Organizer

Data Structure & Algorithms

Computer Branch

IIIrd Sem.

Strictly as per new syllabus

Aditya Prakashan

SYLLABUS

Module 1 : Lecture 4 hrs.

Introduction : Bsic Terminologies : Elementary Data Organizations, Data Structure Operations; insertion, deletion, traversal etc.; Analysis of an Algorithm, Asymptotic Notations, Time-Space trade off.

Module 2 : Lecture 10 hrs.

Stacks and Queues : ADT Stack and its operations : Algorithms and their complexity analysis, Applications of Stacks : Expression Conversion and evaoluation corresopnding algorithms and complexity analysis. ADT queue, Types of Queue : Simple Queue, Circular Queue, Priority Queue; Operations on each Type of Queues : Algorithm and their analysis.

Module 3 : Lecture 6 hrs.

Linked Lists : Singly linked lists : Representation in memory, Algorithms of several operations : Traversing, Searching, Insertion into, Deletion from linked list; Linked representation of Stack and Queue, Header nodes, doubly linked list; operations on it and algorithmic analysis; Circular Linked Lists; all operations their algorithms and th complexity analysis.

Module 4 Lecture 12 hrs.

Searching, Sorting and Hashing : Linear and Binary Search Technique and their complexity analysis. Objective and properties of different sorting algorithms Section Sort, Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort; Performance and Comparison among all the methods, Hashing.

Module 5 Lecturs 8 hrs.

Trees : Basic Tree Terminologies, Different types of Trees; Binary Tree, Threaded Binary Tree, Binary Search Tree, AVL Tree; Tree operations on each of the trees and their algorithms with complexity analysis. Applications of Binary Trees. B Tree, B^+ Tree : definitions, algorithms and analysis.

Graph : Basic Terminologies and Representations, Graph search and traversal algorithms and complexity analysis.

CONTENTS

Module 1 :

Introduction : Bsic Terminologies : Elementary Data Organizations, Data Structure Operations; insertion, deletion, traversal etc.; Analysis of an Algorithm, Asymptotic Notations, Time-Space trade off.

Module 2 :

Stacks and Queues : ADT Stack and its operations : Algorithms and their complexity analysis, Applications of Stacks : Expression Conversion and evaoluation corresopnding algorithms and complexity analysis. ADT queue, Types of Queue : Simple Queue, Circular Queue, Priority Queue; Operations on each Type of Queues : Algorithm and their analysis.

Module 3 :

Linked Lists : Singly linked lists : Representation in memory, Algorithms of several operations : Traversing, Searching, Insertion into, Deletion from linked list; Linked representation of Stack and Queue, Header nodes, doubly linked list; operations on it and algorithmic analysis; Circular Linked Lists; all operations their algorithms and th complexity analysis.

Module 4

Searching, Sorting and Hashing : Linear and Binary Search Technique and their complexity analysis. Objective and properties of different sorting algorithms Section Sort, Bubble Sort, Insertion Sort, Quick Sort, Merge Sort, Heap Sort; Performance and Comparison among all the methods, Hashing.

Module 5

Trees : Basic Tree Terminologies, Different types of Trees; Binary Tree, Threaded Binary Tree, Binary Search Tree, AVL Tree; Tree operations on each of the trees and their algorithms with complexity analysis. Applications of Binary Trees. B Tree, B^+ Tree : definitions, algorithms and analysis.

Graph : Basic Terminologies and Representations, Graph search and traversal algorithms and complexity analysis.

1 INTRODUCTION

Chapter at a Glance

- **Data** (plural of "**datum**"), refers to qualitative or quantitative attributes of a variable or set of variables. Typically, data are the results of measurements and can be the basis of graphs, images or observations of a set of variables.
- **Metadata** is a data, containing a description of other data. Earlier term for metadata is ancillary data.
- **Data structure:** Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called data structure.
- **Classification:** Data structure can be classified as ***linear*** and ***non-linear.***
  Linear data structure: The data structure is said to be linear, if its element forms a sequence or linear list. Linear data structures organize their data elements in a linear fashion.
  Non-linear data structure: In nonlinear data structures, data elements are not organized in a sequential fashion. A data item in a nonlinear data structure could be attached to several other data elements to reflect a special relationship among them and all the data items cannot be traversed in a single run.
- Various operations can be performed on data structure. List of some frequently used operations are given below:
  1. Traversing: sometimes also called as visiting, means accessing the required record so as to process the data.
  2. Searching: finding the location of record to be processed.
  3. Inserting: adding a new record to the structure
  4. Deleting: removing the record from the structure.
  5. Sorting: arranging the record in some logical order.
  6. Merging: combining two different sets of records to form a final set of records.
- **Abstract data type (ADT):** Abstract Data Types are a set of data values and associated operations that are precisely independent of any particular implementation. The term abstract signifies that the data type will only set the rule of it's usage but how it will be used depends on the implementation. For example stacks and queues are called abstract data types. The stack data type defines two abstract methods PUSH and POP.
- **The basic properties of an algorithm** are:
  Input: Each algorithm is supplied with a zero or more external quantities.
  Output: Each algorithm must produce atleast one quantity.
  Definiteness: Each instruction must be clear and unambiguous.
  Finiteness: The algorithm must terminate after a finite number of steps within finite time.
  Effectiveness: Each instruction must be sufficiently basic and also be feasible.
- **Analysis of an algorithm** means, to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the *efficiency or running time* of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity).
- In theoretical analysis of algorithms it is common to estimate their complexity in the *asymptotic* sense, i.e., to estimate the complexity function for arbitrarily large input. **Big O**

INTRODUCTION

Chapter at a Glance

- **Data** (plural of "**datum**"), refers to qualitative or quantitative attributes of a variable or set of variables. Typically, data are the results of measurements and can be the basis of graphs, images or observations of a set of variables.
- **Metadata** is a data, containing a description of other data. Earlier term for metadata is ancillary data.
- **Data structure:** Data may be organized in many different ways. The logical or mathematical model of a particular organization of data is called data structure.
- **Classification:** Data structure can be classified as ***linear*** and ***non-linear.***
  Linear data structure: The data structure is said to be linear, if its element forms a sequence or linear list. Linear data structures organize their data elements in a linear fashion.
  Non-linear data structure: In nonlinear data structures, data elements are not organized in a sequential fashion. A data item in a nonlinear data structure could be attached to several other data elements to reflect a special relationship among them and all the data items cannot be traversed in a single run.
- Various operations can be performed on data structure. List of some frequently used operations are given below:
  1. **Traversing:** sometimes also called as visiting, means accessing the required record so as to process the data.
  2. **Searching:** finding the location of record to be processed.
  3. **Inserting:** adding a new record to the structure
  4. **Deleting:** removing the record from the structure.
  5. **Sorting:** arranging the record in some logical order.
  6. **Merging:** combining two different sets of records to form a final set of records.
- **Abstract data type (ADT):** Abstract Data Types are a set of data values and associated operations that are precisely independent of any particular implementation. The term abstract signifies that the data type will only set the rule of it's usage but how it will be used depends on the implementation. For example stacks and queues are called abstract data types. The stack data type defines two abstract methods PUSH and POP.
- **The basic properties of an algorithm** are:
  Input: Each algorithm is supplied with a zero or more external quantities.
  Output: Each algorithm must produce atleast one quantity.
  Definiteness: Each instruction must be clear and unambiguous.
  Finiteness: The algorithm must terminate after a finite number of steps within finite time.
  Effectiveness: Each instruction must be sufficiently basic and also be feasible.
- **Analysis of an algorithm** means, to determine the amount of resources (such as time and storage) necessary to execute it. Most algorithms are designed to work with inputs of arbitrary length. Usually the *efficiency or running time* of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity).
- In theoretical analysis of algorithms it is common to estimate their complexity in the *asymptotic* sense, i.e., to estimate the complexity function for arbitrarily large input. **Big O**

notation, **omega** notation and **theta** notation are used for this. Usually asymptotic estimates are used because different implementations of the same algorithm may differ in *efficiency*. However the efficiencies of any two "reasonable" implementations of a given algorithm are related by a constant multiplicative factor called a *hidden constant*.

- **Complexity**: There are two types of complexities of an algorithm, time complexity and space complexity.
  The ***time complexity*** of an algorithm is the amount of time the computer requires to execute the algorithm.
  The ***space complexity*** of an algorithm is the amount of memory space the computer requires, completing the execution of the algorithm.

Multiple Choice Type Questions

1. Each element of an array arr[20][50] requires 4 bytes of storage. Base address of arr is 2000. The location of arr[10][10] when the array is stored as column maior is '

a) 2820 **b) 2840** **c) 4048** **d) 4840**

Answer: (b)

2. In C language, malloc( ) returns

a) integer pointer **b) structure pointer**
c) null pointer **d) void pointer**

Answer: (c)

3. Which of the following is the best time for an algorithm?

a) $O(n)$ **b)** $O(\log_2 n)$ **c)** $O(2^n)$ **d)** $O(n\log_2 n)$

Answer: (b)

4. Four algo.s do the same task. Which algo. should execute the slowest for large values of n?

a) $O(n^2)$ **b)** $O(n)$ **c)** $O(\log_2 n)$ **d)** $O(2^n)$

Answer: (d)

5. In C language, arrays are stored in which representation?

a) Column major **b) Row major** **c) Layer major** **d) None of these**

Answer: (b)

6. Which of the following algorithm should execute the slowest for large values of N?

a) O (N) **b) O (N^2)** **c) O($\log_2 N$)** **d) None of these**

Answer: (b)

7. Which is better computing time (in analysis of algorithm)?

a) $O(n)$ **b)** $O(2n)$ **c)** $O(\log 2n)$ **d) None of these**

Answer: (c)

8. A dynamic data structure where we can search for desired records in $O(\log n)$ time is

a) heap
b) binary search tree
c) circularly linked list
d) array

Answer: (b)

9. For Column Major, what is the address of [3, 2] th element of a 3×4 Matrix (contains integer number)? It is given that the 'Base Address is 2000'.

a) 2010
b) 2012
c) 2016
d) 2018

Answer: should be 2020

10. Which of the following expressions access the (i, j)th entry of a (m × n) matrix stored in column major order?

a) n × (i – 1) + j
b) m × (j – 1) + i
c) m × (n – j) + j
d) n × (m – i) + j

Answer: (b)

11. Which of the following is non-linear data structure?

a) Stacks
b) List
c) Strings
d) Trees

Answer: (d)

Short Answer Type Questions

1. Let the size of the elements stored in an 8 × 3 matrix be 4 bytes each. If the base address of the matrix is 3500, then find the address of A [4, 2] for both row major & column major cases.

What is sparse matrix?

OR,

What are sparse matrices? How such a matrix is represented in memory? What are the types of sparse matrices?

Answer:

1st Part:

In row major order the address will be

$$\text{Loc}(A(i,j)) = \text{Lo} + ((i-1) * n + (j-i)) * c$$

Therefore

$$\text{Loc}(A(4,2)) = 3500 + ((4-1) * 3 + (2-1)) * 4 = 3500 + (10) * 4$$
$$= 3500 + 40 = 3540$$

In column-major order the address will be

$$\text{Loc}(A(i,j)) = \text{Lo} + ((j-i) * m + (i-1)) * c$$

Therefore

$$\text{Loc}(A(4,2)) = 3500 + ((2-1) * 8 + (4-1)) * 4 = 3500 + (11) * 4$$
$$= 3500 + 44 = 3544$$

2nd Part:

A matrix is represented in memory as 2D array, like A [i, j], where i is a row and j is a column.

If there are m rows and n columns, then total elements in a matrix or size of a matrix is m x n elements. This m x n elements requires m x n units of storage.

A matrix, that is filled with mostly zeroes (more than 2/3 elements are zero), is called as a sparse matrix.

Now, if there is a sparse matrix of order m × n, and if we use a 2D array of order m × n to store this matrix, then there is basically wastage of large amount of memory.

Now, a sparse matrix can be stored as a list of three tuples of the form (i, j, value).

In this way we can store only non-zero elements.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 30 \end{bmatrix}$$

Let us consider a matrix M(5×6)

Here the number of non-zero elements t = 5.

The array A [0...t, 1...3] can store all the non zero elements as given below:

	1	2	3
A[0]	5	6	5
A[1]	1	5	1
A[2]	2	1	2
A[3]	3	2	−3
A[4]	4	4	7
A[5]	5	6	30

The element A [0, 1] and A [0, 2] contain the number of rows and columns of the matrix. A[0, 3] contains total number of non-zero items.

2. Show that the function $f(n)$ defined by:

$$f(1) = 1$$

$$f(n) = f(n-1) + \frac{1}{n} \text{ for } n > 1$$

has the complexity O (log n).

Define Big – O, Ω, θ notations. Explain the conceptual differences among these three representatives.

And, Define Big O notation.

Answer:

1st Part:

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

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

$= f(n-3)+1/(n-2)+1/(n-1)+1/n$

.....

.....

$= f(2-1)+1/2+1/3+.....+1/(n-1)+1/n$

$= 1/1 + 1/2 +1/3 +.....+ 1/n$

These types of numbers are called Harmonic numbers.

We can evaluate this type of series just by integrating $1/x$ from $½$ to $n + ½$.

After integrating, we get the result as $\ln(n + ½) - \ln ½$

$$\approx \ln n - 0.7$$

Hence, we can write the complexity as O(log n).

Big O notation:

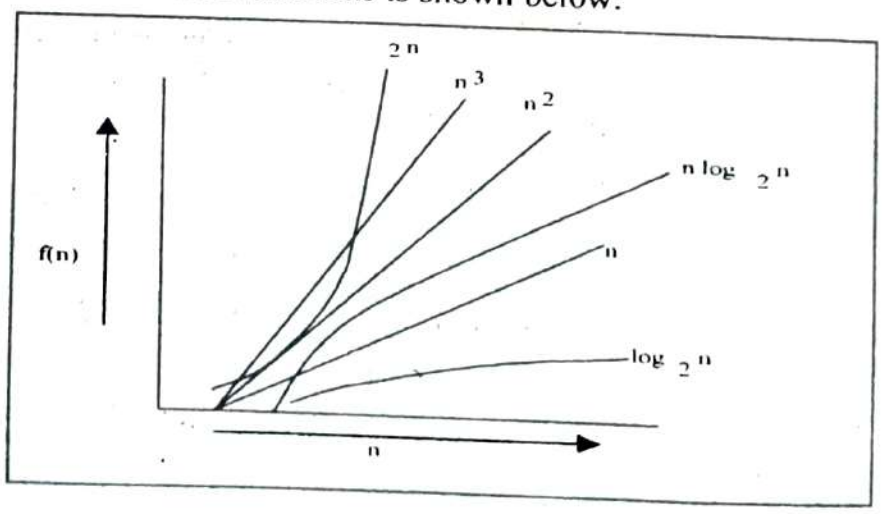
The Big Oh (the "O" stands for "order of") notation is used to classify functions by their asymptotic growth function and hence finding the time complexity of an algorithm.

There are two types of complexities of an algorithm, time complexity and space complexity. The time complexity of an algorithm is the amount of time the computer requires to execute the algorithm. The space complexity of an algorithm is the amount of memory the computer needs to run to complete the execution of the algorithm. The algorithms are compared based on their performances. The performance is measured in terms of time complexity & space complexity. Based on the nature of the input, time complexity can be of three different types: best case, average case and worst case.

The different computing functions are measured as:

$$n, n^2, n^3, \log_2 n, n\log_2 n, 2^n.$$

The rate of growth of all these functions is shown below:



Ω:
Let $f(n)$ and $g(n)$ be functions, where n is a positive integer. We write $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$

θ:
Let $f(n)$ and $g(n)$ be functions, where n is a positive integer. We write $f(n) = \theta(g(n))$ if and only if $g(n) = O(f(n))$. and $g(n) = \Omega(f(n))$.

3. Derive values related to the average case and worst case behaviour of Bubble Sort algorithm. Also, confirm that the best case behavior is O(n).

OR,

Write an algorithm for sorting a list numbers in ascending order using bubble sort technique and find its time complexity.

Answer:

The algorithm for bubble sort is given as follows:

```
Procedure: BubbleSort(List[])
 Inputs: List[] - A list of numbers
 Locals: i, j - integers
 Begin:
    For i = 0 to List.Size-1
       For j = i+1 to List.Size-1
          If List[i] > List[j], Then
             Swap List[i] and List[j]
          End If
       Next j
    Next i
```

Analysis:

The size of the sorting problem is related to the size N of the list. As N increases, we expect the execution time to increase as well.

The innermost instruction is to swap two list elements. Regardless of how long the list is, this always takes the same amount of time, and so the swap instruction is O(1) with respect to N. Surrounding that instruction is an if statement, which either executes its body, or does not; in the worst case, it will execute its body. Again, the if statement is O(1).

The if statement is executed as many times as the inner loop iterates, and the inner loop executes as many times as the outer loop iterates. During the first iteration ($i = 0$), the inner loop executes $N - 1$ times, during the second iteration ($i = 1$), $N - 2$ times, and so on.

$$O\Big(\underbrace{\sum_{i=0}^{N-1}(N-i-1)}_{\text{number-of-loops}} \times \big(\underbrace{1}_{if} + \underbrace{1}_{swap}\big)\Big) = O(N^2 - N).$$

time-per-loop

So the complexity is O(N2) in the average case as well worst case.

The best case would be if the list were already sorted.

There will be comparisons as it is but no exchanges and execution time is in O(N2).

But if we keep a track of exchanges in each pass and terminate the program checking if no there are no exchanges, then the program would require only one pass and max. (N-1)

comparisons are required in that single pass and we can say that the complexity is of order of O(N).

4. What is linear data structure?

Answer:

A data structure is said to be *linear* if its elements form a sequence or a linear list.

Examples:

- Arrays
- Linked Lists
- Stacks, Queues

The Operations that can be done on linear structures are:

- *Traversal*: Travel through the data structure
- *Search*: Traversal through the data structure for a given element
- *Insertion*: Adding new elements to the data structure
- *Deletion*: Removing an element from the data structure
- *Sorting*: Arranging the elements in some type of order
- *Merging*: Combining two similar data structures into one.

5. Prove that, $O(f(x)) + O(g(x)) = O(max(f(x),g(x))$

Answer:

Suppose q(x) is in O(f(x) + g(x)) then q(x) is in O(max(f(x), g(x)) because two times the greatest of f(x) and g(x) is greater than or equal to the sum. More precisely:

$q(x) <= c(f(x) + g(x)) <= 2c(max(f(x), g(x))$.

Next, suppose q(x) is in O(max(f(x), g(x)) then q(x) is in O(f(x) + g(x)) because the greatest of f(x) and g(x) is less than or equal the sum of the two (assuming the two functions are positive valued). In formulas:

$q(x) <= c \max (f(x),g(x),h(x)) <= c (f(x) + g(x))$.

Hence proved.

6. What is an Abstract Data Type? What do you mean by a Dynamic Data Structure?

Answer:

An **abstract data type (ADT)** is an object with a generic description independent of implementation details. If for a particular collection of data only the structure of data and the functions to be performed on the data is defined but the implementation is not defined, then such a collection of data is called Abstract data type.

In dynamic Data Structure memory allocation for the data structure takes place at the run time and only required amount of memory is allocated.

E.g.: Link lists, Stacks, Queues, Trees

7. $T(n) = 4n^2 + 3n \log n$, express T(n) in Big (O) notations.

Answer: In Big (O) notation the complexity is $O(n^2)$.

8. Write the difference between a[ ] [ ] and **a.

Answer:

The main difference between multidimensional arrays **(a[][])** and arrays of pointer **(**a)** refers to an array of particular datatype arrays (a pointer to pointers to that datatatype) is the fact that multidimensional arrays are rectangular data structures, with the same number of elements allocated for each specified dimension. This constraint is not necessarily present in an array of pointers.

9. Do a comparison among Data Type, Abstract Data Type and Data structure.

Answer:

Data type consists of the values it represents and the operations defined upon it.. Integer or character data types are found in nearly all computers, and they have well-defined operations that can be done with them. For example, the integer data type has operations for addition, subtraction, multiplication, and division. The computer knows how to perform these arithmetic operations with any two integers because these operations are part of the integer data type.

A data type can be considered **abstract** when it is defined in terms of operations on it, and its implementation is hidden (so that we can always replace one implementation with another for, e.g., efficiency reasons, and this will not interfere with anything in the program). Thus, speaking about such a type, we leave its implementation aside considering it irrelevant to the topic, unless we directly discuss the implementation.

A data structure is an arrangement of data in a computer's memory or even disk storage. An example of several common data structures are arrays, linked lists, queues, stacks, binary trees, and hash tables. **Data Structure** is an implementation of ADT. Many ADT can be implemented as the same Data Structure.

10. What are the differences between linear and non-linear data structures?

Answer:

Data structure can be classified as linear and non-linear.

Linear data structure: The data structure is said to be linear, if its element forms a sequence or linear list. There are two basic ways of representing such structure in memory. Here the elements are traversed sequentially starting from the beginning.

(1) Linear relationship between data elements is represented by means of sequential memory locations. Ex: Arrays.

(2) Linear relationship between data elements is represented by means of pointers and links. Ex: Linked list.

Non-linear data structure: They are used to represent the data having a hierarchical relationship between elements. Here the elements are not traversed sequentially, rather they are traversed in a non linear fashion. For example, in case of the tree, we have to start from the root but we have to traverse either left subtree or right subtree, but not the both.

Ex: Graphs and Trees.

11. What are the characteristics of algorithm?

Answer:

- **Precision** – the steps are precisely stated(defined).
- **Uniqueness** – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- **Finiteness** – the algorithm stops after a finite number of instructions are executed.
- **Input** – the algorithm receives input.
- **Output** – the algorithm produces output.
- **Generality** – the algorithm applies to a set of inputs.

12. a) Suppose one 2-D array is initialized as int a [5][7]; Base address is 4000. Find the location of element a [2][4] in row major form and column major form.
b) Define Sparse Matrix.

Answer:

a) Assume that the size of each element is 4 bytes.

In row major order the address will be

$$Loc(A(i,j)) = Lo + ((i-1) * n + (j-1)) * c$$

Therefore,

$$Loc(A(2,4)) = 4000 + ((2-1) * 7 + (4-1)) * 4 = 4000 + (7+3) * 4$$
$$= 4000 + 40 = 4040$$

b) *Refer to Question No. 1(2nd Part) of Short Answer Type Questions.*

13. What is the default return type of malloc( )? Why do we need to typecast it?

Answer:

`malloc` returns a void pointer (`void *`), which indicates that it is a pointer to a region of unknown data type.

The **(char*)** or (int*) is an explicit type conversion, converting the pointer returned by malloc from a pointer to anything, to a pointer to char or integer. This is unnecessary in C, since it is done implicitly, and it is recommended not to do this, since it can hide some errors.

14. If $T(n) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_nx^n$, then prove $T(n) = \Theta(x^n)$

Answer:

To prove that f(n) = Θ(g(n)) we have to find c1, c2 > 0 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$.

Now c_1 and c_2 are two valid constants for $n => n_0$:

$$c_1 = \frac{1}{2^k} a_k \qquad c_2 = \left(2 - \frac{1}{2^k}\right) a_k$$

where

$$n_0 = 2 \cdot \max\left\{ \sqrt[i]{\frac{|a_i|}{a_k}} \;\middle|\; 0 \le i \le k-1 \right\}$$

Hence proved.

Long Answer Type Questions

1. Write short note on Abstract Data Type.

Answer: ***Refer to Question No. 6 of Short Answer Type Questions.***

2. Define pseudo-code. **[MODEL QUESTION]**

Answer:

Actual purpose of using pseudo-code is that it is easier for humans to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm.

Pseudo-code is an artificial and informal language that helps programmers develop algorithms. It is a "text-based" detail (algorithmic) design tool.Pseudo-code is a compact and informal high-level description of a computer programming algorithm that uses the structural conventions of a programming language, but is intended for human reading rather than machine reading. Pseudo-code typically omits details that are not essential for human understanding of the algorithm, such as variable declarations, system-specific code and subroutines.

Examples:

```
1. If student's grade is greater than or equal to 80
Print "passed"
else
Print "failed"

2. Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
Input the next grade
Add the grade into the total
Set the class average to the total divided by ten
Print the class average.

3. Initialize total to zero
Initialize counter to zero
Input the first grade
while the user has not as yet entered the sentinel
add this grade into the running total
add one to the grade counter
```

```
input the next grade (possibly the sentinel)
if the counter is not equal to zero
set the average to the total divided by the counter
print the average
else
print 'no grades were entered'
```

```
4. initialize passes to zero
initialize failures to zero
initialize student to one
while student counter is less than or equal to ten
input the next exam result
if the student passed
add one to passes
else
add one to failures
add one to student counter
print the number of passes
print the number of failures
if eight or more students passed
print "raise tuition"
```

2 STACKS & QUEUES

Chapter at a Glance

- **Stacks:** A stack is an abstract data type, which declares two methods PUSH and POP. Stacks are implemented either by an array or by a linked list.
- The **PUSH** operation allows us to insert data at the end of an array or linked list.
  The **POP** operation allows us to remove data from the end of an array or linked list.
  A **STACK** is called a last in first out (LIFO) system.
- A stack may be represented by a linked list. The first node of the list will be the topmost element of the stack and is pointed by a top pointer. This type of stack representation is called **linked stack**. The stack can be declared as follows:

```
typedef struct linked_list
{
   int data;
   struct linked_list *next;
} lstack;
lstack *top;
```

- **Various types of expression:** A mathematical expression involves constants (operands) and operations (operators).
  Infix notation: operand1 *operator* operand2, **A + B**
  Prefix notation: *operator* operand1 operand2, **+ A B**
  Postfix notation: operand1 operand2 *operator*, **A B +**
- **Conversion from INFIX to POSTFIX expression:** In order to convert the infix to its corresponding postfixes form; we need to do the following steps:
  - Fully parenthesize the expression according to the priority of different operators.
  - Move all operators so that they replace their corresponding right parentheses.
  - Delete all parentheses.

  The priorities of different operators are given below:

Operators	Priority
Unary - , unary + , not (!)	4
* , / , % , and (& / &&)	3
+ , - , or (| / ||)	2
<, <=, >, >=, = =, !=	1

- **Priority Queue:** A priority queue is essentially a list of items in which each item has associated with it a ***priority***. In general, different items may have different priorities and we speak of one item having a higher priority than another. Given such a list we can determine which is the highest (or the lowest) priority item in the list. Items are inserted into a priority queue in any, arbitrary order. However, items are withdrawn from a priority queue in order of their priorities starting with the highest priority item first. Two elements with the same priority are processed according to the order in which they were added to the queue.
- **De-queue:** De-queue is a linear data structure, where insertions and deletions are made to or from either end of the structure.

Multiple Choice Type Questions

1. A linear list in which elements can be added or removed at either end but not the middle is known as
   a) queue b) deque c) stack d) tree
   Answer: (b)

2. The prefix expression for the infix expression a * (b + c) / e – f is
   a) /*a + bc –ef b) –/* + abcef c) –/*a + bcef d) None of these
   Answer: (a)

3. The number of stacks required to implement mutual recursion is
   a) 3 b) 2 c) 1 d) none of these
   Answer: (c)

4. Priority queue can be implemented using
   a) array b) linked list c) heap d) all of these
   Answer: (d)

5. Reserve Polish notation is often known as
   a) Infix b) Prefix c) Postfix d) none of these
   Answer: (c)

6. The evaluation of the postfix expression 3, 5, 7, *, +, 12, % is
   a) 2 b) 3 c) 0 d) 3.17
   Answer: (a)

7. The operation for adding an entry to a stack is traditionally called
   a) Add b) Append c) Insert d) Push
   Answer: (d)

8. The best data structure to evaluate an arithmetic expression in postfix form is
   a) Queue b) Stack c) Tree d) Linked List
   Answer: (b)

9. The heap (represented by an array) constructed from the list of numbers 30, 10, 80, 60, 15, 55, 17 is –
   a) 60, 80, 55, 30, 10, 17, 15 b) 80, 55, 60, 15, 10, 30, 17
   c) 80, 60, 30, 17, 55, 15, 10 d) none of these
   Answer: (b)

10. The postfix equivalent of the prefix * + $ab - cd$ is
    a) $ab + cd - *$ b) $abcd + - *$ c) $ab + cd * -$ d) $ab + - cd *$
    Answer: (a)

11. The following sequence of operations is performed on a stack: push (1), push (2), pop, push (1), push (2), pop, pop, pop, push (2), pop. The sequence of popped out values are

a) 2, 2, 1, 1, 2 b) 2, 2, 1, 2, 2 c) 2, 1, 2, 2, 1 d) 2, 1, 2, 2, 2

Answer: (a)

12. Stack cannot be used to

a) evaluate an arithmetic expression in postfix form
b) implement recursion
c) allocate resources (like CPU) by the operating system
d) convert infix expression to its equivalent postfix expression

Answer: (c)

13. The following sequence of operations is performed on a stack push(1), push(2), pop( ), push(1), push(2), pop( ), pop( ), pop( ), push(2), pop( ), the sequence of popped out values are

a) 2, 2, 1, 2, 1 b) 2, 2, 1, 1, 2 c) 2, 1, 2, 2, 2 d) 2, 1, 2, 2, 1

Answer: (b)

14. The deque can be used

a) as a stack
b) as a queue
c) both as a stack and as a queue
d) none of these

Answer: (b)

15. Inserting an item into the stack when stack is not full is called ..... operation and deletion of item from the stack, when stack is not empty is called ..... operation.

a) push, pop
b) pop, push
c) insert, delete
d) delete, insert

Answer: (a)

16. To make a queue empty, elements can be deleted till

a) front = rear + 1
b) front = rear – 1
c) front = rear
d) None of these

Answer: (a)

Short Answer Type Questions

1. a) Convert the following infix expression into equivalent postfix expression using stack:
(A + B) * C – (D – E) / (F + G).
b) What is a Max Heap?

Answer:

a)

Symbol scanned	Stack	Output
(	(	
A	(	A
+	(+	A
B	+	AB
)		AB+
*	*	AB+
C	*	AB+C
-	-	AB+C*
(	-(	AB+C*
D	-(	AB+C*D
-	-(-	AB+C*D
E	-(-	AB+C*DE
)	-	AB+C*DE-
/	-/	AB+C*DE-
(	-/(	AB+C*DE-
F	-/(	AB+C*DE-F
+	-/(+	AB+C*DE-F
G	-/(+	AB+C*DE-FG
)	-/	AB+C*DE-FG+
NONE	EMPTY	AB+C*DE-FG+/-

b) A heap is a tree-based data structure that satisfies the *heap property*: if B is a child node of A, then key(A) ≥ key(B). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a *max-heap*.

2. What is a priority queue?
Mention the different design options for priority queue.

Answer:

A priority queue is essentially a list of items in which each item has associated with it a *priority*. In general, different items may have different priorities and we speak of one item having a higher priority than another. Given such a list we can determine which is the highest (or the lowest) priority item in the list. Items are inserted into a priority queue in any, arbitrary order. However, items are withdrawn from a priority queue in order of their priorities starting with the highest priority item first.

Two elements with the same priority are processed according to the order in which they were added to the queue.

Often a type of binary tree called a heap is used to store the items in a priority queue. A heap has a root node (usually draw at the top) and it has two children, a left child and a right child. Each node (parent, and left or right child) has a value associated with it, with the property that the parent's value is more than either of its children. When we add an

item to the heap we check to see if the parent's value is more or less than it. If it is more, then we swap the child with the parent. We check again, and maybe swap again, until the tree is "balanced," (i.e. that it obeys the heap property).
There are some other methods of implementing priority queue which are not so efficient. They are
Sorted list implementation:
Unsorted list implementation: Keep a list of elements as the queue. To add an element, append it to the end. To get the next element, search through all elements for the one with the highest priority.

3. Define dequeue?

OR,

What is dequeue?

Answer:
Dequeue is a linear data structure, where insertions and deletions are made to or from either end of the structure.

4. Convert the following infix expression to postfix notation by showing the operator stack and output string after reading each input token:

$A * B + C * (D - E) - F * G$

Answer:

Symbol scanned	Stack	Output
A		A
*	*	A
B	*	AB
+	+	AB*
C	+	AB*C
*	+*	AB*C
(	+* (	AB*C
D	+* (	AB*CD
-	+*(-	AB*CD
E	+* (-	AB*CDE
)	+*	AB*CDE-
-	+-	AB*CDE-*
F	+-	AB*CDE-*F
*	+-*	AB*CDE-*F-
G	+-*	AB*CDE-*FG
NONE	NONE	AB*CDE-*FG*-+

5. a) Consider the array int a [10] [10] and the base address 2000, then calculate the address of the array a [2] [3] in the row and column major ordering.

Answer:

In row major ordering the linear offset from the beginning of the array to any given element A[row][column] can then be computed as:

offset = row*NUMCOLS + column

Thus a[2][3] will have address = 2000 + 2*10+3 = 2023

In column major ordering the memory offset could then be computed as:

offset = row + column*NUMROWS

Thus a[2][3] will have address = 2000 + 2+3*10 = 2032

b) Write the advantage of circular queue over linear queue.

OR,

Why circular queue is better than simple queue?

OR,

Why circular queue is used over simple queue

Answer:

Refer to Question No. 5 of Long Answer Type Questions.

6. Evaluate the following postfix expression:

4, 5, 4, 2, ∧ , +, *, 2, 2, ∧ , 9; 3, /, *, –

Write pseudo code for evaluating postfix expression.

Answer:

1st Part:

4, 5, 4, 2, ∧ , +, *, 2, 2, ∧ , 9, 3, /, *, –

≡ 4, 5, 16, +, *, 2, 2, ∧ , 9, 3, /, *, –

≡ 4, 21, *, 2, 2, ∧ , 9, 3, /, *, –

≡ 84, 2, 2, ∧ , 9, 3, /, *, –

≡ 84, 4, 9, 3, /, *, –

≡ 84, 4, 3, *, –

≡ 84, 12 –

≡ 76

2nd Part: ***Refer to Question No. 2 of Long Answer Type Questions.***

7. How many types of priority queues are there? Can you consider stack as a priority queue? If yes, how?

Answer:

Mainly there are two kinds of priority queue: 1) Static priority queue 2) Dynamic priority queue.

Stack can be considered as a priority queue where the priorities of each element being inserted are considered higher than the previous one. This will let it behave e like a LIFO structure (i.e., stack).

8. Suppose a queue is implemented by an array. Write an algorithm to insert a new element at the kth position of the array.

Answer:

```
static int head, tail; // Declare global indices to head and tail
of queue
static char theQueue[MAX_SIZE];        // The queue
//-----------------------------------------
// Function: InitQueue()
// Purpose: Initialize queue to empty.
// Returns: void
//-----------------------------------------
void InitQueue()
{
       head = tail = -1;
}
//-----------------------------------------
// Function: Enqueue()
// Purpose: Enqueue an item into the kth position of queue.
// Returns: TRUE if enqueue was successful
//          or FALSE if the enqueue failed.
//-----------------------------------------
int Enqueue(char ch, k)
{
       int c;
       // Check to see if the Queue is full
       if(isFull()) return FALSE;

       // Increment tail index
       tail++;
      for (c = MAX_SIZE - 1; c >= k - 1; c--)
            array[c+1] = array[c];

        array[position-1] = ch;
       return TRUE;
```

Long Answer Type Questions

1. Write the difference between stack and queue and implement the operations of priority queue.

Answer:

1st Part:

A Stack is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the end of the sequence. In principle a stack is a container of data items, from which we get data items out in reverse order compared to the order they have been put into the container. We can also say that the item that has been put last in is coming first out. That's why a stack is also called LIFO

(Last In First Out list). We can as well say that the item, which is put first in the container is get last out (First In Last Out: FILO).
A Queue is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the beginning of the sequence. In principle a queue is container from which data items are retrieved out in the same order they are put in. This means that the queue is a container that preserves the order of items put there. We can also say that the item that is put last into the queue is taken last out from the queue. We can also say that the item which is put first into the queue is taken first out. (First In First Out: FIFO).

2nd Part:
A priority queue is an abstract data type that supports the following three operations:
insertWithPriority: add an element to the queue with an associated priority
getNext: remove the element from the queue that has the *highest priority*, and return it
/* C implementation using array of size MAX. Assume structure is as below*/

```
struct data
      {
       int val,p,o;
      }d[MAX];

    int front=rear=-1; /*initial values*/

/* insertWithPriority*/
    void insert(data d1)
    {
          if(rear==MAX-1)
             printf("Priority Queue is Full");
          else
          {
             rear++;
             d[rear]=d1;
             if(front==-1)
                  front=0;
             data temp;
             for(int i=front;i<=rear;i++)
                 for(int j=i+1;j<=rear;j++)
                 {
                       if(d[i].p > d[j].p)
                       {
                            temp=d[i];
                            d[i]=d[j];
                            d[j]=temp;
                       }
                       else
                       {
                            if(d[i].p==d[j].p)
                            {
                                  if(d[i].o > d[j].o)
```

```
                                {
                                        temp=d[i];
                                        d[i]=d[j];
                                        d[j]=temp;
                                }
                        }
                }
        }
}
data getNext()
{
        data d1;
        if(front==-1)
           printf("Priority Queue is Empty");
        else
        {
           d1=d[front];
           if(front==rear)
                front=rear=-1;
           else
                front++;
        }
        return d1;
}
```

2. Write an algorithm to convert an infix expression to postfix using stack.

OR,

Write an algorithm to evaluate a postfix expression

Answer:

- Scan the Infix string from left to right.
- Initialise an empty stack.
- If the scannned character is an operand, add it to the Postfix string. If the scanned character is an operator and if the stack is empty Push the character to stack.
- If the scanned character is an Operand and the stack is not empty, compare the precedence of the character with the element on top of the stack (topStack). If topStack has higher precedence over the scanned character Pop the stack else Push the scanned character to stack. Repeat this step as long as stack is not empty and topStack has precedence over the character.
- Repeat this step till all the characters are scanned.
  (After all characters are scanned, we have to add any character that the stack may have to the Postfix string.) If stack is not empty add topStack to Postfix string and Pop the stack. Repeat this step as long as stack is not empty.

Return the Postfix string.

The C-function to convert an infix expression to it's equivalent postfix form is as given below:

```
void postfix(char x[])
{
  int i = 0;
  while (x[i]! = '\0')
  {
    if (isalpha(x[i]))
      printf("%c", x[i]);
    else if (x[i] == ')')
    {
      while (s[top] != '(')
        printf(" %c", pop());
      pop(); // delete '(' symbol
    }
    else
    {
      while (isp(s[top])>= icp(x[i]))
        printf(" %c", pop()); push(x[i]);
    }
      i++;
      while (top > 0)
        printf(" %c", pop());
  }
}
int isp(char a)
{
  if (a == ' *' || a == ' /' || a == '%')
    return (3);
  else if (a == ' + ' || a == ' - ')
    return (2);
  else if (a == ' (')
    return (1);
  else if (a == ' # ')
    return ( - 1);
  else if (a = ') ')
    return (0);
}
int icp(char a)
{
  if (a == '*' || a == '/' || a == '%')
    return (3);
  else if (a == ' + ' || a == ' - ')
    return (2);
  else if (a == ' (')
    return (4);
  else if (a == ') ')
    return (0);
```

3. a) Define circular queue.

Answer:

Refer to Question No. 5(1st & 2nd Part) of Long Answer Type Questions.

b) Write an algorithm to insert an item in circular queue.

Answer:

```
void QInsert(int item)
{
  if (rear == N-1)
  {
    printf("Queue Is Full");
    return;
  }
  CQ[++rear] = item;
  if (front ==  - 1)
    front = 0;
}
```

c) What is input restricted Dequeue?

Answer:

An input-restricted Dequeue is one where deletion can be made from both ends, but input can only be made at one end.

4. a) What is a Stack ADT?

b) Write a C function of popping an element from a stack implemented using linked list.

c) Explain three uses of stack data structure.

Answer:

a) A Stack ADT is a (ordered) collection of items, where all insertions are made to the end of the sequence and all deletions always are made from the end of the sequence. In principle a stack is a container of data items, from which we get data items out in reverse order compared to the order they have been put into the container. We can also say that the item that has been put last in is coming first out. That's why a stack is also called LIFO ((Last In First Out list). We can as well say that the item, which is put first in the container is get last out (First In Last Out: FILO).

b) Assume that the list is defined as below:

```
typedef struct node *nptr;
struct node
{
int data;
nptr next;
};
/* function pop. */
int pop(nptr s) /*Function to pop the elements*/
{
nptr temp;
int y;
```

```
if(s->next==NULL)
          {
printf("Underflow on Pop");
return(-1);
}
Else
{
y=s->next->data;
temp=s->next;
s->next=temp->next;
free(temp);
return(y);
}
}
```

c) Use of stack

Reversing Data: We can use stacks to reverse data.

(example: files, strings). It is very useful for finding palindromes.

Consider the following pseudocode:

```
1)     read (data)
2)     loop (data not EOF and stack not full)
              1) push (data)
              2) read (data)
3)     Loop (while stack notEmpty)
              1) pop (data)
              2) print (data)
```

```
Converting Decimal to Binary:
Consider the following pseudocode
Read (number)
Loop (number > 0)
      1) digit = number modulo 2
      2) print (digit)
      3) number = number / 2
```

The problem with this code is that it will print the binary number backwards. (ex: 19 becomes 11001000 instead of 00010011.)

To remedy this problem, instead of printing the digit right away, we can push it onto the stack. Then after the number is done being converted, we pop the digit out of the stack and print it.

Evaluating arithmetic expressions.

In high level languages, infix notation cannot be used to evaluate expressions. We must analyze the expression to determine the order in which we evaluate it. A common technique is to convert a infix notation into postfix notation, then evaluating it. This is done using stack.

5. i) What is Circular queue?

ii) Write Q-insert algorithm for the circular queue.

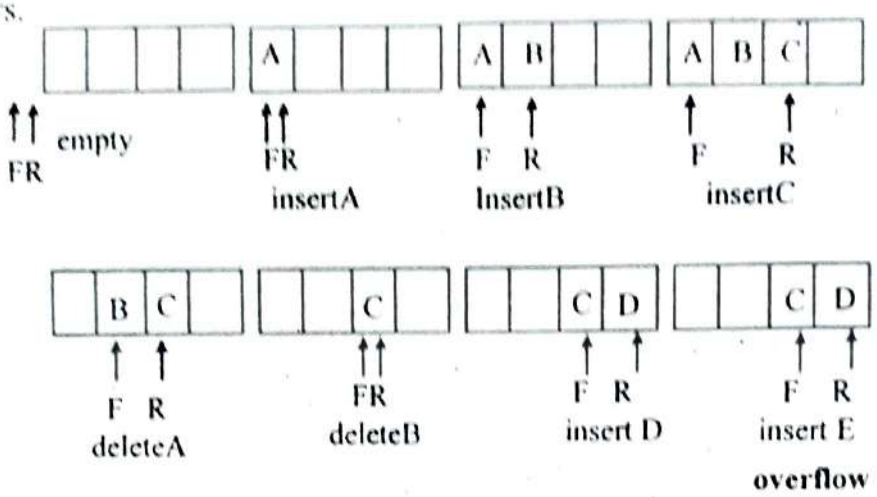
OR,

Write an algorithm to insert an element into circular queue.

Answer:

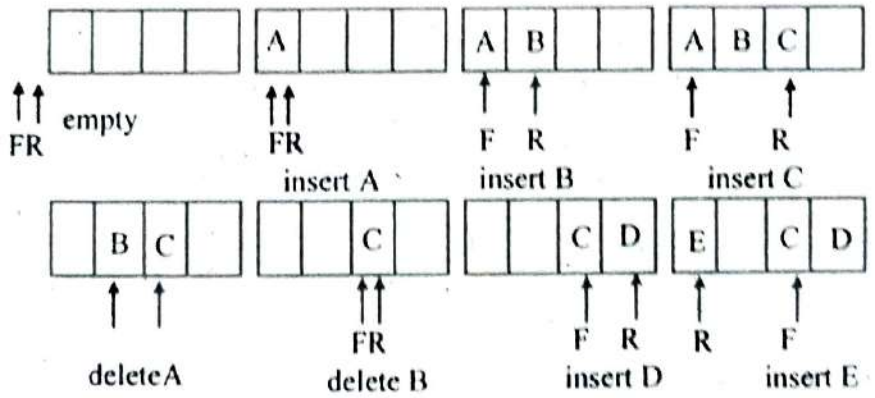
i) The two algorithms QINSERT & QDELETE can be very wasteful of storage if the front pointer F never manages to catch up the rear pointer. Actually an arbitrary large amount of memory would be required to accommodate the elements. The method of performing operations on a Queue should only be used when the queue is emptied at certain intervals.

Let us consider the following sequence of operations. F and R represents the front and rear pointers.



To avoid this problem we can think an alternative representation of a queue, which prevents an excessive use of memory. In this representation elements are arranged as in a circular fashion where the rear again points to the front. This type of queue is known ***circular queues***.

So the above sequence of operations can be represented are shown below (in circular queue).



As we can see that the overflow issue in the previous case while inserting E is resolved by redirecting the rear to the front. This is the essence of circular queue.

ii) Let us now write the insert and delete functions of the circular queue implementation. Let us also assume that F and R represents front and rear pointers respectively

```
void CQInsert(int item)
{
  if (rear == N-1)
  {
    printf("Queue Is Full");
```

```
    return;
  }
  CQ[++rear] = item;
  if (front ==  - 1)
    front = 0;
}
int CQDelete()
{
  int x;
  if (front ==  - 1)
  {
    printf("Queue Is Empty");
    exit(0);
  }
  x = CQ[front];
  CQ[front] =  - 1;
  if (front == rear)
    front = rear =  - 1;
  else
    front++;
  return x;
}
```

In linear queue the condition for queue full is
QREAR==MAXLIMIT
Suppose maxlimit is ten and queue is full now if we delete 9 element from queue then inspite of queue is empty we cannot insert any element in the queue. This wastage of memory is solved through circular queue where queue full condition is
QREAR==Qfront+1

6. Define the ADT for stack. Show the implementation of the stack data structure using linked list.

Answer:

The Stack ADT has two main functions:

Push

input: a stack data object; an element of the base type
output: a boolean indicating whether the operation was successful
effect: the element is added to the sequence in the stack, in the top position

Pop

input: a stack data object
output: a boolean indicating whether the operation was successful
effect: the element in the top position in the sequence in the stack is removed

Stack Implementation

First, let us define the linked list data structure.

```
typedef struct linked_list
{
    int data;
    struct linked_list *next;
} Node;
```

For us to be able to test our code, we need to define a way to display our stack. The following code uses recursion to display stack.

```
// recursively display the contents
// of the stack
void DisplayStack(Node* currentNode)
{
    // recursive termination condition
    if (currentNode == NULL)
    {
        return;
    }
    // the node is not null
    // display the data
    printf(" -> %d", currentNode->data);
    // recursively call the display to
    // display the next element in the stack
    DisplayStack(currentNode->next);
}
```

The code for pushing an element onto the stack is as given below.

```
// push item on the stack
// this is same as adding a node
// at the top of the list
void Push(int dataToAdd)
{
    // assumption: head is already defined elsewhere in the
program
    // 1. create the new node
    Node *temp = new Node;
    temp->data = dataToAdd;

    // 2. insert it at the first position
    temp->next = head;

    // 3. update the head to point to this new node
    head = temp;
}
```

The code for popping an element from stack is given below.

```
// pop an element from the stack
// this is same as removing the first element
// from the list
Node* Pop()
{
    // check for empty list
    if (head == NULL)
    {
        printf("Stack is empty\n");
        return NULL;
```

```
}
// get the top node
Node *firstNode = head;
// move the head
head = head->next;
// disconnect the node
// from the list
firstNode->next = NULL;

// return the top node
return firstNode;
}
```

7. What are the differences between stack and Queue? Write the algorithm for insertion and deletion in a circular Queue.

Answer:

1st Part: ***Refer to Question No. 1(1st Part) of Long Answer Type Questions.***

2nd Part: ***Refer to Question No. 5 of Long Answer Type Questions.***

8. a) Evaluate the postfix expression using stack:
3, 16, 2, +, *, 12, 6, /, –
b) Convert the infix expression into its equivalent prefix expression using stack:
$a+b*c+(d*e+f)*g$.

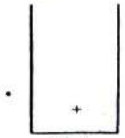
Answer:

a) The steps are as shown below:

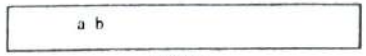
3, 16, 2, +, *, 12, 6, /, –

Input token	Operation	Stack contents (top on the right)	Details
3	Push on the stack	3	
16	Push on the stack	3, 16	
2	Push on the stack	3, 16, 2	
+	Add	3, 18	Pop two values: 16 and 2 and push the result 18 on the stack
*	Multiply	54	Pop two values: 3 and 18 and push the result 54 on the stack
12	Push on the stack	54, 12	
6	Push on the stack	54, 12, 6	
/	Divide	54, 2	Pop two values: 12 and 6 and push the result 2 on the stack
-	Divide	54, 2	Pop two values: 54 and 2 and push the result 52 on the stack
(End of tokens)	(Return the result)	52	Pop the only value 52 and return it

b) First, the symbol "a" is read, so it is passed through to the output. Then '+' is read and pushed onto the stack. Next "b" is read and passed through to the output. The state of affairs at this juncture is as follows:

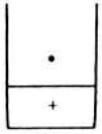


Stack

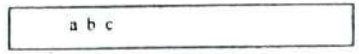


Output

Next a '*' is read. The top entry on the operator stack has lower precedence than '*', so nothing is output and '*' is put on the
stack. Next, "c" is read and output. Thus far, we have



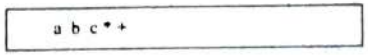
Stack



Output

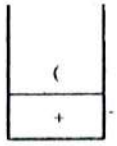
The next symbol is a '+'. Checking the stack, we find that we will pop a '*' and place it on the output, pop the other '+', which is not of lower
but equal priority, on the stack, and then push the '+'.

Stack

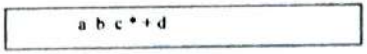


Output

The next symbol read is an '(', which, being of highest precedence , is placed on the stack. Then "d" is read and output.

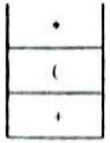


Stack

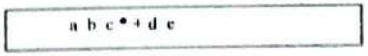


Output

We continue by reading a '*'. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, "e" is read and output.

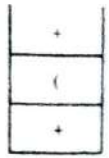


Stack

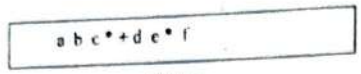


Output

The next symbol read is a '+'. We pop and output '*' and then push '+'. Then we rea and output.



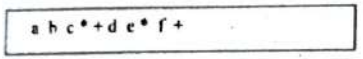
Stack



Output

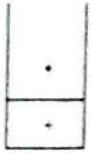
Now we read a ')', so the stack is emptied back to the '('. We output a '+'.

Stack

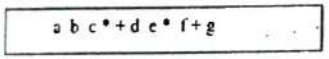


Output

We read a '*' next; it is pushed onto the stack. Then "g" is read and output.



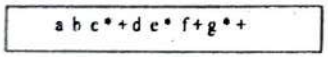
Stack



Output

The input is now empty, so we pop and output symbols from the stack until it is empty.


Stack



Output

9. Convert the infix expression 9+5*7-6^2+15/3 into its equivalent postfix expression and evaluate that postfix expression, clearly showing the state of the stack.

Answer:

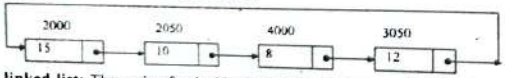
Symbol scanned	Stack	Output
9		9
+	+	9
5	+	95
*	+*	95
7	+*	957
-	-	957*+
6	-	957*+6
^	-^	957*+6
2	-^	957*+62
+	+	957*+62^-
15	+	957*+62^15
/	+/	957*+62^15
3	+/	957*+62^153
NONE	NONE	957*+62^153/+

3

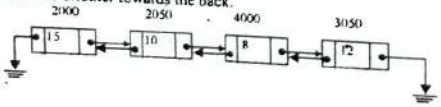
LINKED LI

Chapter at a Glance

- **Singly linked lists:** A linked list is a data structure where data can be represented as a chain of nodes. Each node of a linked list contains two parts: one is the address part another is the data part. The address part holds the address of the node which is next to or previous to the current node.
  Typically the first node of a linked list is called the HEAD node. If the head node is destroyed then the entire list gets destroyed as well. Depending on the traversal requirement a linked list can be designed as circular, bi-directional etc. In it's simplest form the following diagram shows the structure of a uni-directional linked list also known as singly linked list.
- **Several operations:** (Insertion, deletion, Traversing, Searching)
- **Insertion:** The algorithm for inserting in the above manner is given below:
  Let us assume that x is data value to be inserted after the node containing the data value y. p initially contains the address of the head node.
  Step 1: start traversing the linked list from the head node
  Step 2: if the data value of the head node is not equal to y goto step 8
  Step 3: create a new node
  Step 4: place x to the data field of the new node
  Step 5: place the next address of the head node to the next address of the new node
  Step 6: change the next address of the head node by the new node address
  Step 7: Goto Step 13
  Step 8: Identify the address of the node containing data value y as follows
  Step 8a) if the data value of the next node of p is equal to y
  Step 8b) else change p by its next node address
  Step 9: create a new node
  Step 10: place y to the data field of the new node
  Step 11: place the next address of p node to the next address of new node
  Step 12: change the next address of the p node by the new node address
  Step 13: End
- **Circular linked list:** the circular linked list, the address of the last node contains the address of the first node, as shown in the figure below:



- **Doubly linked list:** The node of a doubly linked list contains two link fields, instead of one. One link is used to point to the predecessor node, i.e., the previous node & the other link is used to point to the successor node, i.e., the next node. So, the two links are directed, one towards the front and another towards the back.



- **Linked representation of polynomial:** In general any polynomial A(x) can be written as
  $A(x) = a_0 + a_1x + a_2x^2 + \text{-----} + a_{n-1}x^{n-1} + a_nx^n$.
  Each $a_i x^i$ is i^{th} term of the polynomial, where x is variable, a_i is its coefficient & e is the exponent. If $a_i=0$, then the term is zero term, otherwise it is nonzero term.

Multiple Choice Type Questions

1. In a circularly linked list organization, insertion of a record involves the modification of
a) no pointer b) 1 pointer c) 2 pointers d) 3 pointers
Answer: (c)

2. Linked list are not suitable for
a) Stack b) Deque c) AVL Tree d) Binary search
Answer: (d)

3. Dynamic memory allocation use
a) Calloc b) Malloc c) Free d) all of these
Answer: (d)

4. Which type of linked List contains a pointer to the next as well as previous node in the sequence?
a) Singly Linked List b) Circular Linked List
c) Doubly Linked List d) All of these
Answer: (c)

5. Linked list is not suitable data structure for which one of the following problems?
a) insertion sort b) radix sort
c) binary search d) polynomial addition
Answer: (c)

6. Self-referential pointer is used in defining
a) an array b) a node of linked-list
c) a queue d) all of these
Answer: (d)

7. Inserting a node after a given node in a doubly linked list requires
a) four pointer exchanges b) two pointer exchanges
c) one pointer exchange d) no pointer exchange
Answer: (b)

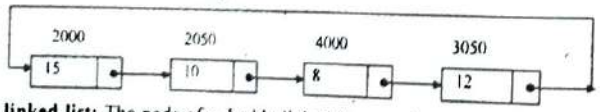
8. Binary search is not possible for
a) array b) linked list c) stack d) queue
Answer: (b)

3

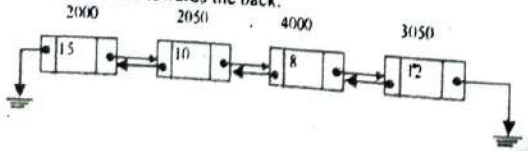
LINKED LI

Chapter at a Glance

- **Singly linked lists:** A linked list is a data structure where data can be represented as a cha of nodes. Each node of a linked list contains two parts: one is the address part another is th data part. The address part holds the address of the node which is next to or previous to th current node.
  Typically the first node of a linked list is called the HEAD node. If the head node is destroyed then the entire list gets destroyed as well. Depending on the traversal requirement a linked lis can be designed as circular, bi-directional etc. In it's simplest form the following diagram shows the structure of a uni-directional linked list also known as singly linked list.
- **Several operations:** (Insertion, deletion, Traversing, Searching)
- **Insertion:** The algorithm for inserting in the above manner is given below:
  Let us assume that x is data value to be inserted after the node containing the data value y. p initially contains the address of the head node.
  Step 1: start traversing the linked list from the head node
  Step 2: if the data value of the head node is not equal to y goto step 8
  Step 3: create a new node
  Step 4: place x to the data field of the new node
  Step 5: place the next address of the head node to the next address of the new node
  Step 6: change the next address of the head node by the new node address
  Step 7: Goto Step 13
  Step 8: Identify the address of the node containing data value y as follows
  Step 8a) if the data value of the next node of p is equal to y
  Step 8b) else change p by its next node address
  Step 9: create a new node
  Step 10: place y to the data field of the new node
  Step 11: place the next address of p node to the next address of new node
  Step 12: change the next address of the p node by the new node address
  Step 13: End
- **Circular linked list:** the circular linked list, the address of the last node contains the address of the first node, as shown in the figure below:



- **Doubly linked list:** The node of a doubly linked list contains two link fields, instead of one. One link is used to point to the predecessor node, i.e., the previous node & the other link is used to point to the successor node, i.e., the next node. So, the two links are directed, one towards the front and another towards the back.



- **Linked representation of polynomial:** In general any polynomial A(x) can be written as
  $A(x) = a_0 + a_1x + a_2x^2 + \text{-----} + a_{n-1}x^{n-1} + a_nx^n$.
  Each a_i x^i is i^{th} term of the polynomial, where x is variable, a_i is its coefficient & e is the exponent. If $a_i=0$, then the term is zero term, otherwise it is nonzero term.

Multiple Choice Type Questions

1. In a circularly linked list organization, insertion of a record involves the modification of
a) no pointer b) 1 pointer c) 2 pointers d) 3 pointers
Answer: (c)

2. Linked list are not suitable for
a) Stack b) Deque c) AVL Tree d) Binary search
Answer: (d)

3. Dynamic memory allocation use
a) Calloc b) Malloc c) Free d) all of these
Answer: (d)

4. Which type of linked List contains a pointer to the next as well as previous node in the sequence?
a) Singly Linked List b) Circular Linked List
c) Doubly Linked List d) All of these
Answer: (c)

5. Linked list is not suitable data structure for which one of the following problems?
a) insertion sort b) radix sort
c) binary search d) polynomial addition
Answer: (c)

6. Self-referential pointer is used in defining
a) an array b) a node of linked-list
c) a queue d) all of these
Answer: (d)

7. Inserting a node after a given node in a doubly linked list requires
a) four pointer exchanges b) two pointer exchanges
c) one pointer exchange d) no pointer exchange
Answer: (b)

8. Binary search is not possible for
a) array b) linked list c) stack d) queue
Answer: (b)

9. A linear link list can be traversed using
   a) recursion
   c) stack
   b) both (a) and (c) are correct
   d) both (a) and (c) are wrong

Answer: (a)

10. The data structure used to solve recursive problem is
   a) Linked list b) Queue c) Stack d) none of the

Answer: (c)

Short Answer Type Questions

1. Write an algorithm to add two polynomials.

Answer:

Let us assume that the two polynomials are represented using linked list and the result is also using a linked list.

Let phead1, phead2 and phead3 represent the pointers of the three lists under consideration.

Let each node contain two integers: exp and coff.

Let us assume that the two linked lists already contain relevant data about the two polynomials.

Also assume that we have got a function append to insert a new node at the end of the given list.

```
p1 = phead1;
p2 = phead2;
```

Let us call malloc to create a new node p3 to build the third list

```
p3 = phead3;
/* now traverse the lists till one list gets exhausted */
while ((p1 != NULL) || (p2 != NULL))
{
 /* if the exponent of p1 is higher than that of p2 then the next term in final list going to be the node of p1*/
   while (p1 ->exp   >  p2 -> exp )
      {
          p3 -> exp  = p1 -> exp;
          p3 -> coff = p1 -> coff ;
          append (p3, phead3);
         /* now move to the next term in list 1*/
         p1 = p1 -> next;
      }
 /* if p2 exponent turns out to be higher than make p3 same as p2 and append to final list */
while (p1 ->exp   <  p2 -> exp )
   {
       p3 -> exp  = p2 -> exp;
       p3 -> coff = p2 -> coff ;
       append (p3, phead3);
       p2 = p2 -> next;
   }
/* now consider the possibility that both exponents are same , then we must add the coefficients to get the term for the final list */
while (p1 ->exp    =  p2 -> exp )
   {
   p3-> exp = p1-> exp;
   p3->coff = p1->coff + p2-> coff ;
          append (p3, phead3) ;
          p1 = p1->next ;
   p2 = p2->next ;
    }
}
/* now consider the possibility that list2 gets exhausted , and there are terms remaining only in list1. So all those terms have to be appended to end of list3. However, one does not have to do it term by term, as p1 is already pointing to remaining terms, so simply append the pointer p1 to phead3 */
 if ( p1 != NULL)
     append (p1, phead3) ;
else
     append (p2, phead3);
```

2. Write the key features of circular linked list and state why it is important in case of Josephus problem.

Answer:

A circular list node is identical to a singly-linked list node. However, the circular list has a reference to its tail node instead of its head node. The tail node has a reference to the head node. This makes it possible to get to both ends of the list in constant time.

Josephus problem is stated as:

There are people standing in a circle waiting to be executed. After the first man is executed, certain number of people are skipped and one man is executed. Then again, people are skipped and a man is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last man remains, who is given freedom.

The task is to choose the place in the initial circle so that you survive (are the last one remaining).

The easiest and most logical way to solve the problem is to simply simulate it using a circular link list as shown below.

```
/*linked list structure */
struct node {
      int item;
      node* next;
};
/* function*/
int simple_simulation(int N, int M)
```

```
{
  int i;
  node *t=new node();
  node *x=t;
  t->item = 1; t->next = t;
  //Construct the list
  for (i = 2; i <= N; i++)
  {
    x = (x->next = new node ());
    x->item = i;
    x->next = t;
  }
  //Find the survivor
  while (x != x->next)
  {
    for (i = 1; i < M; i++)
      x = x->next;
    x->next = x->next->next; N--;
  }
  return x->item;
}
```

3. What are the advantages of linked list over an array?
Write an algorithm to insert a data X after a specific data item Y in a linked list.

Answer:

1st Part:

The advantages of linked list over an array are:

Refer to Question No. 5 of Short Answer Type Questions.

2nd Part:

Assuming the linked list already contains n integer elements. The algorithm to insert data X after a specific data item Y in a linked list is as follows:

```
node *insertXafterY(node *p, int y, int x)
{
 node *q,  *r,  *m;
 q = p;
 r = (node*)malloc(sizeof(node));
 while (p->next != NULL)
 {
  m = p;
  if (p->data == x)
  {
    break;
  }
  else
    p = p->next;
}
```

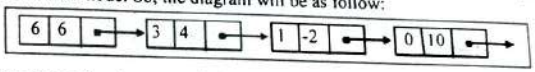


```
r->data = y;
r->next = m->next;
m->next = r;
return (q);
```

4. How can a polynomial such as $6x^6 + 4x^3 - 2x + 10$ be represented by a linked list?

Answer:

We can represent the polynomial by a linked list, where each node contains deg. coef and a pointer to the next node. So, the diagram will be as follow:



5. What are the advantages and disadvantages of linked list data structure over array?

Answer:

Linked list advantages over array

1. Linked lists do not need contiguous blocks of memory; extremely large data sets stored in an array might not be able to fit in memory.
2. Linked list storage does not need to be preallocated (again, due to arrays needing contiguous memory blocks).
3. Inserting or removing an element into a linked list requires one data update, inserting or removing an element into an array requires n (all elements after the modified index need to be shifted).

Linked list disadvantages over array:

1. Arrays have contiguous memory allocation which makes it easy to access elements in between.
2. As memory is allocated during compilation makes the program faster
3. Fixed in size so if we are aware of the exact size of datas then there can be no memory wastage which is also an advantage linked list has.
4. Insertion and deletion at the end of the array is easy but not in between.
5. Accessing data is easy. Example, a[2].

6. Write a C language function to delete nth node of a singly-linked list. The error conditions are to be handled properly.

Answer:

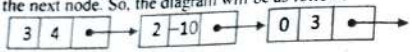
```
struct node* DeleteNode(struct node* head, int n) {
     struct node* temp = head;
     int length = LinkedListLength(temp);
     int i;
    if(n <= 0 || n > length){
       printf("ERROR: Node does not exist!\n");
    }else{
       if(n == 1){
```

```
            head = head->next; //move from head (1st node) to
second node
        }else{
            for(i = 1; i < n-1; ++i){ //move through list
                temp = temp->next;
            }
            temp->next = temp->next->next;
        }
    }
    return head;
}
```

7. a) How the polynomial $4x^3 - 10x^2 + 3$ can be represented using a linked list?
b) Compare and contrast between an array and a single linked list.

Answer:

a) We can represent the polynomial by a linked list, where each node contains deg, co and a pointer to the next node. So, the diagram will be as follow:



b) *Refer to Question No. 5 of Short Answer Type Questions.*

8. What is a self referential structure?

Answer:

A self-referential structure is one of the data structures which refer to the pointer (points) to another structure of the same type. For example, a linked list is supposed to b a self-referential data structure. The next node of a node is being pointed, which is of th same struct type. For example,

```
typedef struct listnode {
void *data;
struct listnode *next;
} linked_list;
```

In the above example, the listnode is a self-referential structure – because the *next is o the type sturct listnode.

9. Write an algorithm to find the largest and smallest element in a single linear list.

Answer:

Linked List definition in C:

```
typedef struct linked_list
{
    int data;
    struct linked_list *next;
} Node;
```

Code to find largest and smallest elements.

```
// finds the largest and smallest in the list
// assumes that head pointer is defined elsewhere
int MaxMinInList(int *max, int *min)
{
    // start at the root
    Node *currentNode = head;
    if (currentNode == NULL)
        return 0; // list is empty
    // initialize the max and min values to the first node
    *max = *min = currentNode->data;
    // loop through the list
    for (currentNode = currentNode->next; currentNode != NULL;
currentNode = currentNode->next)
    {
        if (currentNode->data > *max)
            *max = currentNode->data;
        else if (currentNode->data < *min)
            *min = currentNode->data;
    }
    // we found our answer
    return 1;
}
```

10. Write an algorithm to delete the last node of a linked-list.

Answer:

C function to delete last node is as given below:

```
struct node *delete(struct node *head)
{
  struct node *temp =head;
  struct node *t;
  if(head->next==NULL)
  {
    free(head);
    head=NULL;
  }
  else
  {
     while(temp->next != NULL)
     {
        t=temp;
        temp=temp->next;
     }
     free(t->next);
     t->next=NULL;
  }
  return head;
}
```

11. Write an algorithm to insert an element in the middle of a linked list.

Answer:

Node Structure for Singly Linked List:

```
struct node {
        int data;
        struct node *next;
}start = NULL;
```

Insert node at middle position : Singly Linked List

```
void insert_mid()
{
    int pos,i;
    struct node *new_node,*current,*temp,*temp1;
    new_node=(struct node *)malloc(sizeof(struct node));
    printf("nEnter the data : ");
    scanf("%d",&new_node->data);
    new_node->next=NULL;
    st :
    printf("nEnter the position : ");
    scanf("%d",&pos);
    if(pos>=(length()+1))
       {
       printf("nError : pos > length ");
       goto st;
       }
    if(start==NULL)
       {
       start=new_node;
       current=new_node;
       }
    else
       {
       temp = start;
              for(i=1;i< pos-1;i++)
              {
              temp = temp->next;
              }
       temp1=temp->next;
       temp->next = new_node;
       new_node->next=temp1;
       }
}
```

12. What is the difference between array and linked-list?

Answer:

The primary difference between an array or ArrayList and a linked list is that the array o ArrayList permits random access into the list structure using a subscript; each data ite has a "named" storage location (named by the subscript). In contrast, with a linked list a locations are relative to the next or previous locations. With an array or ArrayList, we ca access the 10^{th} data item, say, with one probe to subscript 9 (indexing zero-up). With

linked list, we cannot access the 10^{th} item without first traversing the first nine items. Searching through a linked list is thus inherently a sequential process.

On the other hand, because all locations are relative, additions and deletions to a linked list can be made without affecting any of the data items already stored in the list. Insertions and deletions require only that the next and previous pointers be changed exactly as needed; no data needs to be moved. When a deletion takes place, no hole is left. The deleted node becomes unused memory.

13. Show how linked list can be used to add the following polynomials:

$5x^4 + 5x^3 + 10x^2 + 8x + 3$

$3x^3 + 2x^2 + 7x + 8.$

Answer:

Let us assume that the two polynomials (p1 and p2) are represented using linked list and the resultant is also using a linked list.

Let each node contain two integers: exp and coff.

The two linked lists already contain relevant data about the two polynomials. We will create a linked list list3 (p3) with a single node to begin with.

Steps: We traverse the lists till one list gets exhausted

1) if the exponent of p1 is higher than that of p2 then the next term in final list is going to be the node of p1
2) if p2 exponent turns out to be higher than make p3 same as p2 and append to final list
3) now consider the possibility that both exponents are same , then we must add the coefficients to get the term for the final list
4) now consider the possibility that list2 gets exhausted , and there are terms remaining only in list1. So all those terms have to be appended to end of list3.

The final solution would be 5x^4 + 8x^3 + 12x^2 + 15x + 11 based on the above algorithm.

14. How a polynomial such as $8x^5 + 4x^3 - 9x^2 + 2x - 17$ can be represented using a linked list? What are the advantages and disadvantages of linked list over an array?

Answer:

1st part:

We can represent the polynomial by a linked list, where each node contains deg, coef and a pointer to the next node. So, the diagram will be as follow:

5|8 →3|5→ 2|-9→1|2→0|-17

2nd part: ***Refer to Question No. 5 of Short Answer Type Questions.***

15. Compare and contrast linked list with static and dynamic array.

Answer:

In contrast, linked lists are dynamic and flexible and it can expand and contract its size. In an static array, memory is assigned during compile time. While in a dynamic array as linked list, it is allocated during execution or runtime. Elements are stored consecutively in arrays whereas it is stored randomly in linked lists.

Compare:

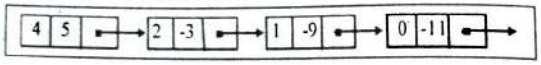
1. An array is the data structure that contains a collection of similar type data elements whereas the linked list is considered as non-primitive data structure contains a collection of unordered linked elements known as nodes.
2. Accessing an element in an array is fast, while linked list takes linear time, so it is quite a bit slower.
3. Arrays are of fixed size. Linked lists are dynamic and flexible and can expand and contract its size.
4. In addition memory utilization is inefficient in an array. Conversely, memory utilization is efficient in the linked list.

Long Answer Type Questions

1. a) How can a polynomial such as $5x^4 - 3x^2 + 9x - 11$ be represented by a linked list?

Answer:

We can represent the polynomial by a linked list, where each node contains deg, coefficient and a pointer to the next node. So, the diagram will be as follow:



b) Write an algorithm to delete a node from a doubly linked list, where a node contains one data and two address (prev & next) portion.

OR,

Write an algorithm for deletion of a node from a doubly-linked list.

Answer:

```
/*C function*/
void deleteNode(node *n)
{
      node *np = n->prev;
      node *nn = n->next;
      np->next = n->next;
      nn->prev = n->prev;
      delete n;
}
```


2. a) Do you consider the following data-structure as linear? Circular doubly linked list.

b) Represent the following polynomial by linked list (show the diagram only)

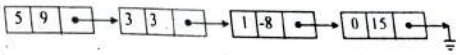
$9x^5 + 3x^3 - 8x + 15$.

c) Write an algorithm to delete all nodes having value greater than X from a given singly linked list. **[WBUT 2008]**

Answer:

a) Linear, because traverse all the list of elements sequentially.

b)



c)

```
node *removeNodeAfterX(node *p, int x)
{
  // r holds the address of the previous node
  node *q,  *r,  *m;
  q = p;
  while (q!= NULL)
  {
      m = q;
      if(q->next == NULL)
      {
          if (q->data > x)
          {
              r->next = NULL;
              free(q);
          }
          break;
      }
      else if (q->data > x)
      {
          r->next = m->next;
          free(q);
          q = r;
      }
      else
      {
          r = q;
          q = q->next;
      }
  }
    return q;
}
```

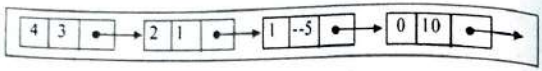
3. a) Represent the given polynomial using a link list.

$3x^4 + x^3 - 5x + 2$

b) Write the pseudo code / C code for adding two polynomials (already given user, no need to take input). Also comment on the complexity of your algorithm.

Answer:

a) We can represent the polynomial by a linked list, where each node contains deg and a pointer to the next node. So, the diagram will be as follow:



b) ***Refer to Question No. 1 of Short Answer Type Questions.***

Running is $\Theta(\max(x,y)-(x+y))$ where x and y are the number of terms in the polynom Poly1 and Poly2 respectively.

4. a) Write an algorithm for creating a linked-list with n nodes.

b) How it can be made a circular linked-list? Write a function for that purpose.

Answer:

a) //C functions showing how to create a linked list with n nodes with values 1 to n sequence

```
#include<stdio.h>
#include<stdlib.h>

// A linked list node
struct Node
{
  int data;
  struct Node *next;
};

/* Given a reference (pointer to pointer) to the head
   of a list and an int, appends a new node at the end  */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct Node* new_node = (struct Node*) malloc(sizeof(struc
Node));
     struct Node *last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. This new node is going to be the last node, so mak
next of
          it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as
head */
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}
/*this creates a list*/
int createList()
{
  /* Start with the empty list */
  struct Node* head = NULL;
for (i=0; i<n; i++)
// Insert data at the end
  append(&head, i);
}
```

b)

```
// Function that convert singly linked list
// into circular linked list.
struct Node* circular(struct Node* head)
{
    // declare a node variable start and
    // assign head node into start node.
    struct Node* start = head;

    // check that while head->next not equal
    // to NULL then head points to next node.
    while (head->next != NULL)
        head = head->next;

    // if head->next points to NULL then
    // start assign to the head->next node.
    head->next = start;
    return start;
}
```

5. How a linked-lists can be used to implement stack?

Answer:

To implement stack using linked list, we need to set the following things before implementing actual operations.

Step 1: Define a 'Node' structure with two members data and next.

Step 2: Define a Node pointer 'top' and set it to NULL.

Step 3: Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method.

push(value) - Inserting an element into the Stack

We can use the following steps to insert a new node into the stack...

Step 1: Create a newNode with given value.

Step 2: Check whether stack is Empty (top == NULL)

Step 3: If it is Empty, then set newNode → next = NULL.

Step 4: If it is Not Empty, then set newNode → next = top.

Step 5: Finally, set top = newNode.

pop() - Deleting an Element from a Stack

We can use the following steps to delete a node from the stack...

Step 1: Check whether stack is Empty (top == NULL).

Step 2: If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function

Step 3: If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.

Step 4: Then set 'top = top → next'.

Step 7: Finally, delete 'temp' (free(temp)).

6. Write short notes on the following:

a) Linear Lists

b) Circular link list

Answer:

a) Linear Lists:

A **linear list** stores a collection of objects of a certain type, usually denoted as the **elements** of the list. The elements are ordered within the linear list in a linear sequence. Linear lists are usually simply denoted as **lists**.

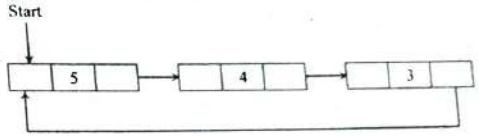
Unlike an array, a list is a data structure allowing insertion and deletion of elements at an arbitrary position of the sequence. If the position in question is given, for example by a reference, such a modification takes *only a constant number* of operations, that is, no effortful copying of entries is necessary and all insertion and deletion operations take an equally short time. Conversely, however, one cannot access a single element via an (integral) index in constant time, as in the case of an array, without having searched for it before and having received a reference to it. Furthermore, lists are not limited to a certain maximum number of elements from the beginning on (like an array).

b) Circular link list:

Circular Linked List is Divided into 2 Categories.

- Singly Circular Linked List
- Doubly Circular Linked List

- In Circular Linked List Address field of Last node contain address of "First Node".
- In short First Node and Last Nodes are adjacent.
- Linked List is made circular by linking first and last node, so it looks likecircular chain as shown in following diagram.



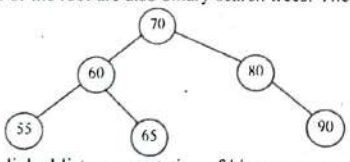
- Two way access is possible only if we are using "Doubly Circular Linked List"
- Sequential movement is possible
- No direct access is allowed.

4

TRE

Chapter at a Glance

- **Tree:** A tree is a finite set of one or more nodes connected by edges such that
  i) There is a specially designated node called the root
  ii) The remaining nodes are partitioned into n (>=0) mutually exclusive disjoint sets
- **Basic Terminology:**
  i) Node: Each data item in a tree is called a node. It specifies the data information & link (branches) to other data items. In the example of Fig (1) the tree has 13 nodes.
  ii) Root: It is the parent of all other nodes of a tree. A root node does not have any parent
  iii) Degree of a node: The number of sub trees of a node is called its degree. In our exampl the root node has three sub trees (B, C, D). Hence, the degree of the root node A is 3. Degre of B is 2. Degree of C is 1. And so on.
  iv) Degree of a tree: It is the maximum degree of nodes in a given tree. In our example th root node A has the highest degree. Hence, the degree of the tree is 3.
  v) Terminal node(s): A node with degree zero is called a terminal node(s) or leaf node(s) external node(s). In our example E, F, G, I are leaf nodes.
  vi) Non Terminal node(s): Any node whose degree is not zero is called non-terminal node(s) or internal nodes or interior node(s). In our example A, B, C, D & H are the non-termina node(s).
  vii) Path: It is a sequence of consecutive edges from the source node to the destination node In our example the path between A & I is given by the edges (A, D) (D, H) & (H, I).
- **Binary tree:** A binary tree is either empty or consists of one single vertex called the root together with two disjoint binary trees called left sub-tree & right sub-tree. Number o branches of any node is either zero or one or at most two.
- **Binary search tree:** A binary search tree (BST) is a binary tree that is either empty or every data node that forms the tree satisfies the following conditions.
  i) The data in the left child of a node is less than the data in its parent node.
  ii) The data in the right child of a node is greater than the data in its parent node.
  The left and right sub trees of the root are also binary search trees. The fig. shown below is Binary search tree.



- **Threaded Binary tree:** In linked-list representation of binary trees, additional storage space is required to store the two links of each node. The null pointers just results in wastage o memory. To overcome this drawback, the null pointers are replaced by appropriate pointe values called *threads*.

Multiple Choice Type Questions

1. If a binary tree is threaded for in-order traversal a right NULL link of any node is replaced by the address of its
   a) successor b) predecessor c) root d) own
   Answer: (a)

2. In a height balanced tree the heights of two sub-trees of every node never differ by more than
   a) 2 b) 0 c) 1 d) -1
   Answer: (c)

3. Maximum possible height of an AVL Tree with 7 nodes is
   a) 3 b) 4 c) 5 d) 6
   Answer: (a)

4. The in-order and post-order traversal of a binary tree are DBEAFC and DEBFCA respectively. What will be the total number of nodes in the left subtree of the given tree?
   a) 1 b) 4 c) 5 d) None of these
   Answer: (d)

5. A binary tree is a special type of tree
   a) That is ordered
   b) Such that no node has degree more than 2
   c) For which both (a) and (b) above are correct
   d) In which non-leaf nodes will have degree 2
   Answer: (b)

6. A B-tree is
   a) Always balanced b) An ordered tree
   c) A direct tree d) All of these
   Answer: (d)

7. Which tree structure is used for efficient access of records residing in disc memory?
   a) AVL Tree b) B Tree c) 2-3 Tree d) Binary Tree
   Answer: (b)

8. Which of the following is essential for converting an infix expression to postfix notation?
   a) A parse tree b) An operand stack
   c) An operator stack d) None of these
   Answer: (c)

9. The values in a BST can be sorted in ascending order by using which of the following traversals?
a) Pre – order b) In - order c) Post – order d) Level - order
Answer: (b)

10. The prefix expression for the infix expression a. (b+c) / e – f is
a) / * a + bc – ef b) - / * + abcef
c) - / * a + bcef d) None of these
Answer: (c)

11. In array representation of Binary tree, if the index number of a child node is 6 then the index number of its present node is
a) 2 b) 3 c) 4 d) 5
Answer: (d)

12. The depth of a complete binary tree with n nodes
a) log(n+1)–1 b) log (n) c) log(n–1)+1 d) log(n)+1
Answer: (a)

13. In a binary search tree , if the number of nodes of a tree is 9, then the minimum height of the tree is
a) 9 b) 5 c) 4 d) None of these
Answer: (d)

14. Which method of traversal does not stack to hold nodes that are waiting to be processed:
a) Breadth - first b) Depth – first
c) D- search d) None of these
Answer: (a)

15. Which of the following is essential for converting an infix expression to the postfix expression efficiently?
a) An operator stack b) An operand stack
c) An operand stack and operator stack d) A parse tree
Answer: (a)

16. Number of all possible binary trees with 4 nodes is –
a) 13 b) 12 c) 14 d) 15
Answer: (c)

17. If the inorder and preorder traversal of a binary tree are D, B, F, E, G, H, A, C and A, B, D, E, F, G, H, C respectively then, the postorder traversal of that tree is –
a) D, F, G, A, B, C, H, E b) F, H, D, G, E, B, C, A
c) C, G, H, F, E, D, B, A d) D, F, H, G, E, B, C, A
Answer: (d)

18. The number of possible distinct binary trees with 12 nodes is
a) 4082 b) 4084 c) 3082 d) 3084
Answer: (b)

19. A binary tree has n leaf nodes. The number of nodes of degree 2 in this tree is
a) $\log n$ b) $n-1$ c) n d) cannot be said
Answer: (d)

20. The minimum height of a binary tree of n nodes is
a) n b) n/2 c) n/2 – 2 d) $\log_2(n+1)$
Answer: (d)

21. The number of edges in a full binary tree of height h is
a) $2^{h+1}-1$ b) 2^h-1 c) $2^{h+1}-2$ d) 2^h-2
Answer: (c)

22. Minimum number of nodes required to make a complete binary tree of height h is
a) 2^h-1 b) 2^h c) 2^h+1 d) 2^{h-1}
Answer: (b)

23. Which one is required to reconstruct a binary tree?
a) Only inorder sequence
b) Both preorder and postorder sequences
c) Both inorder and postorder sequences
d) Only postorder sequence
Answer: (c)

24. Number of nodes in a complete binary tree of depth k is
a) 2^k b) $2k$ c) 2^k-1 d) None of these
Answer: (c)

Short Answer Type Questions

1. Prove that for any non-empty binary true T, if n_0 is the number of leaves and n_2 be the number of nodes having degree 2, then $n_0 = n_2 + 1$.

Answer:
Let n and B denote the total number of nodes & branches in T.
Let $n0$, $n1$, $n2$ represent the nodes with no children, single child, and two children respectively.
$n = n0+n1+n2$,

$B+1=n.$

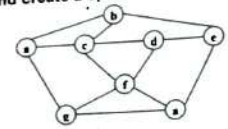
$B=n1+2n2$

$\Longrightarrow n1+2n2+1=n.$

$n1+2n2+1=n0+n1+n2$

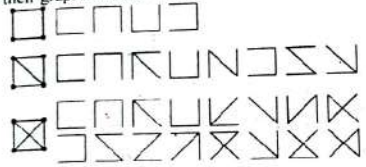
$\Longrightarrow n0=n2+1$

2. Explain spanning tree and create a spanning from the following graph.

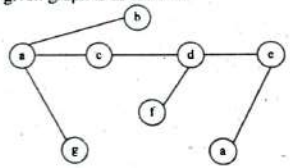


Answer:

A spanning tree of a graph is a subset of edges that form a tree. Examples of various spanning trees from their graphs are shown below:



The spanning tree for the given graph is as follows:



3. Do you consider the following data-structure as linear?

- **Binary tree**

Answer:

Nonlinear, traversing the nodes in a nonlinear pattern, root, then left or right, any one like that. So in worst case the traversing will be the height of the tree.

4. Show how the following integers can be inserted in an empty binary search tree in the order they are given:

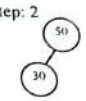
50, 30, 10, 90, 100, 40, 60, 20, 110, 5.

Draw the tree in each step.

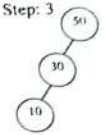
Answer:

Step 1: 50

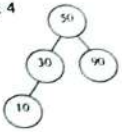
Step: 2



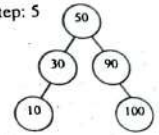
Step: 3



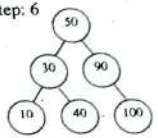
Step: 4



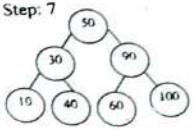
Step: 5



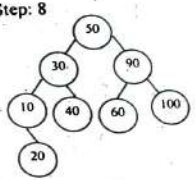
Step: 6



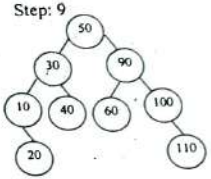
Step: 7



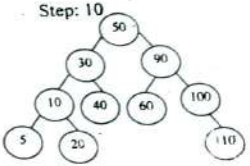
Step: 8



Step: 9



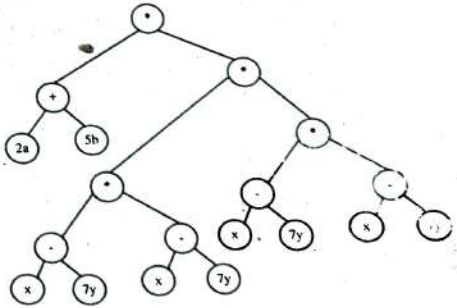
Step: 10



5. Construct the expression tree for the following expression:

$$E=(2a+5b)(x-7y)^4$$

Answer:



6. Write an algorithm for non-recursive in-order traversal of a threaded binary tree.

Answer:

Step-1: For the current node check whether it has a left child which is not there in the visited list. If it has then go to step-2 or else step-3.

Step-2: Put that left child in the list of visited nodes and make it your current node in consideration. Go to step-6.

Step-3: For the current node check whether it has a right child. If it has then go to step-4 else go to step-5

Step-4: Make that right child as your current node in consideration. Go to step-6.

Step-5: Check for the threaded node and if it's there make it your current node.

Step-6: Go to step-1 if all the nodes are not over otherwise quit

7. Write a C language function to find the in-order successor of the root of a binary tree.

Answer:

Let the structure of any node be and let d be the root node.

```
struct Node
{
    int data;
    Node* lptr;     // pointer to the left subtree
    Node* rptr;     // pointer to the right subtree
};
Node* successor(Node * d)
{
   if(d == NULL)
       return NULL;

   // If d has a Right Child
   if(d->rptr != NULL)
   {
      // Move to Right Node
      d = d->rptr;

      // Move to the extreme left
      while(d->lptr != NULL)
         d = d->lptr;

      return d;
   }
```

Below are two functions – first one to find out the minimum key in the binary tree and the second one to find the inorder successor of the root which uses first function.

```
Node *treeminimum (Node *root, Node *nil)
{
    /* returns a pointer to the minimum key in a NONEMPTY tree */

    while (root->left != nil)
        root = root->left;
    return(root);
}
Node *treesuccessor (Node *root, Node *nil)
{
    /* returns pointer to inorder successor of root, and to
       nil if root is the maximum of the tree. This code
       assumes that root != nil. */
    Node *treeminimum(Node *, Node *);
    Node *y;
    if (root->right != nil)
        return(treeminimum(root->right,nil));
    y = root->parent;
    while (y != nil && root == y->right)
        {
        root = y;
        y = y->parent;
        }
    return(y);
}
```


8. Write an algorithm to test whether a given binary tree is a binary search tree.

Answer:

The solution is to check, for every node in the tree, the min and max key values of the nodes in its left sub tree is less than the value of its key and the min and max key values in its right sub tree is greater than the value of its key. This can be achieved by recursion. The idea is to "bubble up" the min and max values of a node's sub tree, after satisfying the conditions above, to its parent node, which will in turn repeat the same process. The algorithm is designed around pre-order tree traversal and solves the problem in O(n) (linear) time.

Algorithm

For the current node:

Step1: Get the min and max values for left sub tree if left child exists

Step2: If the min and max values are greater or equal to current node's key, return "Not BST"

Step 3: If there is no left child, set the min value to current node's key value

Step 4: Get the min and max values for right sub tree if right child exists

Step 5: If the min and max values are lesser or equal to current node's key, return "Not BST"

Step 6: If there is no right sub tree, set the min value from step 1 and the max value to current node's key value

Step 7: Return the min value from step 1 and max value from step 6

9. Write an algorithm to left rotate a binary tree.

Answer:

Input: A binary tree *u*. (*u* may be a subtree of a larger tree).

Output: A (new) binary tree obtained from *u* by performing a left rotation about *u*. If *u* empty or has an empty right subtree, an empty tree is returned.

Algorithm:

```
BinTree rotateLeft( BinTree u)
    if ( u == nil or rightSubtree(u) == nil )
        return nil;
    v = rightSubtree(u);
    return buildTree(v.rightSubtree(v),buildTree( u, leftSubtree(v), leftSubtree(u)),
```

10. What is binary tree? Construct a binary tree using the Inorder and Postorder traversal of the node given below;

Inorder:	D	B	F	E	A	G	C	L	J	H	K
Postorder:	D	F	E	B	G	L	J	K	H	C	A

Answer:

1st Part:

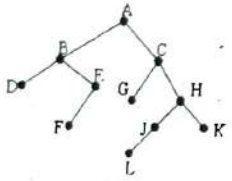
A binary tree is either empty or consists of one single vertex called the root, together with two disjoint binary trees called left subtree & right subtree. Number of branches of any node is either zero or one or at most two.

Binary trees are used to implement binary search trees and binary heaps.

2nd Part:

Based on the postorder traversal, we find A is the root. Now, DBFE is the left child and GCLJHK is the right child of A.

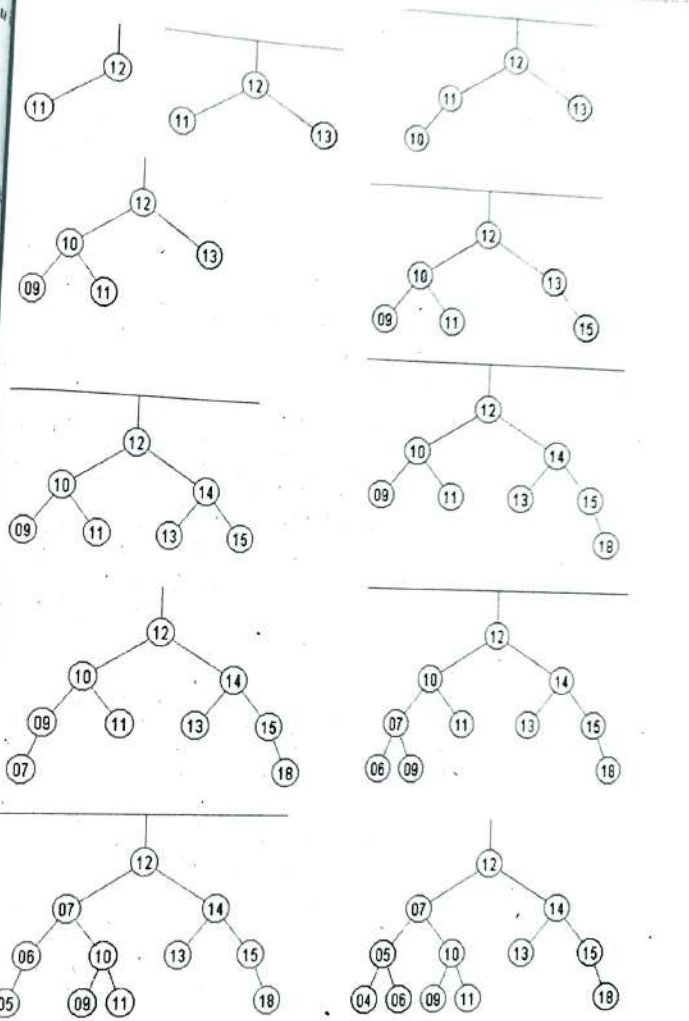
The root of right subtree would be the second last element in preorder i.e., C. We can now further divide the right subtree(with C as root), giving {G} as right subtree and {L H K} as left. In this way the final trees is given below:



11. Construct an AVL tree using the below list. Show all the steps 12, 11, 13, 10, 09, 15, 14, 18, 7, 6, 5, 4 **[WBUT 2012, 2015]**

Answer:

The steps are as shown below:



12. Prove that the maximum no. of nodes in a binary tree of depth k is $2^k - 1$.

Answer:

The maximum number of nodes in a binary tree in level i is 2i (according theorem). Hence the maximum number of nodes in a binary tree of depth k is , summation of maximum number of nodes from level 0 upto level k – 1, i.e.,

$$\sum_{i=0}^{k-1} 2^i = 2^0 + 2^1 + 2^2 + \ldots\ldots\ldots + 2^{k-1}$$

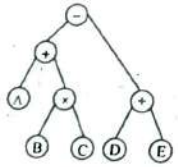
$$= 2^0 * \left(\frac{2^k - 1}{2 - 1}\right) \left[a \cdot \left((r^n - 1)/(r - 1)\right)\right]$$

$$= 2^k - 1$$

13. For the following expression draw the corresponding expression tree:

$a + b * c - d / e$

Answer:



14. a) Does a B tree grow at its leave or at its root? Why?
b) In deleting a key from a B tree, when it is necessary to combine nodes?

Answer:

a) B Tree grows at its root (bottom up). B-trees are balanced search trees designed to work well on magnetic disks or other direct-access secondary storage devices. In a B-tree, new nodes are inserted into the leaf level however, if the B-tree must increase in height, it is the root level or top of the tree that changes. This automatically preserves the balance of the tree and no rotation is necessary.

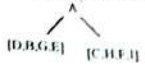
b) If a node has the minimum number of keys, then deleting a key from the node will cause an underflow and it would violate the B Tree property. It needs to be merged to another node to fix the B Tree back.

15. The post-order and in-order traversal sequences of codes in a binary tree are given below:
Post-order: D G E B H I F C A
Pre-order: D B G E A C H F I
Construct the binary tree.

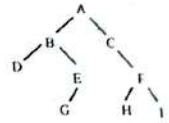
Answer:

Assuming in-order is given instead of pre-order

1) We first find the last node in post-order. The last node is "A", we know this value is root as root always appear in the end of postorder traversal.
2) We search "A" in in-order to find left and right subtrees of root. Everything on left of "A" in inorder is in left subtree and everything on right is in right subtree.



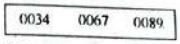
3) We recur the above process for two subtrees.



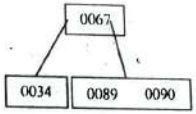
16. Construct one B-Tree of order 4 with the following data. 34, 67, 89, 90, 100, 2, 36, 76, 53, 51, 12, 10, 77, 69.

Answer:

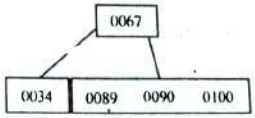
After inserting 34, 67, 89



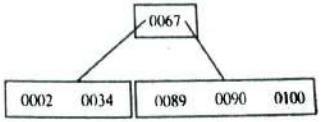
After inserting 90



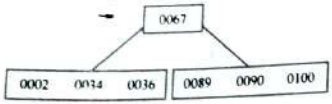
After inserting 100



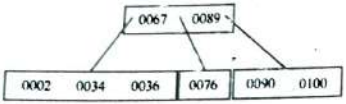
After inserting 2



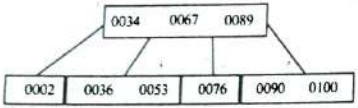
After inserting 36



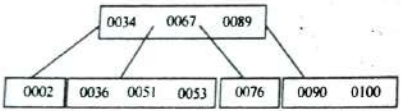
After inserting 76



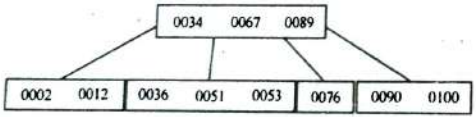
After inserting 53



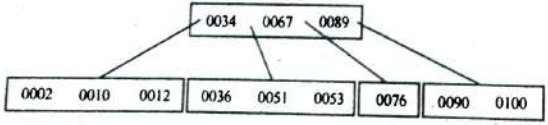
After inserting 51



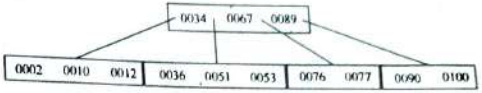
After inserting 12



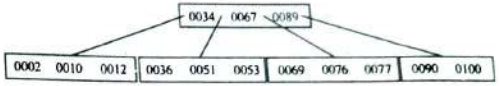
After inserting 10



After inserting 77

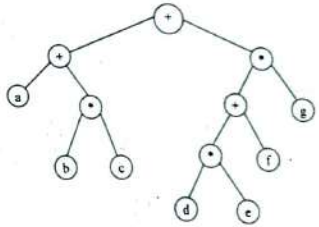


After inserting 69



17. Construct a tree from the given postfix expression $abc*+de*f+g*+$

Answer:



Postcode traversal: a b c * + d e *f + g * +

18. The inorder and preorder tree traversals are given. Draw the binary tree.

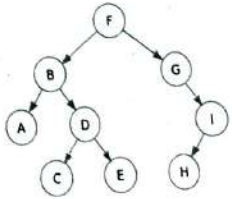
Inorder: ABCDEFGHI

Preorder: FBADCEGIH

Is it possible to build up a unique binary tree when its preorder and postorder traversals are given?

Answer:

1st part:



2nd part: ***Refer to Question No. 13(b) (1st Part) of Long Answer Type Questions.***

19. Insert the following keys into a B-Tree of given order mentioned below:
a, f, b, k, h, m, e, s, r, c. (Order 3)
a, g, f, b, k, d, h, m, j, e, s, l, r, x, c, l, n, t, u, p. (Order 5)

Answer:

1st part:

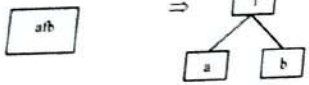
Construction of B-tree: (of order 3)

a, f, b, k, h, m, e, s, r, c

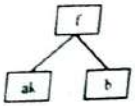
a:

f:

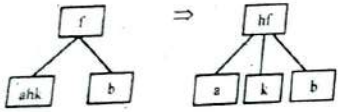
b:



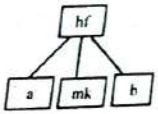
k:



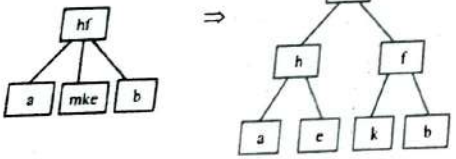
h:



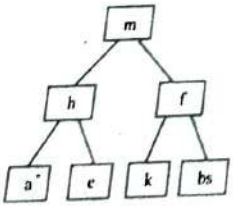
m:



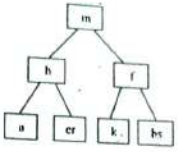
e:



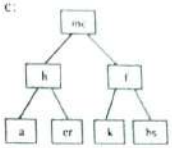
s:



r:



c:

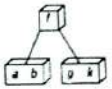


2nd Part:

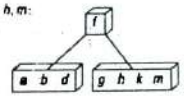
The insertion steps are shown in the figure below:

1. a, g, f, b:

2.

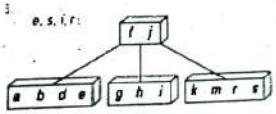


3. d, h, m:

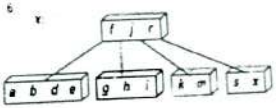


4.

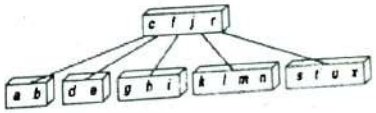
5. e, s, i, r:



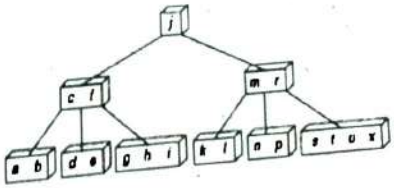
6. x:



7. c, l, n, t, u:



8. p:



20. What is expression tree? Draw the expression tree and write the In, Pre & Post Order traversals for the given expression tree: $E = (2x + y)(5a - b)^3$.

Answer:

1st Part:

An expression tree is a representation of expressions arranged in a tree like data structure. It is a tree with leaves as operands of the expression and nodes contains the operators. Data interaction is also possible in an expression tree.

2nd Part: *Refer to Question No. 2(c) of Long Answer Type Questions.*

Long Answer Type Questions

1. Write an algorithm to insert a node in a binary search tree.

Answer:

Implementing binary search tree.

We assume that a tree node has left and right pointers and a key element.

```
SearchTree Insert( ElementType X, SearchTree T )
{
    if( T == NULL )
    {
        /* Create and return a one-node tree */
        T = malloc( sizeof( struct TreeNode ) );
        if( T == NULL )
            FatalError( "Out of space!!!" );
        else
        {
            T->Element = X;
            T->Left = T->Right = NULL;
        }
    }
    else
        if( X < T->Element )
            T->Left = Insert( X, T->Left );
        else
            if( X > T->Element )
                T->Right = Insert( X, T->Right );
            /* Else X is in the tree already; we'll do nothing */
    return T;  /* Do not forget this line!! */
}
```

2. a) The inorder and preorder traversal sequence of nodes in a binary tree are given below.

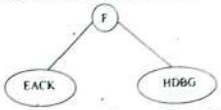
Inorder: E A C K F H D B G

Preorder: F A E K C D H G B

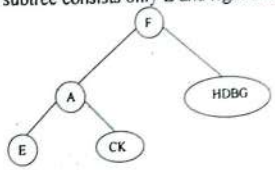
Draw the binary tree. State briefly the logic used to construct the tree.

Answer:

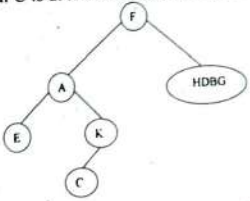
From the preorder traversal, we know that the first node is always the root node of the binary tree. So, F is the root node of the binary tree. Now, from the inorder traversal, we have to find the position of F. All the nodes of F are in the left subtree and all the nodes in right will constitute the right subtree. So, the tree looks like:



Again from the preorder traversal, next element is A. So, A is the root of the left subtree. Again if we find the position of A in the inorder traversal, left element is E & CK are right elements. So, the left subtree consists only E and right subtree consists CK.

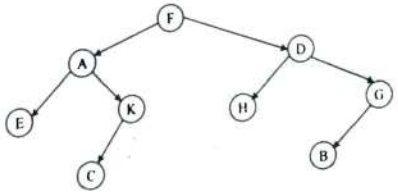


Again for CK, K occurs before C in preorder traversal, so K is the root of right subtree and from the inorder traversal C is at left of K, so, C is the left child of K.



Now, for the right subtree, HDBG of F, the first element in the preorder traversal is D and from the inorder traversal, we say that H is the left child and BG is the right subtree. Again among BG, G is the first element, so G is the root and from the inorder traversal B is the left child.

So, the final tree is



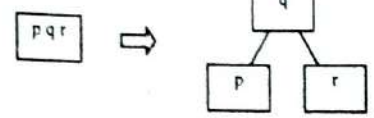
b) Insert the following keys into a B-Tree of order 3:
p, q, r, d, h, m, l, s, k, n.

Answer:

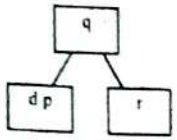
p

q

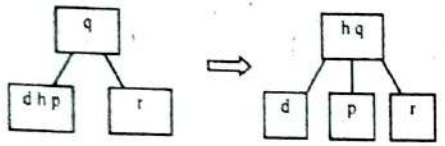
r:



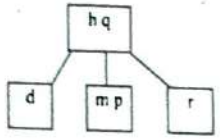
d:



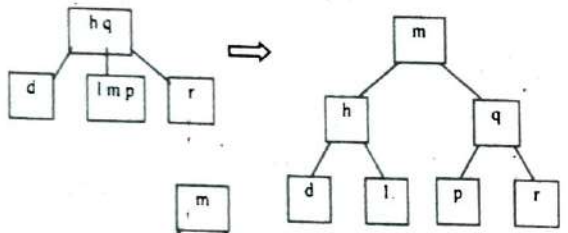
h



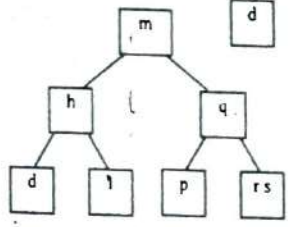
m:



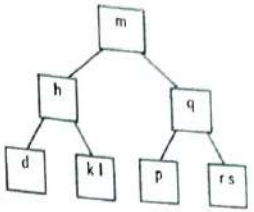
l



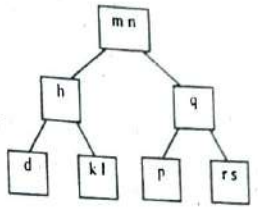
s:



k

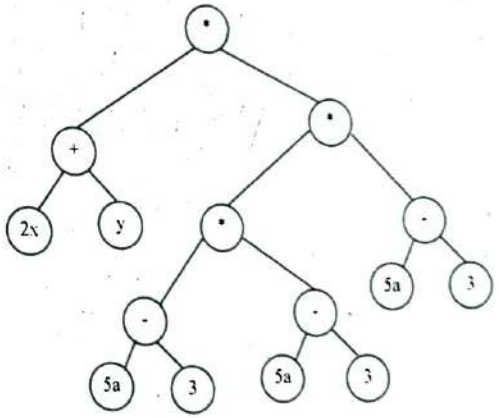


n



c) Construct an expression tree for the expression $E = (2x + y)*(5a - b)^3$.

Answer:



d) Write a non-recursive algorithm for in-order traversal of a binary tree.
[WBUT 2007, 2010, 2016]

Answer:

Assume that a stack ADT exists. The following is the non recursive C function whose input is the root pointer of the binary tree stored as a linked list with left, right and key as the elements.

```
print_inorder(Node *root)
{
 Node temp;
while(left(root)!=null)
 push(stack,left(root));
```

```
while(isEmpty(stack))
 {
  temp = pop(stack);
   print temp->key;
   push(stack,right(temp));
}
```

3. a) In a 2-tree, if E be the external path length, P be the internal path length and Q be the number of vertices that are not leaves, then prove that $E = P + 2Q$

Answer:

If a tree contains only its root and no other vertices, then E = P = Q = 0, and the first case of the above proof is trivially correct. Now let us consider a larger tree, & let v be some vertex that is not a leaf, but for which both the children of the vertex v are leaves. Let k be the number of branches on the path from the root to v.

Now let us delete the two children of v from the 2-tree. Since b is not a leaf but its children are, the number of non-leaf nodes goes down from Q to Q – 1. The internal path length I is reduced by the distance to v, that is, to P - k. The distance to each child of v is k + 1, so the external path length is reduced from E to E – 2 * (k+1), but v is now a leaf, so its distance, k, must be added, given a new external path length of

$$E - 2 * (k+1) + k = E - k - 2.$$

Since the new tree has fewer vertices then the old one; by the induction hypothesis we know that

$$E - k - 2 = (P - k) + 2 * (Q - 1)$$

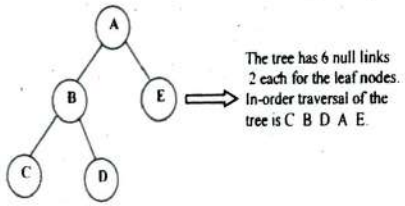
or, $E - k - 2 = P - k + 2q - 2$

or, $E = P + 2Q$. Proved.

b) What is threaded binary tree?

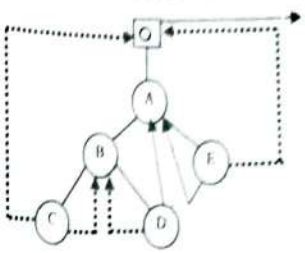
Answer:

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.



Let us consider the tree show above.

- The immediate predecessor of C will be the head node. So the left child of C, which is a null link, will point to the head node. (as per in-order traversal)
- The right null link of C will point to the immediate successor which is B (as per in-order traversal).
- The left null link of D will point to the immediate predecessor which is B (as per in-order traversal).
- The right null link of D will point to the immediate successor which is A (as per in-order traversal).



The above procedure is followed for the node E also and we get the resultant threaded binary tree.

The node structure for a threaded binary tree varies a bit and its like this --

```
struct NODE
{
 struct NODE *leftchild;
 int node_value;
 struct NODE *rightchild;
 struct NODE *thread;
}
```

c) Write an algorithm to delete a node from a binary search tree.

Answer:

In order to delete a node from a binary search tree there may be three possible scenarios:

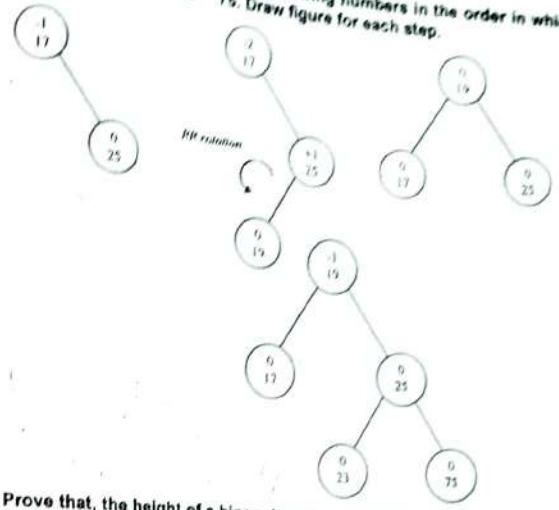
1. Node to be deleted has no children
2. The node to be deleted has only one sub tree
3. The node to be deleted has both left & right subtrees

```
node *delete(node *p, int x)
{
  node *q,  *rp,  *a,  *b,  *root;
  q = NULL;
  root = p;
  while (p!= NULL & &x != p->data)
  {
    q = p;
    if (x < p->data)
      p = p->lc;
    else
      p = p->rc;
  }
  if (p == NULL)
  {
```

```
    printf("key not present");
    exit(0);
  }
  if (p->lc == NULL) /* node p has no left child*/
    rp = p->rc;
  else if (p->rc == NULL)      /* p has no right child*/
    rp = p->lc;
  else
  {
    a = p; /* a contains the parent node p */
    rp = p->rc;
    b = rp->lc; /* b is always left child of rp */
    while (b != NULL) /* to find the inorder successor with no
left child */
    {
      a = rp;
      rp = b;
      b = rp->lc;
    }
      /* At this point rp is the inorder successor of p */
    if (a != p)
    {
      a->lc = rp->rc;
      rp->rc = p->rc;
    }
    if (q == NULL)
      nroot = rp;
    else if (p == q->lc)
    {
      q->lc = rp;
      rp->lc = p->lc;
    }
    else
    {
      q->rc = rp;
      rp->lc = p->lc;
    }
    free(p); // delete the node
    return (root);
  }
```


d) Create a AVL tree by inserting the following numbers in the order in which they are given: 17 25 19 23 75. Draw figure for each step.

Answer:



4. a) Prove that, the height of a binary tree that contains n elements, n≥0, is at most n and at least [log (n + 1)].

b) The order of nodes of a binary tree in Preorder and in order traversal are as under:

In order: D B F E G H I A C

Pre-order: A B D E F G H I C

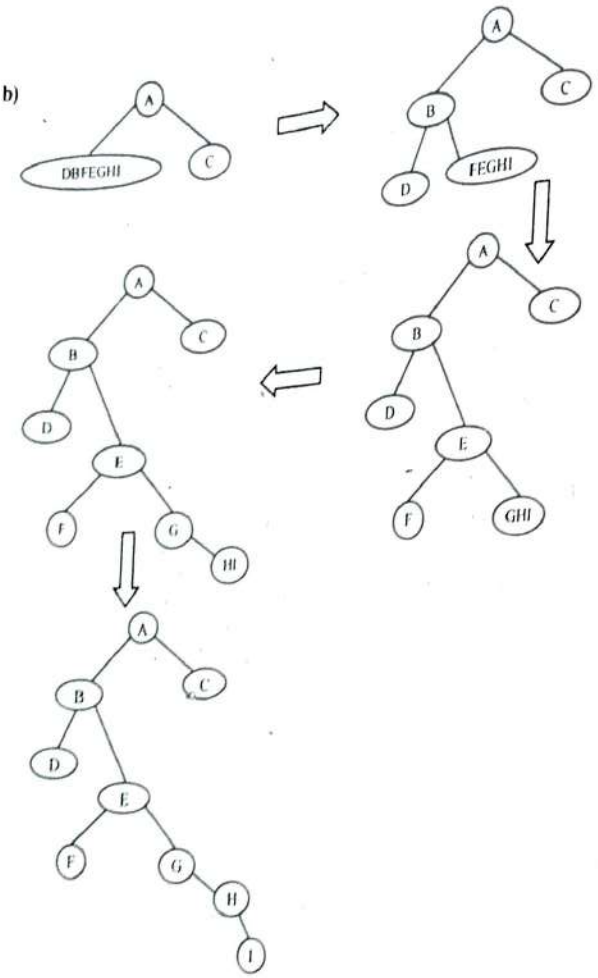
Draw the corresponding binary tree.

c) How does static allocation differ from dynamic allocation of memory?

Answer:

a) The worst case is a linear tree with only one leaf, and thus has n + 1 nodes, that is $h \leq O(n+1)$. The best base is a binary tree, with 2^h internal nodes at each level, except for the bottom level (they are all leaf nodes). However, the completion of nodes at the bottom level can effect the number of external nodes from 2^{h-1} (the bottom level has only the left-most pair of nodes) to 2^h (a complete binary tree) external nodes, where h is the height of the tree. Thus, we can have this inequality for external nodes for a true with n internal nodes: $2^{h-1} < n + 1 \leq 2^h \Rightarrow h - 1 < \lg (n + 1) \leq h$. Thus, $\lg n \leq h \leq n + 1$.

b)



c) Static allocation is done at the time the program is written. The programmer reserves a set amount of memory for each use. That setting will be used each time the program runs. Dynamic allocation is done while the program runs, adjusting the amount of memory for each use according to how much memory required at that time.

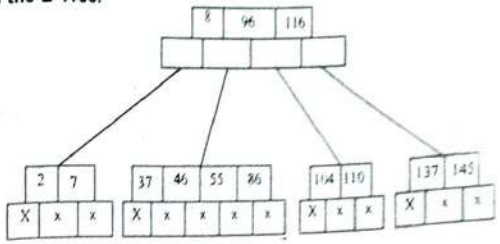
5. a) Show the steps in creation of a height balanced binary AVL TREE using insertion of items in the following order – Show the balancing steps required. (March, May, November, August, April, January, December, July, February, June, October, September)

b) What do you mean by a B-Tree and what are the uses of such a tree in data structures?

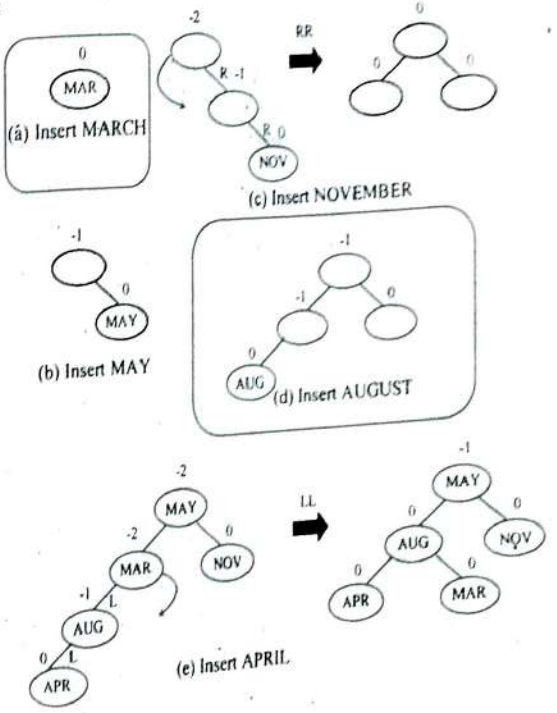
OR,

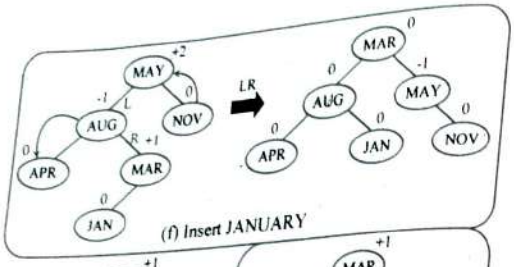
For what purposes are B trees especially appropriate?

c) Consider a B-Tree of order 5 as shown below – insert the elements 4, 5, 58, 6 in this order in the B-Tree.

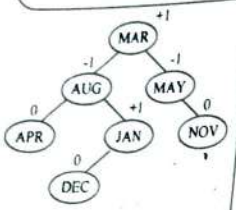


Answer:

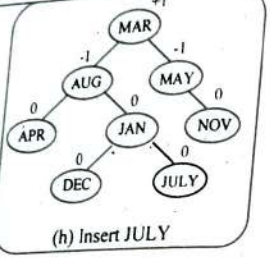




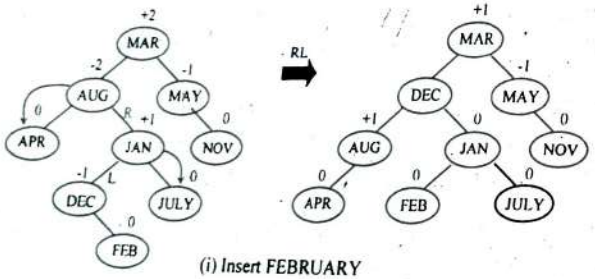
(f) Insert JANUARY



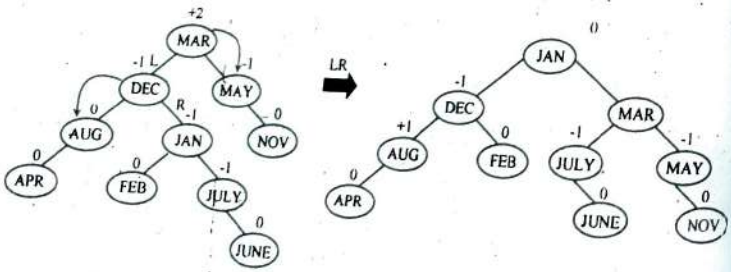
(g) Insert DECEMBER

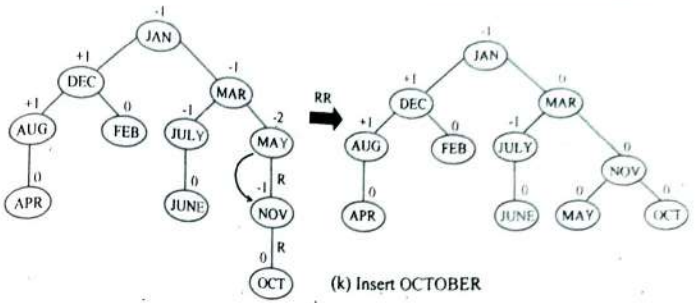


(h) Insert JULY

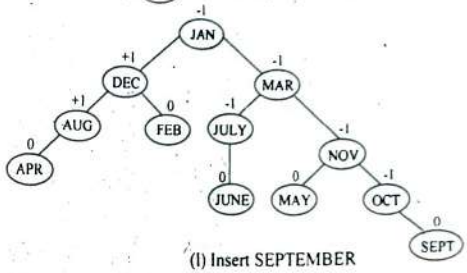


(i) Insert FEBRUARY





(k) Insert OCTOBER

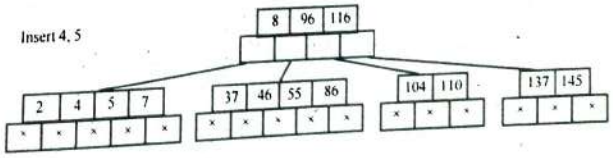


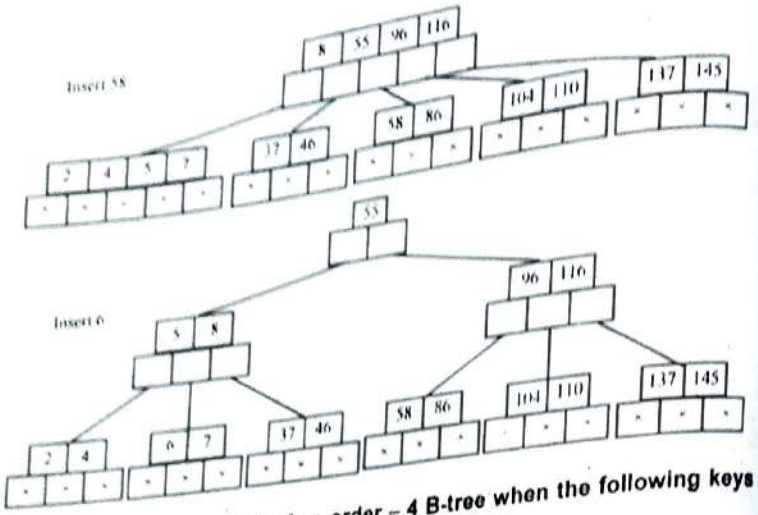
(l) Insert SEPTEMBER

b) B-trees are balanced trees that are optimized for situations when part or the entire tree must be maintained in secondary storage such as a magnetic disk. Since disk accesses are expensive (time consuming) operations, a B-tree tries to minimize the number of disk accesses. Each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node.

Magnetic disks (secondary memory) are cheaper than RAM and have higher capacity. But they are much slower because they have moving parts. B-trees try to read as much information as possible in every disk access operation.

c)

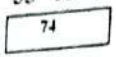




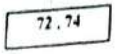
6. a) Show the stages in growth of an order – 4 B-tree when the following keys are inserted in the order given:
74 72 19 87 51 10 35 18 39 60 76 58 19 45

Answer:
74 72 19 87 51 10 35 18 39 60 76 58 19 45

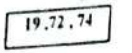
Insert 74:



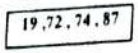
Insert 72:



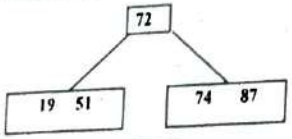
Insert 19:



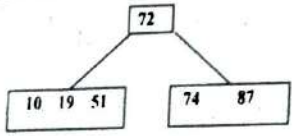
Insert 87:



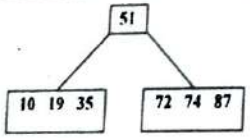
Insert 51:



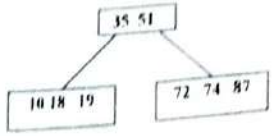
Insert 10:



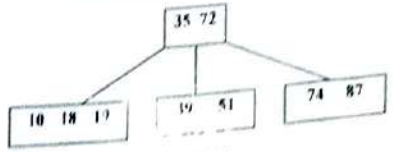
Insert 35:



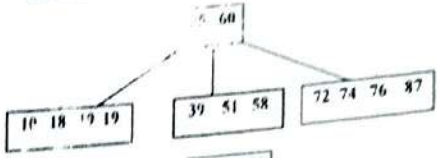
Insert 18:



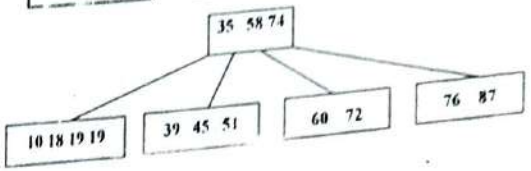
Insert 39:



Insert 60,76,58,19:



Insert 45:



b) How an AVL tree differs from binary search tree?
OR,
What are the differences between AVL Tree & Binary Search Tree?

Answer:
A binary search tree is a binary tree that is either empty or in which each node satisfies the following conditions:

i) The left child has a value smaller than the parent
ii) The right child has a value greater than the parent.

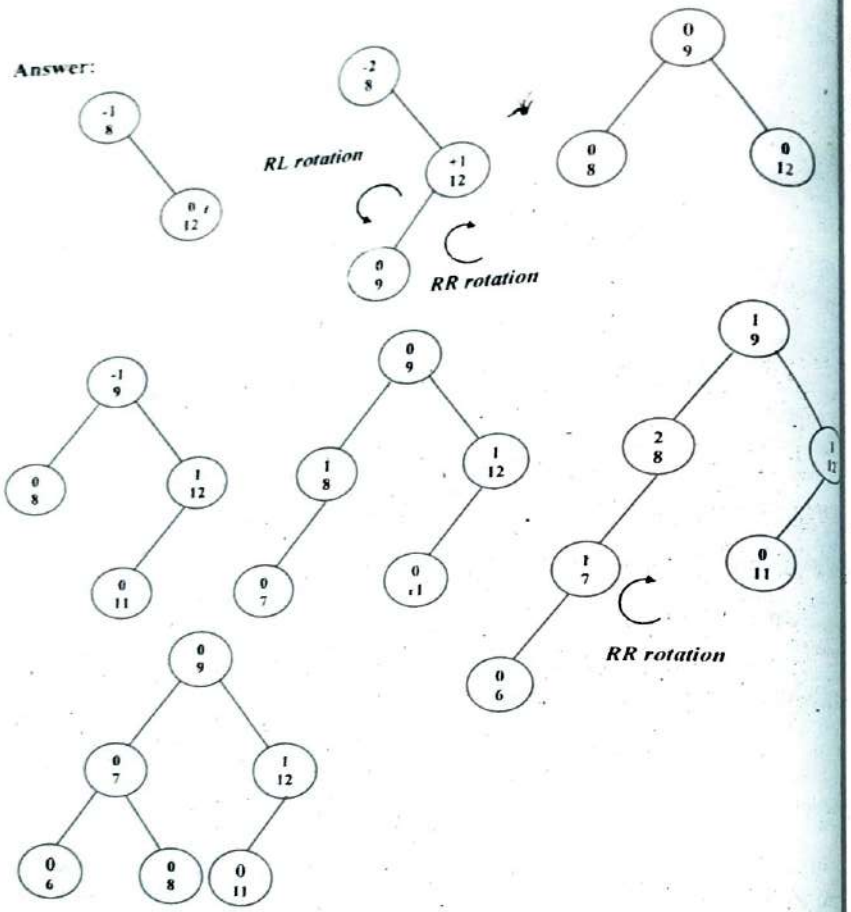
Now, an AVL tree is a height-balanced tree. If T is a non-empty binary tree with T_L & T_R as its left and right subtrees respectively, then T is height balanced if

i) T_L & T_R are height balanced
ii) $|h_L - h_R| <= 1$, where h_L & h_R are the heights of T_L & T_R respectively.

Now, an AVL tree is also a binary search tree, but it is a height balanced binary search tree. In case of binary search tree, if all the data are already sorted, then the height will be O(n), where n is the number of elements in tree. So, then the worst case search time will be O(n). But since AVL tree is height balanced, this type of case will never occur. Here we can perform searching in O (Lgn) time. Now, to construct an AVL tree, we have to perform some extra operations to get the tree height balanced.

c) Insert the following keys in the order given below to build them into an AVL tree.
8 12 9 11 7 6
Clearly mention different rotations used and balance factor of each node.

Answer:



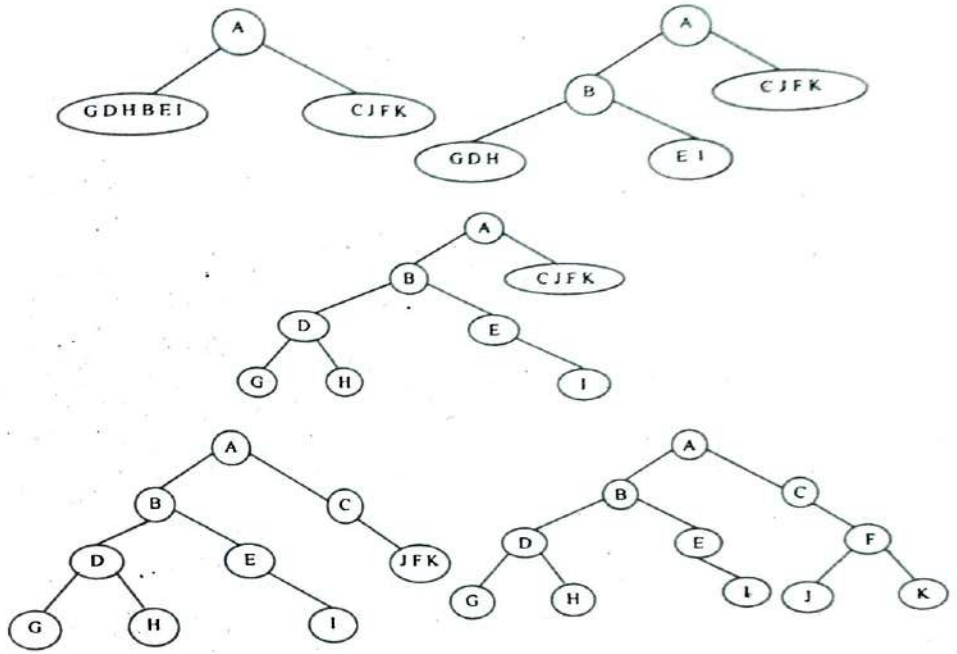
7. a) Given the preorder and inorder sequence and draw the resultant binary tr and write its postorder traversal:

Pre-order: A B D G H E I C F J K
In-order: G D H B E I A C J F K

b) Write an algorithm to search a node in a binary search tree.

Answer:

a) In preorder traversal, the first node is always the root of the binary tree. Here it is A. We look for the position of A in the corresponding in-order traversal. In order the root node is a pivot around which the left and right subtrees are constructed. So in our example all nodes lying to the left of A will represent the left subtree, & all other nodes lying to the right of A will represent the right sub-tree.
The tree looks like.



The Post Order traversal of the above tree is: G , H , D , I , E , B , J , K , F , C , A

b) p is either the root node or the node from where the search has to begin

```
void search(node *p, int digit)
{
  if (p == NULL)
    printf("Error! Tree structure not found\n");
  else if (digit == p->data)
    printf("Number=%d\n", digit);
  else if (digit < p->data)
```

```
    search(p->lc, digit); //lc = left child
  else
    search(p->rc, digit); // rc =  right child
}
```

8. What are the problems of binary tree? Explain the improvement of performance by the use of height-balanced tree.
Explain how a height-balanced tree can be formed by inserting the following elements in the given order:

1, 2, 3, 4, 5, 6, 8, 9, 10, 7, 11

Show the root element the can be deleted from the above tree.

Answer:

1st part:

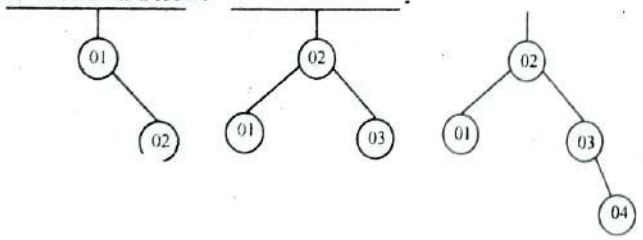
Most operations on a binary search tree (BST) take time directly proportional to the height of the tree, so it is desirable to keep the height small. Since a binary tree with height h contains at most $20+21+\cdots+2h = 2h+1-1$ nodes, it follows that the minimum height of a tree with n nodes is $\log_2(n)$, rounded down; that is, $[\log_2 n]$. However, the simplest algorithms for BST item insertion may yield a tree with height n in rather common situations. For example, when the items are inserted in sorted key order, the tree degenerates into a linked list with n nodes. The difference in performance between the two situations may be enormous: for n = 1,000,000, for example, the minimum height is $[\log_2(n)]-19$.

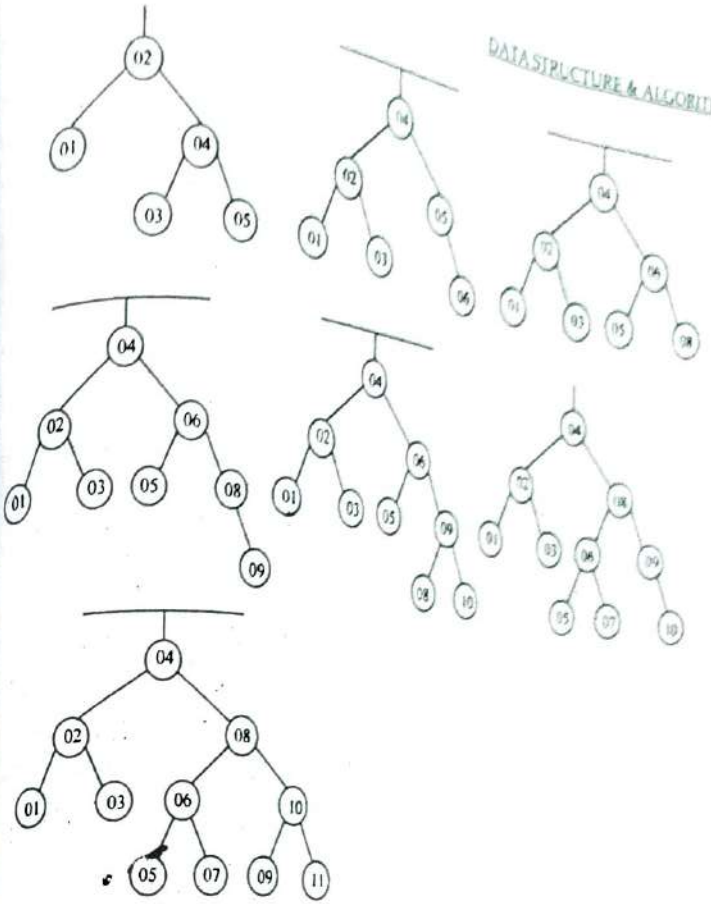
Self-balancing binary trees or height balanced tree solve this problem by performing transformations on the tree (such as tree rotations) at key times, in order to keep the height proportional to $\log_2(n)$. Although a certain overhead is involved, it may be justified in the long run by ensuring fast execution of later operations.

Maintaining the height always at its minimum value $[\log_2(n)]$ is not always viable; it can be proven that any insertion algorithm which did so would have an excessive overhead. Therefore, most self-balanced BST algorithms keep the height within a constant factor of this lower bound. In the asymptotic ("Big-O") sense, a self-balancing BST structure containing n items allows the lookup, insertion, and removal of an item in O(log n) worst-case time, and ordered enumeration of all items in O(n) time. For some implementations these are per-operation time bounds, while for others they are amortized bounds over a sequence of operations. These times are asymptotically optimal among all data structures that manipulate the key only through comparisons.

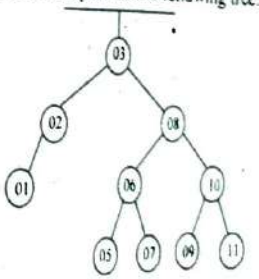
2nd part:

The steps are as shown below:





The root element 4 when deleted will produce the following tree:



9. What is a B-tree? Show how the letters A to P of English alphabet can be entered into a B-tree of order 4.

Answer:

1st Part:

A B-Tree of order m is an m-way tree in which

i) All leaves are on the same level.

ii) All internal nodes except the root have at most m nonempty children, and at least $\lceil m/2 \rceil$ nonempty children.

iii) The number of keys in each internal node is one less than the number of its nonempty children, and these keys partition the keys in the children in the fashion of a search tree.

iv) The root has at most m children, but may have as few as 2 if it is not a leaf, or none if the tree consists of the root alone.

2nd Part:

First A to H would be inserted into the root node one at a time as shown below.

When I has to be inserted, the node will be split at E and then all the alphabets up to M would be inserted as below

When J has to be inserted, the nodes will be split at E and then all the alphabets up to P would be inserted as below

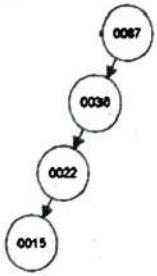
10. Insert the following number into a binary search tree in the order that they are given and draw the resulting tree.

87; 36; 22; 15; 56; 85; 48; 91; 72; 6

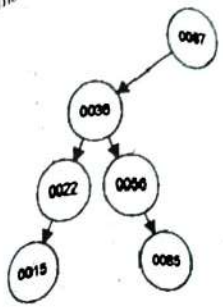
Delete 48 and draw the resulting tree. Delete 15 and draw the resulting tree.

Answer:

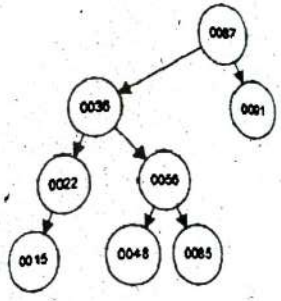
The first 4 nodes are inserted as follows:



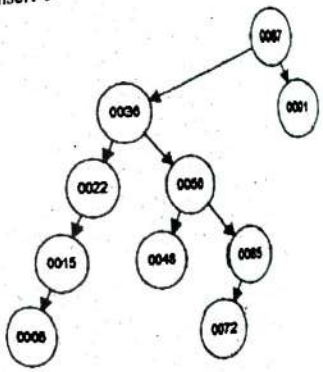
Insert 56, 85



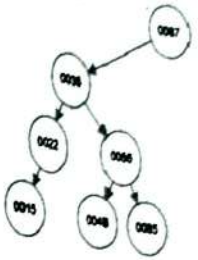
Insert 91



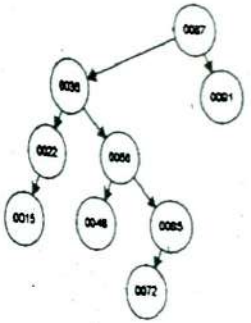
Insert 6



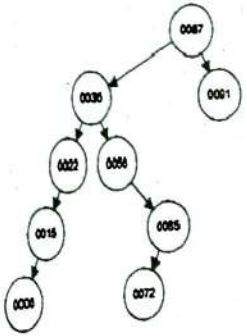
Insert 48



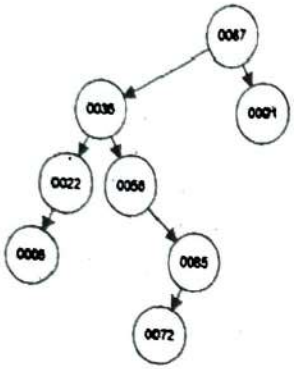
Insert into 72



Delete 48



Delete 15



11. a) Show the stages in growth of an order -4B- Tree when the following keys are inserted in the order given:

84 82 29 97 61 10 45 28 49 70 86 68 19 55 22 11 55 77 16

b) How does an AVL tree differ from a binary search tree? Insert the following keys in the order given below to build them into an AVL tree:

8 12 9 11 7 6 66 2 1 44

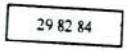
Clearly mention different rotation used and balance factor of each node.

[WBUT 2014]

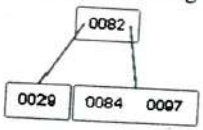
Answer:

a) As order of B-tree is 4, therefore maximum keys in one node is 3 and each node can have maximum of 4 children.

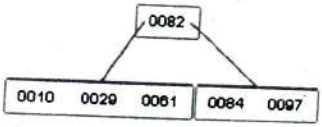
B-tree after inserting 84, 82, 29



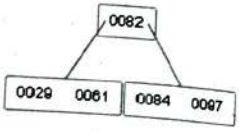
B-tree after inserting 97



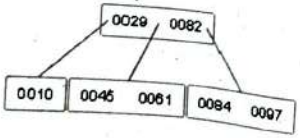
Insert 10



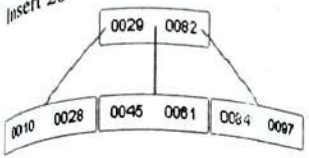
Insert 61



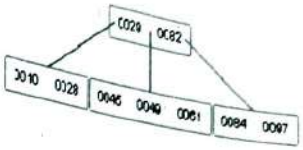
Insert 45



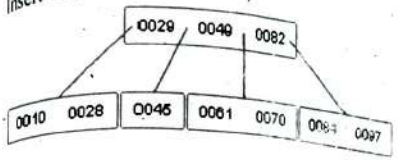
Insert 28



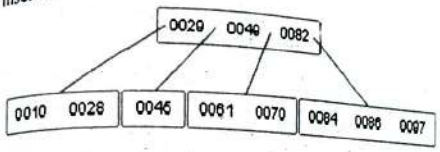
Insert 49



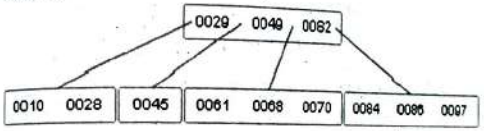
Insert 70



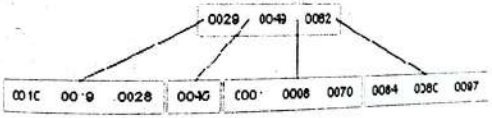
Insert 86



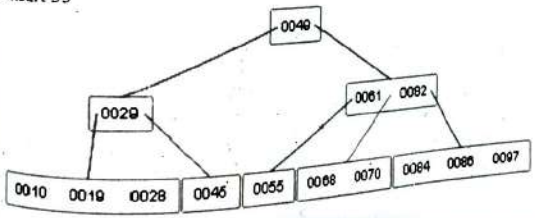
Insert 68



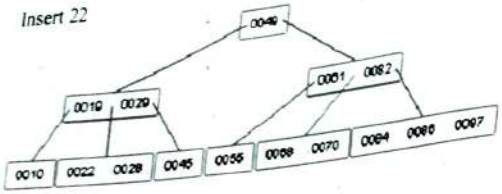
Insert 19



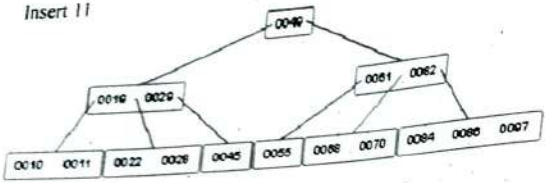
Insert 55



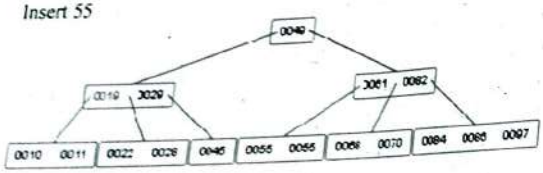
Insert 22



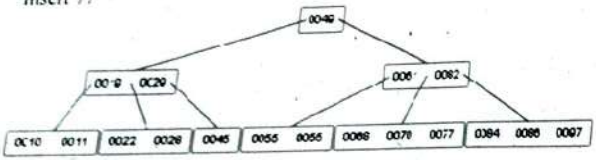
Insert 11



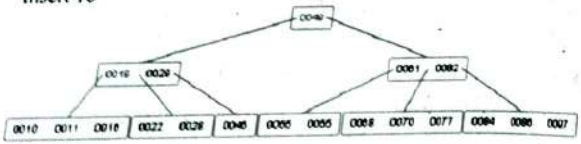
Insert 55



Insert 77



Insert 16



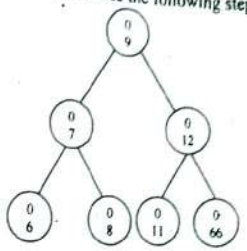
b) 1st Part: ***Refer to Question No. 6(b) of Long Answer Type Questions.***

2nd Part:

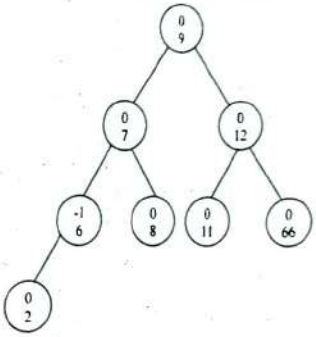
Up to insertion of 8 12 9 11 7 6:

Refer to Question No. 6(c) of Long Answer Type Questions.

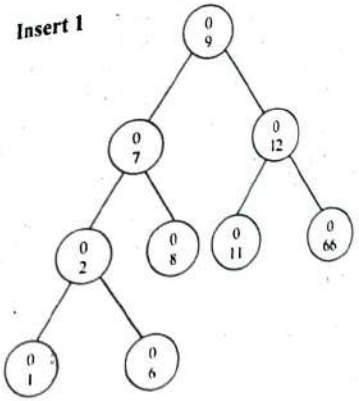
For insertion of 66, 2, 1, 44 in the above tree the following steps are needed.



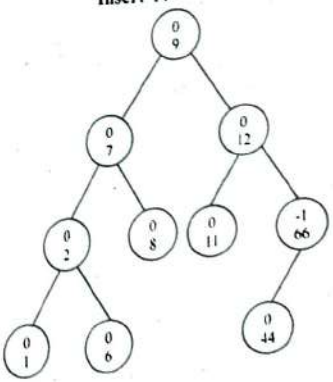
Insert 2



Insert 1



Insert 44



12. a) The in-order and pre-order traversal sequence of nodes in a binary tree are given below:

Post-order:	I	E	J	F	C	G	K	L	H	D	B	A
In-order:	E	I	C	F	J	B	G	D	K	H	L	A

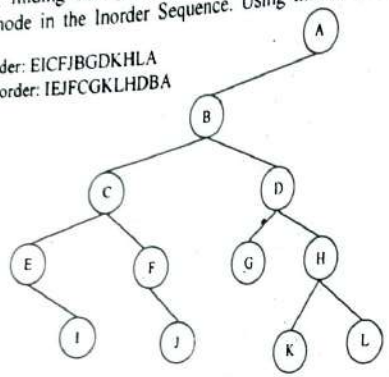
Construct the tree.

b) Write an algorithm for inserting an element into a Binary tree with example.

Answer:

a) The last node in the postorder sequence is the root. We traverse right to left in the postorder sequence, finding each node's position (left or right) with respect to the previously located node in the Inorder Sequence. Using this we construct the tree as shown below.

inorder: EICFJBGDKHLA
postorder: IEJFCGKLHDBA



b) The following C program does the insertion into a binary tree

```
/* Structure */
struct bin_tree {
 int data;
 struct bin_tree * right, * left;
};
typedef struct bin_tree node;
/*insert function*/
1 void insert(node ** tree, int val) {
2 node *temp = NULL;
3 if(!(*tree)) {
4   temp = (node *)malloc(sizeof(node));
5   temp->left = temp->right = NULL;
6   temp->data = val;
7   *tree = temp;
8   return;
9 }
10 if(val < (*tree)->data) {
11    insert(&(*tree)->left, val);
12  } else if(val > (*tree)->data) {
13    insert(&(*tree)->right, val);
14  }
15 }
16 }
```

Explanation:

[Lines 3-9] Check first if tree is empty, then insert node as root.

[Line 11] Check if node value to be inserted is lesser than root node value, then

- a. [Line 12] Call insert() function recursively while there is non-NULL left node
- b. [Lines 3-9] When reached to leftmost node as NULL, insert new node.

[Line 13] Check if node value to be inserted is greater than root node value, then

- a. [Line 14] Call insert() function recursively while there is non-NULL right node
- b. [Lines 3-9] When reached to rightmost node as NULL, insert new node.

13. a) Write a C function to find out the maximum and the minimum elements in a binary search tree.

b) Given the pre-order sequence and the post-order sequence, why cannot you reconstruct the tree?

Answer:

a) Assume the binary search tree nodes are defined as follows:

```
struct node
{
struct node *lchild;
int info;
struct node *rchild;
};
```

/*Min function -recursive */

```
struct node *Min(struct node *ptr)
{
if(ptr==NULL)
return NULL;
else if(ptr->lchild==NULL)
        return ptr;
else
return Min(ptr->lchild);
)/*End of min()*/
```

/*Max function -recursive */

```
struct node *Max(struct node *ptr)
{
if(ptr==NULL)
return NULL;
```

```
else if(ptr->rchild==NULL)
        return ptr;
else
return Max(ptr->rchild);
}/*End of max()*/
```

b) It is not possible to construct a binary tree just from pre and post order traversals of a binary tree because only inorder traversals give the position of the sub-tree elements with respect to the root.

If one has inorder along with pre or post order one can always construct a unique binary tree because the pre/post order information gives you the root of the tree. Once the root of the tree is known one can recursively construct the left and right sub-trees which are binary trees themselves thereby making the final product unique.

14. a) What do you mean by a binary search tree?
b) Construct a binary search tree by inserting the list of elements one by one:
13, 10, 3, 5, 18, 15, 14
c) Write an algorithm for pre-order traversal of a tree represented by a linked-list.

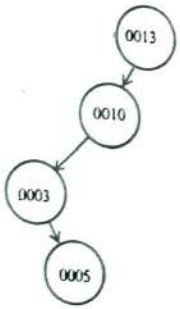
Answer:

a) A binary search tree is a binary tree that is either empty or in which each node satisfies the following conditions:

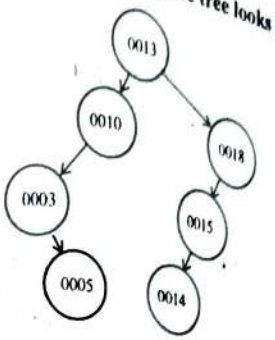
i) The left child has a value smaller than the parent

ii) The right child has a value greater than the parent.

b) After inserting 13, 10, 3, 5 the tree looks like



Now, after inserting 18, 15, 14 the tree looks like



c) //C program implementation

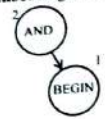
```
/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left;
    struct node* right;
};

/* Given a binary tree, print its nodes in preorder*/
void printPreorder(struct node* node)
{
     if (node == NULL)
          return;

     /* first print data of node */
     printf("%d ", node->data);

     /* then recur on left subtree */
     printPreorder(node->left);
     /* now recur on right subtree */
     printPreorder(node->right);
}
```

15. a) What is an AVL tree?
b) Construct an AVL search tree for the data list:
AND, BEGIN, CASE, DO, END, FOR GOTO.
c) For the AVL tree you have constructed delete the following keys in the order:
DO, FOR END.

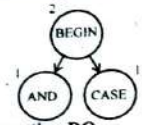
Answer:

a) *Refer to Question No. 16(a) of Long Answer Type Questions.*

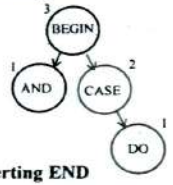
b) After inserting AND and BEGIN:



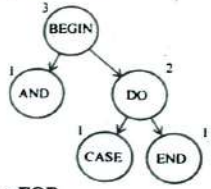
After inserting CASE



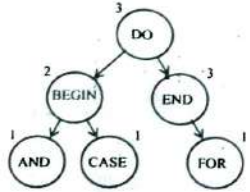
After inserting DO



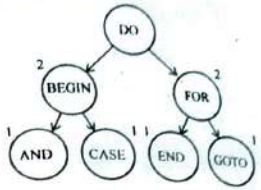
After inserting END



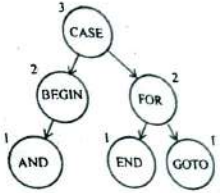
After inserting FOR



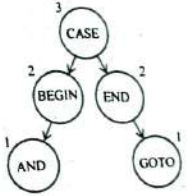
After inserting GOTO



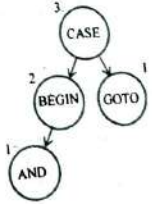
c) After removing DO



After removing FOR



After removing END



16. Write short notes on the following:
a) AVL Tree
b) Threaded Binary Tree
c) B – tree
d) Binary Search Tree
Answer:
a) AVL Tree:
An AVL Tree is a form of binary tree, however unlike a binary tree, the worst case scenario for a search is O (log n). The AVL data structure achieves this property by placing restrictions on the difference in height between the sub-trees of a given node, and re-balancing the tree if it violates these restrictions.
The complexity of an AVL Tree comes from the balance requirements it enforces on each node. A node is only allowed to possess one of three possible states (or balance factors):
Left-High (balance factor -1)
The left-sub tree is one level taller than the right-sub tree
Balanced (balance factor 0)
The left and right sub-trees are both the same heights
Right-High (balance factor +1)
The right sub-tree is one level taller than the left-sub tree.
If the balance of a node becomes -2 (it was left high and a level was lost from the left sub-tree) or +2 (it was right high and a level was lost from the right sub-tree) it will require re-balancing. This is achieved by performing a rotation about this node.

b) Threaded Binary Tree:
Refer to Question No. 3(b) of Long Answer Type Questions.

c) B – tree:
Definition: A B-Tree of order m is an m-way tree in which
i) All leaves are on the same level.
ii) All internal nodes except the root have at most m nonempty children, and at least $\lceil m/2 \rceil$ nonempty children.
iii) The number of keys in each internal node is one less than the number of its nonempty children, and these keys partition the keys in the children in the fashion of a search tree.
iv) The root has at most m children, but may have as few as 2 if it is not a leaf, or none if the tree consists of the root alone.
There are some disadvantages also
- The nodes of a B-tree do not normally contain the same number of value entries.
- They normally do contain a certain amount of free space.
- Difficulty in traversing the keys sequentially.

Note: B-tree is not a binary tree. A B-tree is also known as the balanced sort tree.

d) Binary Search Tree:
Refer to Question No. 14(a) & (b) of Long Answer Type Questions.

5 GRAPHS

Chapter at a Glance

- A graph is a mathematical tool used to represent a physical problem. It is also used to model networks, data structures, scheduling, computation and a variety of other systems where the relationship between the objects in the system plays a dominant role.
- **Types of graph:** A graph often symbolized as G can be of two types:
  1. **Undirected graph:** where a pair of vertices representing an edge is unordered.
  2. **Directed graph:** where a pair of vertices representing an edge is ordered.
- **Graph Traversal:** A graph can be traversed in two ways:
  Depth first search traversal: often analogous to pre-order traversal of an ordered tree.
  Breadth first search traversal: often analogous to level-by-level traversal of an ordered tree.
- **Spanning Tree:** Given a connected, undirected graph, a spanning tree of that graph is a sub graph which is a tree and connects all the vertices together. A single graph can have many different spanning trees. We can also assign a *weight* to each edge, which is a number representing how unfavourable it is, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree.
- **Shortest Path:** In graph theory, the shortest path problem is the problem of finding a path between two vertices in a weighted graph such that the sum of the weights of its constituent edges is minimized.
  The most important algorithms for solving this problem are:
  Dijkstra's algorithm solves the single-source shortest path problems.
  Bellman-Ford algorithm solves the single source problem if edge weights may be negative.
  A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search.

Multiple Choice Type Questions

1. The vertex, removal of which makes a graph disconnected is called
a) Pendant vertex **b) bridge**
c) articulation point **d) colored vertex**
Answer: (c)

2. A vertex of in-degree zero in a directed graph is called
a) articulation point **b) sink**
c) isolated vertex **d) root vertex**
Answer: (c)

3. Adjacency matrix of a digraph is
a) identity **b) symmetric**
c) asymmetric **d) none of these**
Answer: (b)

4. Which data structure is used for breadth first traversal of a graph?
   a) Stack b) Queue
   c) Both stack and queue d) None of these
Answer: (b)

5. The adjacency matrix of an undirected graph is
   a) Unit matrix b) Asymmetric matrix
   c) Symmetric matrix d) None of these
Answer: (c)

6. BFS
   a) scans all incident edges before moving to the other vertex
   b) scans adjacent unvisited vertex as soon as possible
   c) is same as backtracking
   d) none of these
Answer: (b)

7. A non-planar graph with minimum number of vertices has
   a) 9 edges, 6 vertices b) 6 edges, 4 vertices
   c) 10 edges, 5 vertices d) 9 edges, 5 vertices
Answer: (c)

8. Any connected graph with x vertices must have at least
   a) x+ 1 edges b) x-1 edges c) x edges d) x/2 edges
Answer: (b)

9. Maximum number of edges in a n-node undirected graph without self loop is

a) n^2 b) $\frac{n(n-1)}{2}$ c) $n-2$ d) $\frac{(n+1)(n)}{2}$

Answer: (b)

10. BFS constructs
   a) a minimal cost spanning tree of a graph
   b) a depth first spanning tree of a graph
   c) a breath first spanning tree of a graph
   d) none of these
Answer: (a)

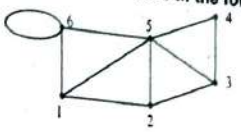
11. A complete directed graph of 5 nodes has.....
   a) 5 b) 10 c) 20 d) 25
Answer: (b)

12. A vertex with degree one in a graph is called
   a) leaf b) pendant vertex c) end vertex d) none of these
Answer: (b)

13. To implement DFS which data structure is generally used?
   a) Stack b) Queue c) Both (a) & (b) d) None of these
Answer: (a)

14. Adjacency matrix for a digraph is –
   a) unit matrix
   c) asymmetric matrix b) symmetric matrix
   d) none of these
Answer: (b)

15. What is the sum of the degrees of all the vertices in the following graph?



   a) 19
   b) 20
   c) 5
   d) none of these
Answer: (a)

16. The adjacency matrix of an undirected graph is
   a) Unit matrix b) Asymmetric matrix
   c) Symmetric matrix d) none of these
Answer: (c)

17. A path is
   a) a closed walk with no vertex repetition
   b) an open walk with no vertex repetition
   c) an open walk with no edge repetition
   d) a closed walk with no edge repetition
Answer: (c)

Short Answer Type Questions

1. Describe Kruskal's minimal spanning tree algorithm.

Answer:

Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest (a minimum spanning tree for each connected component). Kruskal's algorithm is an example of a greedy algorithm.

It works as follows:

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty
- remove an edge with minimum weight from S

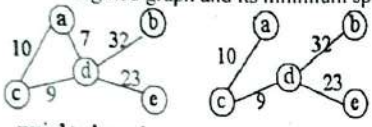
- if that edge connects two different trees, then add it to the forest, combining two trees into a single tree
- Otherwise discard that edge.
- At the termination of the algorithm, the forest has only one component and forms a minimum spanning tree of the graph.

2. What is a minimum spanning tree? Describe Huffman's Algorithm. [WBUT 2012]

Answer:

1st Part:

A Minimum Spanning Tree in an undirected connected weighted graph is a spanning tree of minimum weight (among all spanning trees). The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique. The following figures show a weighted graph and its minimum spanning tree.



weighted graph Tree 1. w=74

2nd Part:

Huffman's algorithm is a method for building an extended binary tree with a minimum weighted path length from a set of given weights. Initially construct a forest of singleton trees, one associated with each weight. If there are at least two trees, choose the two trees with the least weight associated with their roots and replace them with a new tree, constructed by creating a root node whose weight is the sum of the weights of the roots of the two trees removed, and setting the two trees just removed as this new node's children. This process is repeated until the forest consists of one tree.

Pseudocode

Huffman(W, n)
Input: A list W of n (positive) weights.
Output: An extended binary tree T with weights
taken from W that gives the minimum weighted path length.
Procedure:
Create list F from singleton trees formed from elements of W
WHILE (F has more than one element) DO
Find T1, T2 in F that have minimum values associated with their roots
Construct new tree T by creating a new node and setting T1 and T2 as its children
Let the sum of the values associated with the roots of T1 and T2 be associated with the root of T
Add T to F
End DO
Huffman := tree stored in F

3. What is the adjacency matrix representation of a graph?

Answer:

The adjacency matrix of a graph is

	1	2	3	4	5	6	7
1	0	1	1	0	0	1	0
2	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1
5	0	0	0	0	1	1	0
6	0	1	0	0	0	0	0
7	0	0	0	0	1	0	0

4. Write the Prim's algorithm for finding MST from a graph.

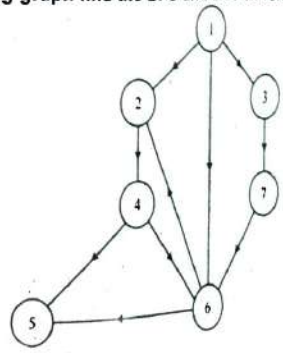
Answer:

Prim's algorithm is a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Algorithm

1) Create a set ***mstSet*** that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3) While mstSet doesn't include all vertices
....a) Pick a vertex ***u*** which is not there in ***mstSet*** and has minimum key value.
....b) Include ***u*** to mstSet.
....c) Update key value of all adjacent vertices of ***u***. To update the key values, iterate through all adjacent vertices. For every adjacent vertex ***v***, if weight of edge ***u-v*** is less than the previous key value of ***v***, update the key value as weight of ***u-v***

5. For the following graph find the BFS and DFS traversal with proper algorithm.



Answer:

a) (i) The BFS traversal of the graph is:

Event	*Queue (Front to Rear)*	*BFS*
Visit 1		1
Visit 2	2	12
Visit 3	23	123
Visit 6	236	1236
Remove 2	36	
Visit 4	364	12364
Remove 3	64	
Visit 7	647	123647
Remove 6	47	
Visit 5	475	1236475
Remove 4	75	
Remove 7	5	
Remove 5		
Done		

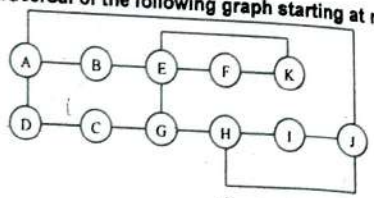
Required BFS is 1 2 3 6 4 7 5

(ii) The DFS traversal of the graph is:

Event	*Stack*	*DFS*
Visit 1	1	1
Visit 2	12	12
Visit 4	124	124
Visit 5	1245	1245
Pop 5	124	
Pop 4	12	
Visit 6	126	12456
Pop 6	12	
Pop 2	1	
Pop 1	-	
Visit 3	3	124563
Visit 7	37	1245637
Remove 7	3	
Remove 3	-	
Done		

Required DFS is 1 2 4 5 6 3 7.

6. Find out the DFS traversal of the following graph starting at node A.



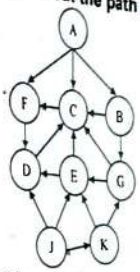
Answer:

For DFS traversal we employ the following rules using stack

- Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

Using the above rules we get can get the following traversal when starting with A:
A B E G H I J F K C D

7. Apply BFS/DFS Algorithms and find out the path of the given graph:



Answer:

The steps involved in **breadth first traversal** are as follows:

| Search Steps
|----------
| Step 1 -> Node A
| Step 2 -> Node B
| Step 3 -> Node C
| Step 4 -> Node F
| Step 5 -> Node G
| Step 6 -> Node D
| Step 7 -> Node E

The steps involved in **depth first traversal** are as follows:

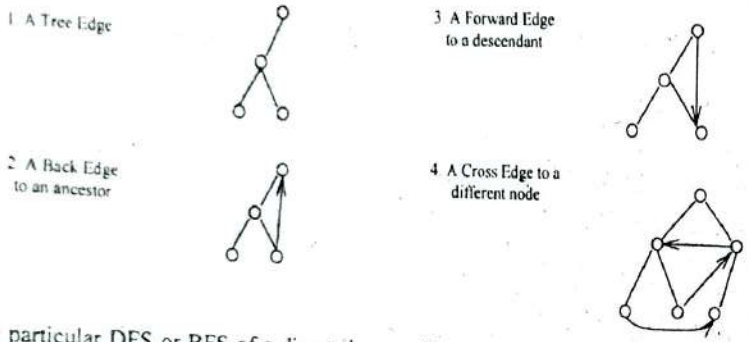
| Step 1 -> Node A
| Step 2 -> Node.B
| Step 3 -> Node C
| Step 4 -> Node F
| Step 5 -> Node D
| Step 6 -> Node G
| Step 7 -> Node E

8. Describe the Edge classification of DFS algorithm. **[MODEL QUESTION]**

Answer:

Consider a directed graph G = (V, E). After a DFS of graph G we can put each edge into one of four classes:

1. A ***tree edge*** is an edge in a DFS-tree.
2. A ***back edge*** connects a vertex to an ancestor in a DFS-tree. Note that a self-loop is a back edge.
3. A ***forward edge*** is a non-tree edge that connects a vertex to a descendent in a DFS-tree.
4. A ***cross edge*** is any other edge in graph G. It connects vertices in two different DFS-tree or two vertices in the same DFS-tree neither of which is the ancestor of the other.

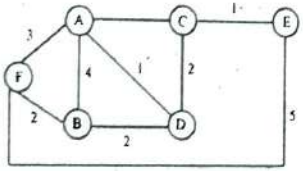


Any particular DFS or BFS of a directed or un-directed graph, each edge gets classified as one of the above.

In a DFS of an undirected graph, every edge is either a tree edge or back edge. Here no cross edge goes to a higher numbered or rightward vertex. The reason behind the DFS algorithm is so important is that it defines a very nice ordering of edges of the graph.

Long Answer Type Questions

1. a) Compare BFS and DFS. Discuss the two different ways of representing a graph.



OR,

Describe BFS algorithm.

Answer:

1st Part:

Depth First Search: DFS of a graph is analogous to the per-order traversal of an ordered tree. For a given graph the DFS will have the following rules.

Rule 1: If possible, visit an adjacent unvisited vertex, mark it visited, and push it on the stack.

Rule 2: If Rule 1 fails, then if possible pop a vertex off the stack follow Rule 1 from it.

Rule 3: Repeat Rule 1 and Rule 2 until the all the vertices are visited.

Breadth First Search: BFS of a graph is analogous to level-by-level traversal of an ordered tree.

For a given graph the BFS will have the following rules.

Rule 1: Visit all the **next** unvisited vertices (if any) that is adjacent to the current vertex, and insert them into a queue one at a time on every visit.

Rule 2: If there is no unvisited vertex, remove a vertex from the Queue and make it the current vertex and then follow Rule 1.

Rule 3: Repeat Rule 1 and Rule 2 until the all the vertices visited.

Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

The time complexity can also be expressed as $O(|E|+|V|)$ since every vertex and every edge will be explored in the worst case.

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking. The time complexity is also given by $O(|E|+|V|)$.

2nd Part:

There are three ways by which graphs can be represented in memory.

i) Adjacency matrices.
ii) Adjacency list.
iii) Adjacency multilists

i) ***Adjacency matrices:*** For a graph G with n vertices the adjacency matrix is a 2D array with n x n element.

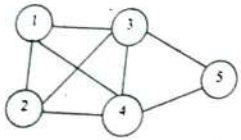
For undirected graphs

A[i,j] = 1, if there is an edge between i & j
A[i,j] = 0, if there is no edge between i & j.

For directed graphs

A [i, j] = 1, if there is an edge directed i & j

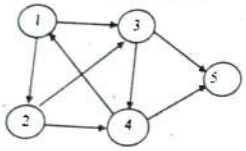
A [i, j] = 0, otherwise



Undirected graph

Adjacency Matrix

	1	2	3	4	5
1	0	1	1	1	0
2	1	0	1	1	0
3	1	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0



Directed graph

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	1	1	0
3	0	0	0	1	1
4	1	0	0	0	1
5	0	0	0	0	0

The adjacency matrix for an undirected graph is symmetrical i.e., diagonal elements are zero.

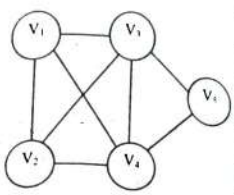
Advantage: It is possible to determine whether an edge is connecting two vertices or not.

Disadvantage:

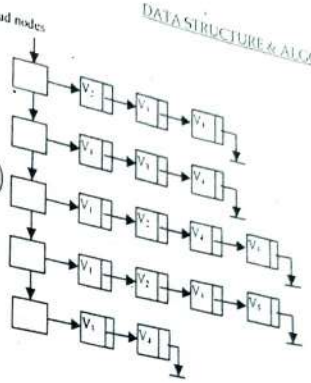
- It requires n x n memory locations. If the graph is not complete, there will be a number of zero entries & this will waste the memory area. To avoid this, the adjacency list method is used.
- There will be two entries for the same edge. This information is redundant.

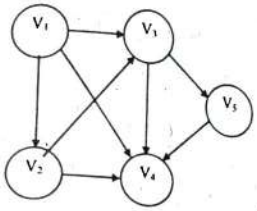
ii) ***Adjacency List:*** It is a linked list representation of a graph where each vertex represents a head node. Each head node has a list associated with it which represents the edges. The nodes of the list form a path between the head node and all other reachable vertices in the graph from that head node.

Example

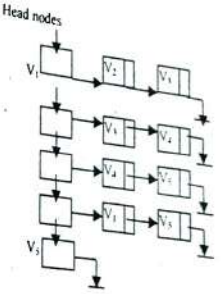


Undirected Graph





Directed graph



Drawbacks: When a graph is large, it is necessary to handle to many numbers of pointers, which can be complex.

iii) ***Adjacency Multilist:*** From the adjacency list representation of an undirected graph, it is clear that each edge (Vi, Vj) is represented by two entries, one is list Vi and another is list Vj. This is unnecessary wastage of the memory as the information present is same at both the nodes (1,2)= (2,1).

To avoid this, the adjacency multilist is used. Using this representation, the nodes are shared amongst the several lists.

In this method for each edge of the graph, there will be exactly one node. But this node will provide the information about two more nodes to which it is incident.

The node structure with this representation will be:

m	V_1	V_2	P_1	P_2

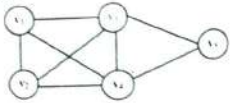
m --- tagfield

V_1, V_2 ....Vertex

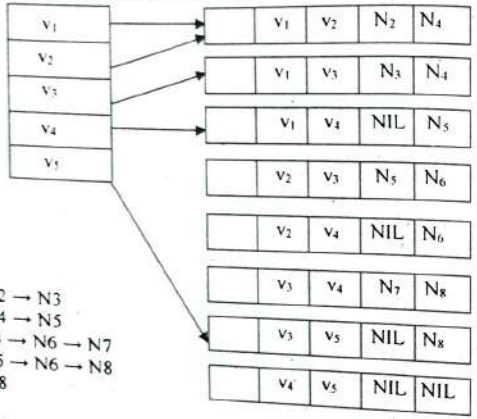
P_1, P_2----- Incident paths

m is a one bit mark field that is used to indicate whether or not the edge has been examined.

Consider the following graph.



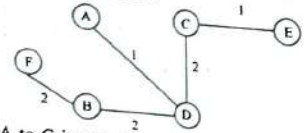
Total distinct edges are:
$\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_2, v_5\}$, $\{v_2, v_4\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$ & $\{v_4, v_5\}$. All other edges represent the same information, because this is an undirected graph.
The adjacency multilist representation is as follows



The lists are:
Vertex 1 : N1 → N2 → N3
Vertex 2 : N1 → N4 → N5
Vertex 3 : N2→ N4 → N6 → N7
Vertex 4 : N3 → N5 → N6 → N8
Vertex 5 : N7 → N8

b) Draw the minimum cost spanning tree for the graph given below and also find its cost.

Answer:
The tree has edges AD, BD, CD, CE and BF.
Cost is 8.



Note: Since the cost of A to C is not given in the question it is assumed that the cost is ≥5.

c) What is Complete Graph?
Show that the sum of degree of all the vertices in a graph is always even.

OR,

Show that the number of vertices of odd degree in a finite graph is even.

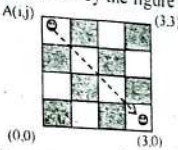
OR,

Prove that the number of odd degree vertices in a graph is always even.

Answer:
1st Part: A graph is said to be complete if there exist an edge between every pair of vertices.
2nd Part: The number of odd degree vertices in a graph is always even:
Degree of a vertex is defined as no. of edges incident on a particular vertex with the self loop counted twice. Now, let us take the sum of degree of all the vertices. Now, this summation is an even number, because each edge contributes twice when we are calculate degree of different vertices. Now, if we take the summation of all the degrees that is nothing but the twice the number of edges $\sum_{i=1}^{n} d(v_i) = 2e \Rightarrow$ *even number*

2. A rat has entered a checkerboard maze through one corner, whose the white boxes are open and black boxes represent obstacles. Develop an algorithm by which the rat can exit the maze through the opposite corner. Clearly explain the representation of the maze and any specific data structure you have used for the algorithm.

Answer:
Let the checkerboard maze be represented by the figure below. For simplicity we show a 4×4 checkerboard.



The maze can be represented by a $n \times n$ two dimensional array. Since black boxes are obstacles, the rat will only travel through white boxes. So the best path to follow will be the diagonal.
The algorithm will be as follow:
Steps:
i←0; j←0
1. Start with A(i, j)
2. While (i < n, j < n)
   i = i + 1
   j = j + 1
3. Final position A(n, n)

3. Write short notes on the following:
a) BFS vs DFS
b) BFS
c) Dijkstra's Algorithm
d) DFS in graph

Answer:

a) BFS vs. DFS: *Refer to Question No. 1(a) of Long Answer Type Questions.*

b) BFS: *Refer to Question No. 1(a) of Long Answer Type Questions.*

c) Dijkstra's Algorithm:

Dijkstra's algorithm works on the principle that the shortest possible path from the source has to come from one of the shortest paths already discovered. A way to think about this is the "explorer" model--starting from the source, we can send out explorers each travelling at a constant speed and crossing each edge in time proportional to the weight of the edge being traversed. Whenever an explorer reaches a vertex, it checks to see if it was the first visitor to that vertex: if so, it marks down the path it took to get to that vertex. This explorer must have taken the shortest path possible to reach the vertex. Then it sends out explorers along each edge connecting the vertex to its neighbors.

It is useful for each vertex of the graph to store a "prev" pointer that stores the vertice from which the "explorer" came from. This is the vertex that directly precedes the current vertex on the path from the source to the current vertex.

The pseudocode for Dijkstra's algorithm is fairly simple and reveals a bit more about what extra information needs to be maintained. Vertices will be numbered starting from 0 to simplify the pseudocode.

```
Given a graph, G, with edges E of the form (v1, v2) and vertices
V, and a
source vertex, s
dist : array of distances from the source to each vertex
prev : array of pointers to preceding vertices
i    : loop index
F    : list of finished vertices
U    : list or heap unfinished vertices

/* Initialization: set every distance to INFINITY until we
discover a path */
for i = 0 to |V| - 1
   dist[i] = INFINITY
   prev[i] = NULL
end
/* The distance from the source to the source is defined to be
zero */
dist[s] = 0
/* This loop corresponds to sending out the explorers walking the
paths, where
```



```
, the step of picking the vertex, v, with the shortest path to
s' corresponds
, to an explorer arriving at an unexplored vertex */
while(F is missing a vertex)
   pick the vertex, v, in U with the shortest path to s
   add v to F
   for each edge of v, (v1, v2)
      /* The next step is sometimes given the confusing name
      "relaxation"
         if(dist[v1] + length(v1, v2) < dist[v2])
         dist[v2] = dist[v1] + length(v1, v2)
         prev[v2] = v1
         possibly update U, depending on implementation
            end if
         end for
end while
```

d) DFS in graph: *Refer to Question No. 1(a) of Long Answer Type Questions.*

6 SORTING & HASHING

Chapter at a Glance

- **A hash table** data structure is just like an array. Data is stored into this array at specific index generated by a hash function. **A hash function** hashes (converts) a number in a large range, into a number in a smaller range.
- **Linear search:** Linear or Sequential Search is a method where the search begins at one end of the list, scans the elements of the list from left to right (if the search begins from left) until the desired record is found.
  This type of searching can be performed in an ordered list or an unordered list. For ordered lists that must be accessed sequentially, such as linked lists or files with variable-length records lacking an index, the average performance can be improved by giving up at the first element which is greater than the unmatched target value, rather than examining the entire list.
- **Binary Search:** In Binary Search the entire sorted list is divided into two parts. We first compare our input item with the mid element of the list and then restrict our attention to only the first or second half of the list depending on whether the input item comes left or right of the mid-element. In this way we reduce the length of the list to be searched by half.
- **Hashing:** In hash tables, there is always a possibility that two data elements will hash to the same integer value. When this happens, a **collision** occurs i.e. two data members try to occupy the same place in the hash table array. There are methods to deal with such situations like **Open Addressing and Chaining**.
- **Bubble sort:** Given an array of unsorted elements, Bubble sort performs a sorting operation on the first two adjacent elements in the array, then between the second & third, then between third & fourth & so on.
- **Insertion sort:** In insertion sort data is sorted data set by identifying an element that is out of order relative to the elements around it. It removes that element from the list, shifting all other elements up one place.
  Finally it places the removed element in its correct location.
  For example, when holding a hand of cards, players will often scan their cards from left to right, looking for the first card that is out of place. If the first three cards of a player's hand are 4, 5, 2, he will often be satisfied that the 4 and the 5 are in order relative to each other, but, upon getting to the 2, desires to place it before the 4 and the 5. In that case, the player typically removes the 2 from the list, shifts the 4 and the 5 one spot to the right, and then places the 2 into the first slot on the left.
- **Quick sort:** In Quick-Sort we divide the array into two halves. We select a pivot element (normally the middle element of the array) and perform a sorting in such a manner that all the elements to the left of the pivot element is lesser than it & all the elements to it's right is greater than the pivot element. Thus we get two sub arrays.
- **Merge sort:** In this method, we divide the array or list into two sub arrays or sub lists as nearly equal as possible and then sort them separately. Then the sub-arrays are again divided into another sub arrays.
- **Heap sort:** A heap takes the form of a binary tree with the feature that the maximum or minimum element is placed in the root. Depending upon this feature the heap is called max-heap or min-heap respectively. After the heap construction the elements from the root are taken out from the tree and the heap structure is reconstructed. This process continues until the heap is empty.

Multiple Choice Type Questions

1. The ratio of the number of items in a hash table, to the table size is called the
   a) load factor b) item factor c) balanced factor d) all of these
   Answer: (a)

2. Which of the following is not a requirement of good hashing function?
   a) Avoid collision b) Reduce the storage space
   c) Make faster retrieval d) None of these
   Answer: (b)

3. Stability of Sorting Algorithm is important for
   a) Sorting records on the basis of multiple keys
   b) Worst case performance of sorting algorithm
   c) Sorting alpha numeric keys as they are likely to be the same
   d) None of these
   Answer: (a)

4. Which of the following is the best time for an algorithm?
   a) O(n) b) O(log2 n) c) O(2n) d) O(n log2 n)
   Answer: (b)

5. The Linear Probing Technique for collision resolution can lead to
   a) Primary clustering b) Secondary clustering
   c) Overflow d) Efficiency storage utilization
   Answer: (a)

6. The fastest sorting algorithm for an almost already sorted array is
   a) quick sort b) merge sort c) selection sort d) insertion sort
   Answer: (d)

7. The time complexity of binary search is
   a) $O(n^2)$ b) O(n) c) O(log n) d) O(n log n)
   Answer: (c)

8. The best case time complexity of Bubble sort technique is
   a) $O(n)$ b) $O(n^2)$ c) $O(n \log n)$ d) $O(\log n)$
   Answer: (a)

9. Which of the following sorting procedures is the slowest?
a) Quick sort b) Heap sort c) Merge sort d) Bubble sort
Answer: (d)

10. Which of the following traversal techniques lists the elements of a binary search tree in ascending order?
a) pre-order b) Post-order c) Inorder d) None of these
Answer: (c)

11. Binary search cannot be used in linked lists.
a) True b) False
Answer: (b)

12. Breadth-first-search algorithm uses.....data structure
a) stack b) queue c) binary tree d) none of these
Answer: (b)

13. The best case complexity of insertion sort is –
a) $O(n^2)$ b) $O(n \log_2 n)$ c) $O(n^3)$ d) $O(n)$
Answer: (d)

14. Which of the following is not related to hashing?
a) Synonyms b) Collision c) Balance d) Load factor
Answer: (c)

15. A machine needs a minimum of 100sec to sort 1000 names by quick sort. The minimum time needed to sort 100 names will be approximately
a) 72.7 sec b) 11.2 sec c) 50.2 sec d) 6.7 sec
Answer: (d)

16. What will be the time complexity for selection sort to sort an array of n elements?
a) O(log n) b) O(n log n) c) O(n) d) $O(n^2)$
Answer: (d)

17. The best sorting technique when the data is almost sorted is
a) Selection sort b) Bubble sort c) Quick sort d) Insertion sort
Answer: (d)

18. Which of the following is a hash function?
a) Quadratic probing b) chaining
c) open addressing d) folding
Answer: (a)

19. The number of swapping needed to sort numbers 8, 22, 7, 9, 31, 19, 5, 13 in ascending order using bubble sort is
a) 11 b) 12 c) 13 d) 14
Answer: (d)

20. Binary search uses
a) divide and reduce strategy b) divide and conquer strategy
c) heuristic search d) both (a) and (b)
Answer: (b)

21. Merge sort uses
a) divide and conquer strategy b) backtracking approach
c) heuristic search d) greedy approach
Answer: (a)

22. The prerequisite condition of Binary search is
a) unsorted array b) ascending order array
c) descending order array d) sorted array
Answer: (d)

Short Answer Type Questions

1. Explain the advantages of binary search over sequential search.

Answer:
A sequential search of either a list, an array, or a chain looks at the first item, the second item, and so on until it either finds a particular item or determines that the item does not occur in the group. Average case of sequential search is $O(n)$
A binary search of an array requires that the array be sorted. It looks first at the middle of the array to determine in which half the desired item can occur. The search repeats this strategy on only this half of the array.
The benefit of binary search over linear search becomes significant for lists over about 100 elements. For smaller lists linear search may be faster because of the speed of the simple increment compared with the divisions needed in binary search.
Thus for large lists binary search is very much faster than linear search, but is not worthwhile for small lists.
Binary search is not appropriate for linked list structures (no random access for the middle term).

2. What is hashing? OR,

Define Hashing.
What is collision?
Explain Linear Probing & Quadratic Probing with example.

OR,

Define 'Hashing'. Explain with a suitable example the collision resolution scheme using linear probing with open addressing.

OR,

Define Hashing. Explain one collision resolution scheme citing one example.

OR,

Write two hash functions

Answer:

1st Part:

Hashing is a method for storing and retrieving records from a database. It lets you insert, delete, and search for records based on a search key value. When properly implemented, these operations can be performed in constant time. In fact, a properly tuned hash system typically looks at only one or two records for each search, insert, or delete operation. This is far better than the O(log *n*) average cost required to do a binary search on a sorted array of *n* records, or the O(log *n*) average cost required to do an operation on a binary search tree. However, even though hashing is based on a very simple idea, it is surprisingly difficult to implement properly. Designers need to pay careful attention to all of the details involved with implementing a hash system.

A hash system stores records in an array called a **hash table**, which we will call **HT**. Hashing works by performing a computation on a search key *K* in a way that is intended to identify the position in **HT** that contains the record with key *K*. The function that does this calculation is called the **hash function**, and is usually denoted by the letter **h**. Since hashing schemes place records in the table in whatever order satisfies the needs of the address calculation, records are not ordered by value. A position in the hash table is also known as a **slot**. The number of slots in hash table **HT** will be denoted by the variable *M* with slots numbered from 0 to *M* - 1.

2nd Part:

A **collision** between two keys K & K' occurs when both have to be stored in the table & both hash to the same address in the table.

3rd Part:

Open addressing: It is a general collision resolution scheme for a hash table. In case of collision, other positions of the hash table are checked (**a probe sequence**) until an empty position is found.

The different types of Open addressing scheme includes

a) Linear Probing (Sequential Probing)
b) Quadratic Probing
c) Double Hashing (Re Hashing)

Linear probing is a used for resolving hash collisions of values of hash functions by sequentially searching the hash table for a free location. This is accomplished using two values - one as a starting value and one as an interval between successive values in modular arithmetic. The second value, which is the same for all keys and known as the

stepsize, is repeatedly added to the starting value until a free space is found, or the entire table is traversed.

The function for the rehashing is the following:

rehash(key)=(n+1) % k;

For example, we have a hash table that could accommodate 9 information, and the data to be stored were integers. To input 27, we use hash(key)= 27 % 9=0. Therefore, 27 is stored at 0. If another input 18 occurs, and we know that 18 % 9= 0, then a collision would occur. In this event, the need to rehash is needed. Using linear probing, we have the rehash(key)= (18+1) % 9= 1. Since 1 is empty, 18 can be stored in it.

Quadratic probing:

In order to prevent collision we use quadratic probing scheme.

- In quadratic probing,
- We start from the original hash location i
- If a location is occupied, we check the locations $i+1^2$, $i+2^2$, $i+3^2$, $i+4^2$ ...
- We wrap around from the last table location to the first table location if necessary.

Let us take the following example:

Table Size is 11 (0..10)

- Hash Function: h(x) = x mod 11
- Insert keys (20, 30,2,13,25,24,10,9):
  - 20 mod 11 = 9
  - 30 mod 11 = 8
  - 2 mod 11 = 2
  - 13 mod 11 = 2 → $2+1^2=3$
  - 25 mod 11 = 3 → $3+1^2=4$
  - 24 mod 11 = 2 → $2+1^2$, $2+2^2=6$
  - 10 mod 11 = 10
  - 9 mod 11 = 9 → $9+1^2$, $9+2^2$ mod 11, $9+3^2$ mod 11 =7

The figure below shows the corresponding entries in the hash table.

0	
1	
2	2
3	13
4	25
5	
6	24
7	9
8	30
9	20
10	10

3. Prove that, the best case time complexity for quick sort is O(nlogn) for input size of n.

Answer:

The analysis of the procedure QUICK_SORT is given by

$T(N) = P(N) + T(J-LB) + T(UB - J)$

where P(N), T(J-LB), and T(UB-J) denote the times to partition the given table, sort the left subtable, and sort the right subtable, respectively. Note that the time to partition a table is O(N). The worst case occurs when, at each invocation of the procedure, the current table is partitioned into two subtables with one of them being empty (that is J = LB or J = UB). Such a situation, for example, occurs when the given key set is already sorted.

The worst case time analysis, assuming J = LB, then becomes

$Tw = P(N) + Tw(0) + Tw(N-1)$
$= c*N + Tw(N-1)$
$= c*N + c*(N-1) + Tw(N-2)$
$= c*N + c*(N-1) + c*(N-2) + Tw(N-3)$

$= c * \{(N+1)(N)\}/2 = O(N2)$

The best case analysis occurs when the table is always partitioned in half, that is, J = [(LB+UB)/2]. The analysis becomes:

$Tb = P(N) + 2Tb(N/2)$
$= c*N + 2Tb(N/2)$
$= c*N + 2c(N/2) + 4Tb(N/4)$
$= c*N + 2c(N/2) + 4c(N/4) + 8Tb(N/8)$

$= (log2N)*c*N + 2log2N*Tb(1)$
$= O(Nlog2N)$

4. Give an algorithm to search for an element in an array using binary search.

OR,

What is the precondition of performing binary search in an array? Write the Binary Search algorithm.

Answer:

Let us consider an array A of size N = 10. The left index and right index are left = 0 right = (N–1) = 9

```
int bin_search(int A[], int n, int item)
{
int left = 0,
right = n - 1;
int flag = 0; //a flag to indicate whether the element is found
int mid = 0;
while (left <= right) // till left and right cross each other
{
mid = (left + right)/2; // divide the array into two halves
if (item == A[mid])
{
flag = 1; //set flag = 1if the element is found
break;
}
else if (item < A[mid])
right = mid - 1; /* compute new right from the right sub array */
else
left = mid + 1; /* compute new left from the left sub array */
}
return flag;
}
```

5. "Binary search technique cannot be implemented using Linked list." — Justify the validity of the statement.

Answer:

The statement is not true as it can Binary Search can be implemented using Linked List – however it would be less efficient than array.

Binary search on an array so fast and efficient because of its the ability to access any element in the array in constant time. Thus one can get to the middle of the array just by saying array[middle].

The same cannot be done with a linked list. One needs to write an algorithm to get the value of the middle node of a linked list. In a linked list, one loses the ability to get the value of any node in a constant time.

One solution to the inefficiency of getting the middle of the linked list during a binary search is to have the first node contain one additional pointer that points to the node in the middle. Decide at the first node if one needs to check the first or the second half of the linked list. Continue doing that with each half-list.

6. Draw a minimum heap tree from the below list:

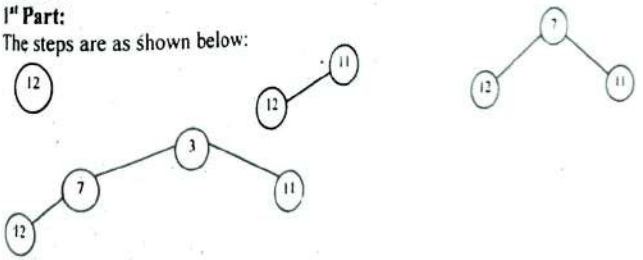
12, 11, 7, 3, 10, –5, 0, 9, 2

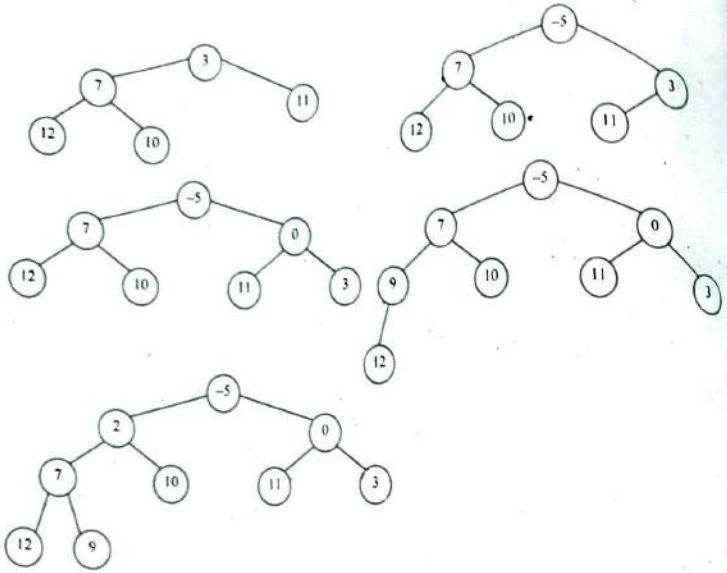
Now do the heap sort operation over the heap tree which you have formed. Write the insertion sort algorithm.

Answer:

I" Part:

The steps are as shown below:





Performing sorting in min heap will result in the array being sorted in descending order. The array now looks like: -5, 2, 0, 7, 10, 11, 3, 12, 9

The steps for sorting are:

Tree nodes: 9, 2, 0, 7, 10, 11, 3, 12 Sorted Array: -5
Tree nodes: 0, 2, 9, 7, 10, 11, 3, 12 Sorted Array: -5
Tree nodes: 12, 2, 9, 7, 10, 11, 3 Sorted Array: 0, -5
Tree nodes: 3, 2, 9, 7, 10, 11, 12 Sorted Array: 0, -5
Tree nodes: 2, 3, 9, 7, 10, 11, 12 Sorted Array: 0, -5
Tree nodes: 3, 9, 7, 10, 11, 12 Sorted Array: 2, 0, -5
Tree nodes: 9, 7, 10, 11, 12 Sorted Array: 3, 2, 0, -5
Tree nodes: 7,9, 10, 11, 12 Sorted Array: 3,2, 0, -5
Tree nodes: 9, 10, 11, 12 Sorted Array: 7, 3,2, 0, -5
Tree nodes: 10, 11, 12 Sorted Array: 9, 7, 3,2, 0, -5
Tree nodes: 11, 12 Sorted Array: 10, 9, 7, 3,2, 0, -5
Tree nodes: 12 Sorted Array: 11, 10, 9, 7, 3,2, 0, -5
Tree nodes: Sorted Array: 12, 11, 10, 9, 7, 3,2, 0, -5

2nd Part:

If the first few objects are already sorted, an unsorted object can be inserted in the sorted set in proper place. This is called insertion sort. An algorithm consider the elements one at a time, inserting each in its suitable place among those already considered (keeping them sorted). Insertion sort is an example of an **incremental** algorithm; it builds the sorted sequence one number at a time.

Pseudocode

We use a procedure INSERTION_SORT. It takes as parameters an array $A[1..n]$ and the length n of the array. The array A is sorted in place: the numbers are rearranged within the array, with at most a constant number outside the array at any time.

```
INSERTION_SORT (A)
1.  FOR j ← 2 TO length[A]
2.      DO key ← A[j]
3.         {Put A[j] into the sorted sequence A[1 . . j − 1]}
4.         i ← j − 1
5.         WHILE i > 0 and A[i] > key
6.               DO A[i +1] ← A[i]
7.                  i ← i − 1
8.         A[i + 1] ← key
```

7. Write the pseudo code for Heap sort.

Answer:

The Algorithm for heap-sort is as follows.

```
#include <stdio.h>
#include <conio.h>
#define N 6
void buildheap(int[], int);
void heapsort(int[], int);
int main(void)
{
  int heapArr[N] ={15, 19, 10, 7, 17, 6};
  int i;
  printf("\nBefore Sorting:\n");
  for (i = 0; i < N; i++)
  printf("%d\t", heapArr[i]);
  buildheap(heapArr, N);
  heapsort(heapArr, N);
  printf("\nAfter Sorting:\n");
  for (i = 0; i < N; i++)
   printf("%d\t", heapArr[i]);
  getch();
  return 0;
}
void buildheap(int x[], int n)
{
  int i, val, s, f;
  for (i = 1; i < n; i++)
  {
    val = x[i];
    s = i;
```

```
    f = (s - 1) / 2;
    while (s > 0 && x[f] < val)
    {
      x[s] = x[f];
      s = f;
      f = (s - 1) / 2;
    }
    x[s] = val;
  }
}
void heapsort(int x[], int n)
{
  int i, s, f, ivalue;
  for (i = n - 1; i > 0; i--)
  {
    ivalue = x[i];
    x[i] = x[0];
    f = 0;
    if (i == 1)
      s = -1;
    else
      s = 1;
    if (i > 2 && x[2] > x[1])
      s = 2;
    while (s >= 0 && ivalue < x[s])
    {
      x[f] = x[s];
      f = s;
      s = 2 * f + 1;

      if (s + 1 <= i - 1 && x[s] < x[s + 1])
        s++;
      if (s > i - 1)
        s = -1;
    }
    x[f] = ivalue;
  }
}
```

8. Deduce the average time complexity of Quicksort algorithm.

Answer:

The basic idea of Quicksort is as given below:

1. Pick one element in the array, which will be the *pivot*.
2. Make one pass through the array, called a *partition* step, re-arranging the entries so that:
   - the pivot is in its proper place.
   - entries smaller than the pivot are to the left of the pivot.
   - entries larger than the pivot are to its right.

3. Recursively apply quicksort to the part of the array that is to the left of the pivot, and to the right part of the array.

Analysis

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

The average value of T(i) is 1/N times the sum of T(0) through T(N-1)

1/N S T(j), j = 0 thru N-1

T(N) = 2/N (S T(j)) + cN

Multiply by N

NT(N) = 2(S T(j)) + cN*N

To remove the summation, we rewrite the equation for N-1:

$(N-1)T(N-1) = 2(S\ T(j)) + c(N-1)^2$, j = 0 thru N-2

and subtract:

NT(N) - (N-1)T(N-1) = 2T(N-1) + 2cN -c

Prepare for telescoping. Rearrange terms, drop the insignificant c:

NT(N) = (N+1)T(N-1) + 2cN

Divide by N(N+1):

T(N)/(N+1) = T(N-1)/N + 2c/(N+1)

Telescope:

T(N)/(N+1) = T(N-1)/N + 2c/(N+1)

T(N-1)/(N) = T(N-2)/(N-1)+ 2c/(N)

T(N-2)/(N-1) = T(N-3)/(N-2) + 2c/(N-1)

....

T(2)/3 = T(1)/2 + 2c /3

Add the equations and cross equal terms:

T(N)/(N+1) = T(1)/2 +2c S (1/j), j = 3 to N+1

T(N) = (N+1)(1/2 + 2c S(1/j))

The sum S (1/j), j =3 to N-1, is about LogN

Thus **T(N) = O(NlogN)**

9. What is the primary criterion of performing binary search technique on a list of data?

Answer:

A binary search algorithm is one method of efficiently processing a sorted list to determine rows that match a given value of the sorted criteria. It does so by "cutting" the set of data in half (thus the term binary) repeatedly, with each iteration comparing the supplied value with the value where the cut was made. If the supplied value is greater than the value at the cut, the lower half of the data set is ignored, thus eliminating the need to compare those values. The reverse happens when the skipped to value is less than

the supplied search criteria. This comparison repeats until there are no more values to compare.

The binary search algorithm was able to eliminate the need to do a comparison on each of the records, and in doing so reduced the overall computational complexity of our request for the database server. Using the smaller set of sorted weight data, we are able to avoid needing to load all the record data into memory in order to compare the product weights to our search criteria.

10. What is Load Factor? Why do we need hashing? How does a hash table allow O(1) searching? Why is prime number chosen for computing a hash function?

Answer:

1st Part:

Loads factor refers to the ratio of the number of records to the number of addresses within a data structure.

2nd Part:

Hashing is used to inbox and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.

3rd Part:

A hash table is an array containing all of the keys to search on. The position of each key in the array is determined by the hash function, while can be any function which always maps the same input to the same output. So, we shall assume the hash function as 0(1).

4th Part:

The prime numbers are used to minimize collisions when the data exhibits some particular pattern. The reason prime numbers are used to neutralize the effect of patterns in the keys in the distribution of collisions of a hash function.

Long Answer Type Questions

1. a) Explain with an example the Merge sort algorithm.
b) Write an algorithm for Merge sort.

OR,

Show the operation of merge sort with an example.
c) Compare the best case time complexity of selection sort and insertion sort.

Answer:

a) Merge-sort is based on the divide-and-conquer paradigm. The Merge-sort algorithm can be described in general terms as consisting of the following three steps:

1. Divide Step

If given array A has zero or one element, return S; it is already sorted. Otherwise, divide A into two arrays, A_1 and A_2, each containing about half of the elements of A.

2. Recursion Step

Recursively sort array A_1 and A_2.

3. Conquer Step

Combine the elements back in A by merging the sorted arrays A_1 and A_2 into a sorted sequence.

b)

```
/*Recursive Merge Sort algorithm*/
mergesort(int a[], int low, int high)
{
    int mid;
    if(low<high)
    {
        mid=(low+high)/2;
        mergesort(a,low,mid);
        mergesort(a,mid+1,high);
        merge(a,low,high,mid);
    }
    return(0);
}
/* Merge function*/
merge(int a[], int low, int high, int mid)
{
    int i, j, k, c[50];  /* assume size of array is 50*/
    i=low;
    j=mid+1;
    k=low;
    while((i<=mid)&&(j<=high))
    {
        if(a[i]<a[j])
        {
            c[k]=a[i];
            k++;
            i++;
        }
        else
        {
            c[k]=a[j];
            k++;
            j++;
        }
    }
    while(i<=mid)
        {
            c[k]=a[i];
            k++;
            i++;
        }
    while(j<=high)
        {
            c[k]=a[j];
            k++;
```

```
            j++;
        }
    for(i=low;i<k;i++)
        {
            a[i]=c[i];
        }
}
```

Let the original unsorted array is: 8 4 5 2 5
In the first step it gets split into
8 4 ||| 5 2 5 (||| denotes the divider)
Further splitting and merging is shown in the next steps:
8 ||| **4** ||| 5 2 5 (split)
8 ||| 4 ||| 5 2 5 (can't split, merges a single-element array and returns)
8 ||| **4** ||| 5 2 5 (can't split, merges a single-element array and returns)
4 8 ||| 5 2 5 (merges the two segments into the array)
4 8 ||| **5** ||| **2 5** (split)
4 8 ||| **5** ||| 2 5 (can't split, merges a single-element array and returns)
4 8 ||| 5 ||| **2** ||| **5** (split)
4 8 ||| 5 ||| **2** ||| 5 (can't split, merges a single-element array and returns)
4 8 ||| 5 ||| 2 ||| **5** (can't split, merges a single-element array and returns)
4 8 ||| 5 ||| **2 5** (merges the two segments into the array)
4 8 ||| **2 5 5** (merges the two segments into the array)
2 4 5 5 8 (merges the two segments into the array)

c) Selection sort Performance: Best Case is $O(n^2)$ because even if the list is sorted, the same number of selections must still be performed.
Insertion Sort Performance: The *best-case* time complexity is when the array is already sorted, and is O(n).

2. a) Explain with a suitable example, the principle of operation of Quick sort.

Answer:
In Quick-Sort we divide the array into two halves. We select a pivot element (normally the middle element of the array) & perform a sorting in such a manner that all the elements to the left of the pivot element is lesser than it & all the elements to it's right is greater than the pivot element. Thus we get two sub arrays. Then we recursively call the quick sort function on these two sub arrays to perform the necessary sorting.
Let us consider the following unsorted array:
a [ ] = 45 26 77 14 **68** 61 97 39 99 90

Step 1:
We choose two indices as left = 0 and right = 9. We find the pivot element by the formula a[left + right] / 2 = a [0 + 9] / 2 = a [4]. So the **pivot element is a [4] = 68**. We also start with a [left] = a [0] = 45 and a [right] = a [9] = 90

Step 2:
We compare all the elements to the left of the pivot element. We increase the value of left by 1 each time, when an element is less than 68. We stop when there is no such element. We record the latest value of left. In our example the left value is 2 i.e., **left=2.**
Step 3:
We compare all the elements to the right of the pivot element. We decrease the value of right by 1 each time when an element is greater than 68. We stop when there is no such element. We record the latest value of right. In our example the right value is 7 i.e., **right=7.**
Step 4:
Since left < = right we swap between a [left] and a [right]. That is between 77 and 39.
The array now looks like
45 26 39 14 **68** 61 97 77 99 90.
Step 5:
We further increment the value of left and decrement the value of right by 1 respectively i.e., **left = 2 + 1 = 3 and right = 7 - 1 = 6.**
We repeat the above steps until left < = right.
We will see that at one stage that the left and right indices will cross each other and we will find that our array has been subdivided. That all the elements lying to the left of the pivot element is less than it and all to it's right are greater. We then make recursive calls of quick sort on this two sub arrays.

b) Find the complexity of Quick sort algorithm.
Answer:
Refer to Question No. 8 of Short Answer Type Questions.

3. a) Why is hashing referred as a heuristic search method?
Answer:
A **heuristic algorithm**, is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios, in the fashion of a general heuristic, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution, or to use reasonable resources. Heuristics are typically used when there is no known method to find an optimal solution, under the given constraints (of time, space *etc.*) or at all. It is a technique whereby or items are placed into a structure based on a key to-address transformation.
We use Hashing for
1) Performing optimal searches & retrieval of data at a constant time i.e. O(1)
2) It increases speed, betters ease of transfer, improves retrieval, optimizes searching of data and reduces overhead.

That's why hashing are sometimes referred as heuristic search method.

b) What is the primary advantage of hashing over deterministic search algorithms?

Step 2:
We compare all the elements to the left of the pivot element. We increase the value of left by 1 each time, when an element is less than 68. We stop when there is no such element. We record the latest value of left. In our example the left value is 2 i.e. **left=2.**

Step 3:
We compare all the elements to the right of the pivot element. We decrease the value of right by 1 each time when an element is greater than 68. We stop when there is no such element. We record the latest value of right. In our example the right value is 7 i.e. **right=7.**

Step 4:
Since left < = right we swap between a [left] and a [right]. That is between 77 and 39
The array now looks like

45 26 39 14 **68** 61 97 77 99 90

Step 5:
We further increment the value of left and decrement the value of right by 1 respectively i.e., **left = 2 + 1 = 3 and right = 7 - 1 = 6.**
We repeat the above steps until left < = right.
We will see that at one stage that the left and right indices will cross each other and we will find that our array has been subdivided. That all the elements lying to the left of the pivot element is less than it and all to it's right are greater. We then make recursive calls of quick sort on this two sub arrays.

b) Find the complexity of Quick sort algorithm.

Answer:

Refer to Question No. 8 of Short Answer Type Questions.

3. a) Why is hashing referred as a heuristic search method?

Answer:

A **heuristic algorithm**, is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios, in the fashion of a general heuristic, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution, or to use reasonable resources. Heuristics are typically used when there is no known method to find an optimal solution, under the given constraints (of time, space *etc.*) or at all. It is a technique whereby or items are placed into a structure based on a key to-address transformation.

We use Hashing for

1) Performing optimal searches & retrieval of data at a constant time i.e. O(1)
2) It increases speed, betters ease of transfer, improves retrieval, optimizes searching of data and reduces overhead.

That's why hashing are sometimes referred as heuristic search method.

b) What is the primary advantage of hashing over deterministic search algorithms?

```
        j++;
    }
for(i=low;i<=k;i++)
    {
        a[i]=c[i];
    }
}
```

Let the original unsorted array is 8 4 5 2 5

In the first step it gets split into

8 4 ||| 5 2 5 (||| denotes the divider)

Further splitting and merging is shown in the next steps:

8 ||| **4** ||| 5 2 5 (split)

8 ||| 4 ||| 5 2 5 (can't split, merges a single-element array and returns)

8 ||| **4** ||| 5 2 5 (can't split, merges a single-element array and returns)

4 8 ||| 5 2 5 (merges the two segments into the array)

4 8 ||| **5** ||| **2 5** (split)

4 8 ||| **5** ||| 2 5 (can't split, merges a single-element array and returns)

4 8 ||| 5 ||| **2** ||| **5** (split)

4 8 ||| 5 ||| 2 ||| 5 (can't split, merges a single-element array and returns)

4 8 ||| 5 ||| 2 ||| 5 (can't split, merges a single-element array and returns)

4 8 ||| 5 ||| **2 5** (merges the two segments into the array)

4 8 ||| **2 5 5** (merges the two segments into the array)

2 4 5 5 8 (merges the two segments into the array)

c) Selection sort Performance: Best Case is $O(n^2)$ because even if the list is sorted, the same number of selections must still be performed.

Insertion Sort Performance: The *best-case* time complexity is when the array is already sorted, and is O(n).

2. a) Explain with a suitable example, the principle of operation of Quick sort.

Answer:

In Quick-Sort we divide the array into two halves. We select a pivot element (normally the middle element of the array) & perform a sorting in such a manner that all the elements to the left of the pivot element is lesser than it & all the elements to it's right is greater than the pivot element. Thus we get two sub arrays. Then we recursively call the quick sort fupction on these two sub arrays to perform the necessary sorting.

Let us consider the following unsorted array:

a [ ] = 45 26 77 14 **68** 61 97 39 99 90

Step 1:

We choose two indices as left = 0 and right = 9. We find the pivot element by the formula a[left + right] / 2 = a [0 + 9] / 2 = a [4]. So the **pivot element is a [4] = 68**. We also start with a [left] = a [0] = 45 and a [right] = a [9] = 90

Step 2:

We compare all the elements to the left of the pivot element. We increase the value of left by 1 each time, when an element is less than 68. We stop when there is no such element. We record the latest value of left. In our example the left value = 2 i.e., **left=2**.

Step 3:

We compare all the elements to the right of the pivot element. We decrease the value of right by 1 each time when an element is greater than 68. We stop when there is no such element. We record the latest value of right. In our example the right value = 7 i.e. **right=7**.

Step 4:

Since left < = right we swap between a [left] and a [right]. That is between 77 and 39.

The array now looks like

45 26 39 14 **68** 61 97 77 99 90

Step 5:

We further increment the value of left and decrement the value of right by 1 respectively i.e., **left = 2 + 1 = 3 and right = 7 - 1 = 6.**

We repeat the above steps until left < = right.

We will see that at one stage that the left and right indices will cross each other and we will find that our array has been subdivided. That all the elements lying to the left of the pivot element is less than it and all to it's right are greater. We then make recursive calls of quick sort on this two sub arrays.

b) Find the complexity of Quick sort algorithm.

Answer:

Refer to Question No. 8 of Short Answer Type Questions.

3. a) Why is hashing referred as a heuristic search method?

Answer:

A **heuristic algorithm**, is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios, in the fashion of a general heuristic, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution, or to use reasonable resources. Heuristics are typically used when there is no known method to find an optimal solution, under the given constraints (of time, space *etc*) or at all. It is a technique whereby or items are placed into a structure based on a key to-address transformation.

We use Hashing for

1) Performing optimal searches & retrieval of data at a constant time i.e. O(1)
2) It increases speed, betters ease of transfer, improves retrieval, optimizes searching of data and reduces overhead.

That's why hashing are sometimes referred as heuristic search method.

b) What is the primary advantage of hashing over deterministic search algorithms?

number of slots in hash table **HT** will be denoted by the variable M with slots numbered from 0 to M - 1.

The goal for a hashing system is to arrange things such that, for any key value K and some hash function **h**, $i = h(K)$ is a slot in the table such that $0 <= i < M$, and we have the key of the record stored at **HT**[i] equal to K.

Hash tables are commonly used to implement many types of in-memory tables. They are used to implement associative arrays (arrays whose indices are arbitrary strings or other complicated objects), especially in interpreted programming languages like AWK, Perl, and PHP. Hash tables can be used to implement caches, auxiliary data tables that are used to speed up the access to data that is primarily stored in slower media. In this application, hash collisions can be handled by discarding one of the two colliding entries—usually erasing the old item that is currently stored in the table and overwriting it with the new item, so every item in the table has a unique hash value.

A **collision** between two keys K & K' occurs when both have to be stored in the table & both hash to the same address in the table.

Linear probing is a used for resolving hash collisions of values of hash functions by sequentially searching the hash table for a free location. This is accomplished using two values - one as a starting value and one as an interval between successive values in modular arithmetic. The second value, which is the same for all keys and known as the *stepsize*, is repeatedly added to the starting value until a free space is found, or the entire table is traversed.

The function for the rehashing is the following:

rehash(key)=(n+1) % k;

For example, we have a hash table that could accommodate 9 information, and the data to be stored were integers. To input 27, we use hash(key)= 27 % 9=0. Therefore, 27 is stored at 0. If another input 18 occurs, and we know that 18 % 9= 0, then a collision would occur. In this event, the need to rehash is needed. Using linear probing, we have the rehash(key)= (18+1) % 9= 1. Since 1 is empty, 18 can be stored in it.

5. a) What do you mean by external sorting? How does it differ from internal sorting?

Answer:

External sorting is the method when the sorting takes place with the secondary memory. The time required for read and write operations are considered to be significant e.g. sorting with disks, sorting with tapes.

Internal sorting on the other hand is defined as a sorting mechanism where the sorting takes place within the main memory. The time required for read and write operations are considered to be insignificant e.g. Bubble Sort, Selection sort, Insertion sort etc.

Internal sorting is faster than external sorting because in external sorting we have to consider the external disk rotation speed, the latency time etc. Such operations are costly than sorting an array or a linked list or a hash table.

b) Write an algorithm for sorting a list numbers in ascending order using selection sort technique.

OR,

Write a C function for selection sort and also calculate the time complexity for selection sort.

Answer:

In selection sorting we select the first element and compare it with rest of the elements. The minimum value in each comparison is swapped in the first position of the array. During each pass elements with minimum value are placed in the first position, then second then third and so on. We continue this process until all the elements of the array are sorted.

```
//n denotes the no. of elements in the array a
void selecsort(int a[], int n) {
  int loc, min, temp, i, j;
  for (i = 0; i < n; i++)
  {
    min = a[i];
    loc = i;
    for (j = i + 1; j < n; j++)
    {
      if (a[j] < min)
      {
        min = a[j];
        loc = j;
      }
    }
    if (loc ! = i)
    {
      temp = a[i];
      a[i] = a[loc];
      a[loc] = temp;
    }
  }
  printf("\nThe sorted array is:\n");
  for (i = 0; i < n; i++)
    printf("%3d", a[i]);
}
```

Time Complexity of Selection Sort

For first pass the inner for loop iterates n - 1 times. For secondpass the inner for loop iterates n - 2 times and so on. Hence the time complexity of selection sort is

$(n-1) + (n-2) + (n-3) + \ldots\ldots\ldots\ldots\ldots + 2 + 1$

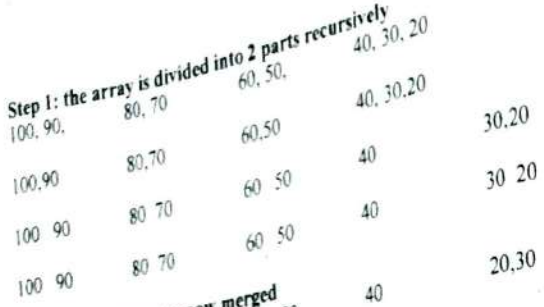
$= [(n-1) * (n-1+1)]/2$

$= [n(n-1)]/2$

$= O(n^2)$

The selection sort minimizes the number of swaps. Swapping data items is time consuming compared with comparing them. This sorting technique might be useful when the amount of data is small.

Step 1: the array is divided into 2 parts recursively



Step 2: the arrays are now merged

Complexity of MergeSort

If the running time of merge sort for a list of length n is $T(n)$, then the recurrence $T(n) = 2T(n/2) + cn$ where c is a constant. This follows from the definition of the algorithm (apply the algorithm to two lists of half the size of the original list, and add the n steps taken to merge the resulting two lists).

$$T(n) = 2.T\left(\frac{n}{2}\right) + cn = 2.\left[2.T\left(\frac{n}{4}\right) + c.\frac{n}{2}\right] + cn = 4.T\left(\frac{n}{4}\right) + 2.cn = 4.\left[2.T\left(\frac{n}{8}\right) + c.\frac{n}{4}\right] + 2.cn$$

$$= 8.T\left(\frac{n}{8}\right) + 3.cn$$

$$= \cdots$$

$$= 2^k.T\left(\frac{n}{2^k}\right) + k.cn$$

$$= 2^{\lg n}.T\left(\frac{n}{2^{\lg n}}\right) + \lg n.cn \left(2^k = n \Rightarrow k = \log_2 n\right)$$

$$= \cdots$$

$$= n.T(1) + cn.\lg(n)$$

$$= \Theta(n.\lg n)$$

8. a) Radix Sort the following list:
189, 205, 986, 421, 97, 192, 535, 839, 562, 674
Write the Radix sort algorithm.
b) Find the time complexity of Binary Search Algorithm.

Answer:

a) 1st Part:

INPUT	1st pass	2nd pass	3rd pass (sorted)
189	421	205	097
205	192	421	189
986	562	535	192
421	674	939	205
097	205	562	421
192	535	674	535
535	986	986	562
839	097	189	674
562	189	192	939
674	939	097	986

2nd Part:

Suppose we have a list of n records each with a key that's a number from 1 to k (we generalize the problem a little so k is not necessarily equal to n).

We can solve this by making an array of linked lists. We move each input record into the list in the appropriate position of the array then concatenate all the lists together in order.

```
bucket sort(L)
{
list Y[k+1]
for (i = 0; i <= k; i++) Y[i] = empty
while L nonempty
{
    let X = first record in L
    move X to Y[key(X)]
}
for (i = 0; i <= k; i++)
concatenate Y[i] onto end of L
}
```

What to do when k is large? Think about the decimal representation of a number

```
x = a + 10 b + 100 c + 1000 d + ...
```

where a,b,c etc all in range 0..9. These digits are easily small enough to do bucket sort.

```
radix sort(L):
{
bucket sort by a
bucket sort by b
bucket sort by c
...
}
```

or more simply

```
radix sort(L):
{
while (some key is nonzero)
{
    bucket sort(keys mod 10)
```

```
        keys = keys / 10
    }
}
```

In radix sorting, the last pass of bucket sorting is the one with the most effect on the overall order. So we want it to be the one using the most important digits. The previous bucket sorting passes are used only to take care of the case in which two items have the same key (mod 10) on the last pass.

b) The time complexity of Binary Search is as follows:

In each iteration, the array is split into two halves. Thereby we can say that the binary search takes the form of a binary tree. The time complexity is thus $O(\log_2 n)$ in worst case also.

9. When will interpolation search be more appropriate than binary search? How does an interpolation search work? Write an algorithm for interpolation search. Show with an appropriate example that worst case time complexity of interpolation search is $O(n)$. What is the average case time complexity of interpolation search?

Answer:

Interpolation search works better than Binary Search for a sorted and uniformly distributed array.

Interpolation search is an algorithm for searching for a given key value in an indexed array that has been ordered by the values of the key. It parallels how humans search through a telephone book for a particular name, the key value by which the book's entries are ordered. In each search step it calculates where in the remaining search space the sought item might be, based on the key values at the bounds of the search space and the value of the sought key, usually via a linear interpolation. The key value actually found at this estimated position is then compared to the key value being sought. If it is not equal, then depending on the comparison, the remaining search space is reduced to the part before or after the estimated position. This method will only work if calculations on the size of differences between key values are sensible.

Algorithm: INTERPOLATON SEARCH

```
/* Given: x, and a_1 < ... < a_n
Return: i such that x = a_i
0 if no such i exists */
 i := 1
 j := n
 LO := a_i
 HI := a_j
 if x < LO then return 0
 if x ≥ HI then i := j
        loop invariant:  x ≥ LO and x ≤ HI
 while (i<j) do
    m := floor( i + (j-i)*(x-LO) / (HI-LO) )
    MID := a_m
    if (x > MID) then
       i := m+1
       LO := MID
    else if (x < MID) then
       j := m-1
       HI := MID
    else
       return MID
 if (x ≠ a_i) then i := 0
 return i
```

On average the interpolation search makes about log(log(n)) comparisons (if the elements are uniformly distributed), where n is the number of elements to be searched. In the worst case (for instance where the numerical values of the keys increase exponentially) it can make up to O(n) comparisons (e.g., searching for 1000 in 1,2.....999,1000,10^9 will take 1000 accesses).

10. a) Define sorting.
b) What is a stable sorting? What is In-Place sorting?
c) Write the Pseudocode for Merge sort implementation. What is its time complexity?
d) If the existing array is sorted and you want to insert a new element in the list without disrupting the sortedness then which sorting technique you should use? Why?

Answer:

a) Sorting is a process by which a collection of items is placed into which is typically based on a data field called a key. Sorting refers to ordering data in an increasing or decreasing fashion according to some linear relationship among the data items.

b) *Stable sort:*

When sorting some kind of data, only part of the data is examined when determining the sort order. The data being sorted can be represented as a record or tuple of values, and the part of the data that is used for sorting is called the *key*. A sorting algorithm is stable if whenever there are two records R and S with the same key, and R appears before S in the original list, then R will always appear before S in the sorted list.

When equal elements are indistinguishable, such as with integers, or more generally, any data where the entire element is the key, stability is not an issue. Stability is also not an issue if all keys are different.

In-place sorting:

A sort algorithm in which the sorted items occupy the same storage as the original ones. These algorithms may use $O(n)$ additional memory for bookkeeping, but at most a constant number of items are kept in auxiliary memory at any time.

c) 1st Part: *Refer to Question No. 1(a) of Long Answer Type Questions.*

2nd Part:

Time complexity of merge sort:

Here at every step, the merge-sort considers only one array. In the next step, the algorithm splits the array into halves and then sorts and merges them. In the k^{th} iteration, the algorithm splits the arrays into sub-arrays, which are $2^k - 1$ in number. In worst case the number of steps required to break the array into sub-arrays of single elements is $\log n$. At each iteration the maximum number of comparisons is O(n). So, the time complexity is O(nlog$_2$n) (in best, average as well as in worst case also). A drawback of mergesort is that it needs an additional space of $\Theta(n)$ for the temporary array *b*. There are different possibilities to implement function *merge*. The most efficient of these is variant b. It requires only half as much additional space, it is faster than the other variants, and it is stable.

d) Insertion Sort as its *best-case* time complexity is when the array is already sorted which is O(n).

11. What is hashing? Describe any three methods of defining a hash function.

Answer:

1st part: *Refer to Question No. 2 of Short Answer Type Questions.*

2nd part:

A hash function maps keys to small integers (buckets). An ideal hash function maps the keys to the integers in a random-like manner, so that bucket values are evenly distributed even if there are regularities in the input data.

This process can be divided into two steps:

- Map the key to an integer.
- Map the integer to a bucket.

The following functions map a single integer key (k) to a small integer bucket value h(k). m is the size of the hash table (number of buckets).

Division method (Cormen) Choose a prime that isn't close to a power of 2. h(k) = k mod m. Works badly for many types of patterns in the input data.

Knuth Variant on Division h(k) = k(k+3) mod m. Supposedly works much better than the raw division method.

Multiplication Method (Cormen). Choose m to be a power of 2. Let A be some random-looking real number. Knuth suggests M = 0.5*(sqrt(5) - 1). Then do the following:

s = k*A
x = fractional part of s
h(k) = floor(m*x)

12. Write a C function to sort positive integers that does not compose the array elements.

Answer:

```
#include<stdio.h>
#include<stdlib.h>
/* structure for a node */
struct Node
{
    int data;
    struct Node *next;
};
/* Function to insert a node at the begining of a linked lsit */
void insertAtTheBegin(struct Node **start_ref, int data);
/* Function to bubble sort the given linked lsit */
void bubbleSort(struct Node *start);
/* Function to swap data of two nodes a and b*/
void swap(struct Node *a, struct Node *b);
/* Function to print nodes in a given linked list */
void printList(struct Node *start);

int main()
{
    int arr[] = {12, 56, 2, 11, 1, 90};
    int list_size, i;

    /* start with empty linked list */
    struct Node *start = NULL;

    /* Create linked list from the array arr[].
      Created linked list will be 1->11->2->56->12 */
    for (i = 0; i< 6; i++)
        insertAtTheBegin(&start, arr[i]);

    /* print list before sorting */
    printf("\n Linked list before sorting ");
    printList(start);

    /* sort the linked list */
    bubbleSort(start);

    /* print list after sorting */
    printf("\n Linked list after sorting ");
    printList(start);

    getchar();
    return 0;
}
/* Function to insert a node at the begining of a linked lsit */
void insertAtTheBegin(struct Node **start_ref, int data)
```

```
    {
        struct Node *ptr1 = (struct Node*)malloc(sizeof(struct Node));
        ptr1->data = data;
        ptr1->next = *start_ref;
        *start_ref = ptr1;
    }
    /* Function to print nodes in a given linked list */
    void printList(struct Node *start)
    {
        struct Node *temp = start;
        printf("\n");
        while (temp!=NULL)
        {
            printf("%d ", temp->data);
            temp = temp->next;
        }
    }
    /* Bubble sort the given linked lsit */
    void bubbleSort(struct Node *start)
    {
        int swapped, i;
        struct Node *ptr1;
        struct Node *lptr = NULL;

        /* Checking for empty list */
        if (ptr1 == NULL)
            return;

        do
        {
            swapped = 0;
            ptr1 = start;

            while (ptr1->next != lptr)
            {
                if (ptr1->data > ptr1->next->data)
                {
                    swap(ptr1, ptr1->next);
                    swapped = 1;
                }
                ptr1 = ptr1->next;
            }
            lptr = ptr1;
        }
        while (swapped);
    }
    /* function to swap data of two nodes a and b*/
    void swap(struct Node *a, struct Node *b)
    {
        int temp = a->data;
        a->data = b->data;
        b->data = temp;
    }
```


13. a) Write an algorithm for linear search.
b) Give an outline of the complexity of your algorithm.
Answer:
a)

```
Linear Search (Array A, Value x)
Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

b) Linear search executes in O(n) time where n is the number of elements in the array. Obviously, the best case of linear search is when VAL is equal to the first element of the array. In this case, only one comparison will be made.
Likewise, the worst case will happen when either VAL is not present in the array or it is equal to the last element of the array. In both the cases, n comparisons will have to be made.

14. Find the time complexity of merge sort technique using the recurrence relation assuming the size of the list $n = 2^k$.
Answer: **[WBUT 2017]**
Refer to Question No. 7 of Long Answer Type Questions.

15. Write short notes on the following:
a) Radix sort
b) Merge Sort
c) Interpolation search
d) Heap
Answer:
a) Radix sort:
Radix sorting is a technique for ordering a list of positive integer values. The values are successively ordered on digit positions, from right to left. This is accomplished by copying the values into "buckets," where the index for the bucket is given by the position of the digit being sorted. Once all digit positions have been examined, the list must be sorted.
The following table shows the sequences of values found in each bucket during the four steps involved in sorting the list 624 852 426 987 269 146 415 301 730 78 593. During pass 1 the ones place digits are ordered. During pass 2 the tens place digits are ordered,

retaining the relative positions of values set by the earlier pass. On pass 3 the hundreds place digits are ordered, again retaining the previous relative ordering. After three passes the result is an ordered list.

bucket	pass 1	pass 2	pass 3
0	73[0]	3[0]1	[0]78
1	30[1]	4[1]5	[1]46
2	85[2]	6[2]4, 4[2]6	[2]69
3	59[3]	7[3]0	[3]01
4	62[4]	1[4]6	[4]15, [4]26
5	41[5]	8[5]2	[5]93
6	42[6], 14[6]	2[6]9	[6]24
7	98[7]	[7]8	[7]30
8	7[8]	9[8]7	[8]52
9	26[9]	5[9]3	[9]87

The C-program of the radix-sort is as shown below:

```
#include <stdio.h>
#include <conio.h>
#define N 11
#define SHOWPASS

void radixsort(int[], int);
void print(int[], int);

int main(void)
{
  int unsortedArr[N] ={624, 852, 426, 987, 269, 146, 415, 301,
  730, 78, 593};
  int i;
  printf("\nBefore Sorting:\n\n");
  for (i = 0; i < N; i++)
    printf("%5d", unsortedArr[i]);
  radixsort(unsortedArr, N);
  printf("\n\nAfter Sorting:\n");
  for (i = 0; i < N; i++)
    printf("%5d", unsortedArr[i]);
  getch();
  return 0;
}
void radixsort(int a[], int n)
{
  int i, b[N], m = 0, exp = 1;
  int count = 0;
  for (i = 0; i < n; i++)
  {
    if (a[i] > m)
      m = a[i];
  }
  while (m / exp > 0)
                    ;
```



```
  {
    count++;
    int bucket[10] ={0};
    for (i = 0; i < n; i++)
      bucket[a[i] / exp % 10]++;
    for (i = 1; i < 10; i++)
      bucket[i] += bucket[i - 1];
    for (i = n - 1; i >= 0; i--)
      b[--bucket[a[i] / exp % 10]] = a[i];
    for (i = 0; i < n; i++)
      a[i] = b[i];
    exp *= 10;

    #ifdef SHOWPASS
      printf("\nPASS %d : ", count);
      print(a, n);
    #endif
  }
}
void print(int a[], int n)
{
  int i;
  printf("\n");
  for (i = 0; i < n; i++)
    printf("%5d", a[i]);
}

/* Output of the above program

Before Sorting:
 624 852 426 987 269 146 415 301 730 78 593
PASS 1 :
 730 301 852 593 624 415 426 146 987 78 269
PASS 2 :
 301 415 624 426 730 146 852 269 78 987 593
PASS 3 :
 78 146 269 301 415 426 593 624 730 852 987

After Sorting:
 78 146 269 301 415 426 593 624 730 852 987

*/
```

Time complexity of radix sort:

The time complexity of the algorithm is as follows: Suppose that the n input numbers have maximum k digits. Then the Counting Sort procedure is called a total of k times. Counting Sort is a linear, or O(n) algorithm. So the entire Radix Sort procedure takes O(n) time.

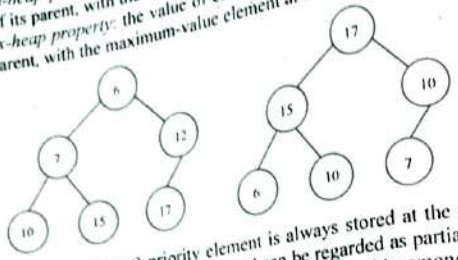
b) Merge Sort: *Refer to Question No. 1 of Long Answer Type Questions.*

c) Interpolation search: *Refer to Question No. 9 of Long Answer Type Questions.*

d) Heap Sort:

A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:

- the *min-heap property*: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
- the *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.



In a heap the highest (or lowest) priority element is always stored at the root, hence the name "heap". A heap is not a sorted structure and can be regarded as partially ordered. As it can be seen from the picture, there is no particular relationship among nodes on any given level, even among the siblings.

Since a heap is a complete binary tree, it has a smallest possible height - a heap with N nodes always has O(log N) height.

A heap is useful data structure when we need to remove the object with the highest (or lowest) priority. A common use of a heap is to implement a priority queue.

7 MISCELLANEOUS

Short Answer Type Questions

1. a) Describe a string reversal algorithm.
b) What is difference between Union & Structure?
Answer:
a) The C code is given below:

```
void reverse(char *str) {
    char * end = str;
    char tmp;

    if (str) {
        while (*end) {
            ++end;
        }
        --end;
        while (str < end) {
            tmp = *str;
            *str++ = *end;
            *end-- = tmp;
        }
    }
}
```

In the above code, in the innermost while loop in each iteration, the chaacters pointed to by `str` and `end` get swapped. `str` gets incremented to point to the next character, and `end` is decremented to point to the previous one.

b) *The differences between structure and union:*

1. Union allocates the memory equal to the maximum memory requird by the member of the union but structure allocates the memory equal to the total memcy required by the members.
2. In union, one block is used by all the member of the union but i case of structure, each member has their own memory space.

While structure enables us treat a number of different variables sored at different in memory, a union enables us to treat the same space in memory as anumber of different variables. That is a Union offers a way for a section of memory to btreated as a variable of one type on one occasion and as a different variable of a diffcent type on another occasion. There is frequent requirement while interacting with hardvare to access a byte or group of bytes simultaneously and sometimes each byte individully. Usually union is the answer.

Long Answer Type Questions

1. a) Write an algorithm to solve the Tower of Hanoi problem. Also calculate the time complexity of your algorithm.

OR,

Write the Pseudocode or C code to implement Tower of Hanoi problem. Also find the complexity of your procedure.

Answer:

```
Tower of Hanoi(n, source, auxiliary, destination)
  {
      If n=1 move disk from source to destination; (base case)
      Else,
          {
                Tower of Hanoi(top n-1, from, to, using);
                 Move the nth disk from 'from' to 'to';
                Tower of Hanoi(n-1, using, from, to);

          }
}
```

First recursive call moves n-1 disks from 'from' to 'using' using 'to'. So after that call n-1 disks are in 'using ' peg in order of size and the 'from' peg contains the nth disk i.e., the largest one. So, now move that disk from 'from' peg to 'to' peg. Then again by the 2nd recursive call move n-1 disk from 'using' peg to 'to' peg using 'from' peg. So, after all these, 'to' peg contains all disks in order of size.

Let the time required for n disks is $T(n)$.

There are 2 recursive call for n-1 disks and one constant time operation to move a disk from 'from' peg to 'to' peg . Let it be k_1

Therefore,

$T(n) = 2\,T(n-1) + k_1$

$T(0) = k_2$, a constant.

$T(1) = 2\,k_2 + k_1$

$T(2) = 4\,k_2 + 2k + k_1$

$T(2) = 8\,k_2 + 4k + 2k_1 + k_1$

Coefficient of $k_1 = 2^n$

Coefficient of $k_2 = 1^n - 1$

Time complexity is $O(2^n)$ or $O(a^n)$ where a is a constant greater than 1.

So it has exponential time complexity.

b) Write the recusive function for the Tower of Hanoi problem. Also draw the recursion tree for any set of initial values.

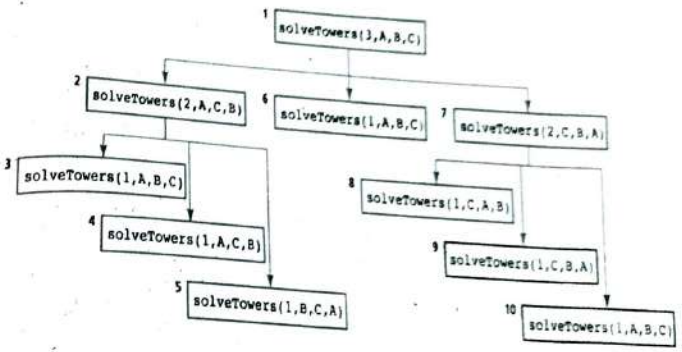
OR,

Write a recursive agorithm to solve tower of Hanoi problem.

Answer:

```
/* c function */
void solveTowers(int count, char source,
                 char destination, char spare) {
  if (count == 1) {
printf("Move top disk from pole" + source +
                 "to pole" + destination);
  }
  else {
    solveTowers(count-1, source, spare, destination); // X
    solveTowers(1, source, destination, spare);       // Y
    solveTowers(count-1, spare, destination, source); // Z
  }  // end if
}  // end solveTowers
```

Recursive tree with n =3.



2. Write short notes on the following:

a) Index sequential file ordering

b) Tail recursion

Answer:

a) Building Indexed-Sequential Files:

The additional entries required on the File specification for an output Indexed-Sequential file (as compared to an ordinary sequential output file) are:

Length of Key Field (columns 29-30)

Record Address Type (column 31) should contain K to specify that records are accessed with record keys

Type of File Organization (column 32)-should contain I to specify Indexed-Sequential organization

The records written to the Indexed-Sequential file being built must be written in ascending key sequence. If an attempt is made to write a record with a key equal to or

less than the key of the record previously written to the file, H0 will be set on and the program will terminate abnormally.

Processing Indexed-Sequential Files Sequentially

There are no additional entries required beyond those listed above (for building Indexed-Sequential files) in order to process an Indexed-Sequential file in its entirety. A sequentially processed file may be used as input (I in column 15 on the File specification) or update (U in column 15). If processed as an update file, records from the file may be updated at either detail or total time. The record available for updating is the record read on the previous input cycle.

Processing Indexed-Sequential Files by Chaining

From one to nine Indexed-Sequential files may be processed by the use of record key fields specified as chaining fields. A data field in an input record is designated as a chaining field by placing the chaining field identifier (C1, C2, C3, ... C9 in columns 61-62 of the Input specification). Each chaining field must also be included on an Extension specification which functions to link the chaining file to the chained file. During the input cycle, the record key(s) contained in one or more chaining fields are used to retrieve the corresponding record(s) from the Indexed-Sequential file specified on the Extension specification. Chaining is the only circumstance in which more than one record identifying indicator may be on during a single cycle.

b) Tail recursion:

A function call is said to be tail recursion if there is nothing to do after the function returns except return its value. Since the current recursive instance is done executing at that point, saving its stalk frame is a waste. Specifically, creating a new stack frame on top of the current, finished, frame is a waste. A compiler is said to implement Tail recursion if it recognizes this case and replaces the caller in place with the callee, so the instead of nesting the stack deeper, the current stack frame is reused. This is equivalent in effect to a "GOTO", and lets a programmer write recursive definitions without worrying about space inefficiency during execution. Tail recursion is as efficient as iteration.

QUESTION 2014

GROUP – A

(Multiple Choice Type Questions)

1. Answer any ten questions.

i) The number of swapping needed to sort numbers 8, 22, 7, 9, 31, 19, 5, 13 in ascending order using bubble sort is

a) 11 b) 12 c) 13 ✓d) 14

ii) Binary search uses

a) divide and reduce strategy ✓b) divide and conquer strategy
c) heuristic search d) both (a) and (b)

iii) The following sequence of operations is performed on a stack: push(1), push(2), pop, push(1), push(2), pop, pop, pop, push(2), pop. The sequence of popped out values are

✓a) 2, 2, 1, 1, 2 b) 2, 2, 1, 2, 2 c) 2, 1, 2, 2, 1 d) 2, 1, 2, 2, 2

iv) The postfix expression for * + a b – c d is

✓a) ab + cd – * b) ab cd + – * c) ab + cd *– d) ab + – cd *

v) Adjacency matrix for a digraph is –

a) unit matrix ✓b) symmetric matrix
c) asymmetric matrix d) none of these

vi) Which of the following is a hash function?

✓a) Quadratic probing b) chaining
c) open addressing d) folding

vii) Linked list is not suitable data structure for which one of the following problems?

a) insertion sort b) radix sort
✓c) binary search d) polynomial addition

viii) Number of all possible binary trees with 4 nodes is –

a) 13 b) 12 ✓c) 14 d) 15

ix) If the inorder and preorder traversal of a binary tree are D, B, F, E, G, H, A, C and A, B, D, E, F, G, H, C respectively then, the postorder traversal of that tree is –

a) D, F, G, A, B, C, H, E b) F, H, D, G, E, B, C, A
c) C, G, H, F, E, D, B, A ✓d) D, F, H, G, E, B, C, A

x) The heap (represented by an array) constructed from the list of numbers 30, 10, 80, 60, 15, 55, 17 is –

a) 60, 80, 55, 30, 10, 17, 15 ✓b) 80, 55, 60, 15, 10, 30, 17
c) 80, 60, 30, 17, 55, 15, 10 d) none of these

xi) In array representation of Binary tree, if the index number of a child node is 6 then the index number of it's parent node is

a) 2 b) 3 c) 4 ✓d) 5

xii) BFS constructs

✓a) a minimal cost spanning tree of a graph b) a depth first spanning tree of a graph
c) a breadth first spanning tree of a graph d) none of these

GROUP – B
(Short Answer Type Questions)

2. Differentiate between Linear and Non Linear data structures? Give two examples of each.
See Topic: INTRODUCTION, Short Answer Type Question No. 10.

3. Write an algorithm to find the largest and smallest element in a single linear list.
See Topic: LINKED LIST, Short Answer Type Question No. 9.

4. a) Suppose one 2-D array is initialized as int a [5][7]: Base address is 4000. Find the location of element a [2][4] in row major form and column major form.
b) Define Sparse Matrix.
See Topic: INTRODUCTION, Short Answer Type Question No. 12.

5. a) Prove that the maximum no. of nodes in a binary tree of depth k is $2^k - 1$.
b) What are the characteristics of algorithm?
a) See Topic: TREES, Short Answer Type Question No. 12.
b) See Topic: INTRODUCTION, Short Answer Type Question No. 11.

6. Draw a minimum heap tree from the list below:
12, 11, 7, 3, 10, –5, 0, 9, 2
Now do the heap sort operation over the heap tree.
See Topic: SORTING & HASHING, Short Answer Type Question No. 6.

GROUP – C
(Long Answer Type Questions)

7. a) Represent the given polynomial using a link list.

$$3x^4 + x^2 - 5x + 2$$

b) Write the pseudo code / C code for adding two polynomials (already given by user, no need to take input). Also comment on the complexity of your algorithm.
c) Write the Pseudocode or C code to implement Tower of Hanoi problem. Also find the complexity of your procedure.
a) & b) See Topic: LINKED LIST, Long Answer Type Question No. 3(a) & (b).
c) See Topic: MISCELLANEOUS, Long Answer Type Question No. 1(a).

8. a) Insert the following number into a binary search tree in the order that they are given and draw the resulting tree.
87; 36; 22; 15; 56; 85; 48; 91; 72; 6
Delete 48 and draw the resulting tree. Delete 15 and draw the resulting tree.
b) Write an algorithm to insert an element into binary search tree.
a) See Topic: TREES, Long Answer Type Question No. 10.
b) See Topic: TREES, Long Answer Type Question No. 1.

9. a) Define sorting.
b) What is a stable sorting? What is In-Place sorting?
c) Write the Pseudocode for Merge sort implementation. What is its time complexity?
d) If the existing array is sorted and you want to insert a new element in the list without disrupting the sortedness then which sorting technique you should use? Why
e) What is Hashing?
a), b), c) & d) See Topic: SORTING & HASHING, Long Answer Type Question No. 10.
e) See Topic: SORTING & HASHING, Short Answer Type Question No. 2.

10. a) Show the stages in growth of an order -4B- Tree when the following keys are inserted in the order given:

84 82 29 97 61 10 45 28 49 70 86 68 19 55 22 11 55 77 16

b) How does an AVL tree differ from a binary search tree? Insert the following keys in the order given below to build them into an AVL tree:

8 12 9 11 7 6 66 2 1 44

Clearly mention different rotation used and balance factor of each node.
c) Write the Prim's algorithm for finding MST from a graph.
a), b) See Topic: TREES, Long Answer Type Question No. 11.
c) See Topic: GRAPHS, Short Answer Type Question No. 4.

11. Write short notes on any three of the following:
a) Radix Sort
b) Index sequential File Organization
c) DFS in graph
d) Interpolation search
e) Threaded binary tree
a) See Topic: SORTING & HASHING, Long Answer Type Question No. 15(a).
b) See Topic: MISCELLANEOUS, Long Answer Type Question No. 2(a).
c) See Topic: GRAPHS, Long Answer Type Question No. 3(d).
d) See Topic: SORTING & HASHING, Long Answer Type Question No. 15(c).
e) See Topic: TREES, Long Answer Type Question No. 16(b).

QUESTION 2015

GROUP – A
(Multiple Choice Type Questions)

1. Answer any ten questions:
i) Which of the following traversal techniques lists the nodes of a binary search tree in ascending order?

a) Post-order ✓b) In-order c) Pre-order d) None of these

ii) The number of possible distinct binary trees with 12 nodes is
a) 4082 ✓b) 4084 c) 3082 d) 3084

iii) Which of the following expressions access the (i, j)th entry of a (m × n) matrix stored in column major order?
a) n × (i − 1) + j ✓b) m × (j − 1) + i c) m × (n − j) + j d) n × (m − i) + j

iv) Stack cannot be used to
a) evaluate an arithmetic expression in postfix form
b) implement recursion
✓c) allocate resources (like CPU) by the operating system
d) convert infix expression to its equivalent postfix expression

v) The postfix equivalent of the prefix * + ab − cd is
✓a) ab + cd − * b) abcd + − * c) ab + cd* − d) ab + − cd*

vi) Merge sort uses
✓a) divide and conquer strategy b) backtracking approach
c) heuristic search d) greedy approach

vii) The following sequence of operations is performed on a stack
push(1), push(2), pop( ), push(1), push(2), pop( ), pop( ), pop( ), push(2), pop( ),
the sequence of popped out values are
a) 2, 2, 1, 2, 1 ✓b) 2, 2, 1, 1, 2 c) 2, 1, 2, 2, 2 d) 2, 1, 2, 2, 1

viii) Which of the following is not a requirement of good hashing function?
a) Avoid collision ✓b) Reduce the storage space
c) Make faster retrieval d) None of these

ix) Self-referential pointer is used in defining
a) an array ✓b) a node of linked-list
c) a queue d) all of these

x) A binary tree has n leaf nodes. The number of nodes of degree 2 in this tree is
a) $\log n$ b) $n-1$ c) n ✓d) cannot be said

GROUP – B
(Short Answer Type Questions)

2. What is the difference between array and linked-list? What is the primary criterion of performing binary search technique on a list of data?
1st Part: See Topic: LINKED LIST, Short Answer Type Question No. 12.
2nd Part: See Topic: SORTING & HASHING, Short Answer Type Question No. 9.

3. Write a recursive algorithm to solve tower of Hanoi problem.
See Topic: MISCELLANEOUS, Long Answer Type Question No. 1(b).

4. Deduce the average time complexity of Quicksort algorithm.
See Topic: SEARCHING & SORTING, Short Answer Type Question No. 8.

5. Suppose a queue is implemented by an array. Write an algorithm to insert a new element at the kth position of the array.
See Topic: STACKS & QUEUES, Short Answer Type Question No. 8.

6. Write an algorithm to delete the last node of a linked-list.
See Topic: LINKED LIST, Short Answer Type Question No. 10.

GROUP – C
(Long Answer Type Questions)

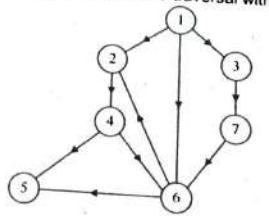
7. a) The in-order and pre-order traversal sequence of nodes in a binary tree are given below:

Post-order:	I	E	J	F	C	G	K	L	H	*D	B	A
In-order:	E	I	C	F	J	B	G	D	K	H	L	A

Construct the tree.
b) Define Hashing.
c) Describe a String reversal Algorithm.
d) Write an algorithm for inserting an element into a Binary tree with example.
a) & d) See Topic: TREES, Long Answer Type Question No. 12(a) & (b).
b) See Topic: SORTING & HASHING, Short Answer Type Question No. 2.
c) See Topic: MISCELLANEOUS, Short Answer Type Question No. 1(a).

8. a) Convert the following infix expression into equivalent postfix expression using stack.
(A + B) * C − (D − E)/(F + G)
b) What is dequeue?
c) How can a polynomial such as $6x^6 + 3x^3 - 2x + 10$ be represented by a linked list?
d) For the following expression draw the corresponding expression tree:
$a + b * c - d / e$
e) Write an algorithm to insert an element in the middle of a linked list.
a) See Topic: STACKS & QUEUES, Short Answer Type Question No. 1(a).
b) See Topic: STACKS & QUEUES, Short Answer Type Question No. 3.
c) See Topic: LINKED LIST, Short Answer Type Question No. 4.
d) See Topic: TREES, Short Answer Type Question No. 13.
e) See Topic: LINKED LIST, Short Answer Type Question No. 11.

9. a) For the following graph find the BFS and DFS traversal with proper algorithm.



b) Insert the following keys in the order given below to build them into an AVL-tree. 12, 11, 13, 10, 09, 15, 14, 18, 7, 6, 5, 4. Clearly mention different rotations used and balance factor of each node.

a) See Topic: GRAPHS, Short Answer Type Question No. 5.

b) See Topic: TREES, Short Answer Type Question No. 11.

10. a) What is hashing? Describe any three methods of defining a hash function.

b) Discuss different collision resolution techniques

a) See Topic: SORTING & HASHING, Long Answer Type Question No. 11.

b) See Topic: SORTING & HASHING, Long Answer Type Question No. 3(c).

11. Write short notes on any three of the following:

a) ADT

b) AVL Tree

c) Circular link list

d) Threaded binary trees

e) Heap

a) See Topic: INTRODUCTION, Long Answer Type Question No. 1.

b) See Topic: TREES, Long Answer Type Question No. 16(a).

c) See Topic: LINKED LIST, Long Answer Type Question No. 6(b)

d) See Topic: TREES, Long Answer Type Question No. 16(b).

e) See Topic: SORTING & HASHING, Long Answer Type Question No. 15(d).

QUESTION 2016

GROUP – A
(Multiple Choice Type Questions)

1. Answer any ten questions:

i) The postfix equivalent of the prefix* + ab – cd is

✓a) $ab+cd-*$ b) $abcd+-*$ c) $ab+cd*-$ d) $ab+-cd*$

ii) If a binary tree is threaded for inorder traversal a right NULL link of any node it is replaced by the address of its

✓a) successor b) predecessor c) root d) own

iii) Adjacency matrix of a digraph is

a) Identity matrix ✓b) Symmetric matrix c) Asymmetric matrix d) None of these

iv) Linked lists are not suitable for

a) Stack b) Dequeue c) AVL tree ✓d) Binary Search

v) The ratio of items present in a hash table of the total size is called

a) balance factor ✓b) load factor c) item factor d) weight factor

vi) Maximum possible height of an AVL tree with 7 nodes is

✓a) 3 b) 4 c) 5 d) 6

vii) The deque can be used

a) as a stack b) as a queue

✓c) both as a stack and as a queue d) none of these

viii) Inserting a node after a given node in a doubly linked list requires

a) four pointer exchanges ✓b) two pointer exchanges

c) one pointer exchange d) no pointer exchange

ix) The minimum height of a binary tree of n nodes is

a) n b) n/2 c) n/2 – 2 ✓d) $\log_2(n+1)$

x) What will be the time complexity for selections sort to sort an array of n elements?

a) $O(\log n)$ b) $O(n\log n)$ c) $O(n)$ ✓d) $O(n^2)$

GROUP – B
(Short Answer Type Questions)

2. Show that the function $f(n)$ defined by

$$f(n)=1;\ n=1$$

$$f(n)=f(n-1)+1/n,\ n>1$$

has complexity $O(\ln n)$.

See Topic: INTRODUCTION, Short Answer Type Question No. 2.

3. a) Does a B tree grow at its leave or at its root? Why?

b) In deleting a key from a B tree, when it is necessary to combine nodes?

c) For what purposes are B trees especially appropriate?

a) & b) See Topic: TREE, Short Answer Type Question No. 14.

c) See Topic: TREE, Long Answer Type Question No. 5(b).

4. The post-order and in-order traversal sequences of codes in a binary tree are given below:

Post-order: D G E B H I F C A

Pre-order: D B G E A C H F I

Construct the binary tree.

See Topic: TREES, Short Answer Type Question No. 15.

5. Construct one B-Tree of order 4 with the following data. 34, 67, 89, 90, 100, 2, 36, 76, 53, 51, 12, 10, 77, 69.

See Topic: TREES, Short Answer Type Question No. 16.

6. What is the default return type of malloc( )? Why do we need to typecast it? Write an algorithm to append a node after a specified node in single linked list.

1st Part: See Topic: INTRODUCTION, Short Answer Type Question No. 13.

2nd Part: See Topic: LINKED LIST, Short Answer Type Question No. 3(2nd Part).

GROUP – C
(Long Answer Type Questions)

7. a) Why circular queue is better than simple queue?
b) Evaluate the postfix expression using stack
3, 16, 2, +, *, 12, 6, /, –
c) Convert the infix expression into its equivalent prefix expression using stack

$$a + b * c + (d * e + f) * g.$$

a) See Topic: STACKS & QUEUES, Short Answer Type Question No. 5(b).
b) & c) See Topic: STACKS & QUEUES, Short Answer Type Question No. 8.

8. a) Write a non-recursive algorithm to traverse a binary tree in its inorder traversal.
b) Write a C function to find out the maximum and the minimum elements in a binary search tree.
c) Given the pre-order sequence and the post-order sequence, why cannot you reconstruct the tree?

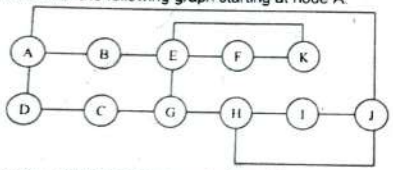
a) See Topic: TREES, Long Answer Type Question No. 2(d).
b) & c) See Topic: TREES, Long Answer Type Question No. 13.

9. a) Construct a tree from the given postfix expression $abc * + de * f + g * +$
b) Write a C function to sort positive integers that does not compose the array elements.
c) Show how linked list can be used to add the following polynomials:

$$5x^4 + 5x^3 + 10x^2 + 8x + 3$$
$$3x^3 + 2x^2 + 7x + 8.$$

a) See Topic: TREES, Short Answer Type Question No. 17.
b) See Topic: SORTING & HASHING, Long Answer Type Question No. 12.
c) See Topic: LINKED LIST, Short Answer Type Question No. 13.

10. a) Describe BFS algorithm
b) Find out the DFS traversal of the following graph starting at node A.



c) Define Prim's algorithm for minimum cost spanning tree with example.

a) See Topic: GRAPHS, Long Answer Type Question No. 1(a).
b) See Topic: GRAPHS, Short Answer Type Question No. 6.
c) See Topic: GRAPHS, Short Answer Type Question No. 4.

QUESTION 2017

GROUP – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for any ten of the following:

i) Which of the following is non-linear data structure?
a) Stacks b) List c) Strings ✓d) Trees

ii) Binary search is not possible for
a) array ✓b) linked list c) stack d) queue

iii) The prerequisite condition of Binary search is
a) unsorted array b) ascending order array
c) descending order array ✓d) sorted array

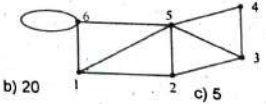
iv) Inserting an item into the stack when stack is not full is called ..... operation and deletion of item from the stack, when stack is not empty is called ..... operation.
✓a) push, pop b) pop, push c) insert, delete d) delete, insert

v) The number of edges in a full binary tree of height h is
a) $2^{h+1} - 1$ b) $2^h - 1$ ✓c) $2^{h+1} - 2$ d) $2^h - 2$

vi) Minimum number of nodes required to make a complete binary tree of height h is
a) $2^h - 1$ ✓b) 2^h c) $2^h + 1$ d) 2^{h-1}

vii) A linear link list can be traversed using
✓a) recursion b) both (a) and (c) are correct
c) stack d) both (a) and (c) are wrong

viii) What is the sum of the degrees of all the vertices in the following graph?



✓a) 19 b) 20 c) 5 d) none of these

ix) The adjacency matrix of an undirected graph is
a) Unit matrix b) Asymmetric matrix
✓c) Symmetric matrix d) none of these

x) A path is
a) a closed walk with no vertex repetition b) an open walk with no vertex repetition
✓c) an open walk with no edge repetition d) a closed walk with no edge repetition

xi) The data structure used to solve recursive problem is
a) Linked list b) Queue ✓c) Stack d) none of these

xii) Which one is required to reconstruct a binary tree?
a) Only inorder sequence b) Both preorder and postorder sequences
✓c) Both inorder and postorder sequences d) Only postorder sequence

Group – B
(Short Answer Type Questions)

2. How a polynomial such as $8x^5 + 4x^3 - 9x^2 + 2x - 17$ can be represented using a linked list? What are the advantages and disadvantages of linked list over an array?
See Topic: LINKED LIST, Short Answer Type Question No. 14.

3. If $T(n) = a_0 + a_1x + a_2x^2 + a_3x^3 + ... + a_nx^n$, then prove $T(n) = \Theta(x^n)$
See Topic: INTRODUCTION, Short Answer Type Question No. 14.

4. Why circular queue is used over simple queue? Write an algorithm to insert an element into circular queue.
1st part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 5(b).
2nd part: See Topic: STACKS & QUEUES, Long Answer Type Question No. 5(ii).

5. The inorder and preorder tree traversals are given. Draw the binary tree.
Inorder: ABCDEFGHI
Preorder: FBADCEGIH
Is it possible to build up a unique binary tree when its preorder and postorder traversals are given?
See Topic: TREES, Short Answer Type Question No. 18.

6. What is Hashing? Write two hash functions. What is collision?
See Topic: SORTING & HASHING, Short Answer Type Question No. 2.

Group – C
(Long Answer Type Questions)

7. a) Write an algorithm to evaluate a postfix expression.
b) Convert the infix expression $9 + 5 * 7 - 6 \wedge 2 + 15/3$ into its equivalent postfix expression and evaluate that postfix expression, clearly showing the state of the stack.
a) See Topic: STACKS & QUEUES, Long Answer Type Question No. 2.
b) See Topic: STACKS & QUEUES, Long Answer Type Question No. 9.

8. a) Write an algorithm for creating a linked-list with n nodes.
b) How it can be made a circular linked-list? Write a function for that purpose.
See Topic: LINKED LIST, Long Answer Type Question No. 4.

9. a) How a linked-lists can be used to implement stack?
b) Write an algorithm for linear search.
c) Give an outline of the complexity of your algorithm.

a) See Topic: LINKED LIST, Long Answer Type Question No. 5.
b) & c) See Topic: SORTING & HASHING, Long Answer Type Question No. 13.

10. a) What do you mean by external sorting? How does it differ from internal sorting?
b) Write an algorithm for sorting a list numbers in ascending order using bubble sort technique and find its time complexity.
c) Find the time complexity of merge sort technique using the recurrence relation assuming the size of the list $n = 2^k$.
a) See Topic: SORTING & HASHING, Long Answer Type Question No. 5(a).
b) See Topic: INTRODUCTION, Short Answer Type Question No. 3.
c) See Topic: SORTING & HASHING, Long Answer Type Question No. 14.

11. a) What is hashing? Describe any three methods of defining a hash function.
b) Discuss different collision resolution techniques.
a) See Topic: SORTING & HASHING, Long Answer Type Question No. 11.
b) See Topic: SORTING & HASHING, Long Answer Type Question No. 3(c).

12. a) What do you mean by a binary search tree?
b) Construct a binary search tree by inserting the list of elements one by one:
13, 10, 3, 5, 18, 15, 14
c) Write an algorithm for pre-order traversal of a tree represented by a linked-list.
d) Show that the number of vertices of odd degree in a finite graph is even.
a), b) & c) See Topic: TREES, Long Answer Type Question No. 14.
d) See Topic: GRAPHS, Long Answer Type Question No. 1(c).

13. a) What is an AVL tree?
b) Construct an AVL search tree for the data list:
AND, BEGIN, CASE, DO, END, FOR GOTO
c) For the AVL tree you have constructed delete the following keys in the order:
DO, FOR END
See Topic: TREES, Long Answer Type Question No. 15.

QUESTION 2018

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:
i) Maximum possible height of an AVL Tree with 7 node is
a) 12 b) 4 c) 5 ✓d) 3

ii) In a circularly linked list organization, insertion of a record involves the modification of
a) no pointer b) 1 pointer ✓c) 2 pointers d) 3 pointers

iii) A B-tree is
a) always balanced b) an ordered tree c) a directed tree ✓d) All of these

iv) Number of nodes in a complete binary tree of depth k is
a) 2^k b) $2k$ ✓c) $2^k - 1$ d) None of these

v) To make a queue empty, elements can be deleted till
✓a) front = rear + 1 b) front = rear – 1 c) front = rear d) None of these

vi) BFS constructs
✓a) a minimal cost spanning tree of a graph b) a depth first spanning tree of a graph
c) a breadth first spanning tree d) None of these

vii) A vertex of in-degree zero in a directed graph is called
a) Articulation point b) Sink
✓c) Isolated matrix d) Root vertex

viii) In a height balanced tree the heights of two sub-trees of every node never differ by more than
a) 2 b) 0 ✓c) 1 d) –1

ix) Inserting a new node after a specific node in a doubly linked requires
a) four pointer exchanges ✓b) two pointer exchanges
c) one pointer exchanges d) no pointer exchanges

x) A non-planar graph with minimum number of vertices has
a) 9 edges, 6 vertices b) 6 edges, 4 vertices
✓c) 10 edges, 5 vertices d) 9 edges, 5 vertices

Group – B
(Short Answer Type Questions)

2. Write an algorithm for in-order traversal of a threaded binary tree.
See Topic: TREES, Short Answer Type Question No. 6.

3. Compare and contrast linked list with static and dynamic array.
See Topic: LINKED LIST, Short Answer Type Question No. 15.

4. Write an algorithm to insert a data X immediately after a specific data item Y in a single linked list.
See Topic: LINKED LIST, Short Answer Type Question No. 3(2nd Part).

5. What is Load Factor? Why do we need hashing? How does a hash table allow O(1) searching? Why is prime number chosen for computing a hash function?
See Topic: SORTING & HASHING, Short Answer Type Question No. 10.

6. Insert the following keys into a B-Tree of given order mentioned below:
a, f, b, k, h, m, e, s, r, c. (Order 3)
a, g, f, b, k, d, h, m, j, e, s, l, r, x, c, l, n, t, u, p. (Order 5)
See Topic: TREES, Short Answer Type Question No. 19.

Group – C
(Long Answer Type Questions)

7. What are sparse matrices? How such a matrix is represented in memory? What are the types of sparse matrices?
Show that the function $f(n)$ defined by

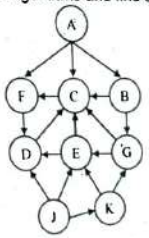
$$f(1) = 1$$

$$f(n) = f(n-1) + \frac{1}{n} \text{ for } n > 1$$

has the complexity O(log n)
Let the size of the elements stored in an 8×3 matrix be 4 bytes each. If the base address of the matrix is 3500, then find the address of A[4, 2] for both row major and column major cases.
1st, 2nd & 3rd Part: See Topic: INTRODUCTION, Short Answer Type Question No. 1(2nd Part).
4th Part: See Topic: INTRODUCTION, Short Answer Type Question No. 2(1st Part).
5th Part: See Topic: INTRODUCTION, Short Answer Type Question No. 1(1st Part).

8. a) What do you mean by external sorting? How does it differ from internal sorting?
b) Write an algorithm for sorting a list numbers in ascending order using selection sort technique.
c) Describe Kruskal's minimal spanning tree algorithm.
a) & b) See Topic: SORTING & HASHING, Long Answer Type Question No. 5(a) & (b).
c) See Topic: GRAPHS, Short Answer Type Question No. 1.

9. What is expression tree? Draw the expression tree and write the In, Pre & Post-Order traversals for the given expression tree: $E = (2x + y)(5a - b)^3$. Prove that the number of odd degree vertices in a graph is always even. Apply BFS/DFS Algorithms and find out the path of the given graph:



1st & 2nd Part: See Topic: TREES, Short Answer Type Question No. 20.
3rd Part: See Topic: GRAPHS, Long Answer Type Question No. 1(c).
4th Part: See Topic: GRAPHS, Short Answer Type Question No. 7.

10. a) Define circular queue.
b) Write an algorithm to insert an item in circular queue.
c) What is input restricted dequeue?
d) Write an algorithm to convert an infix expression to postfix using stack.
a), b) & c) See Topic: STACKS & QUEUES, Long Answer Type Question No. 3(a), (b) & (c).
d) See Topic: STACKS & QUEUES, Long Answer Type Question No. 2.

11. Write short notes on *any three* of the following

i) AVL Tree

ii) Heap Sort

iii) DFS

iv) Tail recursion

v) Binary Search Tree

i) See Topic: TREES, Long Answer Type Question No. 16(a).

ii) See Topic: SORTING & HASHING, Long Answer Type Question No. 15(d).

iii) See Topic: GRAPHS, Long Answer Type Question No. 3(d).

iv) See Topic: MISCELLANEOUS, Long Answer Type Question No. 2(b).

v) See Topic: TREES, Long Answer Type Question No. 16(d).