Q.1 Attempt any FIVE of the following: [10]
Q.1(a) List any four operations on data structure. [2]
Ans.: Operations on data structure:
- Insertion
- Deletion
- Searching
- Sorting
- Traversing
- Merging

Q.1(b) Explain the concept of information, Next, Null pointer and empty list with respect to link list. [2]
Ans.: Node: Each data element in a linked list is represented as a node. Node contains two parts: one is info (data) and other is next pointer (address). Info part stores data and next pointer stores address of next node.

Next pointer: It is a pointer that holds address of next node in the list i.e. next pointer points to next node in the list

Null pointer: It is a pointer that does not hold any memory address i.e. it is pointing to nothing. It is used to specify end of the list. The last element of list contains NULL pointer to specify end of list.

Empty list: Each linked list has a header node. When header node contains NULL value, then that list is said to be empty list.
Q.1(c) List any four applications of queue.

Ans.:
- In computer system to maintain waiting list for single shared resources such as printer, disk, etc.
- It is used as buffers on MP3 players, iPod playlist, etc.
- Used for CPU scheduling in multiprogramming and time sharing systems.
- In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
- Handling of interrupts in real-time systems.
- Simulation

Q.1(d) Differentiate between stack and queue. (Any 2 points).

Ans.:

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Stack</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td>Stack is a data structure in which insertion and deletion operations are performed at same end.</td>
<td>Queue is a data structure in which insertion and deletion operations are performed at different ends.</td>
</tr>
<tr>
<td>(ii)</td>
<td>In stack an element inserted last is deleted first so it is called Last In First Out list.</td>
<td>In Queue an element inserted first is deleted first so it is called First In First Out list.</td>
</tr>
<tr>
<td>(iii)</td>
<td>In stack only one pointer is used called as stack top.</td>
<td>In Queue two pointers are used called as front and rear.</td>
</tr>
<tr>
<td>(iv)</td>
<td>Example: Stack of books</td>
<td>Example: Students standing in a line at fees counter</td>
</tr>
</tbody>
</table>
| (v)     | Application:  
- Recursion  
- Polish notation | Application:  
- In computer system for organizing processes.  
- In mobile device for sending receiving messages. |
| (vi)    | Representation: Using array | Representation: Using array |

Q.1(e) Define the terms: Linear data structure and non-linear data structure.

Ans.: 
- **Linear Data Structure**: A data structure in which all data elements are stored in a particular sequence is known as linear data structure.
Example: stack, queue

Non-Linear data structure: A data structure in which all data elements are not stored in any particular sequence is known as nonlinear data structure. Example: graph and tree.

Q.1(f) Explain time complexity and space complexity. [2]
Ans.: The analysis of the program requires two main considerations:

- Time complexity
- Space complexity

The time complexity of a program / algorithm is the amount of computer time that it needs to run to completion. The space complexity of a program/algorithm is the amount of memory that it needs to run to completion.

Time Complexity
While measuring the time of an algorithm, we concentrate on developing only the frequency count for all key statements (statements that are important and are the basic instructions of an algorithm). This is because, it is often difficult to get reliable timing figure because of clock limitations and the multiprogramming or the sharing environment.

Consider the algorithm given below:

<table>
<thead>
<tr>
<th>ALGORITHM A</th>
<th>a = a + 1</th>
</tr>
</thead>
</table>
| ALGORITHM B | for x = 1 to n step 1  
              | a = a + 1  
              | Loop |
| ALGORITHM C | for x = 1 to n step 1  
              | for y = 1 to n step 1  
              | a = a + 1  
              | Loop |

In the algorithm A we may find that the statement $a = a + 1$ is independent and is not contained within any loop. Therefore, the number of times shall be executed is 1. We can say that the frequency count of algorithm A is 1.

In the second algorithm, i.e. B, the key statement out of three statements is the assignment operation $a = a + 1$. Because this statement is contained
within a loop, the number of times it is executed is n, as the loop runs for n times. The frequency count for this algorithm is n.

According to the third algorithm, the frequency count for the statement \( a = a + 1 \) is \( n^2 \) as the inner loop runs n times, each time the outer loop runs, the outer loop also runs for n times. \( n^2 \) is said to be different in increasing order of magnitude just like 1, 10, 100 depending upon the n.

**Space Complexity**
The space needed by the program is the sum of following components:

**Fixed space requirement**
This includes the instruction space, for simple variables, fixed size structured variables and constants.

**Variable space requirement**
This consists of space needed by structured variables whose size depends on particular instance of variables. It also includes the additional space required when the function uses recursion.

**Q.1(g)** Write any four applications of data structure.  
[2]

**Ans.**:
Following are some real world applications of Data Structures:

(i) Mostly, Dictionaries are built using a Hash Table Data Structure. Hash Tables are also used for caches, database indexing, fast data lookup - symbol table for compilers.

(ii) Trees helps to build File System, Parsers.

(iii) B-Trees helps to build Database Design.

(iv) BSP tree is used in 3D computer graphics.

(v) Radix tree is used in IP routing table.

(vi) Stack : Real word applications like Java virtual machine, expression evaluation, UNDO\/REDO operation in word processors etc.

(vii) Queues : Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.

(ix) Priority queues : process scheduling in the kernel.

(x) Heap is used in Dynamic memory allocation in lisp.

(x) Graphs are used in Connections/relations in social networking sites, Routing, networks of communication, data organization etc.

**Q.2** Attempt any THREE of the following:  
[12]

**Q.2(a)** Explain the working of Binary search with an example.

**Ans.**:
Binary search is performed only on sorted array. Search method starts with calculating mid position from an array and compare the mid position element
with the search element. If a match is found then the search process ends otherwise divide the i/p list into 2 parts. First part contains all the numbers less than mid position element and second part contains all the numbers greater than mid position element. Then select one of the part depending on search element is less or greater than mid position element and calculate mid position for selected part. Again compare mid position element with search element. The binary search performs comparison and division task the element is found or division of list gives one element for comparison.

To calculate mid element perform \((\text{lower} + \text{upper}) / 2\).

**Example:**
Consider Input list 0, 1, 2, 9, 10, 11, 15, 20, 46, 72
Search element: 11

**Iteration 1**
Lower = 0  Upper = 9  mid = \((\text{lower} + \text{upper})/2 = (0 + 9)/2 = 4.5\)

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
0 & 1 & 2 & 3 & 10 & 11 & 15 & 20 & 46 & 72 \\
\end{array}
\]

mid = 11
mid > SE, Lower = mid + 1

**Iteration 2**
Lower = 5  Upper = 9  mid = \((\text{Lower} + \text{Upper})/2 = (5 + 9)/2 = 7\)

\[
\begin{array}{cccc}
5 & 6 & 7 & 8 & 9 \\
11 & 15 & 20 & 46 & 72 \\
\end{array}
\]

mid = 11
mid > SE, upper = mid - 1

**Iteration 3**
Lower = 5  Upper = 6  mid = \((\text{Lower} + \text{Upper})/2 = (5 + 6)/2 = 5.5\)

\[
\begin{array}{cc}
5 & 6 \\
11 & 15 \\
\end{array}
\]

mid = 15
Number is found
Q.2(b) Explain stack overflow and underflow conditions with example. [4]

**Ans.:**

**Stack Overflow:** Sometimes when a new data is to be inserted into a stack but there is no available space, then this situation is known as Stack Overflow.

**Example:** If a stack has maximum capacity of 3 elements, and these three are already occupied and the programmer tries to push in a fourth element, then it will lead to overflow.

**Stack Underflow:** Sometimes when one wants to delete data from a stack but the stack is already empty, then this situation is known as Stack Underflow.

**Example:** If a stack is empty and a programmer tries to execute the POP operation on it, then it will cause the Underflow error.

Q.2(c) Describe the concept of linked list with the terminologies: node, next pointer, null pointer and empty list. [4]

**Ans.:**

**Node:** Each data element in a linked list is represented as a node. Node contains two parts one is info (data) and other is next pointer (address). Info part stores data and next pointer stores address of next node.

![Node Diagram]

**Next pointer:** It is a pointer that holds address of next node in the list i.e. next pointer points to next node in the list.

**Null pointer:** It is a pointer that does not hold any memory address i.e. it is pointing to nothing. It is used to specify end of the list. The last element of list contains NULL pointer to specify end of list.

**Empty list:** Each linked list has a header node. When header node contains NULL value, then that list is said to be empty list.
Q.2(d) Write an algorithm to insert a node in between in a link list. \[4\]

Ans.: Algorithm to insert an element at the beginning of linked list:
1. Start
2. Create the node pointer *temp  
   Struct node * temp
3. Allocate address to temp using malloc  
   temp = malloc(sizeof(struct node));
4. Check whether temp is null, if null then  
   Display “Overflow”
   Else
   temp-> info=data  
   temp-> next=start
5. Start=temp
6. stop

Algorithm to insert an element at the end of linked list:
1. Start
2. Create two node pointers *temp, *q  
   struct node * temp, *q;
3. q = start
4. Allocate address to temp using malloc  
   temp = malloc(sizeof(struct node));
5. Check whether temp is null, if null then  
   Display “Overflow”
   else
   temp-> info=data  
   temp-> next=null
6. While(q->next!=null)
   q= q-> next
7. q->next= temp
8. stop

Q.3 Attempt any Three of the following: \[12\]
Q.3(a) Write an algorithm for inorder traversal of binary tree. \[4\]
Ans.: Inorder Traversal : C-B-A-D-E is the inorder traversal i.e. first we go  
towards the leftmost node i.e C so print that node C. Then go back to the  
node B and print B. Then root node A then move towards the right sub-tree  
print D and finally E. Thus we are following the tracing sequence of LDR. This  
type of traversal is called inorder traversal. The basic principal is to traverse  
left subtree then root and then the right subtree.
Q.3(b) For the following directed graph:
(i) Give adjacency matrix representation.
(ii) Give adjacency list representation.

**Ans.**

**Adjacency List:**
- A: B
- B: D
- C: A, D
- D: A

**Adjacency Matrix:**

\[
\begin{array}{cccc}
A & B & C & D \\
A & 0 & 1 & 0 & 0 \\
B & 0 & 0 & 0 & 1 \\
C & 1 & 0 & 0 & 1 \\
D & 1 & 0 & 0 & 0 \\
\end{array}
\]

Q.3(c) Evaluate the following postfix expression:

5, 6, 2, +, *, 12, 4, /, – Show diagrammatically each step of evolution using stack.

**Ans.:**

<table>
<thead>
<tr>
<th>Scanned Symbol</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Value</th>
<th>Stack Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>5, 6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>5,6,2</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>6</td>
<td>2</td>
<td>8</td>
<td>5, 8</td>
</tr>
<tr>
<td>*</td>
<td>5</td>
<td>8</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>40, 12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>40, 12</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>40, 3</td>
</tr>
<tr>
<td>–</td>
<td>40</td>
<td>3</td>
<td>37</td>
<td>37</td>
</tr>
</tbody>
</table>

Result of above postfix expression evaluation- 37
Q.3(d) Write 'c' program for deletion of an element from an array.
Ans.: #include <stdio.h>
int main()
{
    int array[100], position, c, n;
    printf("Enter number of elements in array\n");
    scanf("%d", &n);
    printf("Enter %d elements\n", n);
    for (c = 0; c < n; c++)
        scanf("%d", &array[c]);
    printf("Enter the location where you wish to delete element\n");
    scanf("%d", &position);
    if (position >= n + 1)
        printf("Deletion not possible.\n");
    else
    {
        for (c = position - 1; c < n - 1; c++)
            array[c] = array[c + 1];
        printf("Resultant array:\n");
        for (c = 0; c < n - 1; c++)
    }
    Return 0;
}

Q.4 Attempt any THREE of the following:

Q.4(a) Construct a binary search tree for following elements:
30, 100, 90, 15, 2, 25, 36, 72, 78, 10 show each step of construction of BST.
Ans.: Stepwise construction of Binary search tree for following elements:
30,100,90,15,2,25,36,72,78,10 is as follows:
Q. 4(b) Explain the concept of double ended queue.

Ans.: Double Ended Queue:

(i) A double-ended queue or dequeue is an abstract data structure that implements a queue for which elements can only be added to or removed from the front (head) or back (tail).

(ii) It is also often called a head-tail linked list.

(iii) Dequeue is a special type of data structure in which insertions and deletions will be done either at the front end or at the rear end of the queue.

(iv) The operations that can be performed on dequeues are

(a) Insert an item from front end
(b) Insert an item from rear end  
(c) Delete an item from front end  
(d) Delete an item from rear end  
(e) Display the contents of queue

Q.4(c) Draw the tree structure of the following expressions:  
(i) $(2a + 5b)^3 * (x - 7y)^4$  
(ii) $(a - 3b) * (2x - 7)^3$  
Ans.: (i) $(2a + 5b)^3 * (x - 7y)^4$  
(ii) $(a - 3b) * (2x - 7)^3$

Q.4(d) Write an algorithm to delete a node from the beginning of a circular linked list.
Ans.: Algorithm to delete a node from the beginning of a circular linked list
Consider the function delatbeg()
(i) Start
(ii) Declare struct node *tmp,*q;
(iii) Set q=last->link;
(iv) While (q! = last)
   Do
      tmp = q;  // Identifies beginning node of Circular Linked List
      last->link=q->link;  // Set the address field before deleting identified node
      free(tmp);  // Delete the beginning node
   End of While
(v) last=NULL; // Set last= NULL if only one node is present in the Circular Linked List
(vi) End of function

Q.4(e) Describe circular linked list with suitable diagram. Also state [4] advantage of circular linked list over linear linked list.

Ans.: Circular Linked List

A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop.

![Circular Linked List Diagram](image1.png)

A circular linked list can be either singly linked or doubly linked.
- for singly linked list, next pointer of last item points to the first item
- In doubly linked list, prev pointer of first item points to last item as well.

We declare the structure for the circular linked list in the same way as follows:

```c
Struct node
{
    Int data;
    Struct node *next;
};
```

```c
Typedef struct node *Node;
Node *start = null;
Node *last = null;
```

For example:

![Circular Linked List Example](image2.png)

**Advantages of Circular Linked Lists:**

(i) 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.

(ii) 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.

(iii) 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.

(iv) Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

Q.5 Attempt any TWO of the following: [12]

Q.5(a) For given binary tree write in-order, pre-order and post-order traversal. [6]

Ans.:


Q.5(b) Write an algorithm to count numbers of nodes in singly linked list. [6]

Ans.:

Step 1: Count = 0. SAVE = FIRST.
Step 2: Repeat step 3 while SAVE != NULL.
Step 3: Count = Count + 1. SAVE = SAVE->LINK.
Step 4: Return Count.

Q.5(c) Consider the graph given in Figure. Find its adjacency matrix and adjacency link representation. [6]
Ans.:

Adjacent nodes can contain two or three fields.

Q.6. Attempt any TWO of the following: [12]
Q.6(a) Create a Singly Linked List using data fields 10, 20, 30, 40, 50. [6]
Search a node 40 from the SLL and show procedure step-by-step with the help of diagram from start to end.

Ans.: Step 1:

```
START
   ↓
  __ |__ |__ |__ |__
  10  20  30  40  50
  ↓   ↓   ↓   ↓   ↓
  10 -> 20 -> 30 -> 40 -> 50 -> NULL
```

VALUE = 40. PTR != VALUE

Step 2:

```
START
  ↓
 __ |__ |__ |__ |__
 10  20  30  40  50
  ↓   ↓   ↓   ↓   ↓
 10 -> 20 -> 30 -> 40 -> 50 -> NULL
```

VALUE = 40. PTR != VALUE
Step 3:

![Linked List Diagram](image)

**VALUE = 40. PTR != VALUE**

Step 4:

![Linked List Diagram](image)

**VALUE = 40. PTR = VALUE**

Result: Value was found in the linked list. The pointer points to the node having the value.

Q.6(b) **Describe breadth first search traversal in a graph with example.** [6]

Ans.: Algorithm for BFS:

(i) Initialize all nodes to ready state.
(ii) Insert starting node in a queue and change its state to waiting state.
(iii) Repeat steps 4 to 6 till the queue becomes empty.
(iv) Remove front node N from queue 2 change its status to visit.
(v) Insert all adjacent nodes of N at the rear end of the queue and change their status to 'waiting state'.
(vi) From the origin find path from source node to destination node or from the queue element list find all nodes that are reachable.
(vii) Stop.

**Example:**

Front of the queue is set to '0'
Rear of the queue is set to '0'
Queue is used to indicate elements of the graph which are visited.

Origin is used to keep track of origin of each node.

Find all nodes reachable from 'A'

1) Insert A into a queue.

<table>
<thead>
<tr>
<th>A</th>
</tr>
</thead>
</table>

Front=1  Rear=1
Queue=A  Origin=0
2) Remove front element A and insert adjacent nodes of a in queue.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>F</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
</table>

Queue = A  front = 1
Origin = O, A, A, A rear = 3

3) Remove element F and insert its adjacent nodes in the queue.

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D  front = 2
Origin = O, A, A, A, F rear = 4

4) Remove front element C and insert its adjacent nodes in the queue
   (F is already visited)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>B</th>
<th>D</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D  front = 3
Origin = O, A, A, A, F rear = 4

5) Remove front element B and insert adjacent nodes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>D</th>
<th>G</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D, G  front = 4

6) Remove front element D and insert adjacent nodes (C is already visited)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>G</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D, G  front = 5

7) Remove front element G and insert adjacent nodes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>E</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D, G, E  front = 6

8) Remove front element E and insert its adjacent nodes (D is already a visited node)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>J</th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D, G, E, J

9) Remove J

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Queue = A, F, C, B, D, G, E, J
All nodes readable from A-A, F, C, B, D, G, E, J
Q.6(c) Create a singly linked list using data fields 90, 25, 46, 39, 56. [6]
Search a node 40 from a SLL and show procedure step-by-step with the help of diagram from start to end.

Ans.: (i) With given data fields, singly linked list is created as follows.

(ii) Operation: Search a node 22 from the above SLL
- Initially \( q = \text{start} \) where \( q \) is a pointer of type struct node used for traversing a linked list.

\[
\begin{align*}
\text{Start} & \quad 15 \quad \rightarrow \quad 20 \quad \rightarrow \quad 22 \quad \rightarrow \quad 56 \quad \rightarrow \quad 60 \quad \rightarrow \quad \text{NULL}
\end{align*}
\]

- \( q \neq \text{NULL} \) and pos = 1
  \( q \rightarrow \text{data} \neq \text{key value} \)
  i.e. is \( \neq 22 \)
  \( \therefore \) \( q = q \rightarrow \text{next} \) and pos++

\[
\begin{align*}
\text{Start} & \quad 15 \quad \rightarrow \quad 20 \quad \rightarrow \quad 22 \quad \rightarrow \quad 56 \quad \rightarrow \quad 60 \quad \rightarrow \quad \text{NULL}
\end{align*}
\]

- \( q \neq \text{NULL} \) and pos = 2
  \( q \rightarrow \text{data} \neq \text{key value} \)
  i.e. \( 20 \neq 22 \)
  \( \therefore \) \( q = q \rightarrow \text{next} \) and pos = 3

\[
\begin{align*}
\text{Start} & \quad 15 \quad \rightarrow \quad 20 \quad \rightarrow \quad 22 \quad \rightarrow \quad 56 \quad \rightarrow \quad 60 \quad \rightarrow \quad \text{NULL}
\end{align*}
\]

\( q \neq \text{NULL} \) and pos = 3 \( q \)
\( q \rightarrow \text{data} = \text{key value} \)
 i.e. \( 22 = 22 \)
 \( \therefore \) node 22 is located at position 3 search is successful.