DATA STRUCTURE & ALGORITHM

INTRODUCTION

Multiple Choice Type Questions

1. In a max heap, both the addition and deletion operations can be performed in time [WBUT 2008, 2015]
   a) $O(\log n)$ b) $O(n \log n)$ c) $O(n)$ d) $O(n^2)$

Answer: (b)

2. Which one is the best time among the following algorithms? [WBUT 2009, 2014]
   a) O (n) b) O ($\log_2 n$) c) O (2^n) d) O ($n \log_2 n$)

Answer: (b)

3. What is the Big Oh notation of the following expression [WBUT 2010, 2011]
$F(n) = n \log n^2 + n^2 + e^{\log n}$
   a) $O(n)$ b) $O(n^2)$ c) $O(n \log n^2)$ d) $O(e^{\log n})$

Answer: (b)

4. Which of the following case does not exist in complexity theory? [WBUT 2012, 2015, 2018, 2019]
   a) best case b) worst case c) average case d) null case

Answer: (d)

5. Which of the following data structure is not linear data structure? [WBUT 2012, 2018]
   a) arrays b) linked lists c) both of these d) none of these

Answer: (c)

6. The data structure which allows deletions at both ends of the list but insertion at only end is [WBUT 2012]
   a) input-restricted deque b) output-restricted deque
   c) priority queues d) none of these

Answer: (a)

7. The elements of an array are stored successively in memory cells because [WBUT 2012]
   a) By this way computer can keep track only the address of the first element and the addresses of other elements can be calculated
   b) The architecture of computer memory does not allow arrays to store other than serially
   c) both of these
   d) none of these

Answer: (a)

8. A mathematical model with a collection of operations defined on that model is called [WBUT 2013]
   a) data structure b) abstract data type
   c) primitive data type d) algorithm

Answer: (b)

9. If $f(n) = 100n \log_2 n + 500n^4 + 0.52^n$ then $f(n)$ [WBUT 2015]
   a) $O(n^4)$ b) $O(n \log_2 n)$ c) $O(2^n)$ d) none of these

Answer: (c)

10. Two main measures for the efficiency of an algorithm are [WBUT 2017]
   a) Processor and memory b) Complexity and capacity
   c) Time and space d) Data and space

Answer: (c)

11. Structure in C is an [WBUT 2018, 2019]
   a) user defined data type b) ADT
   c) Both (a) and (b) d) None of these

Answer: (a)

12. To represent hierarchical relationship between elements, which data structure is suitable? [WBUT 2018, 2019]
   a) Deque b) Priority c) Tree d) All of these

Answer: (c)

13. Which of the followings data structure is not a linear data structure? [WBUT 2019]
   a) Arrays b) linked list c) Tree d) Stack

Answer: (c)

Short Answer Type Questions

1. Define big 'O' notation. [WBUT 2010, 2014, 2018, 2019]

OR,

Explain $f(n) = O(g(n))$. [WBUT 2013]

Answer:

Let $f(n)$ and $g(n)$ be functions, where n is a positive integer. We write $f(n) = O(g(n))$ if and only if there exists a real number c and positive integer n_0 satisfying $0 <= f(n) <= cg(n)$ for all $n >= n_0$.

2. What are the differences between linear and non-linear data structure? Give example. [WBUT 2013]

Answer:

1st Part:

Main difference between linear and nonlinear data structures lie in the way they organize data elements. In linear data structures, data elements are organized sequentially and

therefore they are easy to implement in the computer's memory. In nonlinear data structures, a data element can be attached to several other data elements to represent specific relationships that exist among them. Due to this nonlinear structure, they might be difficult to be implemented in computer's linear memory compared to implementing linear data structures. Selecting one data structure type over the other should be done carefully by considering the relationship among the data elements that needs to be stored.

2nd Part:

Linear data structure: A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. **Ex:** Arrays, Linked Lists

Non-Linear data structure: Every data item is attached to several other data items in a way that is specific for reflecting relationships. The data items are not arranged in a sequential structure. **Ex:** Trees, Graphs.

3. Write a simple code to create an array. **[WBUT 2013]**

Answer:

```
#include <stdio.h>
int main ()
{
   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;
   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ )
   {
     n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

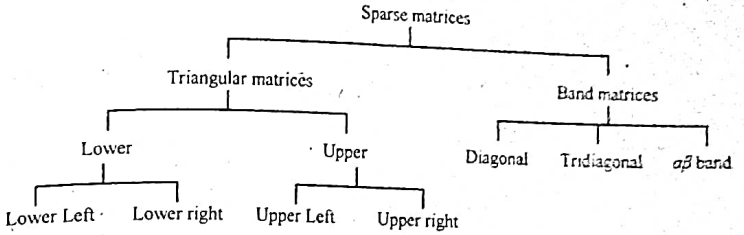
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109

Long Answer Type Questions

1. What are the types of sparse matrices? Show that the function $f(n)$ defined by: $f(1)=1$, $f(n)=f(n-1)+1/n$ 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 [5, 2] for both row major & column major cases. **[WBUT 2012]**

Answer:

The various types of Sparse Matrices are as follows:



$$f(x)=f(n-1)+\frac{1}{n}=f(n-2)+\frac{1}{n-1}+\frac{1}{n}=f(n-3)+\frac{1}{n-2}+\frac{1}{n-1}+\frac{1}{n}$$

$$=f(n-4)+\frac{1}{n-3}+\frac{1}{n-2}+\frac{1}{n-1}+\frac{1}{n}=f(2-1)+\frac{1}{2}+\frac{1}{3}+\ldots\ldots+\frac{1}{n-1}+\frac{1}{n}$$

$$=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots\ldots+\frac{1}{n}$$

These type of numbers are called Harmonic Numbers.

We can evaluate this type of series just by integrating $\frac{1}{n}$ from $\frac{1}{2}$ to $n+\frac{1}{2}$ after integrating, we get the result as $\ln\left(n+\frac{1}{2}\right)-\ln\frac{1}{2}\approx\ln n-0.7$

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

In column major order the address is

$$A[i,j]=L+\left[(j-1)\times m+(i-1)\right]\times w$$

$$\therefore A[5,2]=3500+\left[(2-1)8+(5-1)\right]\times 4=3548$$

In major order the address is

$$A[i,j]=L_0+\left[(i-1)\times n+(j-1)\right]\times w \quad \therefore A[5,2]=3500+\left[(5-1)\times 3+(2-1)\right]\times 4=3552$$

2. a) What is linear data structure? **[WBUT 2014]**

Answer:

In linear data structures, data elements are organized sequentially and therefore they are easy to implement in the computer's memory.

A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached.
Ex: Arrays, Linked Lists

b) Do you consider the following data-structures as linear? **[WBUT 2014]**
(i) Circular double linked list
(ii) Binary tree
Explain for both cases.

Answer:
i) Circular doubly linked list is a linear data structure.
In a circularly linked list, all nodes are linked in a continuous circle, without using null. For lists with a front and a back (such as a queue), one stores a reference to the last node in the list. The next node after the last node is the first node. Elements can be added to the back of the list and removed from the front in constant time.
Circularly linked lists can be either singly or doubly linked.
Both types of circularly linked lists benefit from the ability to traverse the full list beginning at any given node. This often allows us to avoid storing firstNode and lastNode, although if the list may be empty we need a special representation for the empty list, such as a lastNode variable which points to some node in the list or is null if it's empty; we use such a lastNode here. This representation significantly simplifies adding and removing nodes with a non-empty list, but empty lists are then a special case.

ii) Binary tree is a non-linear data structure.
A tree is another data structure that we can use to store information. Unlike stacks and queues, which are linear data structures, trees are hierarchical data structures.
Saying that the structure of a tree is hierarchical means that things are ordered above or below other things. For example, the army is hierarchical, with generals above colonels, and colonels above lieutenants, etc.
Here is an example of a tree holding letters:

```
   tree
   ----
     j   <-- root
    / \
   f   k
  / \   \
 a   h   z   <-- leaves
```

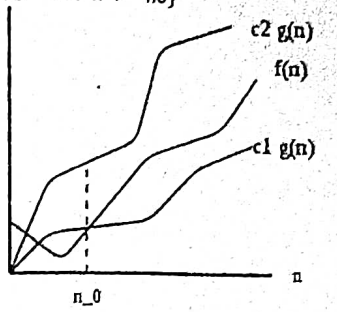
3. What are asymptotic notations? Explain each notation with example and diagram. **[WBUT 2017]**

OR,

Explain different Asymptotic Notations with diagrams. **[WBUT 2018]**

Answer:
Θ Notation: The theta notation bounds a functions from above and below, so it defines exact asymptotic behavior.

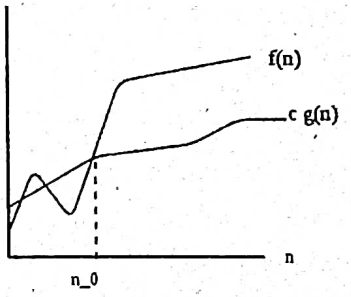

```
θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that 0 <=
c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}
```



f(n) = theta(g(n))

Big O Notation: The Big O notation defines an upper bound of an algorithm, it bounds a function only from above.

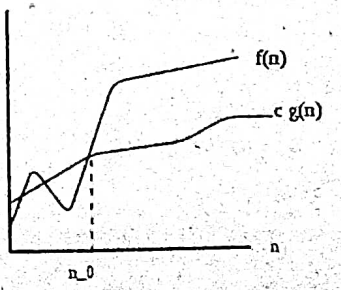
```
O(g(n)) = { f(n): there exist positive constants c and n0 such that 0 <= f(n) <=
cg(n) for all n >= n0}
```



f(n) = Omega(g(n))

Ω Notation: Just as Big O notation provides an asymptotic upper bound on a function, Ω notation provides an asymptotic lower bound.

```
Ω (g(n)) = {f(n): there exist positive constants c and n0 such that 0 <= cg(n)
<= f(n) for all n >= n0}.
```



f(n) = Omega(g(n))

STACKS & QUEUES

Multiple Choice Type Questions

1. A linear list that allows elements to be added or removed at either end but not in the middle is called [WBUT 2007, 2009, 2011]
a) Stack b) Queue c) Dequeue d) Priority queue
Answer: (c)

2. Recursion may be implemented by [WBUT 2008, 2015]
a) linked-list b) stack c) queue d) dequeue
Answer: (b)

3. Reverse polish notation is often called [WBUT 2008, 2011, 2015]
a) post fix b) prefix c) infix d) undefined
Answer: (a)

4. The evaluation of the postfix expression 23 5 7 * – 12 + is [WBUT 2008, 2011]
a) 12 b) 0 c) – 12 d) 35
Answer: (b)

5. The integer pushed 1, 2, 3, 4 are pushed into the stack I that order. They may be popped out of the stack in any valid order. The integers which are popped out produce a permutation of numbers 1, 2, 3, 4. Which of the following permutations can never be produced in such a way? [WBUT 2009, 2010]
a) 1, 2, 3, 4 b) 4, 2, 3, 1 c) 4, 3, 2, 1 d) 3, 2, 4, 1
Answer: (c)

6. If we evaluate the following postfix expression [WBUT 2009, 2015]
23 5 7 * - 12 +
the result will be
a) 12 b) 0 c) –12 d) 35
Answer: (b)

7. Priority queue can be implemented using [WBUT 2013]
a) array b) linked list c) both of these d) none of these
Answer: (c)

8. The prefix notation is also known as [WBUT 2013, 2018, 2019]
a) polish notation b) reverse polish notation
c) symbolic notation d) none of these
Answer: (a)

9. A linear list in which elements can be added or removed at either end but not in the middle, is known as [WBUT 2014]
a) queue b) dequeue c) stack d) tree
Answer: (b)

10. The prefix expression for the infix expression $a*(b+c)/e-f$ is [WBUT 2014]
a) $/*a+bc-ef$ b) $-/*+abcef$ c) $-/*a+bcef$ d) none of these
Answer: (c)

11. The evaluation of the postfix expression 3, 5, 7, *, +, 12, % is [WBUT 2016]
a) 2 b) 3 c) 0 d) 3.17
Answer: (a)

12. In a circular queue, which of the following statements indicates that the queue contains a single element? [WBUT 2016]
a) front = rear + 1 b) front = 0, rear = $n-1$
c) front = rear # –1 d) both (a) and (b)
Answer: front = rear not equals to NULL

13. Complexity is expressed in O notation in [WBUT 2016]
a) lower bound b) upper bound
c) between (a) & (b) d) none of these
Answer: (b)

14. The Postfix expression for the infix expression $a*(b+c)/e-f$ is [WBUT 2017]
a) $abc+*e/f-$ b) $abc+e/f-*$ c) $abc+e/*f-$ d) none of these
Answer: (a)

15. 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 [WBUT 2017]
a) 22112 b) 22122 c) 21221 d) 1222
Answer: (a)

Short Answer Type Questions

1. What is sparse matrix? [WBUT 2008]
Answer:
Refer to Question No. 2 of Long Answer Type Questions.

2. What is stack? [WBUT 2009, 2010, 2011, 2013]
OR,
What is stack & why this is called LIFO? [WBUT 2010, 2018, 2019]
Answer:
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).

3. Explain the operations and applications of the stack. **[WBUT 2009, 2011]**

Answer:

Stack operations: Initially all the elements of the stack are initialized to –1, which indicates an empty stack. The top variable is also initialized to –1. At each insertion, the top is incremented by 1 to point to the latest stack top. At each deletion the top is decremented by 1 and the position from where the item is removed is assigned to –1 to indicate an empty cell.

Stack applications: Conversion of Infix to Postfix Expression algorithm and evaluation of postfix expression.

4. Consider the following queue of characters, where queue Q is a circular array which is allocated 5 memory cells:
Front = 2, Rear = 3, Q : _, P, Q, _, _. Describe the following operations on queue:
(a) R is added to the queue,
(b) Two letters are deleted from the queue,
(c) S, T, U are added to queue. **[WBUT 2012]**

Answer:

FRONT = 2, REAR = 3 & QUEUE: _, **P, Q,** _, _.
a) R is added to the queue:
FRONT = 2, REAR = 4 & QUEUE: _, P, Q, R, _
b) Two letters are deleted from the queue.
FRONT = 4, REAR = 4 & QUEUE: _, _, _, R, _
c) S, T, U are added into the queue.
FRONT = 4, REAR = 3 & QUEUE: T, U, _, F, S

5. Define the following with example:
a) Abstract Data type **[WBUT 2013]**
b) Priority Queue **[WBUT 2013, 2017]**

Answer:

a) Abstract data types or ADTs are a mathematical specification of a set of data and the set of operations that can be performed on the data. They are abstract in the sense that the focus is on the definitions of the constructor that returns an abstract handle that represents the data, and the various operations with their arguments. The actual implementation is not defined, and does not affect the use of the ADT.
For example, rational numbers (numbers that can be written in the form a/b where *a* and *b* are integers) cannot be represented natively in a computer.

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

6. a) Convert the following infix expression into equivalent postfix expression.
$(a+b)*c-(d-e)/(f+g)$
b) What is dequeue? **[WBUT 2014]**

Answer:

a)

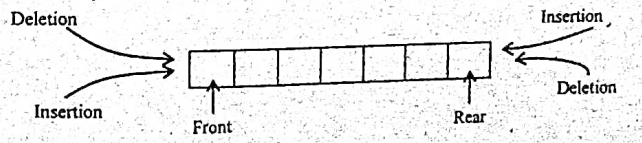
Symbol	Stack	Operator
(		(
A	a	(
+	a	(+
B	ab	
)	ab+	
*	ab+	*
C	ab+c	*
-	ab+c*	-
(	ab+c*	-(
d	ab+c*d	-(
-	ab+c*d	-(-
e	ab+c*de	
)	ab+c*de-	-
/	ab+c*de-	-/
(	ab+c*de-	-/(
f	ab+c*de-f	-/(
+	ab+c*de-f	-/(+
g	ab+c*de-fg	-/(
)	ab+c*de-fg+	-/
	ab+c*de-fg+/	-
	ab+c*de-fg+/-	

b) ***Dequeue means*** double-ended queue. In these types of queues insertion and deletion can be done from both ends.

7. Explain the types of Dequeue with suitable example. **[WBUT 2015, 2017]**

Answer:

A double-ended queue (dequeue or deque) is an abstract data type that generalizes a queue, for which elements can be added to or removed from either the front or rear. Deque differs from the queue abstract data type or First-In-First-Out List (FIFO), where elements can only be added to one end and removed from the other. This general data class has some possible sub-types:
1) An input-restricted deque is one where deletion can be made from both ends, but insertion can be made at one end only.
2) An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only.



We can use Deque as a stack by making insertion and deletion at the same side. Also we can use Deque as queue by making insetting elements at one end and removing elements at other end.
The common way of deque implementations are by using dynamic array or doubly linked list. Here this example shows the basic implementation of deque using a list, which is basically a dynamic array.
The complexity of Deque operations is O(1), when we not consider overhead of allocation/deallocation of dynamic array size.

8. Write down the algorithm to delete elements from a circular queue. **[WBUT 2015]**

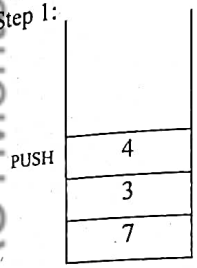
Answer:
Step 1: If FRONT = –1 then
Write ("Circular Queue Underflow")
Step 2: Return (CQ [FRONT])
Step 3: If FRONT = REAR then
FRONT=REAR= –1
Step 4: If FRONT = SIZE–1 then
FRONT=0
Else
FRONT=FRONT+1

9. An infix expression is given. Find out the equivalent postfix expression using operator stack and show the output string stepwise.
Expression: (A–B–C)*(D+E/F)/((G–I)*J)$K
where A$B signifies A^B (apply standard precedence rule). **[WBUT 2015]**
Answer:

Symbol	Postfix String	Operator Stack
(	-	(
A	A	(
-	A	(-
B	AB	(-
-	AB-	(-
C	AB-C	(-
)	AB-C-	
*	AB-C-	*
(	AB-C-	*(
D	AB-C-D	*(
+	AB-C-D	*(+
E	AB-C-DE	*(+
/	AB-C-DE	*(+/
F	AB-C-DEF	*(+/
)	AB-C-DEF/+	*
/	AB-C-DEF/+	*/
(	AB-C-DEF/+	*/(
(	AB-C-DEF/+	*/((

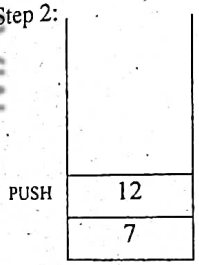
Symbol	Postfix String	Operator Stack
G	AB-C-DEF/+G	*/((
-	AB-C-DEF/+G	*/((-
I	AB-C-DEF/+GI	*/((-
)	AB-C-DEF/+GI-	*/(
*	AB-C-DEF/+GI-	*/(*
J	AB-C-DEF/+GI-J	*/(*
)	AB-C-DEF/+GI-J*	*/
$	AB-C-DEF/+GI-J*	*/$
K	AB-C-DEF/+GI-J*K	*/$
-	AB-C-DEF/+GI-J*K$/*	

10. Evaluate the following Postfix expression by showing the operand stack and intermediate output.
Expression: 7, 3, 4, *, –,3, 9, 1, +, /, *,11, $, 4, + **[WBUT 2017]**
Answer:
Evaluation using stack
Step 1:



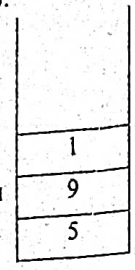
i) '*' Encountered so, POP( ), POP( ) and apply the operator.
Here, 4*3=12

Step 2:



i) '–' Encountered.
Here, 12–7=5

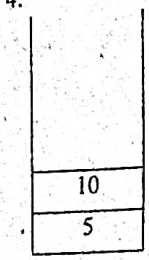
Step 3:



i) '+' Encountered.

Here, 1+9=10

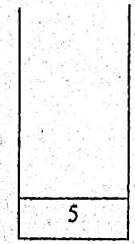
Step 4:



i) '/' Encountered.

Here, 10/5=5

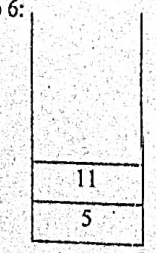
Step 5:



i) '/' Encountered, but one operand present in the stack so, push the next one.

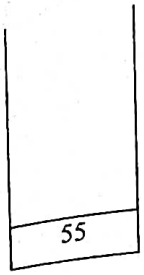
Here, 10/5=5

Step 6:



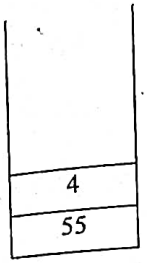
i) Here, 11*5=55

Step 7:



'$' encountered treated as symbol, not an operator

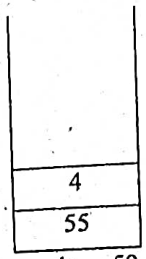
Step 8:



'+' encountered –

4+55=59

Step 9:



∴ This results = 59 (Ans.)

11. Convert the following into postfix: [WBUT 2018, 2019]

(A+B) * C+D / (E+F*G) – H

Answer:

Scanned character	Stack	Postfix
(	(	---
A	(	A
+	(+	A
B	(+	AB
)	(+)	AB+
*	(+)*	AB+
C	(+)*	AB+C

+	(+)*+	AB+C*
D	(+)*+	AB+C*D
/	(+)*+/	AB+C*D
(	(+)*+/(	AB+C*D
E	(+)*+/(	AB+C*DE
+	(+)*+/(+	AB+C*DE
F	(+)*+/(+	AB+C*DEF
*	(+)*+/(+*	AB+C*DEF
G	(+)*+/(+*	AB+C*DEFG
)	(+)*+/(+)	AB+C*DEFG*+
-	(+)*+/(+)-	AB+C*DEFG*+/+
H	(+)*+/(+)-	AB+C*DEFG*+/+H
-----	-----	AB+C*DEFG*+/+H-

12. Evaluate the POSTFIX notation by showing the OPERAND STACK and intermediate output:

9, 4, 6,*, +, 12, 7, 3, –, *, /, 17, $, 24, –

Here, A$B indicates A^B. **[WBUT 2018]**

Answer:

Symbol	Operand	Value	Operand stack
9	9	-	-
4	9,4	-	-
6	9,4,6	-	-
*	9,4,6,*	9,24	9,24
+	9,4,6,+	9,24,+	33
12	33,12	33,12	33,12
7	33,12,7	33,12,7	33,12,7
3	33,12,7,3	33,12,7,3	33,12,7,3
-	33,12,7,3,-	33,12,4	33,12,4
*	33,12,4,/	33,3	33,3
/	33,3,17	33,3,17	33,3,17
17	33,3,17,$	33,4913	33,4913,24
$	33,4913,24	33,4889,24	33,4889
24	33,4913,24,-	33,4889	-
-		33,4889	-

The output is (33,4889).

13. Write algorithm for inserting and deleting elements in a queue. **[WBUT 2019]**

Answer:

Algorithm for inserting a element in a Queue:

```
void addq(item,i)
{/* add an item to the rear of queue i*/
queuepointer temp;
MALLOC(temp,sizeof(*temp));
```



```
        data=item;
temp →  link=NULL;
temp →
if(front[i])
    rear[i] → link=temp;

else
    front[i]=temp;
rear[i]= temp;
}
```

Algorithm for deleting a element in a Queue:

```
void deleteq(int i)
{/* delete an element from queue i*/
queuepointer temp = front[i];
element item;
if(!temp)
    return queueEmpty();
item =temp →    data;
front[i]=temp → link;
    free (temp);
return item;
}
```


14. Construct the following Question of characters where Queue is a circular array which is allocated six memory cells.

FRONT = 2 REAR = 4 QUEUE: _ , A, C, D, _, _

Describe the Queue as the following operations take place:

i) **F is added to the Queue**
ii) **Two letters are deleted from the queue**
iii) **K, L, M are added to the Queue**
iv) **Two letters are deleted from the Queue**
v) **R is added to the Queue**
vi) **One letter is deleted from the Queue** **[WBUT 2019]**

Answer:

FRONT = 2, REAR = 4 & QUEUE: _, **A, C, D,** _ ,_.

i) F is added to the queue:

FRONT = 2, REAR = 4 & QUEUE: _, A, C, D, F, _

ii) Two letters are deleted from the queue.

FRONT = 4, REAR = 4 & QUEUE: _, _, D, F, _.

iii) K, L, M are added into the queue.

FRONT = 4, REAR = 3 & QUEUE: L, M, D, F, K.

iv) Two letters are deleted from the queue.

FRONT = 4, REAR = 3 & QUEUE: L, M, _, _, K.

v) R is added to the queue:

FRONT = 4, REAR = 3 & QUEUE: L, M, R, _, K.

vi) One letter is deleted from the queue.

FRONT = 4, REAR = 3 & QUEUE: L, M, R, _, _.

15. What are the applications of stack? [WBUT 2019]

Answer:

Refer to Question No. 3 (2nd Part) of Short Answer Type Questions.

Long Answer Type Questions

1. Write down the algorithm to convert an expression from in-fix to post-fix. [WBUT 2007, 2011]

OR,

Write an algorithm to convert an infix expression into its corresponding postfix expression using stack. [WBUT 2008, 2009]

Answer:

Scan the Infix string from left to right.

- Initialise an empty stack.
- If the scanned 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.

2. What is sparse matrix? How can be sparse matrix represented efficiently? [WBUT 2007]

OR,

What are sparse matrices? How is it represented in memory? [WBUT 2012]

OR,

Short Notes on sparse matrix [WBUT 2017]

Answer:

A sparse matrix is a matrix populated primarily with zeros. When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix. Operations using standard dense matrix structures and algorithms are slow and consume large amounts of memory when applied to large sparse matrices.

A bitmap image having only 2 colors, with one of them dominant (say a file that stores a handwritten signature) can be encoded as a sparse matrix that contains only row and column numbers for pixels with the non-dominant color.

The most common data structure for a matrix is a two-dimensional array. Each entry in the array represents an element a_{ij} of the matrix and can be accessed by the two indices i and j. For an $m \times n$ matrix, enough memory to store at least $(m \times n)$ entries to represent the matrix is needed.

Substantial memory requirement reductions can be realised by storing only the non-zero entries.

One example of such a sparse matrix format is the Yale Sparse Matrix Format. It stores an initial sparse $N \times N$ matrix M in row from using three arrays, A, IA, JA. NZ denotes the number of nonzero entries in matrix M. The array A then is of length NZ and holds all nonzero entries of M. The array IA stores at $IA(i)$ the position of the first element of row i in the sparse array A. The length of row i is determined by $IA(i+1) - IA(i)$. Therefore IA needs to be of length $N + 1$. In array JA, the column index of the element $A(j)$ is stored. JA is of length NZ.

For example, the matrix

[1 2 0 0]
[0 3 9 0]
[0 1 4 0]

is a three-by-four matrix with six nonzero elements, so

A = [1 2 3 9 1 4]
IA = [0 2 4 6]
JA = [0 1 1 2 1 2]

In this case the Yale representation contains 16 entries, compared to only 12 in the original matrix. The Yale format saves on memory only when $NNZ < (m(n-1) - 1) / 2$.

3. a) Define circular queue. [WBUT 2009, 2014]

OR,

What is circular queue? [WBUT 2015]

b) Write an algorithm to insert an item in circular queue. [WBUT 2009, 2014]

c) What is input restricted dequeue? [WBUT 2009, 2014]

Answer:

a) The type of representation of elements is arranged in a circular fashion where the rear again points to the front is known ***circular queues.***

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

c) Input restricted dequeue
An input-restricted deque is one where deletion can be made from both ends, but input can only be made at one end

4. a) Write algorithms to insert and delete an element from a circular queue implemented as an array.
b) Convert the following infix expression to its posfix equivalent.

$$(A+B)*D+E/(F+A*D)+C$$

[WBUT 2016]

Answer:
a) Insert: *Refer to Question No. 3(b) of Long Answer Type Questions.*
Delete: *Refer to Question No. 8 of Short Answer Type Questions.*

b)

Symbol Scanned	Stack	Output
(	(	-
A	(	A
+	(+	A
B	(+	AB
)	-	AB+
*	*	AB+
D	*	AB+D
+	+	AB+D*
E	+	AB+D*E
/	+/	AB+D*E
(	+/(	AB+D*E
F	+/(	AB+D*EF
+	+/(+	AB+D*EF
A	+/(+	AB+D*EFA
*	+/(+*	AB+D*EFA
D	+/(+*	AB+D*EFAD
)	+/	AB+D*EFAD*+
+	+	AB+D*EFAD*+/+
C	+	AB+D*EFAD*+/+C
		AB+D*EFAD*+/+C+
	None	AB+D*EFAD*+/+C+

5. Write short note on Priority queue. **[WBUT 2007, 2015]**
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.

LINKED LIST

Multiple Choice Type Questions

1. Which is the correct notation to delete the last node p from a doubly linked list? (prev is the pointer pointing to the previous node and next is the pointer pointing to the next node) [WBUT 2010, 2015]
   a) p - > next = NULL
   b) p = NULL
   c) p - > prev - > next = NULL
   d) None of these

Answer: (c)

2. The situation when a linked list Start=NULL is, [WBUT 2012]
   a) underflow
   b) overflow
   c) housefull
   d) saturated

Answer: (a)

3. If Top = NULL then which of following operations is not possible? [WBUT 2013]
   a) push
   b) pop
   c) push and pop both
   d) none of these

Answer: (b)

4. In C language, malloc() returns [WBUT 2014]
   a) integer pointer
   b) structure pointer
   c) null pointer
   d) void pointer

Answer: (d)

5. Fibonacci function $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ is an example of [WBUT 2014]
   a) linear recursion
   b) binary recursion
   c) non-linear recursion
   d) mutual recursion

Answer: (b)

6. 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 [WBUT 2014]
   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)

7. Linked lists are not suitable for implementing [WBUT 2016, 2017, 2018]
   a) insertion sort
   b) Binary Search
   c) Radix sort
   d) Polynomial manipulation

Answer: (b)

8. Linked list is a [WBUT 2016]
   a) Linear data structure
   b) Dynamic data structure
   c) Self-referential data structure
   d) All of these

Answer: (c)

9. Linked lists are not suitable for [WBUT 2019]
   a) stack
   b) AVL tree
   c) deque
   d) binary search

Answer: (b)

Short Answer Type Questions

1. Define tail recursion with suitable example. [WBUT 2007, 2017]
OR,
What is tail recursion? Explain with an example. [WBUT 2008, 2013]
OR,
Write short note on tail recursion. [WBUT 2010, 2012]

Answer:

A function call is said to be tail recursive 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 stack 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 that 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 (from this cause) during execution. Tail Recursion is then as efficient as iteration normally is.

Example:

```
void printLinkedList(list* l)
{
   if (l != NULL)
   {
      cout << l->fname << " " << l->lname << endl;
      printLinkedList(l->next);
   }
}
```


2. Write down the advantages and disadvantages of stack when implementing using array and linked list. [WBUT 2007, 2008]
OR,
What is the advantage of implementing a stack using a linked list instead of using an array? [WBUT 2016]

Answer:

Stack implementing using array

The **array implementation** aims to create an array where the first element (usually at the zero-offset) is the bottom. That is, array[0] is the first element pushed onto the stack and the last element popped off. The program must keep track of the size, or the length of the stack. The stack itself can therefore be effectively implemented as a two-element structure in C.

The push() operation is used both to initialize the stack, and to store values to it. It is responsible for inserting (copying) the value into the ps->items[] array and for

incrementing the element counter (ps->size). In a responsible C implementation, it is also necessary to check whether the array is already full to prevent an overrun.
The pop() operation is responsible for removing a value from the stack, and decrementing the value of ps->size. A responsible C implementation will also need to check that the array is not already empty.
If we use a dynamic array, then we can implement a stack that can grow or shrink as much as needed. The size of the stack is simply the size of the dynamic array. A dynamic array is a very efficient implementation of a stack, since adding items to or removing items from the end of a dynamic array is amortized O(1) time.

Stack implementing using link list

The **linked-list** implementation is equally simple and straightforward. In fact, a simple singly-linked list is sufficient to implement a stack -- it only requires that the head node or element can be removed, or popped, and a node can only be inserted by becoming the new head node.
Unlike the array implementation, our structure typedef corresponds not to the entire stack structure, but to a single node. Such a node is identical to a typical singly-linked list node, at least to those that are implemented in C.
The push() operation both initializes an empty stack, and adds a new node to a non-empty one. It works by receiving a data value to push onto the stack, along with a target stack, creating a new node by allocating memory for it, and then inserting it into a linked list as the new head.
A pop() operation removes the head from the linked list, and assigns the pointer to the head to the previous second node. It checks whether the list is empty before popping from it:

3. Write down the algorithm to implement PUSH and POP operation using stack. Justify if a stack is an abstract data type. **[WBUT 2007, 2015]**

OR,

Write the push and pop operations of stack. **[WBUT 2013]**

OR,

Write the push( ) and pop( ) functions for a stack after describing the Data-Structure clearly. **[WBUT 2019]**

Answer:

```
#define N 10                    // maximum elements in a stack
int top =  - 1   // indicates empty stack
int stck[N] // initially all the elements are -1.
void PUSH(int item)
{
  if (top == N)
  {
    printf("STACK IS FULL");
    exit(0);
  }
  stck[++top] = item;
```



```
}
int POP()
{
  int x;
  if (top ==  - 1)
  {
    printf("STACK IS EMPTY");
    exit(0);
  }
  x = stck[top--];
  stck[top + 1] =  - 1;
  return x;
}
```

A stack can have any abstract data type as an element, but is characterized by only two fundamental operations: *push* and *pop*.

4. What do you mean by recursion? Write a C code to implement Tower of Hanoi problem using recursion. **[WBUT 2007, 2011]**

OR,

Write the recursive function for the tower of Hanoi problem. **[WBUT 2013]**

Answer:

A recursive process or recursion is one in which objects are defined in terms of other objects of the same type. Using some sort of recurrence relation, the entire class of objects can then be built up from a few initial values and a small number of rules. The Fibonacci numbers are most commonly defined recursively. Care, however, must be taken to avoid self-recursion, in which an object is defined in terms of itself, leading to an infinite nesting.

The Fibonacci numbers are the sequence of numbers $\{F_n\}_{n=1}^{\infty}$ defined by the linear recurrence equation $F_n = F_{n-1} + F_{n-2}$ with $F_1 = F_2 = 1$. As a result of the definition (1), it is conventional to define $F_0 = 0$.
The Fibonacci numbers for $n = 1, 2, \ldots$ are 1, 1, 2, 3, 5, 8, 13, 21, ...

Tower of Hanoi Problem Using Recursion

```
1. #include<stdio.h>
2. #include<conio.h>
3. #include<math.h>
4. void hanoi(int x, char from,char to,char aux)
5. {
6.
7. if(x==1)
8. {
9. printf("Move Disk From %c to %c\n",from,to);
10.    }
11.    else
12.    {
13.    hanoi(x-1,from,aux,to);
```

```
14.   printf("Move Disk From %c to %c\n",from,to);
15.   hanoi(x-1,aux,to,from);
16.   }
17.
18.   }
19.   void main()
20.   {
21.   int disk;
22.   int moves;
23.   clrscr();
24.   printf("Enter the number of disks you want to play with:");
25.   scanf("%d",&disk);
26.   moves=pow(2,disk)-1;
27.   printf("\nThe No of moves required is=%d \n",moves);
28.   hanoi(disk,'A','C','B');
29.   getch();
30.   }
```

5. Why is linked list better than array? [WBUT 2007]

OR,

Discuss the advantages & disadvantages of linked list over array as linear data structure. [WBUT 2008, 2009, 2010]

OR,

What are the advantages of linked list over an array? [WBUT 2014]

OR,

What are the advantages and disadvantages of linked list over the array? [WBUT 2015, 2017, 2019]

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 algorithms to insert into and delete elements from a doubly linked list. [WBUT 2009, 2011]

Answer:

```
function insertAfter(List list, Node node, Node newNode)
    newNode.prev := node
    newNode.next := node.next
    if node.next == null
        list.lastNode := newNode
    else
        node.next.prev := newNode
    node.next := newNode
function remove(List list, Node node)
  if node.prev == null
      list.firstNode := node.next
  else
      node.prev.next := node.next
  if node.next == null
      list.lastNode := node.prev
  else
      node.next.prev := node.prev
  destroy node
```

7. Write a recursive algorithm to print Fibonacci numbers. [WBUT 2009]

OR,

Convert the following iterative form of C code to recursive code: [WBUT 2013]

```
int term, fib1 = 0, fib2 =1;
for ( int i = 3; i <=n; i + +)
{
    term = fib1 + fib2;
    fib1 = fib2;
    fib2 = term;
}
```

Answer:

Fibonacci numbers can be defined by the following recurrence relations.

$$F_0 = 0; \quad F_1 = 1;$$

$$F_n = F_{n-1} + F_{n-2} \text{ for } n \geq 2.$$

So, any number is just the summation of previous two numbers.

So, we can write the recursive function as below:

```
int fib(int n)
{
    if (n == 0)
        return (0);
    else if (n == 1)
        return (1);
    else
        return (fib(n - 1) + fib(n - 2));
}
```

8. Polynomials can be represented either by an array or by linked list. Compare & contrast these two types of representations. Represent the following polynomials by a linked data structure (show only the diagram) $-5x^5+4x^4-25x^3+10$.

OR, **[WBUT 2012]**

How a polynomial such as $5x^5+4x^3-2x+10$ can be represented by a linked list? **[WBUT 2014]**

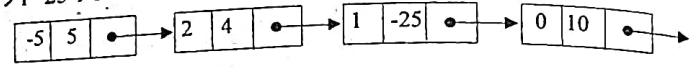
Answer:

1st Part:

In case of representation of polynomials using array; from sparse polynomials uses lots of space. But if they are represented using linked list, the space wastage is reduced.

2nd Part:

-5 5→2 4→1 -25→0 10



9. Write an algorithm to count the total number of nodes in a singly linked list. **[WBUT 2013]**

Answer:

Algorithm:

```
1.   Start
2.   check if the list is empty or not
     if phead ==NULL
     OUTPUT 'Zero nodes' and exit
     else goto step 3
3.   Set the head pointer to a temporary variable
     last=phead
4.   Traverse till the last node
     SET while(last->next!= NULL)
          { increase the node count by 1
          SET count++
          point to the next node in the list
          SET last=last->next
          }
5.   Increase the value of count by 1 for last node if the list
     has more than one node otherwise this condition is
     applicable if the list has exactly one node.
6.   Display the total count of nodes
          OUTPUT count
7.   Stop
```

10. Write an algorithm to insert a node at the end of a linked list. **[WBUT 2013]**

OR,

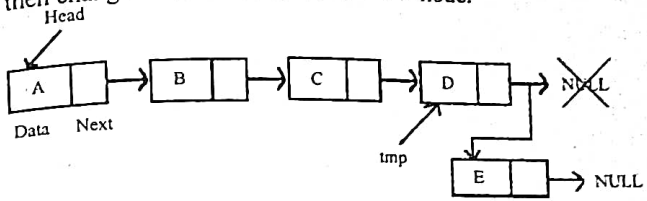
Write an algorithm to insert an element at the end of a linked list pointed by a tail pointer, not a head pointer. **[WBUT 2018]**

Answer:

Add a node at the end: (6 steps process)

The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.

Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.




```
/* 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(struct 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 make 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;
}
```

11. Write an algorithm to insert a data X after a specific data item Y in a linked list. [WBUT 2014]

Answer:

Algorithm to insert item after a specific node

Description: Here START is a pointer variable which contains the address of first node. NEW is a pointer variable which will contain address of new node. Y is the value after which new node is to be inserted and X is the value to be inserted.

1. If (START == NULL) Then
2. Print: Linked-List is empty. It must have at least one node
3. Else
4. Set PTR = START, NEW = START
5. Repeat While (PTR != NULL)
6. If (PTR->INFO == Y) Then
7. NEW = New Node
8. NEW->INFO = X
9. NEW->LINK = PTR->LINK
10. PTR->LINK = NEW
11. Print: X inserted
12. ELSE
13. PTR = PTR->LINK

12. Write a tail recursive algorithm to find out factorial of a number. What are the disadvantages of recursion? [WBUT 2015]

Answer:

1st part:

```
#include<iostream>
using namespace std;

// A tail recursive function to calculate factorial
unsigned factTR(unsigned int n, unsigned int a)
{
    if (n == 0)  return a;
    return factTR(n-1, n*a);
}
// A wrapper over factTR
unsigned int fact(unsigned int n)
{
   return factTR(n, 1);
}
// Driver program to test above function
```



```
int main()
{
    cout << fact(5);
    return 0;
}
```

2nd part:

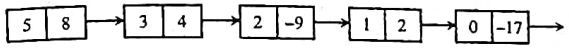
Disadvantages of Recursion

- Recursive solution is always logical and it is very difficult to trace. (debug and understand).
- In recursive we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise the function will never return.
- Recursion takes a lot of stack space, usually not considerable when the program is small and running on a PC.
- Recursion uses more processor time.

13. 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? [WBUT 2016]

Answer:

1st Part:



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

14. Write an algorithm to insert a node at the *k*th position of a doubly linked list. [WBUT 2016]

Answer:

Input: X is the data content of the node to be inserted, and KEY the data content of the node after which the new node is to be inserted.

Output: A double linked list enriched with a node containing data X after the node with data KEY, if KEY is not present in the list then it is inserted at the end.

Data structure: Double linked list structure whose pointer to the header node is the HEADER.

Steps:

1. ptr = HEADER
2. While (ptr→DATA#KEY) and (ptr→RLINK # NULL //Move to the key node if the //current node is not the KEY node or if the list reaches the end
3. ptr = ptr→RLINK
4. EndWhile
5. new = GetNode(NODE) //Get a new node from the pool of free storage
6. If (new = NULL) then //When the memory is not available

```
7.      Print (Memory is not available)          //Quit of the program
8.      Exit
9.  Endif
10. If (ptr→RLINK = NULL) then    //If the KEY is not found in the list
11. new→LLINK = ptr                //Insert at the end
12. ptr → RLINK = new
13. new → RLINK = NULL
14. new → DATA = X                 //Copy the information to the newly inserted node
15. Else                           //The KEY is available
16.     ptr = ptr→RLINK            //Next node after the key node
17.     new → LLINK = ptr          //Change the pointer
18.     new → RLINK = ptr1         //Change the pointer
19.     ptr → RLINK = new          //Change the pointer
20.     ptr1 → LLINK = new         //Change the pointer
21.     ptr = new                  // This becomes the current node
22.     new →DATA = X              //Copy the content to the newly inserted node
23. End if
24. Stop
```

15. Write an algorithm to delete a node from a singly linked list. [WBUT 2018, 2019]

Answer:

1. [Checking for deletion of first node and deleting]
   If (DATA(T) = = KEY)
   P ← T
   T ← LINK (T)
   Call REMOVE NODE(P)
   return (T).
2. [Traversing to the destination node]
   I ← T
   while (DATA (LINK(T)) != key)
   if ((LINK (LINK(T)) == 0)
   return (0)
   else
   T ← LINK(T)
3. [Deleting the target node]

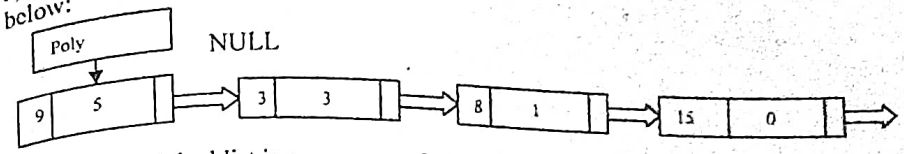
16. a) Represent the following polynomial by linked list (show the diagram only):

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

b) What is circular linked list? [WBUT 2019]

Answer:

a) The Polynomial representation of 9x^5 + 3x^3 – 8x + 15 by linked list as shown in below:



b) A circular linked list is a sequence of elements in which every element has a link to its next element in the sequence and the last element has a link to the first element. That means circular linked list is similar to the single linked list except that the last node points to the first node in the list.

17. What is recursion? What are the difference between iteration and recursion? [WBUT 2019]

Answer:

1st Part: *Refer to Question No. 4 (1st Part) of Short Answer Type Questions.*

2nd Part:

Features	Recursion	Iteration
Size of code	It reduces the size of the code.	It makes the code longer.
Speed	Slow in execution.	Fast in execution.
Overhead	It possesses the overhead of repeated function calls.	No overhead of repeated function call.
Applied	It is always applied to functions.	It is applied to iteration statements or loops.
Stack	The stack is used to store the set of new local variables and parameters each time the function is called.	Does not uses stack.

Long Answer Type Questions

1. Write an algorithm to reverse a singly linked list. [WBUT 2007, 2008, 2010, 2011, 2015, 2016]

Answer:

1st Part:

Reverse a linked list

```
/*C function*/
      //
      // iterative version
      //
      Node* ReverseList( Node ** List )
      {
```

```
        Node *temp1 = *List;
        Node * temp2 = NULL;
        Node * temp3 = NULL;

    while ( temp1 )
        {
        *List = temp1; //set the head to last node
           temp2= temp1->pNext; // save the next ptr in temp2
           temp1->pNext = temp3; // change next to previous
           temp3 = temp1;
           temp1 = temp2;
        }
        return *List;
}
```

2. What is recursion tree? Draw the recursion tree for computing nth Fibonacci numbers, where *n* may be any number & Tower Hanoi having 4/3 disks. Write short notes on Tail Recursion & Tower of Hanoi Problem. **[WBUT 2012]**

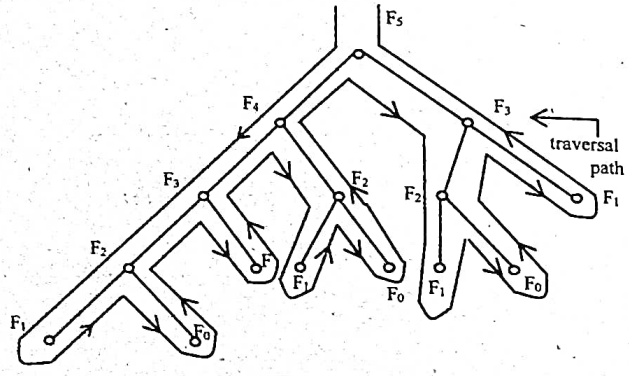
Answer:

1st Part:

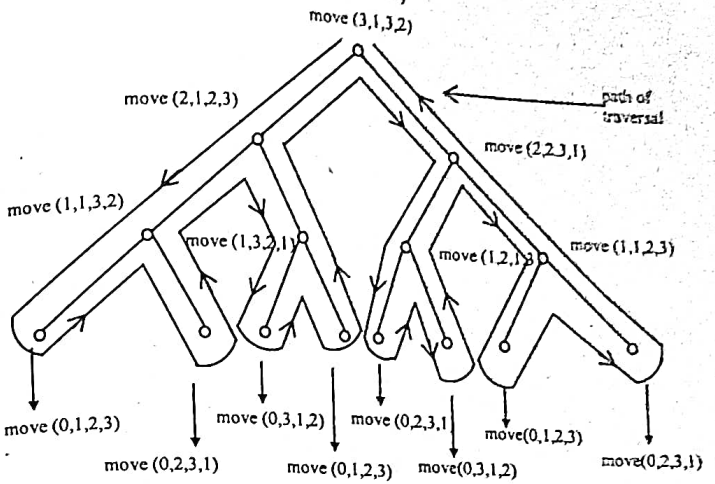
Recursive functions are internally evaluated by using stack. Now, the connection between the stacks and function calls can be shown by using a tree. That tree is called recursion tree.

2nd Part:

We can draw the recursion tree for computing nth Fibonacci numbers for the value of n = 5 as below:



The recursion tree of Tower Hanoi having 3 disks,



3rd Part:

Tail Recursion: ***Refer to Question No. 1 of Short Answer Type Questions.***

Tower of Hanoi Problem:

```
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,

$T(n) = 2\,T(n-1) + k_1$

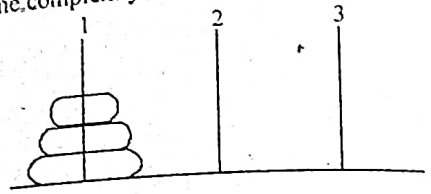
$T(0) = k_2$, a constant.

$T(1) = 2\,k_2 + k_1$

$T(2) = 4\,k_2 + 2k_1 + k_1$

$T(2) = 8\,k_2 + 4k_1 + 2k_1 + k_1$

Coefficient of $k_1 = 2^n$
Coefficient of $k_2 = 2^n - 1$
Time complexity is $O(2^n)$ or $O(a^n)$ where a is a constant greater than 1.
So it has exponential time complexity



Tower of Hanoi

3. a) Write an algorithm to delete a node from the middle of a double link list. [WBUT 2014]

OR,

Write an algorithm to delete a node from a Doubly Linked List. [WBUT 2015]

Answer:

Let us assume that node containing data value x is to be deleted. p initially contains the address of the head node.

Step 1: store the address of the head node then start traversing the linked list from the head node
Step 2: if the data value of the head node is not equal to x goto step 7
Step 3: make the next node of head as new head node
Step 4: make the prev of the new head node as NULL.
Step 5: free the previous head node.
Step 6: return the address of the new head node and Goto Step 12
Step 7: identify the address of the node containing data value x as follows
Step 7a) if the data value of p is equal to x
Step 7b) else change p by its next node address
Step 8: change the prev of next of p by prev of p
Step 9: change the next of prev of p by next of p
Step 10: free the node p
Step 11: return the stored head node address
Step 12: End

b) Write an algorithm to insert a node at a specified position in a single linked list. [WBUT 2014]

Answer:

The algorithm for inserting in the above manner is given below:

Let us assume that x is data value to be inserted at nth position of the list. p initially contains the address of the head node. pos contains the position where the data to be inserted. In this case as there is a possibility that the head node address may change so the algorithm will return the address of head node at the end.

Step 1: store the address of the head node then start traversing the linked list from the head node
Step 2: if the position is not equal to 1 goto step 8
Step 3: create a new node
Step 4: place x to the data field of the new node
Step 5: change the next address of the new node by the head node address
Step 6: make this new node as head node
Step 7: goto Step 13
Step 8: identify the address of the node at (n–1)th position
Step 9: create a new node
Step 10: place x to the data field of the new node
Step 11: place the next address of (n–1)th node to the next address of new node
Step 12: change the next address of the (n–1)th node by the new node address
Step 13: End

4. Write an algorithm to insert an item after a particular key element in doubly linked list. Write the search algorithm separately for the same. [WBUT 2017]

Answer:

1st Part

Inserting At Specific location in the list (After a Node). We can use the following steps to insert a new node after a node in the double linked list...

Step 1: Create a **newNode** with given value.

Step 2: Check whether list is **Empty (head == NULL)**

Step 3: If it is **Empty** then, assign **NULL** to **newNode → previous** & **newNode → next** and **newNode** to **head.**

Step 4: If it is **not Empty** then, define two node pointers **temp1** & **temp2** and initialize **temp1** with **head.**

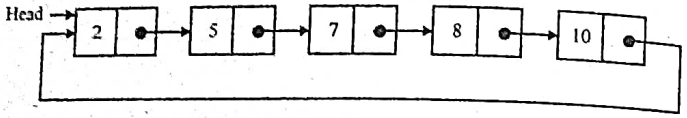
Step 5: Keep moving the **temp1** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to **location**, here location is the node value after which we want to insert the newNode).

step 6: Every time check whether **temp1** is reached to the last node. If it is reached to the last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp1** to next node.

Step7: Assign **temp1→next** to **temp2, newNode** to **temp1→next, temp1** to **newNode → previous, temp2** to **newNode→ next**and **newNode** to **temp2 → previous.**

2nd part:
Display:
Step 1: Check whether list is Empty (head == NULL)
Step 2: If it is Empty, then display 'List is Empty!!!' and terminate the function.
Step 3: If it is not Empty, then define a Node pointer 'temp' and initialize with head.
Step 4: Display 'NULL <— '.
Step 5: Keep displaying temp → data with an arrow (<==>) until temp reaches to the last node
Step 6: Finally, display temp → data with arrow pointing to NULL (temp → data → NULL).

5. Write short notes on the following:
a) Circular link list [WBUT 2014, 2018]
b) Stack ADT [WBUT 2014]
Answer:
a) Circular link list:
Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.



Advantages of Circular Linked Lists:
1) Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
2) Useful for implementation of queue. Unlike this implementation, we don't need to maintain two pointers for front and rear if we use circular linked list. We can maintain a pointer to the last inserted node and front can always be obtained as next of last.
3) Circular lists are useful in applications to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list.
4) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

b) Stack ADT:
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).
In object oriented programming the ADT Stack is specified by

- one *constructor*, namely,
  - stack *create_stack(void)
    - that constructs a data structure of type stack, empty for the moment, and returns a pointer to it,
- two *access functions*, namely,
  - boolean stack_is_empty(stack *s)
    - that, given a pointer s to a stack, returns TRUE if the stack is empty and returns FALSE otherwise,
  - stack_object *top_of_stack(stack *s)
    - that, given a pointer s to a nonempty stack, returns a pointer to the object on the top of the stack,
- two *manipulator functions*, namely,
  - void push_on_stack(stack *s, stack_object *object);
    - that, given a pointer s to a stack and given a pointer object to an object, pushes the object on the top of the stack,
  - void pop_stack(stack *s);
    - that, given a pointer s to a nonempty stack, pops the stack.

TREES

Multiple Choice Type Questions

1. A full binary tree with n leaves contains [WBUT 2007, 2011, 2015]
a) n nodes b) log n nodes c) $2n - 1$ d) 2^n
Answer: (c)

2. Number of possible binary trees with 3 node is [WBUT 2009, 2011]
a) 3 b) 2 c) 4 d) 5
Answer: (a)

3. Total nodes in a 2-tree (strictly binary tree) with thirty leaves are [WBUT 2009, 2017]
a) 60 b) 58 c) 59 d) 57
Answer: (c)

4. In a height balanced tree, heights of two sub-trees of every node never differ by more than [WBUT 2009, 2011]
a) 2 b) 0 c) 1 d) –1
Answer: (c)

5. To represent hierarchical relationship between elements, which data structure is suitable? [WBUT 2012]
a) deque b) priority c) tree d) all of these
Answer: (c)

6. When converting binary tree into extended binary tree, all the original nodes in binary tree are [WBUT 2012]
a) internal nodes on extended tree b) external nodes on extended tree
c) vanished on extended tree d) none of these
Answer: (a)

7. If node having two children is deleted from a binary tree, it is replaced by its
a) inorder predecessor b) inorder successor [WBUT 2013]
c) preorder predecessor d) none of these
Answer: (b)

8. Maximum possible height of an AVL tree with 7 nodes is [WBUT 2014, 2018, 2019]
a) 3 b) 4 c) 5 d) 6
Answer: (a)

9. A B-tree is [WBUT 2014]
a) always balanced b) an ordered tree
c) a directed tree d) all of these
Answer: (d)

10. In a heap, the right child of a nod in position 10 will be in position [WBUT 2015]
a) 20 b) 21 c) 9 d) 5
Answer: (b)

11. A binary tree T has n leaf nodes. The no. of nodes of degree 2 in T is [WBUT 2016]
a) $\log_2 n$ b) $n-1$ c) n d) 2^n
Answer: (b)

12. Which of the following need not be a binary tree? [WBUT 2016]
a) Search tree b) Heap c) AVL d) B-tree
Answer: (d)

13. A binary tree in which if all its levels except possibly the last, have the maximum number of nodes and all the nodes at the last level appear as far left as possible, is known as [WBUT 2016]
a) Full binary tree b) AVL tree
c) Threaded tree d) Complete binary tree
Answer: (a)

14. What is a suitable and efficient data structure to construct a tree? [WBUT 2018, 2019]
a) Linked list b) Stack c) Queue d) None of these
Answer: (d)

15. The height difference of any node in an AVL tree is [WBUT 2018, 2019]
a) –1, 0, 1 b) –2, 0, 1 c) –2, 0, 2 d) –1, 0, 2
Answer: (a)

Short Answer Type Questions

1. Prove that for any non-empty binary tree T, if n_0 is the number of leaves and n_2 is the number of nodes of degree 2, then $n_0 = n_2 + 1$. [WBUT 2008, 2009, 2010, 2011, 2014]

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.

$n = n0+n1+n2$,

$B+1=n$,

$B=n1+2n2$

$\Rightarrow n1+2n2+1=n$,

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

$\Rightarrow n0=n2+1$

2. Define and distinguish between binary tree and B-tree. **[WBUT 2008, 2011]**

Answer:

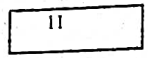
Binary tree is a tree data structure in which each node has at most two children. Typically the first node is known as the parent and the child nodes are called left and right.

A B-tree is a tree data structure that keeps data sorted and allows searches, insertions, deletions, and sequential access in logarithmic amortized time. The B-tree is a generalization of a binary search tree in that more than two paths diverge from a single node. Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. It is most commonly used in databases and file systems.

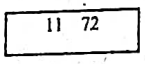
3. Construct a B-tree of order 5 with the following elements:
11, 72, 61, 20, 110, 40, 80, 130, 100, 42, 191, 92, 181, 242, 32, 122, 140, 162.
[WBUT 2009, 2015]

Answer:

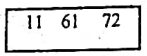
Insert 11:



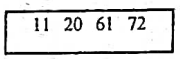
Insert 72:



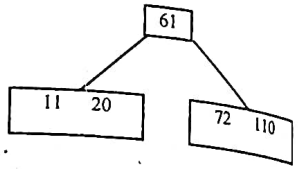
Insert 61:



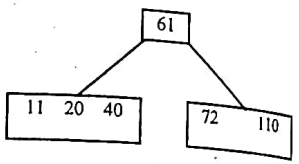
Insert 20:



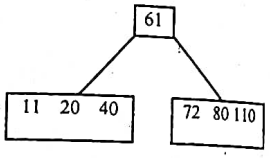
Insert 110:



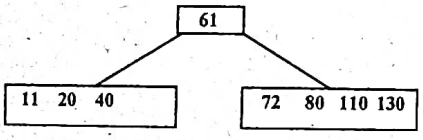
Insert 40:



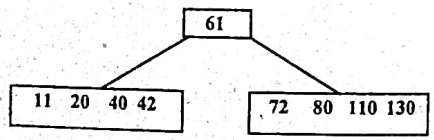
Insert 80:



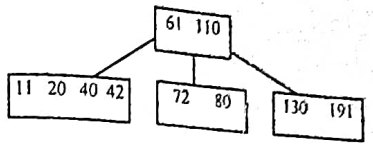
Insert 130:



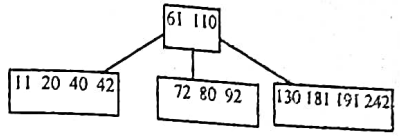
Insert 42:



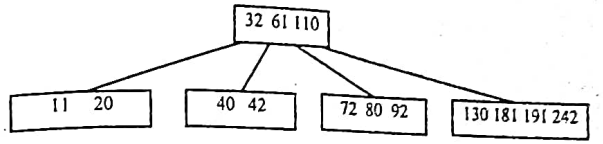
Insert 191:



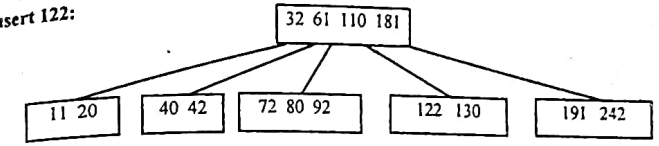
Insert 92, 181, 242:



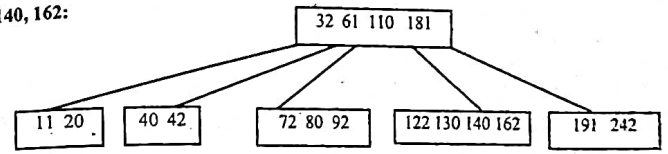
Insert 32:



Insert 122:



Insert 140, 162:

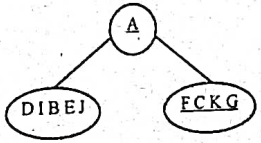


4. Given below are in-order and pre-order traversals of a binary tree. Draw the tree and find its post-order traversal. **[WBUT 2009, 2010]**

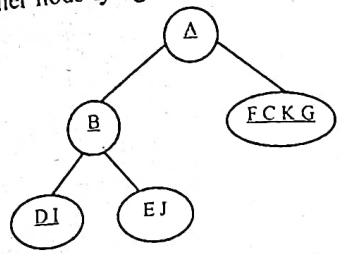
Pre-order: A B D I F J C F G K
In-order: D I B E J A F C K G

Answer:

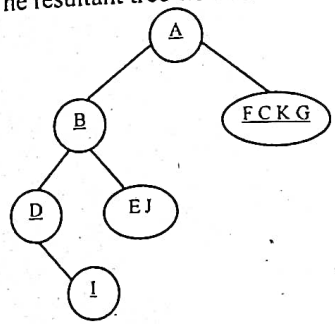
In preorder traversal, the first node is always the root of the binary tree. Here it is A. We look for the position of A in the corresponding in-order traversal. In order the root node is a pivot around which the left and right subtrees are constructed. So in our example all nodes lying to the left of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



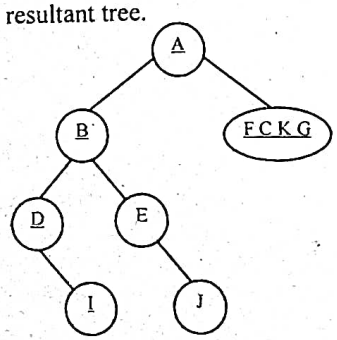
Again from pre-order traversal sequence the next element B is selected. We again find the position of B in in-order traversal sequence. All nodes lying to the left of B will be the left sub-tree of B & all other nods lying to the right of B will represent the right sub-tree. The tree looks like.



We then choose the next node from the pre-order traversal sequence i.e. D & follow the same steps described above. The resultant tree will be.

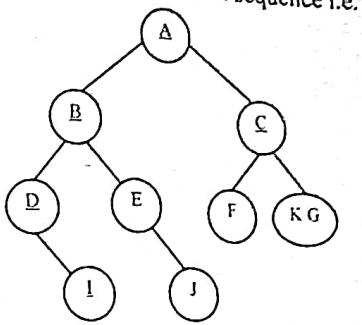


Then I is selected, but I is already placed in its exact position i.e. right of D. Then we move on to the right sub-tree of B & choose E (from pre-order sequence). We follow the same procedure to obtain the resultant tree.

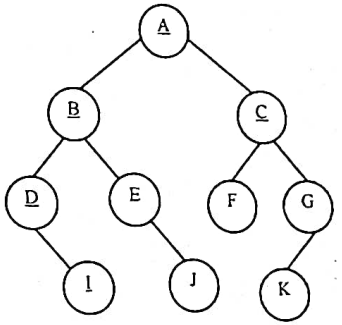


Then we move to the right of A and organize the right subtree in the same manner. The resultant trees in each step are as shown below.

We start with the next node from preorder traversal sequence i.e. C.



Then F is selected, but F is already placed in its exact position i.e. left of C. Then accordingly G and K are selected and we get the final tree as shown below.



The postorder traversal of the above tree is
I, D, J, E, B, F, K, G, C, A

5. Insert the following keys into a B-Tree of given order mentioned below:
a,g,f,b,k,d,h,m,j,e,s,i,r,x,c,l,n,t,u,p. (Order 5) **[WBUT 2012]**

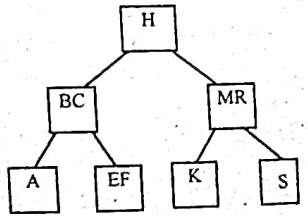
OR,

a,f,b,k,h,m,e,s,r,c. (Order 3) **[WBUT 2017]**

Answer:

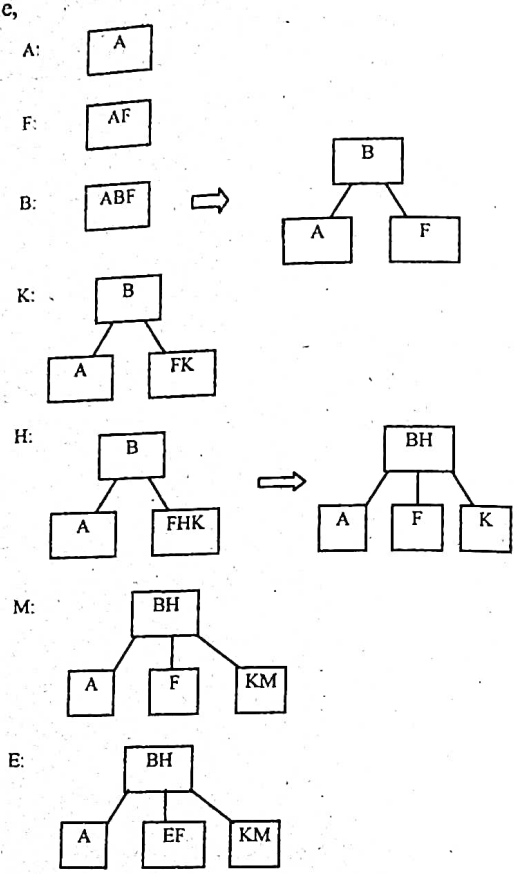
a,f,b,k,h,m,e,s,r,c. (Order 3)

SRC:

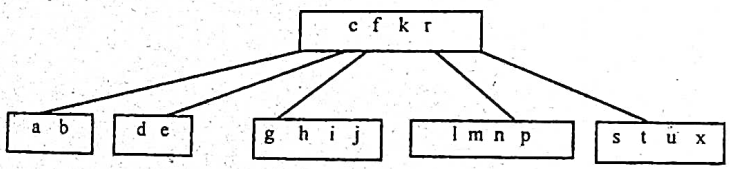


In this way the after inserting S, R, C

The tree become,



a,g,f,b,k,d,h,m,j,e,s,i,r,x,c,l,n,t,u,p. (Order 5)



6. Prove the following for a Complete Binary Tree: **[WBUT 2013]**
Number leaves (L) with number of Nodes (N) is $L=(N+1)/2$.

Answer:

A full binary tree with n nodes has height $\lg(n+1)-1$

Let A full binary tree of height h has 2^h leaf nodes.

Again, we will use induction on the height. When $h=0$, there is $2^0=1$ node and that node is a leaf node. If we assume perfect tree of height h has 2^h leaf nodes then a full binary tree of height $(h+1)$ has two subtrees, both of which are of height h and both have 2^h leaf nodes. Therefore, the perfect tree of height $h+1$, must have $2.2^h=2^{h+1}$ leaf nodes $==2^{\lg(n+1)-1}=(n+1)/2$

7. The inorder and preorder traversal sequence of nodes in a binary tree are given below: **[WBUT 2013]**

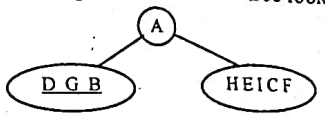
Inorder:	D	G	B	A	H	E	I	C	F
Preorder:	A	B	D	G	C	E	H	I	F

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

Answer:

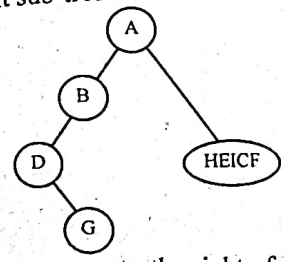
In preorder traversal, the first node is always the root of the binary tree. Here it is A. We look for the position of A in the corresponding in-order traversal. In order the root node is a pivot around which the left and right subtrees are constructed. So in our example all nodes lying to the left of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

1)



Again from pre-order traversal sequence the next element B is selected. We again find the position of B in in-order traversal sequence. All nodes lying to the left of B will be the left sub-tree of B. The tree looks like.

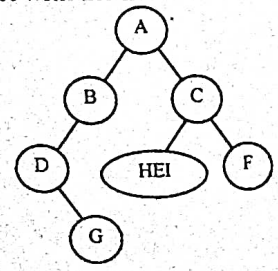
2)



Then we move to the right of A and organize the right subtree in the same manner. The resultant trees in each step are as shown below.

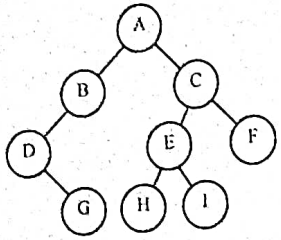
We start with the next node from preorder traversal sequence i.e. C.

3)



Then from inorder sequence F is selected, but F is already placed in its exact position i.e. right of C. Then accordingly E, H and I are selected and we get the final tree as shown below.

4)



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

in-order :	E	A	C	K	F	H	D	B	G
pre-order :	F	A	E	K	C	D	H	G	B

Draw the binary tree. State briefly the logic used to construct the Tree. [WBUT 2014]

OR,

Construct a binary tree using the in-order and pre-order traversal of the nodes given below (Indicate all intermediate steps).

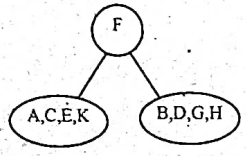
In-order: E A C K F H D B G

Pre- order: F A E K C D H G B [WBUT 2019]

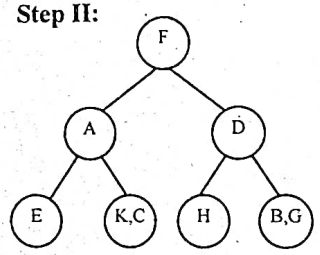
Answer:

From inorder and preorder traversal concept we can decide

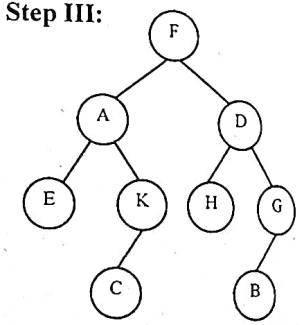
Step I:



Step II:



Step III:



9. What is threaded binary tree? [WBUT 2014]

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.

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;
}
```

10. Write a C function to find out inorder successor of the root of a binary tree. [WBUT 2015]

Answer:

In in-order traversal of binary search tree we get number in ascending order. Hence the successor of a node will be either to the right of node(if exists) or one level above.

In this case the parent pointer is given.

The code for finding in-order successor is broken into two cases:

i) If the right subtree of node x is not empty then the successor of x is just the leftmost node in the right subtree.

ii) If the right subtree of node x is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x.

```
struct node* minValue(struct node* node)
{
struct node* n = node;
while(n>left!=NULL)
n=n->left;
return n;
}
 struct node* inOrderSuccessor(struct node* root,struct node* x)
{
if(x->right)
return minValue(x->right);
else
{
y = x->parent;
while(y !=NULL && x == y->right)
{
x = y;
y = y->parent;
}
return y;
}
}
```

Time Complexity : O(h) where h is height of the tree,since we either follow a path up the tree or follow a path down the tree.

11. How does the AVL tree differ from a binary search tree? [WBUT 2015]

Construct an AVL tree by using following keys: [WBUT 2015, 2018]

8, 12, 9, 11, 7, 6, 14, 10.

Answer:

1st Part:

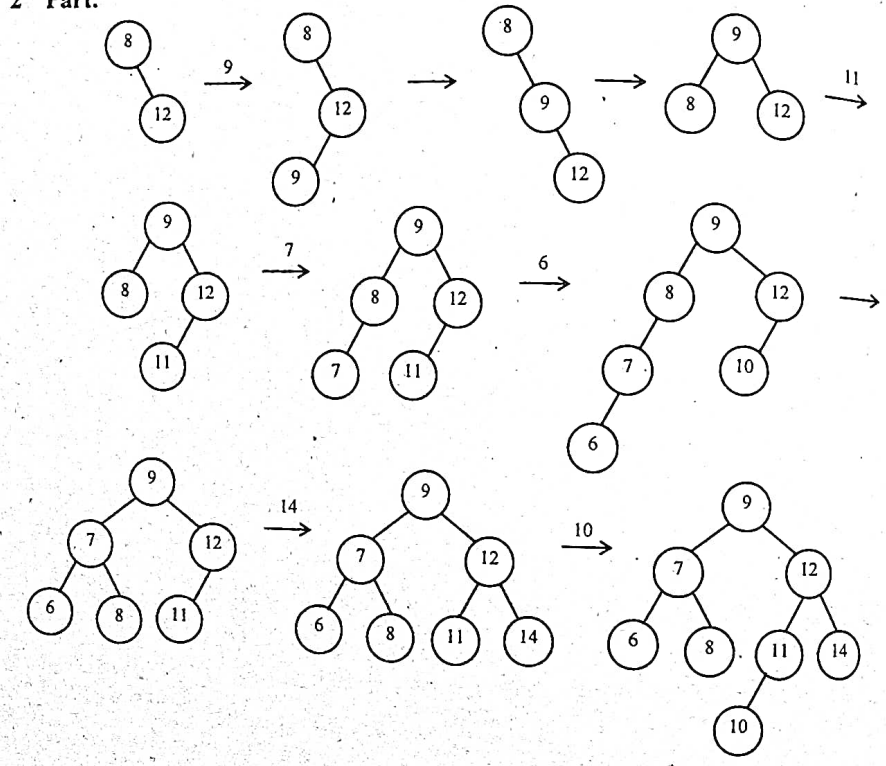
A binary tree is a tree data structure in which each node has at most two children. Typically the child nodes are called left and right. One common use of binary trees is binary search trees; another is binary heaps.

A binary search tree (BST) is a binary tree data structure which has the following properties:

- each node has a value;
- a total order is defined on these values;
- the left subtree of a node contains only values less than the node's value;
- the right subtree of a node contains only values greater than or equal to the node's value.

An AVL tree is a self-balancing binary search tree. In an AVL tree the heights of the two child subtrees of any node differ by at most one, therefore it is also called height-balanced. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases. Additions and deletions may require the tree to be rebalanced by one or more tree rotations.

2nd Part:



12. Preorder and inorder sequence are given for a Binary tree. Reconstruct the Binary Tree.

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

Preorder: G D H B E I A C J F K

Answer: **[WBUT 2015]**

Binary tree

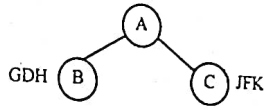
Preorder: A B D G H E I C F J K

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

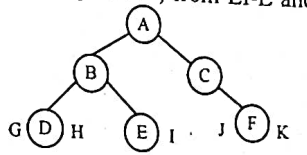
In preorder traversal, the first node is root node.

$\therefore$ G D H B E I (A) C J F K

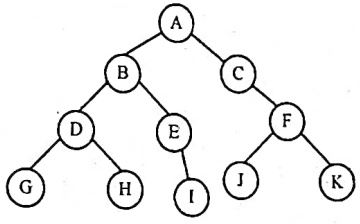
$\therefore$ among G D H B I – B comes first in preorder and among CJFK – C comes first in preorder.



Now form GDH – D comes first in preorder, from EI-E and from JFK-F comes first in preorder traversal.



$\therefore$ Binary tree becomes.



13. Prove that, for any non-empty binary tree *T*, if n_0 is the number of leaves and n_2 be the number of nodes of degree 2, then $n_0 = n_2 + 1$. **[WBUT 2016]**

Answer:

Let n be the total no. of nodes in binary tree.

n_0 no. of nodes of degree 0

n_1 no. of nodes of degree 1

n_2 no. of nodes of degree 2

$n = n_0 + n_1 + n_2$ .... (1)

All the nodes except the root node has a branch coming into it. Let B be the no. of branches in the binary tree.

$n = B + 1$ .... (2)

$B = 0.n_0 + 1 * n_1 + 2*n_2$ since there would be 0 branches from n_0, and 1 branch from n_1 nodes and two branches from n2 nodes .... (3)

$B = n_1 + 2n_2$

Substituting (3) in (2). .... (4)

$n = n_1 + 2n_2 + 1$

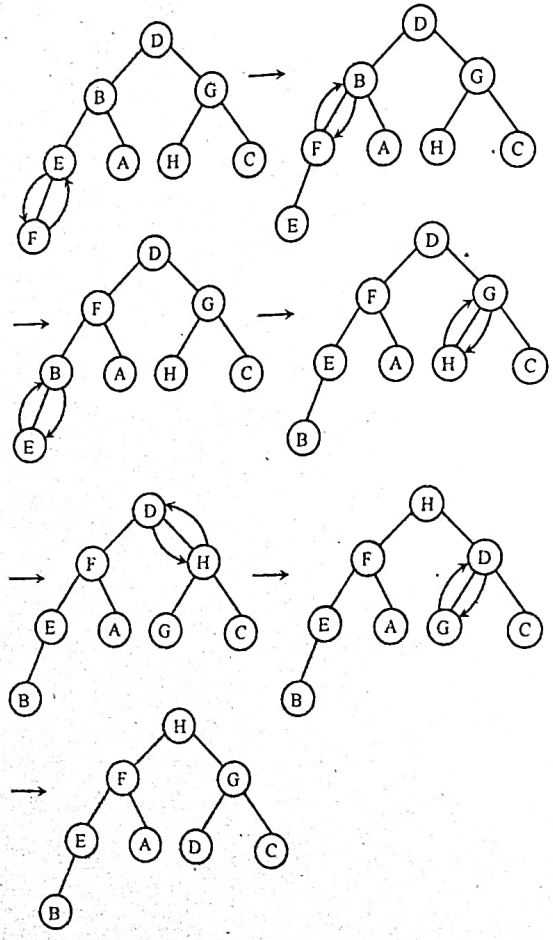
Equate 1 & 4

$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$

$n_0 = n_2 + 1$

14. Construct a max heap tree out of the following set *L* of elements. Illustrate the step by step process of insertion and heap reconstruction before the final heap is obtained. $L = \{D, B, G, E, A, H, C, F\}$ **[WBUT 2016]**

Answer:



15. What is B tree? Write down the characteristics of a B tree. [WBUT 2016, 2017]

Answer:

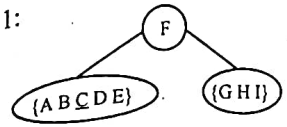
Refer to Question No. 7(a) of Long Answer Type Questions.

16. The inorder and preorder tree traversals are given. Draw the binary tree. [WBUT 2017]

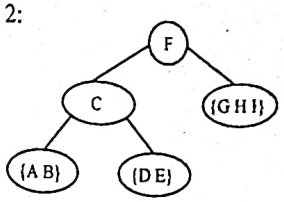
Inorder: ABCDEFGHI
Preorder: FBADCEGIH

Answer:

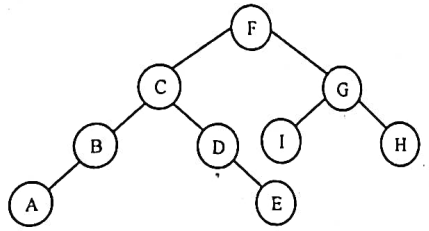
Step 1:



Step 2:



Step 3:



Binary Tree

17. The inorder and preorder sequences are given below, create the postorder sequence. [WBUT 2018]

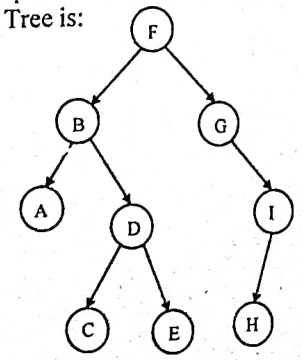
Inorder: A, B, C, D, E, F, G, H, I
Preorder: F, B, A, D, C, E, G, I, H

Answer:

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

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

The Binary Tree is:



Therefore, Post Order sequence is: C E D A B H I G F

18. Define B-Tree with suitable example. [WBUT 2018]

Answer:

Refer to Question No. 7(a) of Long Answer Type Questions.

19. What is binary tree? [WBUT 2019]

Answer:

A binary tree is a special type of tree in which every node or vertex has either no child node or one child node or two child nodes. A binary tree is an important class of a tree data structure in which a node can have at most two children.

Long Answer Type Questions

1. In a nonempty Binary tree, the following list occurs after tree traversal:

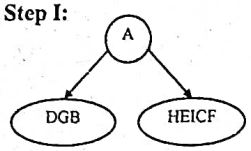
In-order: DGBAHEICF

Post-order: GDBHIEFCA

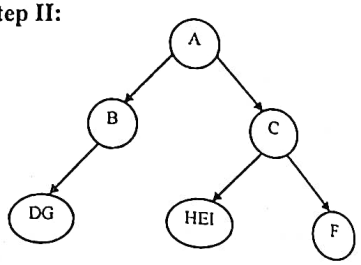
Generate the complete binary tree. [WBUT 2007, 2011]

Answer:

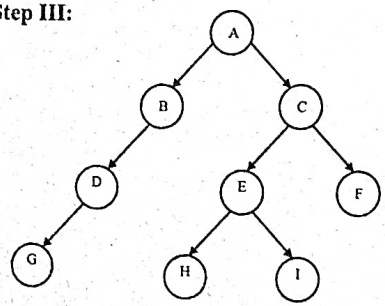
Step I:



Step II:



Step III:



2. Write down a non-recursive function for inorder traversal of a binary tree. [WBUT 2008, 2014]

OR,

Write an algorithm of in-order traversal of a binary tree. [WBUT 2009]

Answer:

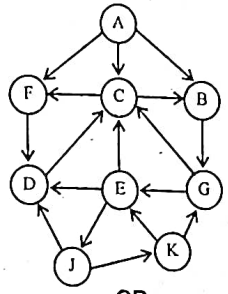
Assume that a stack ADT exists. The following is the non- recursive C function whose input is the root pointer of the binary tree stored as a linked list with left, right and key as the elements.


```
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. 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 BSF/DFS Algorithm find out the path of the given graph: [WBUT 2012]



OR,

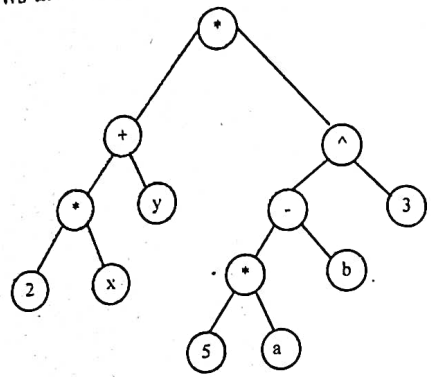
Construct an expression tree for the following expression $E=(2x+y)*(5a-b)^3$. [WBUT 2014]

Answer:

1st Part:

An Expression Tree is a data structure that contains Expressions, which is basically code. So it is a tree structure with pieces of code in it.

2nd Part:
The figure below shows the corresponding expression tree:



The in - order traversal for the tree above is:
`2 * x + y * 5 * a - b ^ 3`
The pre - order traversal for the tree above is:
`* + * 2 x y ^ - * 5 a b 3`
The post - order traversal for the tree above is:
`2 x * y + 5 a * b - 3 ^ *`

3rd Part:
Let us consider a graph G with n vertices $v_1, v_2, \ldots..v_n$. Since each edge contributes two degrees, the sum of the degrees of all vertices in G is twice the number of edge in G. That is,

$$\sum_{i=1}^{n} d(v_i) = 2e = \text{evenno}.....(1)$$

Let us consider the vertices with odd & even degrees separately.
∴ from eqn. (1) we can write as

$$\sum_{i=1}^{n} d(v_i) = \sum_{\text{even } j} d(v_j) + \sum_{\text{odd } k} d(v_k).....(2)$$

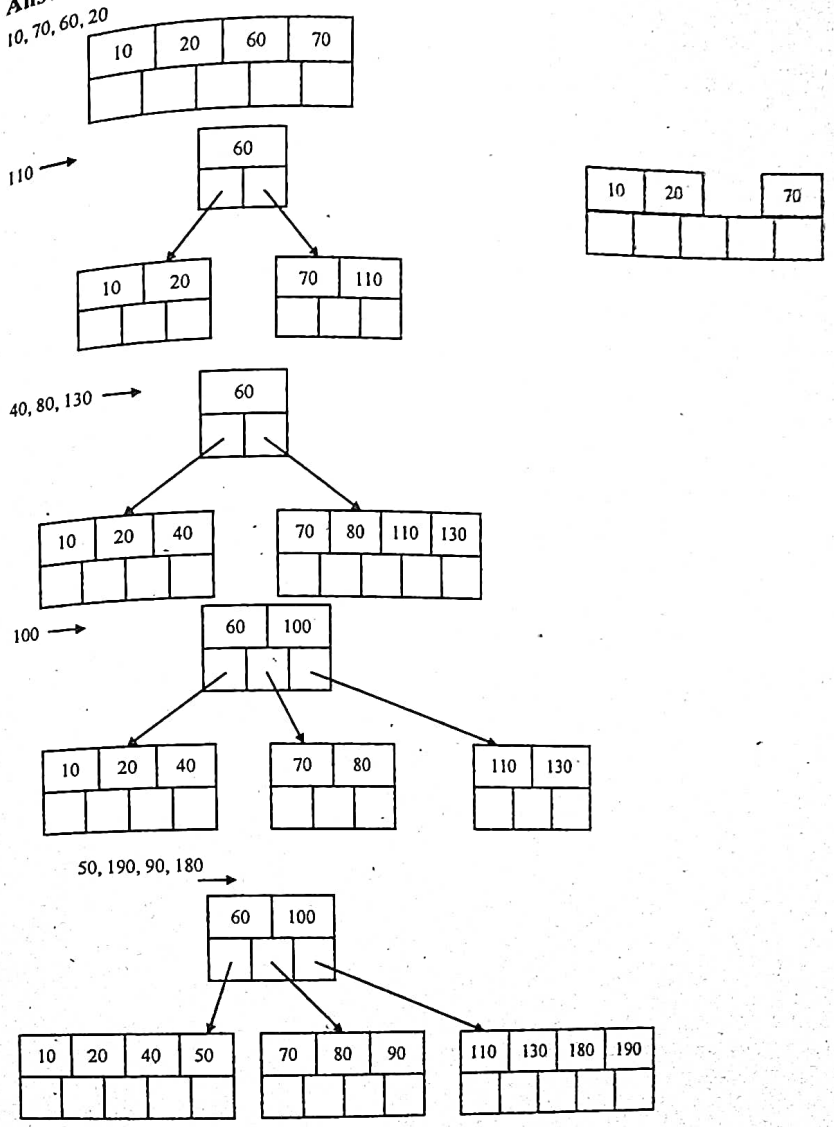
Since L.H.S of eqn. = (2) is even & the first expression on r.h.s. is even, the second expression must be an even number.

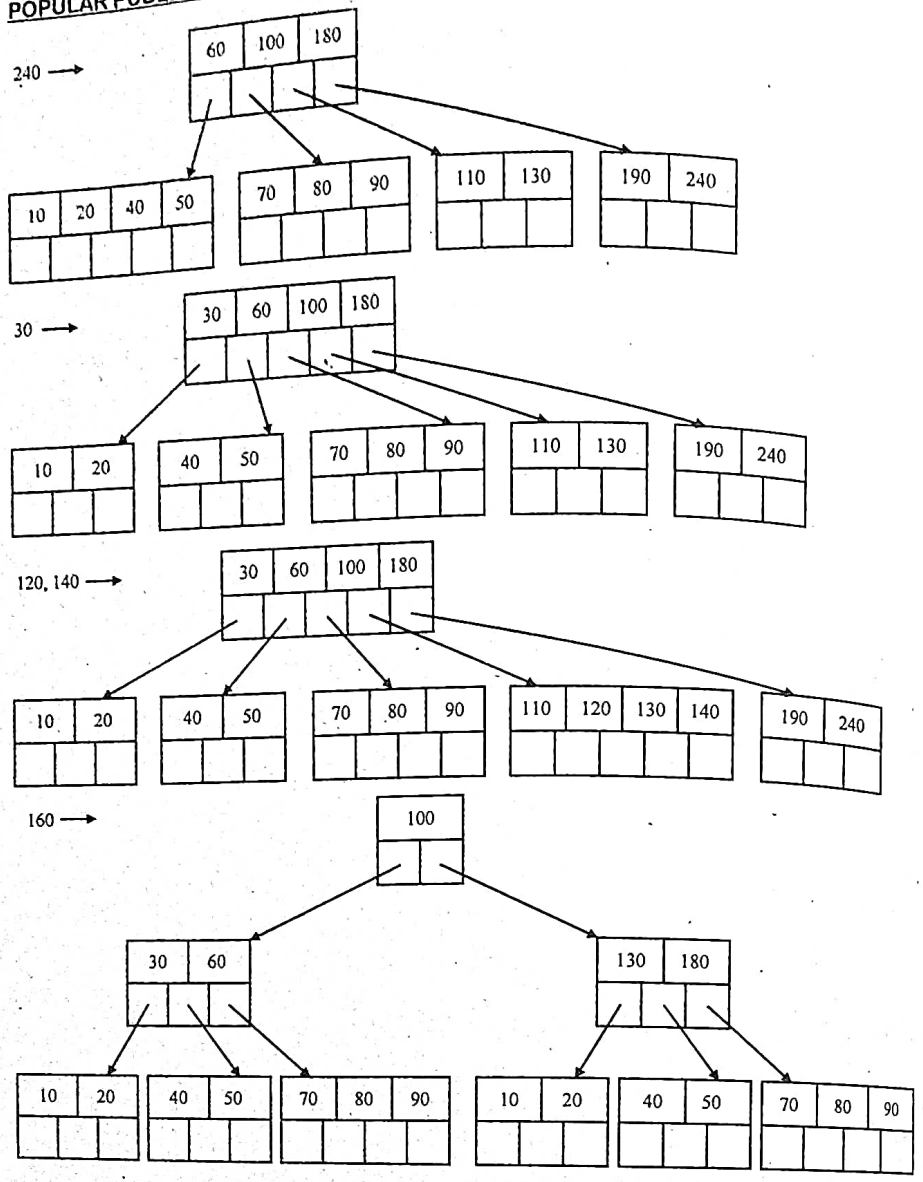
$$\therefore \sum_{\text{odd } k} d(v_k) = \text{an even no}.....(3)$$

Now, in eqn. (3) each $d(v_k)$ is odd, the total number of terms in the sum must be even to make the sum an even number.(Proved)

4th Part:
BFS - A F C B D G E J K
DFS - A F D C B G E J K

4. Construct a B-Tree of Order 5 with the following data: [WBUT 2013]
Data: 10, 70, 60, 20, 110, 40, 80, 130, 100, 50, 190, 90, 180, 240, 30, 120, 140, 160.
Answer:





5. a) If E and I denote the external and internal path length of a binary tree and *n* denotes the number of internal nodes then show that E = I + 2*n.
b) Write down the characteristics of a B-tree. **[WBUT 2015, 2017]**

Answer:

a) We use induction on *n*, the number of internal nodes in the binary tree. If *T* is any binary tree, we use the notation *I(T)* and *E(T)* to denote the internal and external path lengths of *T*, respectively.

Base Case: If a binary tree has no internal nodes , it can either be empty or it can have a single node, the root. Ih either case, E(T) = I(T) =0, and we see that for this case E(T) = I(T) + 2n.

Induction Hypothesis: Suppose the that n_0 is a non negative integer with the property that any full binary tree T with $n= n_0 (\geq 0)$ internal nodes satisfies the formula E(T) = I(T) + 2n.

Induction step: Suppose now that I is a full binary tree with $n= n_0 + 1 \geq 1$ internal nodes. T contains at least one internal node P, both of whose children, L and L', are leaves. Consider the tree T' obtained from T by deleting L and L' from T. Then in T' the node P is a leaf. So T' contains $n= n_0$ internal nodes. Furthermore if Q is any node of T' different from P, the children of Q in T' are the same as the children of Q in T. Hence T' is full , and we can apply the induction hypothesis to conclude that $E(T') = I(T') + 2n_0$.
If d(Q) denotes the depth of a node of T, we know that d(L) = d(L') = d(P) +1. Thus we have that

$$E(T) = E(T') - d(P) + 2d(L) \text{ and}$$
$$I(T) = I(T') + d(P).$$

So we have that

$$\begin{aligned} E(T) &= I(T') + 2n_0 - d(P) + 2d(L) \\ &= I(T') - d(P) + 2n_0 - d(P) + 2d(L) \\ &= I(T) + 2n_0 - 2d(P) + 2d(P) + 2 \\ &= I(T) + 2(n_0 + 1) \end{aligned}$$

b) ***Refer to Question No. 7(a) of Long Answer Type Questions.***

6. a) Write a procedure for deletion of any node from Binary Search Tree. **[WBUT 2016]**

OR,

Write algorithm to delete any node from a binary search tree. **[WBUT 2019]**

b) Construct the binary tree give the array representation of the same.

B	C	G	E		Y	D						Z
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]

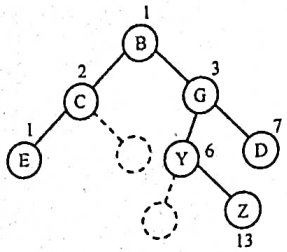
[WBUT 2016, 2018]

Answer:

a) Algorithm: Deletion in a binary search tree
Require: Input node *N* to delete, with path *P* of ancestors to the root
if *N* is a leaf or has at most one child *C* **then**
Replace pointer to *N* in its father by a pointer to *C* or to nil
for all Ancestors *A* in path *P* **do**
if *N* is in the left subtree of *A* **then**
Decrement the left subtree size field within *A*
end if

end for
Return
else
Let M be the left child of N
Let path $Q \leftarrow (P \| N)$
while M has a right child do
Let path $Q \leftarrow (Q \| M)$
Replace M by its right child
end while
Exchange the value fields of M and N
Call deletion algorithm on M with path Q {M has at most one child}
end if

b)



7. Write short notes on the following:

a) B-tree [WBUT 2007, 2010, 2012, 2019]
b) AVL tree [WBUT 2007, 2008, 2010, 2012, 2014, 2016, 2019]
c) Threaded binary tree [WBUT 2007, 2008, 2012, 2015, 2016, 2017]
d) B+ Tree [WBUT 2018]
e) Binary Tree [WBUT 2018]
f) Binary Search Tree [WBUT 2019]

Answer:

a) B-Tree:

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.

Characteristics of B-Tree

1) All leaves are at same level.
2) A B-Tree is defined by the term *minimum degree* 't'. The value of t depends upon disk block size.
3) Every node except root must contain at least t-1 keys. Root may contain minimum 1 key.
4) All nodes (including root) may contain at most 2t – 1 keys.
5) Number of children of a node is equal to the number of keys in it plus 1.
6) All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in range from k1 and k2.
7) B-Tree grows and shrinks from root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
8) Like other balanced Binary Search Trees, time complexity to search, insert and delete is O(Logn).

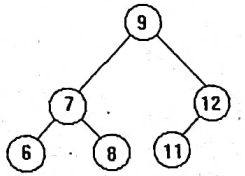
b) AVL tree:

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.



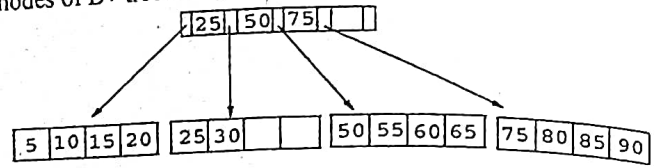
c) Threaded binary tree:

Refer to Question No. 9 of Short Answer Type Questions.

d) B+ Tree:

B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.

In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
B+ Tree are used to store the large amount of data which can not be stored in the main memory. Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.
The internal nodes of B+ tree are often called index nodes.



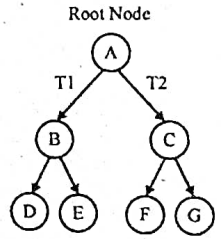
Advantages of B+ Tree:

1. Records can be fetched in equal number of disk accesses.
2. Height of the tree remains balanced and less as compare to B tree.
3. We can access the data stored in a B+ tree sequentially as well as directly.
4. Keys are used for indexing.
5. Faster search queries as the data is stored only on the leaf nodes.

e) Binary Tree:
Binary Tree is a special type of generic tree in which, each node can have at most two children. Binary tree is generally partitioned into three disjoint subsets.

1. Root of the node
2. Left sub-tree which is also a binary tree
3. Right binary sub-tree

A binary Tree is shown in the following image.



Types of Binary Tree:
1. Strictly Binary Tree
In Strictly Binary Tree, every non-leaf node contain non-empty left and right sub-trees. In other words, the degree of every non-leaf node will always be 2. A strictly binary tree with n leaves, will have (2n – 1) nodes.

2. Complete Binary Tree
A Binary Tree is said to be a complete binary tree if all of the leaves are located at the same level d. A complete binary tree is a binary tree that contains exactly 2^l nodes at each level between level 0 and d. The total number of nodes in a complete binary tree with depth d is $2^{d+1}-1$ where leaf nodes are 2^d while non-leaf nodes are 2^d-1.

Binary Tree Traversal:
1. Pre-order Traversal: Traverse the root first then traverse into the left sub-tree and right sub-tree respectively. This procedure will be applied to each sub-tree of the tree recursively.
2. In-order Traversal: Traverse the left sub-tree first, and then traverse the root and the right sub-tree respectively. This procedure will be applied to each sub-tree of the tree recursively.
3. Post-order Traversal: Traverse the left sub-tree and then traverse the right sub-tree and root respectively. This procedure will be applied to each sub-tree of the tree recursively.

f) Binary Search Tree:
A binary search tree (BST) is a binary tree data structure which has the following properties:

- each node has a value;
- a total order is defined on these values;
- the left sub-tree of a node contains only values less than the node's value;
- the right sub-tree of a node contains only values greater than or equal to the node's value.

GRAPH THEORY AND HASHING

Multiple Choice Type Questions

1. The ratio of the number of items in a hash table, to the table size is called the [WBUT 2007, 2010, 2011, 2015]
a) load factor b) item factor c) balanced factor d) all of these
Answer: (a)

2. ..... traversal of binary search tree gives the sorted list in ascending order. [WBUT 2007, 2011, 2015]
a) In-order b) Post-order c) Pre-order d) All of these
Answer: (a)

3. The adjacency matrix of an undirected graph is [WBUT 2013, 2017]
a) unit matrix b) symmetric matrix
c) asymmetric matrix d) none of these
Answer: (b)

4. Maximum no. of edges in a n-node undirected graph in a without self loop is [WBUT 2013]
a) $n-2$ b) $n(n-1)/2$
c) $n(n+1)/2$ d) none of these
Answer: (b)

5. Any connected graph with x vertices must have at least [WBUT 2014]
a) $x+1$ edges b) $x-1$ edges c) x edges d) $x/2$ edges
Answer: (b)

6. Which of the following is not a requirement of good hashing function? [WBUT 2014]
a) avoid collision b) reduce the storage space
c) make faster retrieval d) none of these
Answer: (d)

7. A vertex of in-degree zero in a directed graph is called [WBUT 2014]
a) articulation point b) sink
c) isolated vertex d) root vertex
Answer: (d)

8. A graph in which all nodes are of equal degree is known as [WBUT 2016]
a) Multi graph b) Non-regular graph
c) Regular graph d) Complete graph
Answer: (c)

9. Which amongst the following is related to hashing? [WBUT 2017]
a) Balance factor b) Chaining c) Inverse traversal d) Load factor
Answer: (b)

10. Adjacency matrix of a digraph is [WBUT 2019]
a) O (n) b) O (log n) c) O (2^n) d) O(n log n)
Answer: (a)

Short Answer Type Questions

1. What is hashing? Why is it required? What is the time complexity of hashing? Mention different types of hashing techniques. What do you mean by re-hashing? [WBUT 2007, 2015]

Answer:
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.

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.

Advantages of using hashing

1) Performs 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.

Some common has functions are mentioned below.

a. Digit selection
b. Division
c. Mid-Square
d. Folding

Re-hashing is one of the more effective methods of probing an open addressed hash table. It works by adding the hash coding of two auxiliary hash functions. The function for performing double hashing is $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$ where $0 <= i < m$, m is the number of entries in the table, and h_1 and h_2 are the hash functions. To ensure that all entries in the table are filled before collisions occur we can either make m a power of 2 and have h_2 always return an odd number or we make m prime and have h_2 always return a value less than m. For example if $h_1(k) = k \bmod m$ and $h_2(k) = (k \bmod (m - 1))$ and we

set m to 1699 (a prime number) then when we hash 15385 we probe position 94 first and then every 113th position through the table as i increases until we find the entry we are looking for.

2. a) What is hashing?
b) What is collision? How is it handled? [WBUT 2013, 2017, 2018] [WBUT 2013]

Answer:

a) *Refer to Question No. 1(1st Part) of Short Answer Type Questions.*

b) Since its always not possible to design perfect hash function with minimal overhead which would generate unique key and lead to collision.
To address this problem following are the two main collision resolving techniques:

1) Open Hashing also known as separate chaining
2) Closed Hashing also known as open addressing

3. Find out the shortest path between all pair of nodes in the given graph by Kruskal's algorithm. [WBUT 2013]

Answer:

Kruskal's algorithm:

- 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 and F is not yet spanning
- 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

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Example:

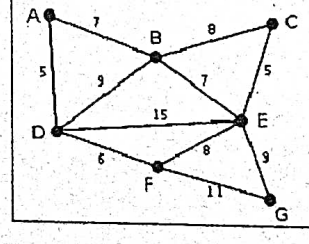
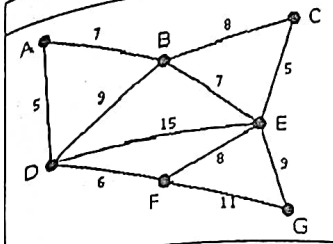
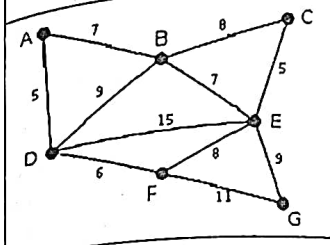
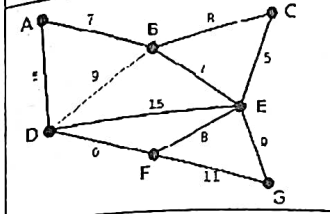
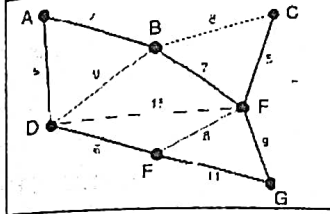
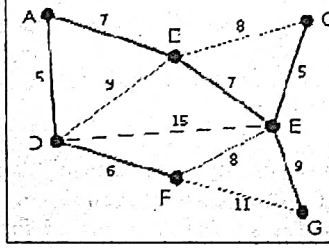
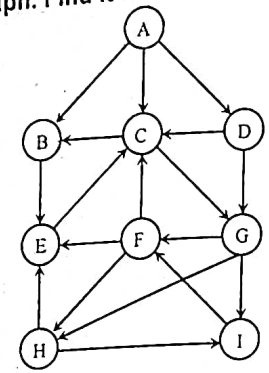
Image	Description
	AD and CE are the shortest edges, with length 5, and AD has been arbitrarily chosen, so it is highlighted.

Image	Description
	CE is now the shortest edge that does not form a cycle, with length 5, so it is highlighted as the second edge.
	The next edge, DF with length 6, is highlighted using much the same method.
	The next-shortest edges are AB and BE, both with length 7. AB is chosen arbitrarily, and is highlighted. The edge BD has been highlighted in red, because there already exists a path (in green) between B and D, so it would form a cycle (ABD) if it were chosen.
	The process continues to highlight the next-smallest edge, BE with length 7. Many more edges are highlighted in red at this stage: BC because it would form the loopBCE, DE because it would form the loop DEBA, and FE because it would form FEBAD.
	Finally, the process finishes with the edge EG of length 9, and the minimum spanning tree is found.

4. What is pendent vertex of a graph? [WBUT 2015, 2017]

Answer:
A vertex of a graph is said to be pendant if its neighborhood contains exactly one vertex.

5. Consider the following graph. Find it BFS traversal starting from A. [WBUT 2016]



Answer:

A			–						
B	C	D	A						
C	D	E	A	B					
D	E	G	A	B	C				
E	G		A	B	C	D			
G			A	B	C	D	E		
F	H	I	A	B	C	D	E	G	
H	I		A	B	C	D	E	G	F
I			A	B	C	D	E	G	F	H	
–			A	B	C	D	E	G	F	H	I

So the BFS traversal is

AB C D E G F H I

6. Describe Prim's minimal spanning tree algorithm. [WBUT 2017]

Answer:
The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets of vertices must be connected to make a ***Spanning*** Tree. And they must be connected with the minimum weight edge to make it a ***Minimum*** Spanning Tree.

- *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) WhilemstSetdoesn'tincludeallvertices
  ….a) Pick a vertex ***u*** which is not there in ***mstSet*** and has minimum key value.
  ….b) Include ***u*** tomstSet.

….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***
- The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

7. What is Hashing? What is Linear Probing? What are the functions of hashing? [WBUT 2018, 2019]

Answer:
1st Part: *Refer to Question No. 1(a) of Short Answer Type Questions.*
2nd Part: *Refer to Question No. 3 (2nd Part) of Long Answer Type Questions.*
3rd Part: *Refer to Question No. 1(c) of Long Answer Type Questions.*

8. Discuss two different ways of representing a graph. [WBUT 2019]

Answer:
Following two are the most commonly used representations of a graph.

1. Adjacency Matrix
2. Adjacency List

Adjacency Matrix:
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs.

Adjacency List:
An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the *i*th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs.

Long Answer Type Questions

1. a) Differentiate between BFS and DFS traversals. [WBUT 2007]

OR,

Compare BFS with DFS. [WBUT 2011, 2019]

b) Explain degree, adjacency matrix, adjacency list, path, cycle of a directed graph. [WBUT 2007]

What is meant by spanning tree? [WBUT 2007]

c) Explain different hash functions.

OR, [WBUT 2008]

Write down the popular hash functions.

OR, [WBUT 2017]

Briefly explain the different commonly used hashing functions.

Answer:

a) 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 $O(|E|+|V|)$.

b) 1st Part:

Degree: Degree of any vertex is defined as the number of edges incident on a particular vertex with self-loop counted twice.
Suppose that D is a digraph with n vertices, numbered 1, 2, . . . , n. Then the **adjacency matrix** A(D) of D is a square n × n array, with rows and columns numbered 1, 2, . . . , n, such that the entry in row i and column j is the number of arcs from vertex i to vertex j.
Adjacency Lists: a representation with a singly linked list.
Path: In an open walk, if no vertex appears more than once then that open walk is called path. In other words, a path does not intersect itself.
In the previous example,

$v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_5$ is a path;

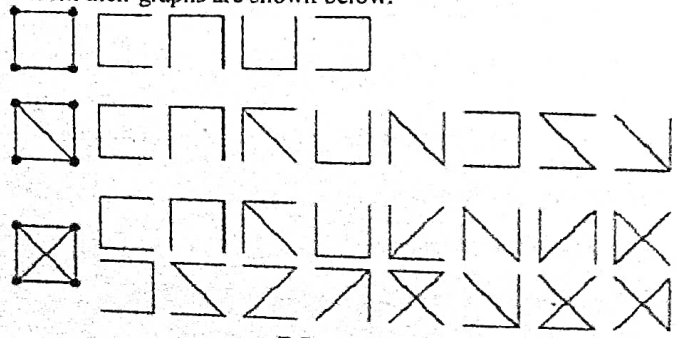
whereas,

$v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_5, e_9, v_3, e_{10}, v_4$

is not a path.
In a graph, a **cycle** is a sequence $v_0, e_1, v_1, \ldots, e_k, v_k$ $(k \geq 1)$ of alternately vertices and edges (where e_i is an edge joining v_{i-1} and v_i), with all the edges different and all the vertices different, except that $v_0 = v_k$.

2nd Part:

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:



c) Some common **hash functions** are described below.

i. Digit selection

Choose digits from certain positions of key (e.g. last 3 digits of SS#).
Unfortunately it is easy to get a biased sample. We can carefully analyze keys to see which will work best. We must watch out for patterns - they should generate all possible table positions. (For example the first digits of SS#'s reflect the region in which they were assigned and hence usually would work poorly as a hashing function.)

ii. Division

Let H(key) = key mod TableSize.
This is very efficient and often gives good results if the TableSize is chosen properly.
If it is chosen poorly then you can get very poor results. If TableSize = 28 = 256 and the keys are integer ASCII equivalent of two letter pairs, i.e. Key(xy) = 28 * ORD(x) + ORD(y), then all pairs ending with the same letter get mapped to same address. Similar problems arise with any power of 2.
The best bet seems to be to let the TableSize be a prime number.
In practice if no divisor of the TableSize is less than 20, the hash function seems to be OK. (Text uses 997 in the sample program)

iii. Mid-Square

In this algorithm you square the key and then select certain bits. Usually the middle half of the bits is taken. The mixing provided by the multiplication ensures that all digits are used in the computation of the hash code.

Example: Let the keys range between 1 and 32000 and let the TableSize be 2048 = 211. Square the Key and remove the middle 11 bits. (Grabbing certain bits of a word is easy to do using shift operators in assembly language or can be done using the div and mod operators using powers of two.)
In general r bits gives a table of size 2r.

iv. Folding

Break the key into pieces (sometimes reversing alternating chunks) and add them up.
This is often used if the key is too big. E.g., If the keys are Social security numbers, the 9 digits will generally not fit into an integer. Break it up into three pieces - the 1st digit, the next 4, and then the last 4. Then add them together. Now one can do arithmetic on them.
This technique is often used in conjunction with other methods (e.g. division)

2. a) What is a complete graph? Show that sum of degree of all the vertices in a graph is always even. **[WBUT 2008, 2011, 2014]**

Answer:

A complete graph is a simple graph in which every pair of distinct vertices is connected by an edge.

Proof of sum of degree of all the vertices in a graph is always even:
Each edge ends at two vertices. If we begin with just the vertices and no edges, every vertex has degree zero, so the sum of those degrees is zero, an even number. Now add edges one at a time, each of which connects one vertex to another, or connects a vertex to itself. Either the degree of two vertices is increased by one (for a total of two) or one vertex's degree is increased by two. In either case, the sum of the degrees is increased by two, so the sum remains even.

b) Write down BFS algorithm for searching a graph. **[WBUT 2008, 2009, 2011]**

Answer:
Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighbouring nodes. Then for each of those nearest nodes, it explores their unexplored neighbour 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.

3. Define "hashing". Explain with suitable example the collision resolution scheme using linear probing with open addressing. **[WBUT 2009, 2010, 2011]**

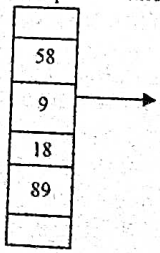
Answer:
Hashing is the transformation of a string of characters into a usually shorter fixed-length value or key that represents the original string. Hashing is used to index and retrieve items in a database because it is faster to find the item using the shorter hashed key than to find it using the original value.
Let us consider the following elements

89, 18, 58, 9, 7

H(89) = 5 (Using Division – Remainder Method)
H(18) = 4 (Using Division – Remainder Method)
H(58) = 2 (Using Division – Remainder Method)
H (9) = 2 (Using Division – Remainder Method)

Now here, already there is one element in the position 2, which is 58 in our example. But 9 is also hashed to position 2, which is occupied by 58 in our example. When this kind of situation occurs we say that a collision has taken place.

Linear Probing: It is one method for dealing with collisions. If a data element hashes to a location in the table that is already occupied, the table is searched sequentially from that location until an open location is found. Taking the collision in our case



Here 9 also wanted this location. But 58 have been allocated here. Thus the next free space is calculated to adjust this collision. Thus by sequential search we find that 3 is the location so we place 9 in address 3.

This is linear probing.

But what happens if the key hashes to the last index in the array & that space is in use? We can consider array as a circular structure & continue looking for an empty room at the beginning of the array.

Quadratic probing: It is almost similar to linear probing, except that, here the space between places in the sequence increases quadratically.
D → if collision occurs at this particular address place, then the next free location is calculated by

$D + 1^2$
Then $D + 2^2$
Then $D + 3^2$
⋮
$D + i^2$

Double Hashing/Rehashing: With this method a collision is resolved by searching the table for an empty place at intervals given by a different hash function,

$H(K) = (K + C) \bmod M$

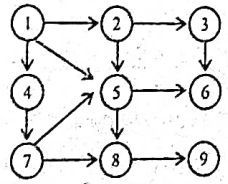
Where a second hash function is used to compute C.
Again we consider the collision situation in our example 58 & 9 → collision takes place.
By linear probing we found that 3 is the free space where 9 can be adjusted.
Now by double hashing method we compute another hashing function to adjust the value 9. Let us consider that addition hash function as

$C = H(K) = K \bmod 5 = H(9) = 9 \bmod 5 = 4$
Thus C = 4
Applying double hashing
$H(K) = (K + C) \bmod 7 = (9 + 4) \bmod 7$ where $C = K \bmod 5 = 6$

So 9 can be adjusted in position 6. This is how double hashing works.
The second hash function must be chosen based on the following characteristics:

- It must not be the same as the primary hash function
- It must never output a 0(otherwise there would be no step; every probe would land on the same cell, and the algorithm would go into an endless loop).

4. Perform DFS of the following graph: **[WBUT 2013]**



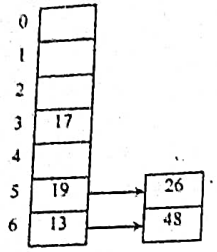
Answer:
A depth-first search starting at 1, assuming that the left edges in the shown graph are chosen before right edges, and assuming the search remembers previously visited nodes and will not repeat them (since this is a small graph), will visit the nodes in the following order: 1,4,7,8,9,5,6,2,3.

5. Consider a hash table of size 7 with hash function $h(k) = k \bmod 7$ draw the hash table that will result after inserting the following values: 19, 26, 13, 48, 17 (in the given order) for each of the three scenarios below:
(i) When collisions are handled by separate chaining method.
(ii) When collisions are handled by linear probing method.
(iii) When collisions are handled by double hashing method using a second hash function $h'(k) = 5 - (k \bmod 5)$ **[WBUT 2016, 2018]**

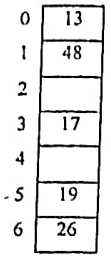
Answer:

a) 19, 26, 13, 48, 17

i)

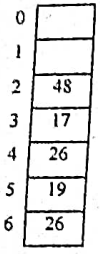


ii)



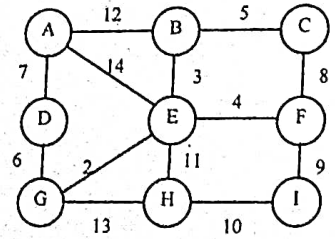
19%7 = 5
26%7 = 5 ≈ 5
13%7 = 6 ≈ 0
48%7 = 6 ≈ 1
17%7 = 3

iii)



19%7 = 5
26%7 = 5 ≈ 4
13%7 = 6
48%7 = 6 ≈ 2
17%7 = 3

6. a) What is a graph? Find out the shortest path between all pairs of nodes in the given graph by Kruskal's algorithm.

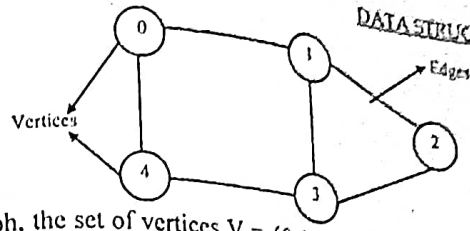


b) Define a directed graph. Provide an example. **[WBUT 2019]**

Answer:

a) 1st Part:

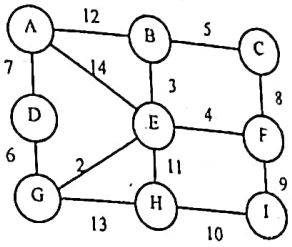
A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges which are lines or arcs that connect any two nodes in the graph. A Graph consists of a finite set of vertices (or nodes) and set of Edges which connect a pair of nodes.



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

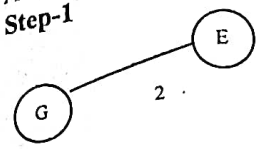
2nd Part:



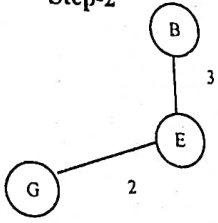
The graph consists 9 vertices and 13 Edges, So the minimum Spanning Tree formed will be having (9 – 1) = 8 Edges.

After sorting,

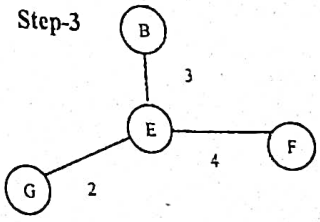
Step-1



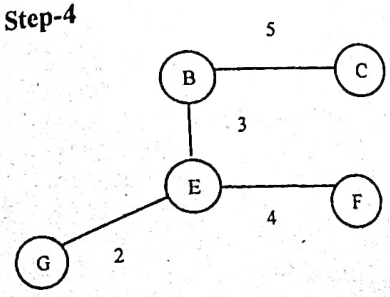
Step-2



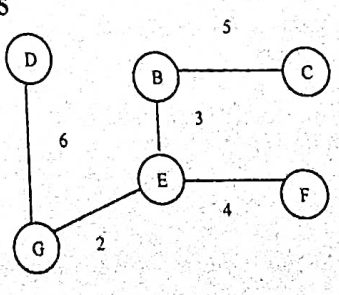
Step-3



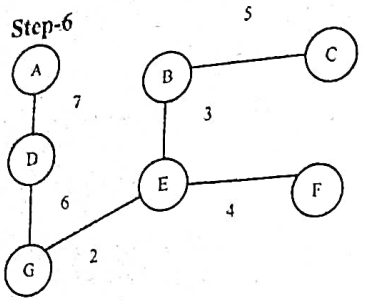
Step-4



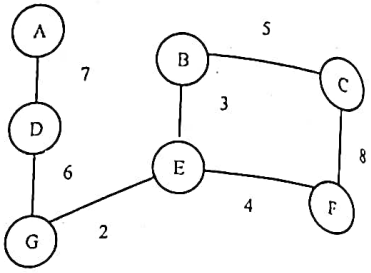
Step-5



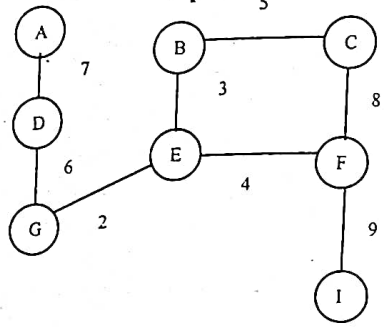
Step-6



Step-7

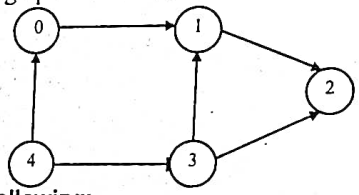


Step-8



Now, the weight of the Graph is = 7+6+2+3+4+5+8+9 = 44.

b) A directed graph is graph, where a set of objects (called vertices or nodes) are connected together, and all the edges are directed from one vertex to another. A directed graph is sometimes called a digraph or a directed network.



7. Write short notes on the following:

a) BFS vs. DFS **[WBUT 2014]**
b) Hashing **[WBUT 2014]**
c) Collision Resolution **[WBUT 2015, 2017]**
d) Spanning Tree **[WBUT 2016]**
e) DAG **[WBUT 2016]**
f) BFS **[WBUT 2019]**

Answer:

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

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

c) *Refer to Question No. 3 of Long Answer Type Questions.*

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

e) DAG:

A *directed acyclic graph* (*DAG* for short) is a graph whose edges are oriented and containing no cycle $(v_1,\cdots\cdots,v_n)$ such that edge (v_i,v_{i+1}) is directed from v_i to v_{i-1}, for $i-1,\cdots n-1$, and edge (v_n,v_1) is directed from v_n to v_1. The *underlying graph* of a DAG G is the undirected graph obtained from G by removing the directions on its edges. An *upward drawing* of a DAG is such that each edge is represented by an increasing curve. An *upward planner drawing* is a drawing that is both upward and planar. An *upward planar DAG* is a DAG that admits an upward planar drawing. In a directed graph, the *outdegree* of a vertex is the number of edges leaving the vertex, and the *indegree* of a vertex is the number of edges entering the vertex. A *source* (respectively, *sink*) is a vertex with indegree zero (respectively, with outdegree zero). An *st-plannar DAG* is a DAG with exactly one sources s and one sink t that admits an upward plannar embedding in which s and t are on the outer face. *Bipartite DAGs* and *directed trees* are DAGs whose underlying graphs are bipartite graphs and trees, respectively. A *series-parallel DAG* is a DAG that can be inductively constructed as follows. An edge (u,v) directed from u to v is a series-parallel DAG with *starting pole* u and *ending pole* v. Denote by u_i and v_i the starting and ending poles of a series-parallel DAG G_i, respectiuvely. Then a *series composition* of a sequence $G_1,G_2,\cdots,G_k$ of series-parallel DAGs, with $k\geq 2$, constructs a series-parallel DAG that has starting pole $u=u_1$, that has ending pole $v=v_k$, that contains DAGs G_i as subgraphs, and such that vertices v_i and u_{i+1} have been identified to be the same vertex, for each $i=1,2,\cdots k-1$. A *parallel composition* of a set $G_1,G_2,\cdots,G_k$ of series-parallel DAGs, with $k\geq 2$, constructs a series-parallel DAG that has starting pole $u=u_1=u_2=\cdots u_k$, that has ending pole $v=v_1=v_2=\cdots v_k$, that contains DAGs G_i as subgraphs, and such that vertices $u_1,u_2,\cdots,u_k$ (vertices $v_1,v_2,\cdots,v_k$) have been identified to be the same vertex. We remark that series-parallel DAGs are a subclass of the upward planar DAGs whose underlying graph is a series-parallel graph.

f) BFS:

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.

SORTING

Multiple Choice Type Questions

1. In external sorting technique all data reside in [WBUT 2009, 2011, 2015]
   a) primary memory b) secondary memory
   c) both (a) & (b) d) none of these
   Answer: (b)

2. The complexity of merge sort algorithm is [WBUT 2010, 2012, 2018, 2019]
   a) $O(n)$ b) $O(n^2)$ c) $O(n \log n)$ d) $O(\log n)$
   Answer: (c)

3. Binary search algorithm cannot be applied to [WBUT 2012]
   a) sorted linked list b) sorted binary trees
   c) sorted linear array d) pointer array
   Answer: (a)

4. The worst case occur in linear search algorithm when [WBUT 2012]
   a) Item is somewhere in the middle of the array
   b) Item is not in the array at all
   c) Item is the last element in the array
   d) Item is the last element in the array or is not there at all
   Answer: (d)

5. The number of interchanges required to sort 5, 1, 6, 2, 4 in ascending order using Bubble sort is [WBUT 2013]
   a) 6 b) 5 c) 7 d) 8
   Answer: (b)

6. In worst case Quick Sort has order [WBUT 2013]
   a) $O(n \log n)$ b) $O(n^2/2)$ c) $O(\log n)$ d) $O(n^2/4)$
   Answer: $O(n^2)$

7. The fastest sorting algorithm for an almost already sorted array is [WBUT 2014]
   a) quick sort b) merge sort c) selection sort d) insertion sort
   Answer: (d)

9. Queue can be used to implement [WBUT 2016, 2017]
   a) Radix sort b) Quick sort c) Recursion d) Depth First Search
   Answer: (a)

10. The best case complexity of insertion sort is [WBUT 2017]
    a) $O(n^2)$ b) $O(n \log n)$ c) $O(n^3)$ d) $O(n)$
    Answer: (d)

11. Binary search uses [WBUT 2017]
    a) divide and reduce strategy b) divide and conquer strategy
    c) heuristic search d) both (a) and (b)
    Answer: (b)

12. Worst case time complexity of the heap sort algorithm is [WBUT 2018, 2019]
    a) $O(n \log_2 n)$ b) $O(n \lg n)$ c) $O(n^2)$ d) $O(n^3)$
    Answer: (b)

Short Answer Type Questions

1. Explain DFS Algorithm with example. [WBUT 2012]
   OR,
   Explain Depth First Search with suitable example. [WBUT 2015, 2017]
   OR,
   Write down the DFS algorithm. [WBUT 2019]

Answer:

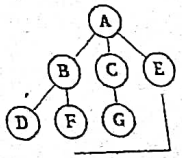
Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Algorithm
DEPTH FIRST SEARCH (G, v)
Input: A graph G and a vertex v.
Output: Edges labelled as discovery and back edges in the connected component.

```
For all edges e incident on v do
  If edge e is unexplored then
    w ← opposite (v,·e) // return the end point of e distant to v
    If vertex w is unexplained then
      - mark e as a discovery edge
      - Recursively call DSF (G, w)
    else
      - mark e as a back edge
```

The running time of DFS is Θ(V + E).

Example



A depth-first search starting at A, assuming that the left edges in the shown graph are chosen before right edges, and assuming the search remembers previously-visited nodes and will not repeat them (since this is a small graph), will visit the nodes in the following order: A, B, D, F, E, C, G.

2. Compare between insertion sort and selection sort. Write the insertion sort algorithm. [WBUT 2012, 2018, 2019]

OR,

Compare the complexity of selection sort and insertion sort. [WBUT 2014]

Answer:

1st Part:

Insertion Sort	Selection Sort
Insertion sort, sorts a set of records by inserting record into an existing sorted file. The closer the file is sorted the more efficient the insertion sort becomes.	A selection sort is one in which successive elements are selected in order and placed into their proper sorted positions.
In insertion sort we know the element and search for its place.	In selection sort we know the place and search the element to be placed.
It is a live sorting technique. It can sort the data as it arrives.	It requires all the data to be present at the beginning of the sort.
This is a stable sorting algorithm.	This is not a stable sorting algorithm.
If the initial data is sorted then -	
Comparisons = n*(n-1)/2 Moves = 0	Comparisons = n*(n-1)/2 Moves = 0
If the initial data is in reverse order then -	
Comparisons =(n-1) Moves = 2+3+4+ ----- +n	Comparisons = n*(n-1)/2 Moves = n/2 * 3 (each swap consist of 3 moves)

2nd Part:

We call a function insertion_sort(a , n) with arguments (an array and a variable, which denotes the number of elements in the array) .We define the function body as shown below.

```
void insertion_sort(int a[], int n)
{
  int step = 0, temp, i, j;
  for (i = 1; i < n; i++)
  {
    temp = a[i];
    j = i - 1;
    while ((temp < a[j]) && (j >  = 0))
    {
      a[j + 1] = a[j];
      j = j - 1;
    }
    a[j + 1] = temp;
  } // end of for loop
  printf("\nThe sorted elements in the array after insertion sort is\n ");
  for (i = 0; i < n; i++)
    printf("%3d", a[i]);
}
```


3. What is index? What are the various types of indexing? State the advantages of using indexing over a sequential file. [WBUT 2013]

Answer:

Index is created on a column in tables. It helps in providing fast access to data based on the values stored in the column.

Types of indexes:

- **Clustered:** Stores the actual row at the leaf level of the index.
- **Non-clustered:** Stores only the values from the indexed column and row locators that point to the actual row at leaf level instead of the actual row itself.
- **Composite:** Index containing more than one column (max 16).
- **Unique:** Ensures the uniqueness of each value in the indexed column.

Advantage of index sequential file organization:

1. In indexed sequential file organization, the item in the table can be examined sequentially if all the records in the file must be accessed.
2. Indexed sequential file organization is very useful when a random access or records by specifying the key is required.
3. Updating is easily accommodated.
4. Random access is possible.

4. Write an algorithm to search a node in binary search tree. [WBUT 2013]

Answer:

The TREE-SEARCH (x, k) algorithm searches the tree root at x for a node whose key value equals k. It returns a pointer to the node if it exists otherwise NIL

```
TREE-SEARCH (x, k)
if x = NIL    .OR.    k = key[x]
  then return x
if k < key[x]
  then return TREE-SEARCH (left[x], k)
  else return TREE-SEARCH (right[x], k)
```

5. Prove that, the best case time complexity for selection sort is $O(n^2)$ for input size of n. [WBUT 2014]

OR,

Write down the algorithm for selection sort. Derive the time complexities. [WBUT 2015]

Answer:

The idea behind selection sort is:

1. Find the smallest value in A; put it in A[0].
2. Find the second smallest value in A; put it in A[1].
3. etc.

The approach is as follows:

- Use an outer loop from 0 to N-1 (the loop index, k, tells which position in A to fill next).

- Each time around, use a nested loop (from k+1 to N-1) to find the smallest value (and its index) in the unsorted part of the array.
- Swap that value with A[k].

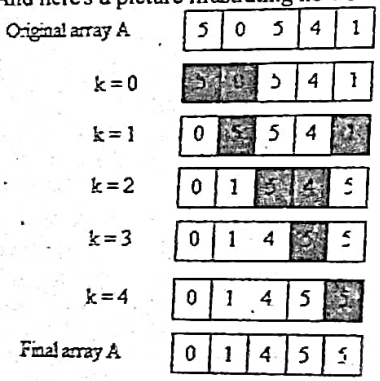
Note that after i iterations, A[0] through A[i-1] contain their final values (so after N iterations, A[0] through A[N-1] contain their final values and we're done!)

Here's the code for selection sort:

```
public static void selectionSort(Comparable[] A) {
  int j, k, minIndex;
  Comparable min;
  int N = A.length;

  for (k = 0; k < N; k++) {
     min = A[k];
     minIndex = k;
     for (j = k+1; j < N; j++) {
        if (A[j].compareTo(min) < 0) {
          min = A[j];
          minIndex = j;
        }
     }
     A[minIndex] = A[k];
     A[k] = min;
  }
}
```

And here's a picture illustrating how selection sort works:



Here the inner loop executes a different number of times each time around the outer loop, so we can't just multiply N * (time for inner loop). However, we can notice that:

- 1st iteration of outer loop: inner executes N - 1 times
- 2nd iteration of outer loop: inner executes N - 2 times
- ...
- Nth iteration of outer loop: inner executes 0 times

This is our old favorite sum: N-1 + N-2 + ... + 3 + 2 + 1 + 0
which we know is O(N2).
What if the array is already sorted when selection sort is called? It is still O(N2); the two loops still execute the same number of times, regardless of whether the array is sorted or not.

6. Find the complexity of Bubble sort. **[WBUT 2014]**

Answer:
In bubble sort algorithm we can see the first pass the inner loop iterates n-1 times. In the next iteration n-2 times and in the last pass only once.
Hence the complexity of the bubble sort is

$$(n-1) + (n-2) + (n-3) + \ldots\ldots\ldots\ldots + 2 + 1$$
$$= [(n-1)*(n-1+1)]/2 = [n(n-1)]/2$$
$$\Rightarrow O(n^2)$$

This expression is true for average case as well as worst case.

7. Explain the principle operation of quick sort with a suitable example. **[WBUT 2015, 2016]**

What is the best case complexity of insertion sort? **[WBUT 2015]**

Answer:
1st part:
Basic idea

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.

Here we don't have the merge step, at the end all the elements are in the proper order.

Algorithm
STEP 1. Choosing the pivot
Choosing the pivot is an essential step.
Depending on the pivot the algorithm may run very fast, or in quadric time:

1. Some fixed element: e.g. the first, the last, the one in the middle
   This is a bad choice - the pivot may turn to be the smallest or the largest element, then one of the partitions will be empty.
2. Randomly chosen (by random generator) - still a bad choice.

3. The median of the array (if the array has N numbers, the median is the [N/2] largest number. This is difficult to compute - increases the complexity.
4. The median-of-three choice: take the first, the last and the middle element. Choose the median of these three elements.

Example:
8, 3, 25, 6, 10, 17, 1, 2, 18, 5
The first element is 8, the middle - 10, the last - 5.
The median of [8, 10, 5] is 8

STEP 2. Partitioning
Partitioning is illustrated on the above example.
1. The first action is to get the pivot out of the way - swap it with the last element
5, 3, 25, 6, 10, 17, 1, 2, 18, 8
2. We want larger elements to go to the right and smaller elements to go to the left.
Two "fingers" are used to scan the elements from left to right and from right to left:
[5, 3, 25, 6, 10, 17, 1, 2, 18, 8]
$\hat{i}$ $\hat{j}$

- While i is to the left of j, we move i right, skipping all the elements less than the pivot. If an element is found greater then the pivot, i stops
- While j is to the right of i, we move j left, skipping all the elements greater than the pivot. If an element is found less then the pivot, j stops
- When both i and j have stopped, the elements are swapped.
- When i and j have crossed, no swap is performed, scanning stops, and the element pointed to by i is swapped with the pivot.

In the example the first swapping will be between 25 and 2, the second between 10 and 1.
3. Restore the pivot.
After restoring the pivot we obtain the following partitioning into three groups:
[5, 3, 2, 6, 1] [8] [10, 25, 18, 17]

STEP 3. Recursively quicksort the left and the right parts.

2nd part:
The insertion sort inserts the *j*th element into the correct position among the first *j-1* element that have already been put into the correct order. It does this by using a linear search technique , successively comparing the *j*th element with successive terms until a term that is greater than or equal to it is found or it compares a_j with itself and stops because a_j is not less than itself. Consequently in the worst case *j* comparisons are required to insert the jth element into the correct position. Therefore the total number of comparisons used by the insertion sort to sort a list of *n* elements is

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

Using the summation formula for the sum of consecutive integers and noting that the first term 1 is missing in the sum. Note that the insertion sort may use considerably fewer

comparisons if the smaller element started out at the end of the lists. We conclude that the insertion sort has worst case complexity $\Theta(n^2)$.

8. What is External sorting? **[WBUT 2015, 2016]**

Answer:
External sorting is required if the data to be sorted are large enough, which cannot be loaded at once in memory, and common sorting algorithms are not applicable. Larger files may be too large to fit in memory simultaneously and require external sorting.

9. If $T(n) = 15n^3 + n^2 + 4$, then prove that $T(n) = \Theta(n^3)$, find the values of n_0, c_1 and c_2. **[WBUT 2016]**

Answer:
Let $c_1 = 15$, $c_2 = 20$ and $n_0 = 1$.
Must show that $c_1 g(n) \le f(n)$ and $f(n) \le c_2 g(n)$.
$c_1 g(n) = 15n^3 \le 15n^3 + n^2 + 4 = f(n)$.
$f(n) = 15n^3 + n^2 + 4 \le 15n^3 + n^3 + 4n^3 = 20n^3 = c_2 g(n)$.

10. Write the algorithm of merge sort and explain its time complexity. **[WBUT 2016]**

Answer:
Refer to Question No. 1(a) & (c) of Long Answer Type Questions.

Long Answer Type Questions

1. a) Write down the algorithm for Merge sort. **[WBUT 2009, 2013]**
b) Show the operation of merge sort with a suitable sample data. **[WBUT 2009]**
OR,
Explain with a suitable example, the principle of Merge sort. **[WBUT 2014]**
c) Show that the running time for merge sort algorithm is O ($n\log_2 n$). **[WBUT 2009, 2012]**
d) Show that the worst case complexity of quick sort is $O(n^2)$. **[WBUT 2009]**

Answer:
a)

```
/*Recursive Merge Sort C function 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];
            }
}
```

b) 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) *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 $\log_2 n$. At each iteration the maximum number of comparisons is O (n). So, the time complexity is O ($n \log_2 n$)

d) Let $T(n)$ be the worst-case time for QUICK SORT on input size n. We have a recurrence

$T(n) = \max_{1 \le q \le n-1} (T(q) + T(n-q)) + \Theta(n)$ ——— (1)

where q runs from 1 to n-1, since the partition produces two regions, each having size at least 1.

Now we guess that $T(n) \le cn^2$ for some constant c.

Substituting our guess in equation 1.We get

$$T(n) = \max_{1 \le q \le n-1} (cq^2) + c(n-q^2)) + \Theta(n)$$
$$= c \max (q^2 + (n-q)^2) + \Theta(n)$$

Since the second derivative of expression $q^2 \div (n-q)^2$ with respect to q is positive. Therefore, expression achieves a maximum over the range $1 \le q \le n-1$ at one of the endpoints. This gives the bound $\max (q^2 + (n-q)^2)) \; 1 + (n-1)^2 = n^2 \div 2(n-1)$.

Continuing with our bounding of $T(n)$ we get

$$T(n) \le c\,[n^2 - 2(n-1)] + \Theta(n)$$
$$= cn^2 - 2c(n-1) + \Theta(n)$$

Since we can pick the constant so that the $2c(n-1)$ term dominates the $\Theta(n)$ term we have

$$T(n) \le cn^2$$

Thus the worst-case running time of quick sort is $\Theta(n^2)$.

2. Write a recursive algorithm for binary search. Compute the time complexity of your algorithm. [WBUT 2008, 2011]

OR,

Write the algorithm of binary search & calculate the complexity for best, worst & average cases. [WBUT 2010]

OR,

Write an algorithm of Binary search with an example. [WBUT 2014]

OR,

Write the algorithm of binary search and explain its strategy in brief. [WBUT 2016, 2017]

Answer:

```
1. //ALGORITHM: Bin search (a, i, l, x).
2. //given an array a(i, l) of elements in non-decreasing.
3. // order, 1<=i<=l, determine whether x is present and
4. //if so, return j such that x=a[j], else return 0.
5. {
6. if(l=i) then
{
if(x=a[i] then return i;
else return 0;
}
else
{
mid=[(i + l)/2]
if(x-a[mid] then return mid
else if(x<a[mid]) then
return Bin search(a,i,mid-1,x)
else return Bin search(a.mid+1,l,x)
}
}
```

Best case complexity O (1)
Average case complexity O (log n)
Worst case complexity O (log n)

3. Explain with suitable example, the principle operation of Selection Sort. Find out the complexity values of Quick Sort Algorithm. [WBUT 2012]

Answer:

1st Part:

1. Data elements are grouped into two sections: a sorted section and an un-sorted section.
2. Assuming the sorting order is from low to high, find the element with the lowest comparable order from the un-sorted section.
3. Place the found element to the end of the sorted section.
4. Repeat step 2 and 3 until no more elements left in the un-sorted section.

For example, consider the following array, shown with array elements in sequence separated by commas:
63, 75, 90, 12, 27

The leftmost element is at index zero, and the rightmost element is at the highest array index, in our case, 4 (the effective size of our array is 5). The largest element in this effective array (index 0-4) is at index 2. We have shown the largest element in this at the highest index in bold. We then swap the element at index 2 with that at index 4. The result is:
63, 75, 27, 12, 90
We reduce the effective size of the array to 4, making the highest index in the effective array now 3. The largest element in this effective array (index 0-3) is at index 1, so we swap elements at index 1 and 3 (in bold):
63, 12, 27, 75, 90
The next two steps give us:
27, 12, 63, 75, 90
12, 27, 63, 75, 90
The last effective array has only one element and needs no sorting. The entire array is now sorted.

Performance: Best Case: $O(n^2)$, Worst Case: $O(n^2)$

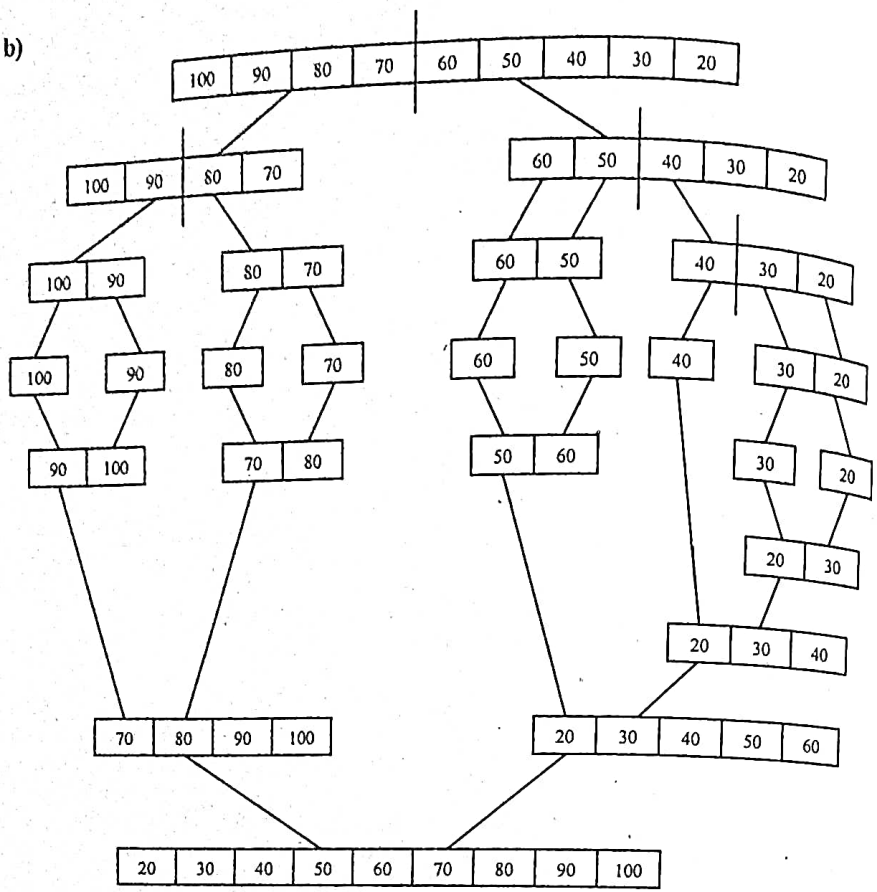
2nd Part: *Refer to Question No. 1(d) of Long Answer Type Questions.*

4. a) Why does it run faster than bubble sort in most of cases? [WBUT 2013]
b) How merge sort will sort the algorithm of the following numbers?
100, 90, 80, 70, 60, 50, 40, 30, 20.

Answer:

a) In bubble sort the worst case time complexity is $O(n^2)$.
But in case of 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 kth iteration, the algorithm splits the arrays into sub-arrays, which are 2k – 1 in number. In worst case the number of steps required to break the array into sub-arrays of single elements is log 2 n. At each iteration the maximum number of comparisons is O (n). So, the time complexity is O (n log 2 n).

b)



5. a) What do you mean by in place sorting?
b) Write the algorithm of quicksort and explain its worst case, best case and average case time complexity. **[WBUT 2017]**

Answer:

a) Sorting and Searching

n-place sorting means sorting without any extra space requirement. An in-place algorithm is an algorithm which transforms input using a data structure with a small, constant amount of extra storage space. Quicksort is one example of In-Place Sorting.

b) QuickSort

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Pseudo Code for recursive QuickSort function:

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

PartitionAlgorithm

There can be many ways to do partition, following pseudo code adopts the method given in CLRS book. The logic is simple, we start from the leftmost element and keep track of index of smaller (or equal to) elements as i. While traversing, if we find a smaller element, we swap current element with arr[i]. Otherwise we ignore current element.

```
/* low  --> Starting index,  high  --> Ending index */
quickSort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // Before pi
        quickSort(arr, pi + 1, high); // After pi
    }
}
```

AnalysisofQuickSort

Time taken by QuickSort in general can be written as following.

$T(n) = T(k) + T(n - k - 1) +$ (n)

The first two terms are for two recursive calls, the last term is for the partition process. k is the number of elements which are smaller than pivot.
The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.

Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.
$T(n) = T(0) + T(n - 1) + ...(n)$
which is equivalent to
$T(n) = T(n - 1) + ...(n)$

The solution of above recurrence is $O(n^2)$.
Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.
$T(n) = 2T(n/2) + \quad (n)$
The solution of above recurrence is (nLogn). It can be solved using case 2 of Master Theorem.

Average Case:
To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.
We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

$$T(n) = T(n/9) + T(9n/10) + \quad (n)$$

Solution of above recurrence is also O(nLogn)
Although the worst case time complexity of QuickSort is $O(n^2)$ which is more than many other sorting algorithms like Merge Sort and Heap Sort, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures, and in most real-world data. QuickSort can be implemented in different ways by changing the choice of pivot, so that the worst case rarely occurs for a given type of data. However, merge sort is generally considered better when data is huge and stored in external storage.

6. a) Write an algorithm to sort an array by insertion sort. If the array contains numbers only then what can you do to reduced computations? [WBUT 2018]
Answer:
1st Part: ***Refer to Question No. 2(2nd Part) of Short Answer Type Questions.***

2nd Part:
The below algorithm is slightly optimized version to avoid swapping key element in every iteration. Here, the key element will be swapped at the end of the iteration (step).
InsertionSort(arr[])


```
for j = 1 to arr.length
  key = arr[j]
  i = j - 1
  while i > 0 and arr[i] > key
    arr[i+1] = arr[i]
    i = i - 1
  arr[i+1] = key
```

b) Explain the working principle of Quick sort with suitable example and show it's time complexity.
Answer: [WBUT 2018]
1st Part: ***Refer to Question No. 7 of Short Answer Type Questions.***

2nd Part:
Quick sort time complexity:
Time taken by QuickSort in general can be written as following.

$$T(n) = T(k) + T(n-k-1) + \theta(n)$$

The first two terms are for two recursive calls; the last term is for the partition process. k is the number of elements which are smaller than pivot. The time taken by QuickSort depends upon the input array and partition strategy. Following are three cases.
Worst Case: The worst case occurs when the partition process always picks greatest or smallest element as pivot. If we consider above partition strategy where last element is always picked as pivot, the worst case would occur when the array is already sorted in increasing or decreasing order. Following is recurrence for worst case.
$T(n) = T(\theta) + T(n-1) + \theta(n)$
which is equivalent to
$T(n) = T(n-1) + \theta(n)$
The solution of above recurrence is $O(n^2)$.

Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.
$T(n) = 2T(n/2) + \theta(n)$
The solution of above recurrence is O(nLogn).

Average Case:
To do average case analysis, we need to consider all possible permutation of array and calculate time taken by every permutation which doesn't look easy.
We can get an idea of average case by considering the case when partition puts O(n/9) elements in one set and O(9n/10) elements in other set. Following is recurrence for this case.

$$T(n) = T(n/9) + T(9n/10) + \theta(n)$$

Solution of above recurrence is also O(nLogn).

7. Write short notes on the following: [WBUT 2008, 2010, 2018]

a) Merge sort [WBUT 2008, 2012, 2015, 2017]
b) Heap sort [WBUT 2012, 2016, 2018]
c) Radix sort [WBUT 2015]
d) Binary Search [WBUT 2017]
e) File Structure [WBUT 2019]
f) Quick sort

Answer:

a) Merge sort:

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) Heap sort:

A heap is an "almost complete binary tree" (i.e. it is a complete binary tree possibly missing some of the right-most entries in the bottom level). Therefore one may implement the heap using a one-dimensional array. This may be done by numbering the nodes of the binary tree 0 through N-1 beginning with the root and moving top to bottom (lower-numbered level to higher-numbered level) and left to right. Recall there is one node at the 0 level, 2 nodes at the one level, 4 nodes at the two level, ... k nodes at the k level.

One may easily demonstrate that the left child of the node numbered k in the array is node 2*k+1 and the right child is 2*k+2. The parent of node k is (k-1)/2.

Consider the following integers, stored in a one-dimentional array, which are to be stored in a heap, H.

index	0	1	2	3	4	5	6
value	17	8	6	25	2	10	4

We will now show how the input array can be simultaneously used as the heap; as the heap is built by adding input data items.

The first item in the list, 17, is the first item to be added to the heap and so should be placed at the root of the binary tree that is the heap, H.

After placing 17 into the heap we have exactly one item in the heap. Therefore we consider the first item of the array to be the heap and items 1 through 6 to be the rest of the input data.

After we add the next two items, 8 and 6, we have a heap that contains three items. At this point items 0 through 2 are the heap and items 3 through 6 are the remaining input data items in the array pictured above. So far the properties of a heap have been preserved with no special effort.

However, the next data item, 25, in its current position violates the heap (largest on top) property. This may be remedied by a process that we shall call "siftUp".

In *siftUp* we compare the current node with its parent. If the value of the current node is greater than its parent's value then we swap the current node and its parent in the array (and thus, in the heap). We continue with this compare and swap-with-parent operation until the new value sifts up to the top of the heap or it encounters a parent that is larger.

Applying this process to item value 25 yields the following heap/input array.

index	0	1	2	3	4	5	6
value	25	17	6	8	2	10	4

Finally the completed heap will look like the following:

index	0	1	2	3	4	5	6
value	25	17	10	8	2	6	4

Next, we may begin the process whereby we remove each element from the heap and place it into the sorted array.

The first element to go is the top one, 25. It is swapped with the last element in the heap yielding the following.

index	0	1	2	3	4	5	6
value	4	17	10	8	2	6	25

25 is now in its proper position (the last item in the sorted array) but the number now at the top of the heap, 4, is not in its proper place. It needs to be "sifted down" to where it belongs. (See the *siftDown* function described below.)

After sifting down 4 we have the following.

index	0	1	2	3	4	5	6
value	17	8	10	4	2	6	25

Now we have sorted exactly one item so position 6 in the array is a list of sorted items and positions 0 through 5 are the items remaining in the heap yet to be placed into the sorted list.

As each successive item in the heap is placed into the sorted output list, the sorted list will expand from the largest (rightmost) index toward 0 as the heap decreases in size toward 0.

The second item to be removed from the heap is 17. It is swapped with 6 yielding the following.

index	0	1	2	3	4	5	6
value	6	8	10	4	2	17	25

Now the sorted list is size two and consists of 17 and 25 in locations 5-6. The heap requires a siftDown to bring element 6 down to where it belongs. So 6 is compared to the largest of its two children, 8 and 10, and swapped with 10 since 10 > 6. Continuing in this manner we restore the heap property and are left with the following.

index	0	1	2	3	4	5	6
value	10	8	6	4	2	17	25

So now the *heap* extends from index 0 to index 4 and the *sorted list* extends from index 5 to index 6.

Eventually we will have sorted the entire array yielding the following.

index	0	1	2	3	4	5	6
value	2	4	6	8	10	17	25

c) 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

d) Binary Search:

Binary search offers comparatively lesser number of iterations than the linear search. In this method, the elements are not searched sequentially like linear search. Instead of this two compartments of the list are done. Each compartment will be having an equal number of items. The lower compartment contains approximately half the elements and this compartment is called lower half. The upper compartment containing upper elements is upper half. Then the given element is searched. The element may be found in the lower half or the upper half of the compartments. It creates two parts of the lists hence it is known as binary search.

Binary search is quicker than the linear search. However, it cannot be applied on unsorted data structure. Before applying binary search, the linear data structure must be sorted in either ascending or descending order. The binary search is based on the approach divide-and-conquer. In this method, the element is successfully searched. Otherwise the element is compared with the items of the two compartments, that is, it is compared with the items falling under the lower half or upper half compartment. It the expected item is less than the middle element, the lower half compartment is search. If it is larger than the middle, upper half compartment is searched.

c) File Structure:

File is a collection of records related to each other. The file size is limited by the size of memory and storage medium.

There are two important features of file:

1. File Activity
2. File Volatility

File activity specifies percent of actual records which proceed in a single run.

File volatility addresses the properties of record changes. It helps to increase the efficiency of disk design than tape.

Operations with a file

- Create a file - this also implies associating an identifier with the file being created. The identifier is subsequently used to unambiguously identify the file.
- Insert a record
- Search a record - whose one or more fields satisfies some specified criteria - exact match, range, relative rank (e.g. largest/smallest value of a field), or external property (e.g. the n-th record in an ordered file, etc.) The field(s) on which a search is made is sometimes referred to as key field.
- Delete a record - involves a search
- Modify a record - involves a search
- List records, usually by some specified order - involves successive searches.
- Delete a file

File Structures:

Sequential File: A very natural way to store a file is in the form of an array, or a linked list of the records. In these representations, the entire file may be traversed in a linear fashion. This file structure is called *sequential file*. It is simple to implement and can be economic in space.

On the negative side, most search operations in such a file are likely to be inefficient since searching requires traversing of the sequence of records according to the storage sequence. In case of large files, particularly when the file is in disk, such traversal times can be too long. If the structure used is an array (and not a linked list), and the records are of equal sizes, then searching can be made efficient by keeping the records sorted on one key field. In that case searching based on that key field can be carried out using binary search. But search based on any other field value cannot derive the benefit of binary search. Moreover, using an array for storing a file makes growth or shrinking of the file (due to insertions and deletions) inconvenient. To overcome this one may use some kind of search tree structure to store records instead of a simple sequential structure. However, even in a search tree we can assume an ordering among the records only on one of the fields.

Indexed Sequential File

To overcome the shortcomings of the sequential file structures, indexed sequential file structure is used. In this the records of the file are stored like in a sequential file. But in addition to this, some *index* of the records is (are) maintained. An index is a collection of (key_value, record_address) pairs. The index is represented using a separate data structure. A search for a record using a key field shall now be carried out in the index based on that key value. Once the index entry is located, the record_address part of the entry can be used to directly access the record. Due to smaller size of the index, it can be structured in a way that facilitates efficient search, e.g. sorted array (for binary search), hash table, search-trees, etc. Also, for the same file several indexes using different key field can be maintained to help search based on different key fields. Since an index can contain many entries (as many as the number of records in the file), structuring them appropriately to facilitate an efficient search is important.

f) Quick sort: ***Refer to Question No. 6(b) of Long Answer Type Questions.***

QUESTION 2015

Group – A

(Multiple Choice Type Questions)

1. Choose the correct answers for the following:

i) A full binary tree with n leaves contains

a) n nodes b) log n nodes ✓c) 2n – 1 nodes d) 2n nodes

ii) _____ traversal of binary search tree gives the sorted list in ascending order.

✓a) In-order b) Post-order c) Pre-order d) All of these

iii) Recursion may be implemented by

a) linked list ✓b) stack c) queue d) deque

iv) In a max heap, both the addition and deletion operations can be performed in time

a) $O(\log n)$ ✓b) $O(n \log n)$ c) $O(n)$ d) $O(n^2)$

v) In a heap, the right child of a nod in position 10 will be in position

a) 20 ✓b) 21 c) 9 d) 5

vi) If $f(n) = 100n\log_2 n + 500n^4 + 0.52^n$ then $f(n)$

a) $O(n^4)$ b) $O(n\log_2 n)$ ✓c) $O(2^n)$ d) none of these

vii) If we evaluate the following postfix expression: 23, 5, 7, *, –, 12, + then the result will be

a) 12 ✓b) 0 c) –12 d) 35

viii) In external sorting technique all data reside in

a) primary memory ✓b) secondary memory

c) both (a) and (b) d) none of these

ix) Which is the correct notation to delete the last node P from a doubly linked list? (prev is the pointer pointing to the previous node and next is the pointer pointing to the next node)

a) P–>next=NULL b) P=NULL

✓c) P–>prev–>next=NULL d) None of these

x) Ratio of number of items in hash table to the table size is called

✓a) load factor b) item factor c) balanced factor d) all of these

xi) Reverse polish notation is also called

✓a) postfix b) prefix c) infix d) undefined

xii) Which of the following case does not exist in complexity theory?
a) Best case b) Worst case c) Average case ✓d) Null case

Group – B
(Short Answer Type Questions)

2. What is hashing? Why is it required? What is the time complexity of hashing? Mention different types of hashing techniques. What do you mean by re-hashing?
See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 1.

3. Write down the algorithm to implement PUSH and POP operation using stack implemented using array. Justify if a stack is an abstract data type.
See Topic: LINKED LIST, Short Answer Type Question No. 3.

4. Write a C function to find out inorder successor of the root of a binary tree.
See Topic: TREES, Short Answer Type Question No. 10.

5. Explain the types of Deque with suitable example. What are the advantages and disadvantages of linked list over the array?
1st Part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 7.
2nd Part: See Topic: LINKED LIST, Short Answer Type Question No. 5.

6. Construct a B-tree of order 5 with the following elements:
11, 72, 61, 20, 110, 40, 80, 130, 100, 42, 191, 92, 181, 242, 32, 122, 140, 162.
See Topic: TREES, Short Answer Type Question No. 3.

Group – C
(Long Answer Type Questions)

7. a) What is circular queue? Write down the algorithm to delete elements from a circular queue.
b) How does the AVL tree differ from a binary search tree? Construct an AVL tree by using following keys: 8, 12, 9, 11, 7, 6, 14, 10.
c) Explain the principle operation of quick sort with a suitable example. What is the best case complexity of insertion sort?
a) 1st part: See Topic: STACKS & QUEUES, Long Answer Type Question No. 3(a).
2nd part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 8.
b) See Topic: TREES, Short Answer Type Question No. 11.
c) See Topic: SORTING, Short Answer Type Question No. 7.

8. a) Preorder and inorder sequence are given for a Binary tree. Reconstruct the Binary Tree.
Inorder: A B D G H E I C F J K
Preorder: G D H B E I A C J F K
b) Write a tail recursive algorithm to find out factorial of a number. What are the disadvantages of recursion?
c) Write an algorithm to delete a node from a Doubly Linked List.
a) See Topic: TREES, Short Answer Type Question No. 12.
b) See Topic: LINKED LIST, Short Answer Type Question No. 12.
c) See Topic: LINKED LIST, Long Answer Type Question No. 3(a).

9. a) An infix expression is given. Find out the equivalent postfix expression using operator stack and show the output string stepwise.
Expression: (A–B–C)*(D+E/F)/((G–I)*J)$K
where A$B signifies AB (apply standard precedence rule).
b) What is External sorting? Write down the algorithm for selection sort. Derive the time complexities.
c) What is pendent vertex of a graph? Explain Depth First Search with suitable example.
a) See Topic: STACKS & QUEUES, Short Answer Type Question No. 9.
b) 1st part: See Topic: SORTING AND SEARCHING, Short Answer Type Question No. 8.
2nd & 3rd part: See Topic: SORTING, Short Answer Type Question No. 5.
c) 1st part: See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 4.
2nd part: See Topic: SORTING, Short Answer Type Question No. 1.

10. a) If E and I denote the external and internal path length of a binary tree and n denotes the number of internal nodes then show that E = I + 2*n.
b) Write down the characteristics of a B-tree.
c) Write an algorithm which will reverse a singly linked list.
a) & b) See Topic: TREES, Long Answer Type Question No. 5(a) & (b).
c) See Topic: LINKED LIST, Long Answer Type Question No. 1.

11. Write short notes on any three of the following:
a) Heap Sort
b) Collision Resolution
c) Threaded Binary Tree
d) Binary Search
e) Priority queue
a) See Topic: SORTING, Long Answer Type Question No. 7(b).
b) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 7(c).
c) See Topic: TREES, Long Answer Type Question No. 7(c).
d) See Topic: SORTING, Long Answer Type Question No. 7(d).
e) See Topic: STACKS & QUEUES, Long Answer Type Question No. 5.

QUESTION 2016

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:
i) A graph in which all nodes are of equal degree is known as
a) Multi graph b) Non-regular graph
✓c) Regular graph d) Complete graph

ii) Complexity is expressed in OOnotation in
a) lower bound ✓b) upper bound
c) between (a) & (b) d) none of these

iii) A binary tree T has n leaf nodes. The no. of nodes of degree 2 in T is
a) $\log_2 n$ ✓b) $n-1$ c) n d) 2^n

iv) Which of the following need not be a binary tree?
a) Search tree b) Heap c) AVL ✓d) B-tree

v) Linked lists are not suitable for implementing
a) insertion sort ✓b) Binary Search
c) Radix sort d) Polynomial manipulation

vi) Queue can be used to implement
✓a) Radix sort b) Quick sort c) Recursion d) Depth First Search

vii) Linked list is a
a) Linear data structure b) Dynamic data structure
✓c) Self-referential data structure d) All of these

viii) The evaluation of the postfix expression 3, 5, 7, *, +, 12, % is
✓a) 2 b) 3 c) 0 d) 3.17

ix) In a circular queue, which of the following statements indicates that the queue contains a single element?
a) front = rear + 1 b) front = O, rear = $n-1$
c) front = rear # – 1 d) both (a) and (b)

Answer: front = rear not equals to NULL

x) A binary tree in which if all its levels except possibly the last, have the maximum number of nodes and all the nodes at the last level appear as far left as possible, is known as
✓a) Full binary tree b) AVL tree
c) Threaded tree d) Complete binary tree

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. 13.

3. If $T(n)=15n^3+n^2+4$, then prove that $T(n)=\Theta(n^3)$, find the values of n_0, c_1 and c_2.
See Topic: SORTING, Short Answer Type Question No. 9.

4. Prove that, for any non-empty binary tree T, if n_0 is the number of leaves and n_2 be the number of nodes of degree 2, then $n_0 = n_2 + 1$
See Topic: TREES, Short Answer Type Question No. 13.

5. Explain with suitable example the principle of operation of Quick sort.
See Topic: SORTING, Short Answer Type Question No. 7(1st Part).

6. Construct a max heap tree out of the following set L of elements. Illustrate the step by step process of insertion and heap reconstruction before the final heap is obtained.
$L=\{D,B,G,E,A,H,C,F\}$
See Topic: TREES, Short Answer Type Question No. 14.

Group – C
(Long Answer Type Questions)

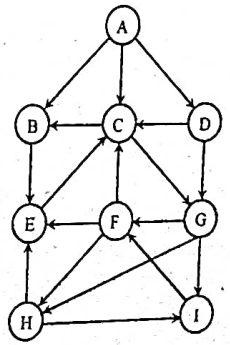
7. a) Write an algorithm to insert a node at the kth position of a doubly linked list.
b) Write algorithms to insert and delete an element from a circular queue implemented as an array.
c) Convert the following infix expression to its posfix equivalent.
$(A+B)*D+E/(F+A*D)+C$
d) What is the advantage of implementing a stack using a linked list instead of using an array?
a) See Topic: LINKED LIST, Short Answer Type Question No. 14.
b) & c) See Topic: STACKS & QUEUES, Long Answer Type Question No. 4(a) & (b).
d) See Topic: LINKED LIST, Short Answer Type Question No. 2.

8. a) What do you mean by external sorting?
b) Write the algorithm of merge sort and explain its time complexity.
c) Write the algorithm of binary search and explain its strategy in brief.
a) See Topic: SORTING, Short Answer Type Question No. 8.
b) See Topic: SORTING, Short Answer Type Question No. 10.
c) See Topic: SORTING, Long Answer Type Question No. 2.

9. a) Write a procedure for deletion of any node from Binary Search Tree.
b) Construct the binary tree give the array representation of the same.

B	C	G	E		Y	D						Z
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]

c) Consider the following graph. Find it BFS traversal starting from A.



a) & b) See Topic: TREES, Long Answer Type Question No. 6(a) & (b).
c) See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 5.

10. a) Consider a hash table of size 7 with hash function $h(k) = k \bmod 7$ draw the hash table that will result after inserting the following values: 19, 26, 13, 48, 17 (in the given order) for each of the three scenarios below:
(i) When collisions are handled by separate chaining method.
(ii) When collisions are handled by linear probing method.
(iii) When collisions are handled by double hashing method using a second hash function $h'(k) = 5 - (k \bmod 5)$
b) What is B tree? Write down the characteristics of a B tree.
c) Write an algorithm which will reverse a singly linked list.
a) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 5.
b) See Topic: TREES, Short Answer Type Question No. 15.
c) See Topic: LINKED LIST, Long Answer Type Question No. 1.

11. Write short notes on any three of the following:
a) Threaded Binary Tree
b) Spanning Tree
c) Radix Sort
d) AVL Tree
e) DAG
a) See Topic: TREES, Long Answer Type Question No. 7(c).
b) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 7(d).
c) See Topic: SORTING, Long Answer Type Question No. 7(c).
d) See Topic: TREES, Long Answer Type Question No. 7(b).
e) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 7(e).

QUESTION 2017

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for the following:
i) The best case complexity of insertion sort is
a) $O(n^2)$ b) $O(n \log n)$ c) $O(n^3)$ ✓d) $O(n)$

ii) Binary search uses
a) divide and reduce strategy
✓b) divide and conquer strategy
c) heuristic search
d) both (a) and (b)

iii) The Postfix expression for the infix expression $a*(b+c)/e-f$ is
✓a) $abc+*e/f-$ b) $abc+e/f-*$ c) $abc+e/*f-$ d) none of these

iv) 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) 22112 b) 22122 c) 21221 d) 1222

v) Total nodes in a 2-tree (Strictly binary tree) with 30 leaves will be
a) 60 b) 58 ✓c) 59 d) 57

vi) The adjacency matrix of an undirected graph is
a) Unit matrix
b) Asymmetric matrix
✓c) Symmetric matrix
d) none of these

vii) Queue can be used to implement
a) Radix sort ✓b) Quicksort c) Recursion d) Depth First Search

viii) Linked lists are not suitable for implementing
a) Insertion sort
✓b) Binary search
c) Radix sort
d) Polynomial manipulation

ix) Two main measures for the efficiency of an algorithm are
a) Processor and memory
b) Complexity and capacity
✓c) Time and space
d) Data and space

x) Which amongst the following is related to hashing?
a) Balance factor
✓b) Chaining
c) Inverse traversal
d) Load factor

Group – B
(Short Answer Type Questions)

2. What is Priority queue? Explain Tail recursion in brief.
1st Part See Topic: STACKS & QUEUES, Short Answer Type Question No. 5(b).
2nd part See Topic: LINKED LIST, Short Answer Type Question No. 1.

3. The inorder and preorder tree traversals are given. Draw the binary tree.
Inorder: ABCDEFGHI
Preorder: FBADCEGIH.
See Topic: TREES, Short Answer Type Question No. 16.

4. Evaluate the following Postfix expression by showing the operand stack and intermediate output.
Expression: 7, 3, 4, *, –,3, 9, 1, +, /, *,11, $, 4, +
See Topic: STACKS & QUEUES, Short Answer Type Question No. 10.

5. Explain the types of Dequeue with suitable example. What are the advantages and disadvantages of linked list over the array?

1st Part See Topic: STACKS & QUEUES, Short Answer Type Question No.7.
2nd part See Topic: LINKED LIST, Short Answer Type Question No. 5.

6. Describe Prim's minimal spanning tree algorithm.

See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 6.

Group – C
(Long Answer Type Questions)

7. a) What are asymptotic notations? Explain each notation with example and diagram.
b) Write an algorithm to insert an item after a particular key element in doubly linked list. Write the search algorithm separately for the same.

a) See Topic: INTRODUCTION, Long Answer Type Question No. 3.
b) See Topic: LINKED LIST, Long Answer Type Question No. 4.

8. a) What do you mean by in place sorting?
b) Write the algorithm of quicksort and explain its worst case, best case and average case time complexity. 7
c) Write the algorithm of binary search and explain its strategy in brief.

a) & b) See Topic: SORTING, Long Answer Type Question No. 5(a) & (b).
c) See Topic: SORTING, Long Answer Type Question No. 2.

9. a) If E and I denotes the external and internal path length of a binary tree and n denotes the number of internal nodes then show that $E = I + 2 * n$.
b) What is B tree? Write down the characteristics of a B tree.
c) What is pendent vertex of a graph? Explain Depth First Search with suitable example.

a) See Topic: TREES, Long Answer Type Question No. 7.
b) See Topic: TREES, Short Answer Type Question No. 15.
c) 1st part See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 4.
2nd part See Topic: SORTING, Short Answer Type Question No. 1.

10. a) Insert the following keys into a B tree of order 3:

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

b) What is Hashing? Briefly explain the different commonly used hashing functions.

a) See Topic: TREES, Short Answer Type Question No. 5(1st Part).
b) 1st part See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 2.
2nd part See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 1(c).

11. Write short notes on any *three* of the following:
a) Heap Sort
b) Collision Resolution
c) Threaded Binary Tree
d) File Structure
e) Sparse Matrix

a) See Topic: SORTING, Long Answer Type Question No. 7(b).
b) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 7(c).
c) See Topic: TREES, Long Answer Type Question No. 7(c).
d) See Topic: SORTING, Long Answer Type Question No. 7(e).
e) See Topic: STACKS & QUEUES, Long Answer Type Question No. 2.

QUESTION 2018

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for any *ten* of the following

i) Maximum possible height of an AVL Tree with 7 nodes is
✓a) 3 b) 4 c) 5 d) 6

ii) Worst case time complexity of the heap sort algorithm is
a) $O(n\log_2 n)$ ✓b) $O(n\lg n)$ c) $O(n^2)$ d) $O(n^3)$

iii) The prefix notation is also known as
✓a) Polish notation b) Reverse Polish notation
c) Both (a) and (b) d) None of these

iv) Linked lists are not suitable for
a) Stack b) AVL tree c) Deque ✓d) Binary search

v) Structure in C is an
✓a) user defined data type b) ADT
c) Both (a) and (b) d) None of these

vi) What is a suitable and efficient data structure to construct a tree?
a) Linked list b) Stack c) Queue ✓d) None of these

vii) To represent hierarchical relationship between elements, which data structure is suitable?
a) Deque b) Priority ✓c) Tree d) All of these

viii) Which of the following data structure is not a linear data structure?
a) Arrays b) Linked lists ✓c) Both of these d) None of these

ix) Adjacency matrix of a digraph is
a) $O(n)$ b) $O(\log_2 n)$ c) $O(2^n)$ d) $O(n\log_2 n)$
Answer: None of these

x) Which of the following cases do not exist in the complexity theory?
a) Best case b) Average case ✓c) Null case d) Worst case

xi) The complexity of merge sort algorithm is
a) $O(n)$ b) $O(\log n)$ c) $O(n^2)$ ✓d) $O(n \log n)$

xii) The height difference of any node in an AVL tree is
✓a) –1, 0, 1 b) –2, 0, 1 c) –2, 0, 2 d) –1, 0, 2

Group – B
(Short Answer Type Questions)

2. Convert the following into postfix:
(A+B) * C+D / (E+F*G) – H

See Topic: STACKS & QUEUES, Short Answer Type Question No. 11.

3. Compare between insertion sort and selection sort. Write an insertion sort algorithm.
See Topic: SORTING, Short Answer Type Question No. 2.

4. Define Big oh Notation. What is stack? Why is this called LIFO?
1st Part: See Topic: INTRODUCTION, Short Answer Type Question No. 1.
Rest Part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 2.

5. What is Hashing? What is Linear Probing? What are the functions of hashing?
See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 7.

6. Write an algorithm to delete a node from a singly linked list.
See Topic: LINKED LIST, Short Answer Type Question No. 15.

Group – C
(Long Answer Type Questions)

7. a) Write an algorithm to sort an array by insertion sort. If the array contains numbers only then what can you do to reduced computations?
b) Write an algorithm to insert an element at the end of a linked list pointed by a tail pointer, not a head pointer.
a) See Topic: SORTING, Long Answer Type Question No. 6(a).
b) See Topic: LINKED LIST, Short Answer Type Question No. 10.

8. a) The inorder and preorder sequences are given below, create the postorder sequence.
Inorder: A, B, C, D, E, F, G, H, I
Preorder: F, B, A, D, C, E, G, I, H
b) Evaluate the POSTFIX notation by showing the OPERAND STACK and intermediate output:
9, 4, 6, *, +, 12, 7, 3, –, *, /, 17, $, 24, –
Here, A$B indicates A^B.
c) Explain different Asymptotic Notations with diagrams.
a) See Topic: TREES, Short Answer Type Question No. 17.
b) See Topic: STACKS & QUEUES, Short Answer Type Question No. 12.
c) See Topic: INTRODUCTION, Long Answer Type Question No. 3.

9. a) Define B-Tree with suitable example
b) Construct a Binary Tree from the given array representation:

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]
B	C	G	E		Y	D						Z

c) Construct an AVL tree by using following elements:
8, 12, 9, 11, 7, 6, 14, 10

a) See Topic: TREES, Short Answer Type Question No. 18.
b) See Topic: TREES, Long Answer Type Question No. 6(b).
c) See Topic: TREES, Short Answer Type Question No. 11(2nd Part).

10. a) What is Hashing?
See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 2(a).

b) Considered a hash table of size 7 with h(k) = k mod 7. Draw the hash table after inserting the following values 19, 26, 13, 48, 17 (in the given order) for each of three scenarios below:
i) When collisions are handled by separate changing method
ii) When collisions are handled by linear probing method
iii) When collisions are handled by double hashing method using a second hash function:
h'(k) = 5– (k mod 5)
See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 5.

c) Explain the working principle of Quick sort with suitable example and show it's time complexity.
See Topic: SORTING, Long Answer Type Question No. 6(b).

11. Write short notes on any three of the following:
a) B+ Tree
b) Merge Sort
c) Circular Linked List
d) Radix Sort
e) Binary Tree
a) See Topic: TREES, Long Answer Type Question No. 7(d).
b) See Topic: SORTING, Long Answer Type Question No. 7(a).
c) See Topic: LINKED LIST, Long Answer Type Question No. 5(a).
d) See Topic: SORTING, Long Answer Type Question No. 7(c).
e) See Topic: TREES, Long Answer Type Question No. 7(e).

QUESTION 2019

Group – A
(Multiple Choice Type Questions)

1. Choose the correct alternatives for any *ten* of the following:
i) Maximum possible height of an AVL tree with 7 nodes is
✓a) 3 b) 4 c) 5 d) 6

ii) Worst case time complexity of heap sort algorithm is
a) O (nlog2 n) ✓b) O (n log n) c) O (n^2) d) O(n3)

iii) The prefix notation is also known as
✓a) polish notation b) reverse polish notation
c) reverse polish notation d) None of these

iv) Linked lists are not suitable for
a) stack ✓b) AVL tree c) deque d) binary search

v) Structure in C is
✓a) user defined data type b) ADT
c) both (a) and (b) d) none of these

vi) What is suitable efficient data structure to construct a tree?
a) Linked list b) Stack c) Queue ✓d) None of these

vii) To represent hierarchical relationship between elements, which data structure is suitable?
a) Deque b) Priority ✓c) Tree d) All of these

viii) Which of the followings data structure is not a linear data structure?
a) Arrays b) linked list ✓c) Tree d) Stack

ix) Adjacency matrix of a digraph is
✓a) O (n) b) O (log n) c) O (2n) d) O(n log n)

x) Which of the following case does not exist in the complexity theory?
a) Best case b) Average case ✓c) Null Case d) Worst case

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

xii) The height difference of any node in an AVL tree is
✓a) – 1, 0, 1 b) – 2, 0, 1 c) - 2, 0, 2 d) – 1, 0, 2

Group – B
(Short Answer Type Questions)

2. Convert the following into postfix:

(A+B) * C+D / (E+F*G) – H

See Topic: STACKS & QUEUES, Short Answer Type Question No. 11.

3. Compare between insertion sort and selection sort. Write the insertion sort algorithm.

See Topic: SORTING, Short Answer Type Question No. 2.

4. Define Big oh Notation. What is Stack? Why is this called LIFO?

1st Part: See Topic: INTRODUCTION, Short Answer Type Question No. 1.
Rest Part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 2.

5. What is Hashing? What is Linear Probing? What are the functions of hashing?

See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 7.

6. Write an algorithm to delete a node from a singly linked list.

See Topic: LINKED LIST, Short Answer Type Question No. 15.

Group – C
(Long Answer Type Questions)

7. a) Represent the following polynomial by linked list (show the diagram only):

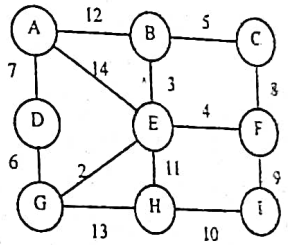
$9x^5 + 3x^3 - 8x + 15$

b) What is circular linked list? What are the advantages and disadvantages of linked list over an array?

c) Write algorithm for inserting and deleting elements in a queue.

a) & b) See Topic: LINKED LIST, Short Answer Type Question No. 16.
2nd Part: See Topic: LINKED LIST, Short Answer Type Question No. 5.
c) See Topic: STACKS & QUEUES, Short Answer Type Question No. 13.

8. a) What is a graph ? Find out the shortest path between all pairs of nodes in the given graph by Kruskal's algorithm.



b) Write down the DFS algorithm.

c) Define a directed graph. Provide an example.

a) & c) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 6.
b) See Topic: SORTING, Short Answer Type Question No. 1.

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

b) Write algorithm to delete any node from a binary search tree.

c) What is binary tree? Construct a binary tree using the in-order and pre-order traversal of the nodes given below (Indicate all intermediate steps).

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

a) 1st Part: See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 1(a).
2nd Part: See Topic: GRAPH THEORY AND HASHING, Short Answer Type Question No. 8.
b) See Topic: TREES, Long Answer Type Question No. 6(a).
c) 1st Part: See Topic: TREES, Short Answer Type Question No. 19.
2nd Part: See Topic: TREES, Short Answer Type Question No. 8.

10. a) Construct the following Question of characters where Queue is a circular array which is allocated six memory cells.

FRONT = 2 REAR = 4 QUEUE: _ , A, C, D, _, _

Describe the Queue as the following operations take place:

i) F is added to the Queue
ii) Two letters are deleted from the queue
iii) K, L, M are added to the Queue
iv) Two letters are deleted from the Queue
v) R is added to the Queue
vi) One letter is deleted from the Queue

b) What is recursion? What are the difference between iteration and recursion? What are the applications of stack?

c) Write the push( ) and pop( ) functions for a stack after describing the Data-Structure clearly.

a) See Topic: STACKS & QUEUES, Short Answer Type Question No. 14.
b) 1st & 2nd Part: See Topic: LINKED LIST, Short Answer Type Question No. 17.
3rd Part: See Topic: STACKS & QUEUES, Short Answer Type Question No. 15.
c) See Topic: LINKED LIST, Short Answer Type Question No. 3.

11. Write the short notes any *three* of the following:

a) AVL tree
b) Quick sort
c) Binary search tree
d) B-Tree
e) BFS

a) See Topic: TREES, Long Answer Type Question No. 7(b).
b) See Topic: SORTING, Long Answer Type Question No. 7(f).
c) See Topic: TREES, Long Answer Type Question No. 7(f).
d) See Topic: TREES, Long Answer Type Question No. 7(a).
e) See Topic: GRAPH THEORY AND HASHING, Long Answer Type Question No. 7(f).