
Node at the "top" of a tree - the one from which all operations on the tree commence. The root node may not exist (a NULL tree with no nodes in it) or have 0, 1 or 2 children in a binary tree.
Leaf Node

Node at the "bottom" of a tree - farthest from the root. Leaf nodes have no children.

Complete Tree

Tree in which each leaf is at the same distance from the root. A more precise and formal definition of a complete tree is set out later.

Height
Number of nodes which must be traversed from the root to reach a leaf of a tree.

Graph terminology
Graph a collection of distinct vertices (nodes) and distinct edges (links) G = {V, E} where V is a set of all (distinct) vertices, E is a set of all (distinct) edge

Vertices simple (node) objects that can have labels (names) and other properties

Edges connections between two vertices, often have name, distance, cost, time, etc. associated

Path a sequence of edges connecting from vertex v_i to v_j

Connected Graph: a graph is connected if there is a path from every vertex to every other vertex in the graph

Cycle a simple path that starts and ends at the same vertex

Tree : A graph with no cycles

undirected graph : A graph whose edges are not directed (assume connection goes both ways) directed graph a graph whose edges are directed with an arrow in one or both ends

adjacent (neighbor) : Vertex v_i is adjacent to v_j if there is an edge (directed or undirected) connecting from v_j to v_i
Adjacent vertices are also called neighbors.

Spanning tree

A *spanning tree* of a graph is just a subgraph that contains all the vertices and is a tree. A graph may have many spanning trees; for instance the complete graph on four vertices

Tree - a non-empty collection of nodes & edges

Node - can have a name and carry other associated information

Path - a list of distinct nodes in which successive nodes are connected by edges. Any two nodes must have one and only one path between them, else it is not a tree

Root - starting point (top) of the tree

Parent (ancestor) of a node A - the node "above" node A

Child (descendent) of a node A - the node "below" node A

Siblings - nodes that have same parent

Leaf (terminal node) - a node with no children

Level of a node - the number of edges between this node and the root

Depth of a node - the number of edges from the root to that node. The depth of the root is 0.

Height of a node - the largest number of edges from that node to a leaf. The height of each leaf is zero

Height of a tree - the longest path from the root to a leaf node.

Balanced tree - the difference between the height of the left subtree and the right subtree is not more than 1.

Ordered trees - The order of the children matters, i.e. children are listed in a particular order.

Tree - a non-empty collection of nodes & edges

Node - can have a name and carry other associated information

Path - a list of distinct nodes in which successive nodes are connected by edges. Any two nodes must have one and only one path between them, else it is not a tree

Root - starting point (top) of the tree

Parent (ancestor) of a node A - the node "above" node A

Child (descendent) of a node A - the node "below" node A

Siblings - nodes that have same parent

Leaf (terminal node) - a node with no children

Level of a node - the number of edges between this node and the root

Depth of a node - the number of edges from the root to that node. The depth of the root is 0.

Height of a node - the largest number of edges from that node to a leaf. The height of each leaf is zero

Height of a tree - the longest path from the root to a leaf node.

Balanced tree - the difference between the height of the left subtree and the right subtree is not more than 1.

Ordered trees - The order of the children matters, i.e. children are listed in a particular order.

Pre-order traversal:

Visit the root first then the left child then the right child

C O M P U T E R L A N D

In-order Traversal

Visit the left child then the root then the right child

P M U O C T E L R N A D

Post-order Traversal

Visit the left child then the right child then the root

P U M O L N D A R E T C

Multiple Choice Questions

1.Two main measures for the efficiency of an algorithm are

a. Processor and memory
b. Complexity and capacity
c. Time and space
d. Data and space

Ans: Time and space

2. The time factor when determining the efficiency of algorithm is measured by

a. Counting microseconds
b. Counting the number of key operations
c. Counting the number of statements
d. Counting the kilobytes of algorithm

Ans: b. Counting the number of key operations

3. The space factor when determining the efficiency of algorithm is measured by

a. Counting the maximum memory needed by the algorithm
b. Counting the minimum memory needed by the algorithm
c. Counting the average memory needed by the algorithm
d. Counting the maximum disk space needed by the algorithm

Ans: a. Counting the maximum memory needed by the algorithm

4. Which of the following case does not exist in complexity theory

a. Best case
b. Worst case
c. Average case
d. Null case

Ans: d. Null case

5. The Worst case occur in linear search algorithm when

a. Item is somewhere in the middle of the array
b. Item is not in the array at all
c. Item is the last element in the array
d. Item is the last element in the array or is not there at all

Ans: d. Item is the last element in the array or is not there at all

6. The Average case occur in linear search algorithm

a. When Item is somewhere in the middle of the array
b. When Item is not in the array at all
c. When Item is the last element in the array
d. When Item is the last element in the array or is not there at all

Ans: a. When Item is somewhere in the middle of the array

7. The complexity of the average case of an algorithm is

a. Much more complicated to analyze than that of worst case
b. Much more simpler to analyze than that of worst case
c. Sometimes more complicated and some other times simpler than that of worst case
d. None or above

Ans: a. Much more complicated to analyze than that of worst case

8. The complexity of linear search algorithm is

a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Ans: a. O(n)

9. The complexity of Binary search algorithm is

a. O(n)
b. O(log)
c. O(n2)
d. O(n log n)

Ans: b. O(log n)

10. The complexity of Bubble sort algorithm is

a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Ans: c. $O(n^2)$

11. The complexity of merge sort algorithm is

a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Ans: c

12.Merge sort uses

a.Divide and conquer strategy
b. Greedy
c. Array
d. Link List

Ans : a

13. Which of the following data structure is not linear data structure?

a. Arrays

b. Linked lists

c. Both of above

d. None of above

Ans:d. None of above

14. Which of the following data structure is linear data structure?

a. Trees
b. Graphs
c. Arrays
d. None of above

Ans: c. Arrays

15. The operation of processing each element in the list is known as

a. Sorting
b. Merging
c. Inserting
d. Traversal

Ans: d. Traversal

16. Finding the location of the element with a given value is:

a. Traversal
b. Search
c. Sort
d. None of above

Ans: b. Search

17. Which sorting algorithn is faster :
a.O(n^2)
b.O(nlogn)
c.O(n+k)

d.O(n^3)

Ans: c. O(n+k)

19. The running time of quick sort depends heavily on the selection of
a. No of inputs
b. Arrangement of elements in array
c. Size o elements
d. pivot element

Ans: d. Pivot element

20. The elements of an array are stored successively in memory cells because

a. by this way computer can keep track only the address of the first element and the addresses of other elements can be calculated

b.To avoid this unnecessary repetitions by writing down the results of recursive calls and looking them up again if we need them later
c.To make the process accurate
d.None of the above

Ans : a

21. The most appropriate matching for the following pairs

X: depth first search 1: heap
Y: breadth-first search 2: queue
Z: sorting 3: stack

is :
(a) X—1 Y—2 Z-3
(b) X—3 Y—1 Z-2
(c) X—3 Y—2 Z-1
(d) X—2 Y—3 Z-1

Answer: (c)

21. Complexity of Kruskal's algorithm for finding the minimum spanning tree of an undirected graph containing n vertices and m edges if the edges are sorted is

a) O(mn) c) O(m)
b) O(m+n) d) O(n)

Answer: b

23. Memorization is?
a.To store previous results for future use
b.To avoid this unnecessary repetitions by writing down the results of recursive calls and looking them up again if we need them later
c.To make the process accurate
d.None of the above

Ans: b.

24.Which sorting algorithm is faster
O (n log n)
O n^2
o(n+k)
O n^3

Ans:c

25. We do sorting to,
a. keep elements in random positions
b. keep the algorithm run in linear order
c. keep the algorithm run in (log n) order
d. keep elements in increasing or decreasing order

Ans : D

26. Which sorting algorithn is faster :
a. O(n^2)
b. O(nlogn)

c. O(n+k)
d. O(n^3)
Ans : B

27.Two main measures for the efficiency of an algorithm are
a. Processor and memory
b. Complexity and capacity
c. Time and space
d. Data and space
Ans.c

28.The time factor when determining the efficiency of algorithm is measured by
a. Counting microseconds
b. Counting the number of key operations
c. Counting the number of statements
d. Counting the kilobytes of algorithm

Ans.b

29.The space factor when determining the efficiency of algorithm is measured by
a. Counting the maximum memory needed by the algorithm
b. Counting the minimum memory needed by the algorithm
c. Counting the average memory needed by the algorithm
d. Counting the maximum disk space needed by the algorithm

Ans.a.

30.Which of the following case does not exist in complexity theory
a. Best case
b. Worst case
c. Average case
d. Null case
Ans.d

31.The Worst case occur in linear search algorithm when
a. Item is somewhere in the middle of the array
b. Item is not in the array at all
c. Item is the last element in the array
d. Item is the last element in the array or is not there at all
Ans.d

33.The Average case occur in linear search algorithm
a. When Item is somewhere in the middle of the array
b. When Item is not in the array at all
c. When Item is the last element in the array
d. When Item is the last element in the array or is not there at all

Ans.a.

34.The complexity of the average case of an algorithm is
a.Much more complicated to analyze than that of worst case
b. Much more simpler to analyze than that of worst case
c. Sometimes more complicated and some other times simpler than that of worst case
d. None or above

Ans.a.

35.The complexity of linear search algorithm is
a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)
Ans.a

36.The complexity of Binary search algorithm is
a.O(n)
b. O(log)
c. O(n2)
d. O(n log n)

Ans.b

37.The complexity of Bubble sort algorithm is
a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Ans.c

38.The complexity of merge sort algorithm is
a. O(n)
b. O(log n)
c. O(n2)
d. O(n log n)

Ans.d

39.The indirect change of the values of a variable in one module by another module is called
a. internal change
b. inter-module change
c. side effect
d. side-module update

Ans.c

40. Which of the following data structure is not linear data structure?
a.Arrays
b. Linked lists
c. Both of above
d. None of above

Ans.d

41.Which of the following data structure is linear data structure?
a. Trees
b. Graphs
c. Arrays
d. None of above

Ans.a

42.The operation of processing each element in the list is known as
a. Sorting
b. Merging
c. Inserting
d. Traversal

Ans.d

42.Finding the location of the element with a given value is:
a.Traversal
b. Search
c. Sort
d. None of above

Ans.b

43.The elements of an array are stored successively in memory cells because
a. by this way computer can keep track only the address of the first element and the addresses of other elements can be calculated
b. the architecture of computer memory does not allow arrays to store other than serially
c. both of above
d. none of above

Ans.a

44.The difference between Prim's algorithm and Dijkstra's algorithm is that Dijkstra'salgorithm uses a different key.
True or False
Ans.True

45.What is the time complexity to extract a vertex from the priority queue in Prim'salgorithm?
a.log (v)
b.v.v
c.e.e
d.log(e)
Ans.a

46. Which statement is true?
a.If a dynamic-programming problem satisfies the optimal-substructure property,then a locally optimal solution is globally optimal.
b.If a greedy choice property satisfies the optimal-substructure property, then alocally optimal solution is globally optimal.
c.both of above
d.none of above
Ans.c

47. Kruskal's algorithm (choose best non-cycle edge) is better than Prim's (choose best treeedge) when the graph has relatively few edges.
a.True
b.False
Ans.a.

48. A dense undirected graph is:
a.A graph in which E = O(V^2)
b.A graph in which E = O(V)
c.A graph in which E = O(log V)
d.All items above may be used to characterize a dense undirected graph.
Ans.a

49: Suppose that a graph G = (V,E) is implemented using adjacency lists. What is thecomplexity of a breadth-first traversal of G?
a.O(|V |^2)
b.O(|V | |E|)
c.O(|V |^2|E|)
d.O(|V | + |E|)
Ans.b

50.The relationship between number of back edges and number of cycles in DFS is,
a.Both are equal
b.Back edges are half of cycles
c.Back edges are one quarter of cycles
d.There is no relationship between no. of edges and cycles

51 Heaps can be stored in arrays without using any pointers; this is due to the _____ nature of the binary tree,Select correct option:
A. left-complete
B. right-complete
C. tree nodes
D. tree leave

Ans.A

52. The appropriate big thita classification of the given function. f(n) = 4n2 + 97n + 1000 is
1. ?(n)
2. O(2^n)
3. O(n^2)
4. O(n^2logn)
Ans : 3

54. If algorithm A has running time 7n^2 + 2n + 3 and algorithm B has running time 2n^2,
then
1. Both have same asymptotic time complexity
2. A is asymptotically greater
3. B is asymptotically greater
4. None of others
Ans.1

55.What is the solution to the recurrence T(n) = T(n/2)+n .
1. O(logn)
2. O(n)
3. O(nlogn)
4. O(n^2)
Ans : 1

56. Consider the following Algorithm:
Factorial (n){
if (n=1)
return 1
else
return (n * Factorial(n-1))
}
Recurrence for the following algorithm is:
1. T(n) = T(n-1) +1
2. T(n) = nT(n-1) +1
3. T(n)= T(n-1) +n
4. T(n)=T(n(n-1)) +1
Ans: 4

57. How many elements do we eliminate in each time for the Analysis of Selection algorithm?
1. n / 2 elements
2. (n / 2) + n elements
3. n / 4 elements
4. n elements
Ans:4

58. A (an) _____ is a left-complete binary tree that conforms to the heap order
1. heap
2. binary tree
3. binary search tree
4. array
Ans: 1

59. We do sorting to,
1. keep elements in random positions
2. keep the algorithm run in linear order
3. keep the algorithm run in (log n) order
4. keep elements in increasing or decreasing order
Ans : 1

60. Dynamic programming algorithms need to store the results of intermediate sub-problems.
1. TRUE
2. FALSE
Correct Choice : 1

61. The Knapsack problem belongs to the domain of _____ problems.
1. Optimization
2. NP Complete
3. Linear Solution
4. Sorting
Ans : 1

62. Suppose we have three items as shown in the following table, and suppose the capacity of the knapsack is 50 i.e. W = 50. Item Value Weight
1 60 10
2 100 20
3 120 30
The optimal solution is to pick

1. Items 1 and 2
2. Items 1 and 3
3. Items 2 and 3
4. None of these
Ans: 3

63. An optimization problem is one in which you want to find,
1. Not a solution
2. An algorithm
3. Good solution
4. The best solution
Ans: 4

References

http://www.scribd.com/doc/81904196/81/BASIC-SEARCH-AND-TRAVERSAL-TECHNIQUE

Design And Analysis Of Algorithms by V.V. Muniswamy

Analysis And Design Of Algorithms by A.A.Puntambekar

Algorithms: Design Techniques and Analysis by M. H. Alsuwaiyel
Introduction to Algorithms by corman, Rivest,