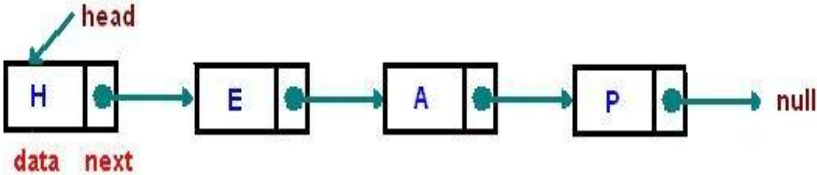
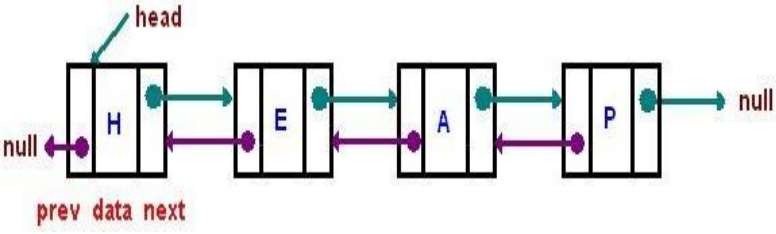
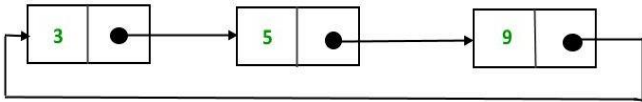
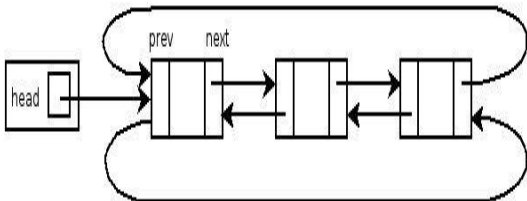


# Question Bank for FDS(Practical/OR Examination )

S. No.	Question
1	<b>What is a data structure?</b> <ul style="list-style-type: none"><li>• A data structure is a method for organizing and storing data which would allow efficient data retrieval and usage.</li><li>• A data structure is a way of organizing data that considers not only the items stored, but also their relationships to each other.</li></ul>
2	<b>Why do we need data structures?</b> <ul style="list-style-type: none"><li>• Data structures allow us to achieve an important goal: component reuse.</li><li>• Once data structure has been implemented, it can be used again and again in various applications.</li></ul>
3	<b>List some common data structures.</b> <ul style="list-style-type: none"><li>• Stacks</li><li>• Queues</li><li>• Lists</li><li>• Trees</li><li>• Graphs</li><li>• Tables</li></ul>
4	<b>How data structures are classified?</b> <p>Data structures are classified into two categories based on how the data items are operated:</p> <ol style="list-style-type: none"><li>i. Primitive data structure</li><li>ii. Non-Primitive data structure</li></ol>

	a. Linear data structure b. Non-linear data structure												
5	<b>Differentiate linear and non-linear data structure.</b> <table><tr><th>Linear data structure</th><th>Non-linear data structure</th></tr><tr><td>Data are arranged in linear or sequential manner</td><td>Data are not arranged in linear manner</td></tr><tr><td>Every items is related to its previous and next item</td><td>Every item is attached with many other items</td></tr><tr><td>Data items can be traversed in a single run.</td><td>Data items cannot be traversed in a single run.</td></tr><tr><td>Implementation is easy</td><td>Implementation is difficult.</td></tr><tr><td>Example: array, stack, queue, linked list</td><td>Example: tree, graph</td></tr></table>	Linear data structure	Non-linear data structure	Data are arranged in linear or sequential manner	Data are not arranged in linear manner	Every items is related to its previous and next item	Every item is attached with many other items	Data items can be traversed in a single run.	Data items cannot be traversed in a single run.	Implementation is easy	Implementation is difficult.	Example: array, stack, queue, linked list	Example: tree, graph
Linear data structure	Non-linear data structure												
Data are arranged in linear or sequential manner	Data are not arranged in linear manner												
Every items is related to its previous and next item	Every item is attached with many other items												
Data items can be traversed in a single run.	Data items cannot be traversed in a single run.												
Implementation is easy	Implementation is difficult.												
Example: array, stack, queue, linked list	Example: tree, graph												
6	<b>Define ADT (Abstract Data Type)</b> <p>An abstract data type (ADT) is a set of operations and mathematical abstractions , which can be viewed as how the set of operations is implemented. Objects like lists, sets and graphs, along with their operation, can be viewed as abstract data types, just as integers, real numbers and Booleans.</p>												
7	<b>Mention the features of ADT.</b> <p>a. Modularity i. Divide program into small functions ii. Easy to debug and maintain iii. Easy to modify b. Reuse i. Define some operations only once and reuse them in future c. Easy to change the implementation</p>												
8	<b>Define List ADT</b> <p>A list is a sequence of zero or more elements of a giventype. The list is represented as sequence of elements separated by comma.A1, A2, A3.....AN Where N&gt;0 and A is of type element</p>												
9	<b>What are the ways of implementing linked list?</b> <p>The list can be implemented in the following ways: i. Array implementation ii. Linked-list implementation</p>												

	iii. Cursor implementation
10	<p><b>What are the types of linked lists?</b></p> <p>There are three types</p> <ol style="list-style-type: none"> <li>Singly linked list</li> <li>Doubly linked list</li> <li>Circularly linked list</li> </ol>
11	<p><b>How the singly linked lists can be represented?</b></p>  <p>Each node has two elements</p> <ol style="list-style-type: none"> <li>Data</li> <li>Next</li> </ol>
12	<p><b>How the doubly linked list can be represented?</b></p>  <p>Doubly linked list is a collection of nodes where nodes are connected by forward and backward links.</p> <p>Each node has three fields:</p> <ol style="list-style-type: none"> <li>Address of previous node</li> <li>Data</li> <li>Address of next node.</li> </ol>
13	<p><b>What are benefits of ADT?</b></p> <ol style="list-style-type: none"> <li>Code is easier to understand</li> <li>Implementation of ADT can be changed without requiring changes to the program that uses the ADT</li> </ol>
14	<p><b>When singly linked list can be represented as circular linked list?</b></p> <p>In a singly linked list, all the nodes are connected with forward links to the next nodes in the list. The last node has a next field, NULL. In order to implement the circularly linked</p>

	<p>lists from singly linked lists, the last node's next field is connected to the first node.</p> 
15	<p><b>When doubly linked list can be represented as circular linked list?</b></p> <p>In a doubly linked list, all nodes are connected with forward and backward links to the next and previous nodes respectively. In order to implement circular linked lists from doubly linked lists, the first node's previous field is connected to the last node and the last node's next field is connected to the first node.</p> 
16	<p><b>Where cursor implementation can be used?</b></p> <p>The cursor implementation of lists is used by many languages such as BASIC and FORTRAN that do not support pointers. The two important features of the cursor implementation of linked are as follows:</p> <ul style="list-style-type: none"> <li>• The data are stored in a collection of structures. Each structure contains data and a index to the next structure.</li> <li>• A new structure can be obtained from the system's global memory by a call to cursorSpace array.</li> </ul>
17	<p><b>List down the applications of List.</b></p> <ol style="list-style-type: none"> <li>Representation of polynomial ADT</li> <li>Used in radix and bubble sorting</li> <li>In a FAT file system, the metadata of a large file is organized as a linked list of FAT entries.</li> <li>Simple memory allocators use a free list of unused memory regions, basically a linked list with the list pointer inside the free memory itself.</li> </ol>
18	<p><b>What are the advantages of linked list?</b></p> <ol style="list-style-type: none"> <li>Save memory space and easy to maintain</li> <li>It is possible to retrieve the element at a particular index</li> <li>It is possible to traverse the list in the order of increasing index.</li> </ol>

	d. It is possible to change the element at a particular index to a different value, without affecting any other elements.						
19	<b>Mention the demerits of linked list</b> a. It is not possible to go backwards through the list b. Unable to jump to the beginning of list from the end.						
20	<b>The polynomial equation can be represented with linked list as follows:</b> <table border="1"><tr><td>Coefficient</td><td>Exponent</td><td>Next node link</td></tr><tr><td>struct polynomial</td><td></td><td></td></tr></table> <pre>{ int coefficient;int exponent;struct polynomial *next; };</pre>	Coefficient	Exponent	Next node link	struct polynomial		
Coefficient	Exponent	Next node link					
struct polynomial							
21	<b>What are the operations performed in list?</b> The following operations can be performed on a list i. Insertion a. Insert at beginning b. Insert at end c. Insert after specific node d. Insert before specific node ii. Deletion a. Delete at beginning b. Delete at end c. Delete after specific node d. Delete before specific node iii. Merging iv. Traversal						
22	<b>What are the merits and demerits of array implementation of lists?</b> Merits <ul style="list-style-type: none"><li>● Fast, random access of elements</li><li>● Memory efficient – very less amount of memory is required</li></ul> Demerits <ul style="list-style-type: none"><li>● Insertion and deletion operations are very slow since the elements should be moved.</li><li>● Redundant memory space – difficult to estimate the size of array.</li></ul>						
23	<b>What is a circular linked list?</b> A circular linked list is a special type of linked list that supports traversing from the end of the list to the beginning by making the last node point back to the head of the list.						

24	<b>What are the advantages in the array implementation of list?</b> <ol style="list-style-type: none"> <li>Print list operation can be carried out at the linear time</li> <li>Find Kth operation takes a constant time</li> </ol>
25	<b>What is the need for the header?</b> <p>Header of the linked list is the first element in the list and it stores the number of elements in the list. It points to the first data element of the list.</p>
26	<b>List three examples that uses linked list?</b> <ol style="list-style-type: none"> <li>Polynomial ADT</li> <li>Radix sort</li> <li>Multi lists</li> </ol>
27	<b>List out the different ways to implement the list?</b> <ol style="list-style-type: none"> <li>Array Based Implementation</li> <li>Linked list Implementation <ol style="list-style-type: none"> <li>Singly linked list</li> <li>Doubly linked list</li> <li>Cursor based linked list</li> </ol> </li> </ol>
28	<b>Write the routine for insertion operation of singly linked list.</b> <pre> Void Insert (ElementType X, List L, Position P) {     Position TmpCell; TmpCell=malloc(sizeof(struct Node));     if(TmpCell==NULL)         FatalError("Out of space!!!");     TmpCell-&gt;Element =X; TmpCell-&gt;Next=P-&gt;Next;P-&gt;Next=TmpCell; } </pre>
29	<b>Advantages of Array over Linked List.</b> <ol style="list-style-type: none"> <li>Array has a specific address for each element stored in it and thus we can access any memory directly.</li> <li>As we know the position of the middle element and other elements are easily accessible too, we can easily perform BINARY SEARCH in array.</li> </ol>
30	<b>Disadvantages of Array over Linked List.</b> <ol style="list-style-type: none"> <li>Total number of elements need to be mentioned or the memory allocation needs to be done at the time of array creation</li> <li>The size of array, once mentioned, cannot be increased in the program. If number of elements entered exceeds the size of the array ARRAY OVERFLOW EXCEPTION occurs.</li> </ol>

31	<b>Advantages of Linked List over Array.</b> <ol style="list-style-type: none"> <li>1. Size of the list doesn't need to be mentioned at the beginning of the program.</li> <li>2. As the linked list doesn't have a size limit, we can go on adding new nodes (elements) and increasing the size of the list to any extent.</li> </ol>
32	<b>Disadvantages of Linked List over Array.</b> <ol style="list-style-type: none"> <li>1. Nodes do not have their own address. Only the address of the first node is stored and in order to reach any node, we need to traverse the whole list from beginning to the desired node.</li> <li>2. As all Nodes don't have their particular address, BINARYSEARCH cannot be performed</li> </ol>
1	Explain the various operations of the list ADT with examples
2	Write the program for array implementation of lists
3	Write a C program for linked list implementation of list.
4	Explain the operations of singly linked lists
5	Explain the operations of doubly linked lists
6	Explain the operations of circularly linked lists
7	How polynomial manipulations are performed with lists? Explain the operations
8	Explain the steps involved in insertion and deletion into a singly and doubly linked list.

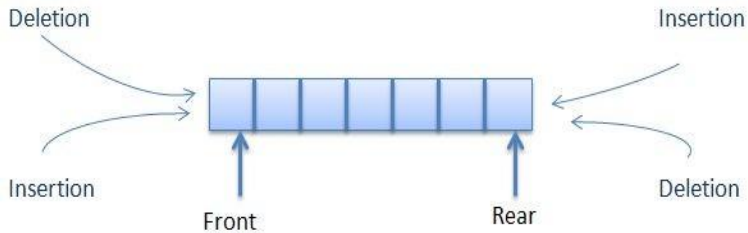
### LINEAR DATA STRUCTURES-STACKS,QUEUES

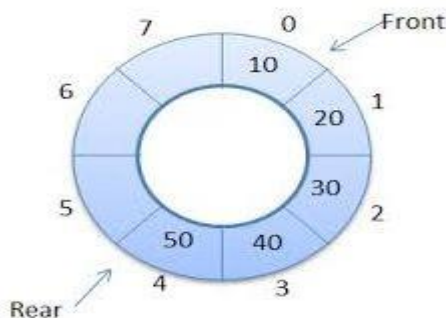
S. No.	Question
1	<b>Define Stack.</b> A stack is an ordered list in which all insertions and deletions are made at one end, called the top. It is an abstract data type and based on the principle of LIFO (Last In First Out).

2	<b>What are the operations of the stack?</b> a. CreateStack/ InitStack(Stack) – creates an empty stack b. Push(Item) – pushes an item on the top of the stack c. Pop(Item) – removes the top most element from the stack d. Top(Stack) – returns the first element from the stack e. IsEmpty(Stack) – returns true if the stack is empty
3	<b>Write the routine to push a element into a stack.</b> Push(Element X, Stack S) { if(IsFull(S) { Error(“Full Stack”); }else S→Array[++S→TopOfStack]=X; }
4	<b>How the operations performed on linked list implementation of stack?</b> a. Push and pop operations at the head of the list. b. New nodes should be inserted at the front of the list, so that they become the top of the stack. c. Nodes are removed from the front(top) of the stack.
5	<b>What are the applications of stack?</b> The following are the applications of stacks <ul style="list-style-type: none"> <li>• Evaluating arithmetic expressions</li> <li>• Balancing the parenthesis</li> <li>• Towers of Hanoi</li> <li>• Function calls Tree traversal</li> </ul>
6	<b>What are the methods to implement stack in C?</b> The methods to implement stacks are: <ul style="list-style-type: none"> <li>• Array based</li> <li>• Linked list based</li> </ul>
7	<b>How the stack is implemented by linked list?</b> It involves dynamically allocating memory space at runtime while performing stack operations. Since it consumes only that much amount of space is required for holding its data elements , it prevents wastage of memory space. <pre> struct stack { </pre>




	<pre> int element; struct stack *next; }*top; </pre>
8	<p><b>Write the routine to pop a element from a stack.</b></p> <pre> int pop() { if(top==NULL) { printf("\n Stack is empty.\n");getch();exit(1);} else { int temp; temp=top-&gt;element; top=top-&gt;next; return temp; }} </pre>
9	<p><b>Define queue.</b></p> <p>It is a linear data structure that maintains a list of elements such that insertion happens at rear end and deletion happens at front end. FIFO – First In First Out principle</p>
10	<p><b>What are the operations of a queue?</b></p> <p>The operations of a queue are</p> <ul style="list-style-type: none"> <li>• isEmpty()</li> <li>• isFull()</li> <li>• insert()</li> <li>• delete()</li> <li>• display()</li> </ul>
11	<p><b>Write the routine to insert a element onto a queue.</b></p> <pre> void insert(int element) { if(front==-1 ) { front = rear = front +1; queue[front] = element;return; } if(rear==99) { printf("Queue is full");getch(); return; } rear = rear +1; queue[rear]=element; } </pre>
12	<p><b>What are the types of queue?</b></p> <p>The following are the types of queue:</p> <ul style="list-style-type: none"> <li>• Double ended queue</li> <li>• Circular queue</li> <li>• Priority queue</li> </ul>
13	<p><b>Define double ended queue</b></p> <ul style="list-style-type: none"> <li>• It is a special type of queue that allows insertion and deletion of elements at both</li> </ul>

	<p>Ends.</p> <ul style="list-style-type: none"> <li>It is also termed as DEQUE.</li> </ul> 
14	<p><b>What are the methods to implement queue in C?</b></p> <p>The methods to implement queues are:</p> <ul style="list-style-type: none"> <li>Array based</li> <li>Linked list based</li> </ul>
15	<p><b>How the queue is implemented by linked list?</b></p> <ul style="list-style-type: none"> <li>It is based on the dynamic memory management techniques which allow allocation and De-allocation of memory space at runtime.</li> </ul> <p><b>Insert operation</b></p> <p>It involves the following subtasks:</p> <ol style="list-style-type: none"> <li>Reserving memory space of the size of a queue element in memory</li> <li>Storing the added value at the new location</li> <li>Linking the new element with existing queue</li> <li>Updating the <i>rear</i> pointer</li> </ol> <p><b>Delete operation</b></p> <p>It involves the following subtasks:</p> <ol style="list-style-type: none"> <li>Checking whether queue is empty</li> <li>Retrieving the front most element of the queue</li> <li>Updating the front pointer</li> <li>Returning the retrieved value</li> </ol>
16	<p><b>Write the routine to delete a element from a queue</b></p> <pre>int del() { int i;   if(front == NULL) /*checking whether the queue is empty*/   {return(-9999);} else   {i = front-&gt;element; front = front-&gt;next; return i;} }</pre>
17	<p><b>What are the applications of queue?</b></p> <p>The following are the areas in which queues are applicable</p> <ol style="list-style-type: none"> <li>Simulation</li> <li>Batch processing in an operating systems</li> <li>Multiprogramming platform systems</li> <li>Queuing theory</li> <li>Printer server routines</li> <li>Scheduling algorithms like disk scheduling , CPU scheduling</li> <li>I/O buffer requests</li> </ol>

18	<p><b>Define circular queue</b></p> <p>A Circular queue is a queue whose start and end locations are logically connected with each other. That means the start location comes after the end location.</p> 				
19	<p><b>What are push and pop operations?</b></p> <ul style="list-style-type: none"><li>• Push – adding an element to the top of stack</li><li>• Pop – removing or deleting an element from the top of stack</li></ul>				
20	<p><b>What are enqueue and dequeue operations?</b></p> <ul style="list-style-type: none"><li>• <b>Enqueue</b> - adding an element to the queue at the rear end</li></ul> <p>If the queue is not full, this function adds an element to the back of the queue, else it prints “<b>OverFlow</b>”.</p> <pre>void enqueue(int queue[], int element, int&amp; rear, int arraySize) { if(rear == arraySize)    // Queue is full    printf(“OverFlow\n”);else{        queue[rear] = element;    // Add the element to the back        rear++;    } }</pre> <ul style="list-style-type: none"><li>• <b>Dequeue</b> – removing or deleting an element from the queue at the front end</li></ul> <p>If the queue is not empty, this function removes the element from the front of the queue, else it prints “<b>UnderFlow</b>”.</p> <pre>void dequeue(int queue[], int&amp; front, int rear) {    if(front == rear)    // Queue is empty        printf(“UnderFlow\n”);    else {        queue[front] = 0;    // Delete the front element        front++;    } }</pre>				
21	<p><b>Distinguish between stack and queue.</b></p> <table><tr><th>STACK</th><th>QUEUE</th></tr><tr><td>Insertion and deletion are made at one end.</td><td>Insertion at one end rear and deletion at other end front.</td></tr></table>	STACK	QUEUE	Insertion and deletion are made at one end.	Insertion at one end rear and deletion at other end front.
STACK	QUEUE				
Insertion and deletion are made at one end.	Insertion at one end rear and deletion at other end front.				

	The element inserted last would be removed first. So LIFO structure.	The element inserted first would be removed first. So FIFO structure.	
	Full stack condition:  If(top==Maxsize)  Physically and Logically full stack	Full stack condition:  If(rear == Maxsize)  Logically full. Physically may or may not be full.	
22	<b>Convert the infix (a+b)*(c+d)/f into postfix &amp; prefix expression</b>  Postfix : a b + c d + * f /  Prefix : / * + a b + c d f		
23	<b>Write postfix from of the expression –A+B-C+D?</b>  A-B+C-D+		
24	<b>How do you test for an empty queue?</b> To test for an empty queue, we have to check whether READ=HEAD where REAR is a pointer pointing to the last node in a queue and HEAD is a pointer that pointer to the dummy header. In the case of array implementation of queue, the condition to be checked for an empty queue is READ<FRONT.		
25	<b>What are the postfix and prefix forms of the expression?</b> A+B*(C-D)/(P-R) Postfix form: ABCD-*PR-/ + Prefix form: +A/*B-CD-PR		
26	<b>Explain the usage of stack in recursive algorithm implementation?</b> In recursive algorithms, stack data structures is used to store the return address when a recursive call is encountered and also to store the values of all the parameters essential to the current state of the procedure.		
27	<b>Define priority queue with diagram and give the operations.</b> Priority queue is a data structure that allows at least the following two operations. 1. Insert-inserts an element at the end of the list called the rear. 2. DeleteMin-Finds, returns and removes the minimum element in the priority Queue.		

	 <p>Operations: Insert, DeleteMin</p>
28	<p><b>Give the applications of priority queues.</b></p> <p>There are three applications of priority queues</p> <ol style="list-style-type: none"> <li>1. External sorting.</li> <li>2. Greedy algorithm implementation.</li> <li>3. Discrete even simulation.</li> <li>4. Operating systems.</li> </ol>
29	<p><b>How do you test for an empty stack?</b></p> <p>To check if the stack is empty, we only need to check whether top and bottom are the same number.</p> <pre>bool stack_empty(stack S) //@requires is_stack(S); { return S-&gt;top == S-&gt;bottom; }</pre>
30	<p><b>What are the features of stacks?</b></p> <ul style="list-style-type: none"> <li>• Dynamic data structures</li> <li>• Do not have a fixed size</li> <li>• Do not consume a fixed amount of memory</li> <li>• Size of stack changes with each push() and pop() operation.</li> </ul> <p>Each push() and pop() operation increases and decreases the size of the stack by 1, respectively.</p>
31	<p><b>Write a routine for isEmpty condition of queue.</b></p> <p>If a queue is empty, this function returns 'true', else it returns 'false'.</p> <pre>bool isEmpty(int front, int rear) {return (front == rear); }</pre>
1	Explain Stack ADT and its operations
2	Explain array based implementation of stacks
3	Explain linked list implementation of stacks
4	Explain the applications of Stacks
5	Explain how to evaluate arithmetic expressions using stacks
6	Explain queue ADT
7	Explain array based implementation of queues
8	Explain linked list implementation of queues

9	Explain the applications of queues
10	Explain circular queue and its implementation
11	Explain double ended queue and its operations
12	Explain priority queue and its operations

### SEARCHING, SORTING

S. No.	Question
1	<p><b>Define sorting</b></p> <p>Sorting arranges the numerical and alphabetical data present in a list in a specific order or sequence. There are a number of sorting techniques available. The algorithms can be chosen based on the following factors</p> <ul style="list-style-type: none"> <li>● Size of the data structure</li> <li>● Algorithm efficiency</li> </ul> <p>Programmer's knowledge of the technique</p>
2	<p><b>Mention the types of sorting</b></p> <ul style="list-style-type: none"> <li>● Internal sorting</li> <li>● External sorting</li> </ul>
3	<p><b>What do you mean by internal and external sorting?</b></p> <p>An internal sort is any data sorting process that takes place entirely within the main memory of a computer. This is possible whenever the data to be sorted is small enough to all be held in the main memory.</p> <p>External sorting is a term for a class of sorting algorithms that can handle massive amounts of data. External sorting is required when the data being sorted do not fit into the main memory of a computing device (usually RAM) and instead they must reside in the slower external memory (usually a hard drive).</p>

4

**How the insertion sort is done with the array?**

It sorts a list of elements by inserting each successive element in the previously sorted Sub list.

Consider an array to be sorted  $A[1], A[2], \dots, A[n]$

- a. Pass 1:  $A[2]$  is compared with  $A[1]$  and placed in sorted order.
- b. Pass 2:  $A[3]$  is compared with both  $A[1]$  and  $A[2]$  and inserted at an appropriate place. This makes  $A[1], A[2], A[3]$  as a sorted sub array.
- c. Pass  $n-1$ :  $A[n]$  is compared with each element in the sub array

	A [1], A [2] ...A [n-1] and inserted at an appropriate position.
5	<b>Define hashing.</b> Hash function takes an identifier and computes the address of that identifier in the hash table using some function
6	<b>What is the need for hashing?</b> Hashing is used to perform insertions, deletions and find in constant average time.
7	<b>Define hash function?</b> Hash function takes an identifier and computes the address of that identifier in the hash table using some function.
8	<b>List out the different types of hashing functions?</b> The different types of hashing functions are, a. The division method b. The mind square method c. The folding method d. Multiplicative hashing e. Digit analysis
9	<b>What are the problems in hashing?</b> a. Collision b. Overflow
10	<b>What are the problems in hashing?</b> When two keys compute in to the same location or address in the hash table through any of the hashing function then it is termed collision.
11	<b>what is insertion sort? How many passes are required for the elements to be sorted ?</b> one of the simplest sorting algorithms is the insertion sort. Insertion sort consist of N-1 passes . For pass P=1 through N-1 , insertion sort ensures that the elements in positions 0 through P-1 are in sorted order .It makes use of the fact that elements in position 0 through P-1 are already known to be in sorted order .
12	<b>Write the function in C for insertion sort ?</b> <pre>void insertionsort(elementtype A[ ], int N) { int j, p; elementtype tmp; for(p=1 ; p &lt;N ;p++ ) { tmp = a[ p] ; for ( j=p ; j&gt;0 &amp;&amp; a [ j -1 ] &gt;tmp ;j--)a [ j ]=a [j-1 ] ; a [ j ] = tmp ; }}</pre>
13	<b>Who invented shellsort ? define it ?</b> Shellsort was invented by Donald Shell . It works by comparing element that are distant . The distance between the comparisons decreases as the algorithm runs until the last phase in which



	adjacent elements are compared . Hence it is referred as diminishing increment sort.												
14	<b>write the function in c for shellsort?</b> Void Shellsort(Elementtype A[ ],int N) { int i , j , increment ;elementtype tmp ; for(elementtype=N / 2;increment > 0;increment /= 2)For( i= increment ; i <N ; i ++) { tmp=A[ i ]; for( j=i; j>=increment; j -=increment)if(tmp< A[ j ]=A[j - increment]; A[ j ]=A[ j - increment];Else Break; A[ j ]=tmp; }} 												
15	<b>differentiate between merge sort and quick sort? Mergesort          quick sort</b> 1. Divide and conquer strategy          Divide and conquer strategy 2. Partition by position          Partition by value												
16	<b>Mention some methods for choosing the pivot element in quicksort?</b> 1. Choosing first element 2. Generate random number 3. Median of three												
17	<b>What are the three cases that arise during the left to right scan in quick sort?</b> 1. I and j cross each other 2. I and j do not cross each other 3. I and j points the same position												
18	<b>What is the need of external sorting?</b> External sorting is required where the input is too large to fit into memory. So external sorting Is necessary where the program is too large												
19	<b>What is sorting?</b> Sorting is the process of arranging the given items in a logical order. Sorting is an example where the analysis can be precisely performed.												
20	<b>What is mergesort?</b> The mergesort algorithm is a classic divide conquer strategy. The problem is divided into two arrays and merged into single array												
21	<b>Compare the various hashing techniques.</b> <table><tr><td>Technique</td><td></td><td>Load Factor</td></tr><tr><td>Separate chaining</td><td>-</td><td>close to 1</td></tr><tr><td>Open Addressing</td><td>-</td><td>should not exceed 0.5</td></tr><tr><td>Rehashing</td><td>-</td><td>reasonable load factor</td></tr></table>	Technique		Load Factor	Separate chaining	-	close to 1	Open Addressing	-	should not exceed 0.5	Rehashing	-	reasonable load factor
Technique		Load Factor											
Separate chaining	-	close to 1											
Open Addressing	-	should not exceed 0.5											
Rehashing	-	reasonable load factor											

22	<p><b>Define collision in hashing.</b></p> <p>When two different keys or identifiers compute into the same location or address in the hash table through any of the hashing functions, then it is termed Collision.</p>
23	<p><b>Define Double Hashing.</b></p> <p>Double Hashing is a collision-resolution technique used in open addressing category. In double hashing, we apply a second hash function to <math>x</math> and probe at a distance of <math>\text{hash}_2(x)</math>, <math>2\text{hash}_2(x)</math>, ..... , and so on.</p>
24	<p><b>What are applications of hashing?</b></p> <p>The applications of hashing are,</p> <ul style="list-style-type: none"> <li>• Compilers use hash table to keep track of declared variables on source code.</li> <li>• Hash table is useful for any graph theory problem, where the nodes have real names instead of numbers.</li> <li>• Hash tables are used in programs that play games.</li> <li>• Online spelling checkers use hashing.</li> </ul>
25	<p><b>What does internal sorting mean?</b></p> <p>Internal sorting is a process of sorting the data in the main memory</p>
26	<p><b>What are the various factors to be considered in deciding a sorting algorithm?</b></p> <p>Factors to be considered in deciding a sorting algorithm are,</p> <ol style="list-style-type: none"> <li>1. Programming time</li> <li>2. Executing time for program</li> <li>3. Memory or auxiliary space needed for the program's environment.</li> </ol>
27	<p><b>How does the bubble sort get its name?</b></p> <p>The bubble sort derives its name from the fact that the smallest data item bubbles up to the top of the sorted array.</p>
28	<p><b>What is the main idea behind the selection sort?</b></p> <p>The main idea behind the selection sort is to find the smallest entry among in <math>a(j), a(j+1), \dots</math> ..... <math>a(n)</math> and then interchange it with <math>a(j)</math>. This process is then repeated for each value of <math>j</math>.</p>
29	<p><b>Is the heap sort always better than the quick sort?</b></p> <p>No, the heap sort does not perform better than the quick sort. Only when array is nearly sorted to begin with the heap sort algorithm gains an advantage. In such a case, the quick sort deteriorates to its worst performance of <math>O(n^2)</math>.</p>
30	<p><b>Name some of the external sorting methods.</b></p> <p>Some of the external sorting methods are,</p> <ol style="list-style-type: none"> <li>1. Polyphase sorting</li> <li>2. Oscillation sorting</li> <li>3. Merge sorting</li> </ol>
31	<p><b>Define radix sort</b></p> <p>Radix Sort is a clever and intuitive little sorting algorithm. <b>Radix sort</b> is a non-comparative integer sorting algorithm that sorts</p>

	data with integer keys by grouping keys by the individual digits which share the same significant position
32	<p><b>Define searching</b></p> <p>Searching refers to determining whether an element is present in a given list of elements or not. If the element is present, the search is considered as successful, otherwise it is considered as an unsuccessful search. The choice of a searching technique is based on the following factors</p> <ol style="list-style-type: none"> <li>Order of elements in the list i.e., random or sorted</li> <li>Size of the list</li> </ol>
33	<p><b>Mention the types of searching</b></p> <p>The types are</p> <ul style="list-style-type: none"> <li>Linear search</li> <li>Binary search</li> </ul>
34	<p><b>What is meant by linear search?</b></p> <p><b>Linear search</b> or <b>sequential search</b> is a method for finding a particular value in a list that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.</p>
35	<p><b>What is binary search?</b></p> <p>For binary search, the array should be arranged in ascending or descending order. In each step, the algorithm compares the search key value with the middle element of the array. If the key match, then a matching element has been found and its index, or Position, is returned. Otherwise, if the search key is less than the middle element, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right.</p>
36	<p><b>What are the collision resolution methods?</b></p> <p>The following are the collision resolution methods</p> <ul style="list-style-type: none"> <li>Separate chaining</li> <li>Open addressing</li> <li>Multiple hashing</li> </ul>
37	<p><b>Define separate chaining</b></p> <p>It is an open hashing technique. A pointer field is added to each record location, when an overflow occurs; this pointer is set to point to overflow blocks making a linked list. In this method, the table can never overflow, since the linked lists are only extended upon the arrival of new keys.</p>
38	<p><b>What is open addressing?</b></p> <p>Open addressing is also called closed hashing, which is an alternative to resolve the</p>

	<p>Collisions with linked lists. In this hashing system, if a collision occurs, alternative cells are tried until an empty cell is found.</p> <p>There are three strategies in open addressing:</p> <ul style="list-style-type: none"> <li>• Linear probing</li> <li>• Quadratic probing</li> <li>• Double hashing</li> </ul>
39	<p><b>What is Rehashing?</b></p> <p>If the table is close to full, the search time grows and may become equal to the table size.</p> <p>When the load factor exceeds a certain value (e.g. greater than 0.5) we do</p> <p>Rehashing: Build a second table twice as large as the original and rehash there all the keys of the original table.</p> <p>Rehashing is expensive operation, with running time <math>O(N)</math> However, once done, the new hash table will have good performance.</p>
40	<p><b>What is Extendible Hashing?</b></p> <p>Used when the amount of data is too large to fit in main memory and external storage is used.</p> <p><b>N</b> records in total to store, <b>M</b> records in one disk block</p> <p><b>The problem:</b> in ordinary hashing several disk blocks may be examined to find an element - a time consuming process.</p> <p><b>Extendible hashing:</b> no more than two blocks are examined.</p>
1	Explain the sorting algorithms
2	Explain the searching algorithms
5	Write a C program to sort the elements using bubble sort, insertion sort and radix sort.
6	Write a C program to perform searching operations using linear and binary search.