Question 1(a) [3 marks]

Differentiate between array and list.

Answer:

Array	List
Fixed size at creation	**Dynamic size** - can grow/shrink
Homogeneous data (same type)	**Heterogeneous** data (mixed types)
Memory efficient - contiguous allocation	**Flexible** but uses more memory
Faster access for calculations	**Built-in methods** for operations

Mnemonic: "Arrays are Fixed Friends, Lists are Loose Leaders"

Question 1(b) [4 marks]

Explain the concept of class and object with the help of python program.

Answer:

Class એ એક blueprint છે જે objects ના structure અને behavior define કરે છે. **Object** એ class નો instance છે.

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display(self):
        print(f"Name: {self.name}, Age: {self.age}")

# Creating objects
s1 = Student("Ram", 20)
s2 = Student("Sita", 19)
s1.display()
```

- **Class**: Template બનાવે છે
- **Object**: Real instance બનાવે છે
- **Constructor**: Object initialize કરે છે

Mnemonic: "Class Blueprints Create Object Instances"

Question 1(c) [7 marks]

Define constructor. Discuss different types of constructors with suitable python program.

Answer:

Constructor એ special method છે જે object creation time પર automatically call થાય છે. Python માં `__init__()` method constructor છે.

```
class Demo:
    # Default Constructor
    def __init__(self):
        self.value = 0

    # Parameterized Constructor
    def __init__(self, x, y=10):
        self.x = x
        self.y = y

# Usage
d1 = Demo(5)      # x=5, y=10 (default)
d2 = Demo(3, 7)   # x=3, y=7
```

Types of Constructors:

Type	Description	Usage
Default	No parameters	Object initialization
Parameterized	With parameters	Custom initialization
Copy	Creates copy of object	Object duplication

Mnemonic: "Default Parameters Copy Objects"

Question 1(c) OR [7 marks]

Define Polymorphism. Write a python program for polymorphism through inheritance.

Answer:

Polymorphism એ same interface વાપરીને different objects પર different operations perform કરવાની ability છે.

```
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

# Polymorphic behavior
animals = [Dog(), Cat()]
```

```
for animal in animals:
    print(animal.sound())
```

- **Method Overriding**: Child class માં same method name
- **Dynamic Binding**: Runtime પર method selection
- **Code Reusability**: Same interface, different implementation

Mnemonic: "Many Objects, One Interface"

Question 2(a) [3 marks]

Explain Python specific data structure List, Tuple and Dictionary.

Answer:

Data Structure	Properties	Example
List	Mutable, ordered, allows duplicates	`[1, 2, 3, 2]`
Tuple	Immutable, ordered, allows duplicates	`(1, 2, 3, 2)`
Dictionary	Mutable, key-value pairs, no duplicate keys	`{'a': 1, 'b': 2}`

Mnemonic: "Lists Change, Tuples Stay, Dictionaries Map"

Question 2(b) [4 marks]

Explain application of stack.

Answer:

Stack Applications:

- **Function Calls**: Call stack management
- **Expression Evaluation**: Infix to postfix conversion
- **Undo Operations**: Text editors, browsers
- **Parentheses Matching**: Syntax checking

```
    +---+
    | 3 | <- Top
    +---+
    | 2 |
    +---+
    | 1 |
    +---+
```

Mnemonic: "Functions Evaluate Undo Parentheses"

Question 2(c) [7 marks]

Define stack. Explain PUSH & POP operation with example. Write an algorithm for PUSH and POP operations of stack.

Answer:

Stack એ LIFO (Last In First Out) principle follow કરતું linear data structure છે.

PUSH Algorithm:

```
1. Check if stack is full
2. If full, print "Stack Overflow"
3. Else, increment top
4. Add element at top position
```

POP Algorithm:

```
1. Check if stack is empty
2. If empty, print "Stack Underflow"
3. Else, remove element from top
4. Decrement top
```

Example:

```
stack = []
stack.append(10)  # PUSH
stack.append(20)  # PUSH
item = stack.pop()  # POP returns 20
```

Mnemonic: "Last In, First Out - Like Plates"

Question 2(a) OR [3 marks]

Define Following terms: I. Time Complexity II. Space Complexity III. Best case

Answer:

Term	Definition	Example
Time Complexity	Algorithm execution time analysis	O(n), O(log n)
Space Complexity	Memory usage analysis	O(1), O(n)
Best Case	Minimum time/space needed	Sorted array search

Mnemonic: "Time Space Best Performance"

Question 2(b) OR [4 marks]

Convert A – (B / C + (D % E * F) / G)* H into postfix expression

Answer:

Step-by-step conversion:

```
Infix: A – (B / C + (D % E * F) / G) * H

1. A B C / D E % F * G / + – H *

Stack operations:
- Operators: -, (, /, +, (, %, *, ), /, ), *
- Final: A B C / D E % F * G / + – H *
```

Postfix Result: `A B C / D E % F * G / + – H *`

Mnemonic: "Operands First, Operators Follow"

Question 2(c) OR [7 marks]

Define circular queue. Explain INSERT and DELETE operations of circular queue with diagrams.

Answer:

Circular Queue એ queue નું modified version છે જ્યાં last position first position સાથે connected હોય છે.

```
     +---+---+---+---+
     | 1 | 2 | 3 |   |
     +---+---+---+---+
       ^           ^
     front       rear
```

INSERT Algorithm:

```
1. Check if queue is full
2. rear = (rear + 1) % size
3. queue[rear] = element
4. If first element, set front = 0
```

DELETE Algorithm:

```
1. Check if queue is empty
2. element = queue[front]
3. front = (front + 1) % size
4. Return element
```

- **Advantage**: Memory efficiency
- **Application**: CPU scheduling, buffering

Mnemonic: "Circle Back When Full"

Question 3(a) [3 marks]

Explain Implementation of Stack using List.

Answer:

Stack operations Python List વડે:

```
stack = []  # Empty stack
stack.append(10)  # PUSH
stack.append(20)  # PUSH
top = stack.pop()  # POP
```

- **PUSH**: `append()` method
- **POP**: `pop()` method
- **TOP**: `stack[-1]` for peek

Mnemonic: "Append Pushes, Pop Pulls"

Question 3(b) [4 marks]

Discuss different applications of linked list.

Answer:

Linked List Applications:

- **Dynamic Memory**: Size varies at runtime
- **Insertion/Deletion**: Efficient at any position
- **Implementation**: Stacks, queues, graphs
- **Undo Functionality**: Browser history, text editors

Application	Advantage	Use Case
Music Playlist	Easy add/remove	Media players
Memory Management	Dynamic allocation	Operating systems
Polynomial Representation	Efficient storage	Mathematical operations

Mnemonic: "Dynamic Implementation Undo Memory"

Question 3(c) [7 marks]

Explain doubly linked list. Write an algorithm to delete a node from the beginning of doubly linked list

Answer:

Doubly Linked List માં દરેક node માં data, next pointer અને previous pointer હોય છે.

```
    +------+------+------+
    | prev | data | next |
    +------+------+------+
         ^          ^
       NULL     points to next
```

Delete from Beginning Algorithm:

```
1. If list is empty, return
2. If only one node:
   - head = NULL
3. Else:
   - temp = head
   - head = head.next
   - head.prev = NULL
   - delete temp
```

```python
def delete_beginning(self):
    if self.head is None:
        return
    if self.head.next is None:
        self.head = None
    else:
        self.head = self.head.next
        self.head.prev = None
```

Mnemonic: "Two Way Navigation"

Question 3(a) OR [3 marks]

Convert this Infix expression into Postfix expression: A+B/C*D-E/F-G

Answer:

Step-by-step conversion:

```
Infix: A+B/C*D-E/F-G

Postfix: A B C / D * + E F / - G -

Operator precedence: *, / > +, -
Left to right associativity
```

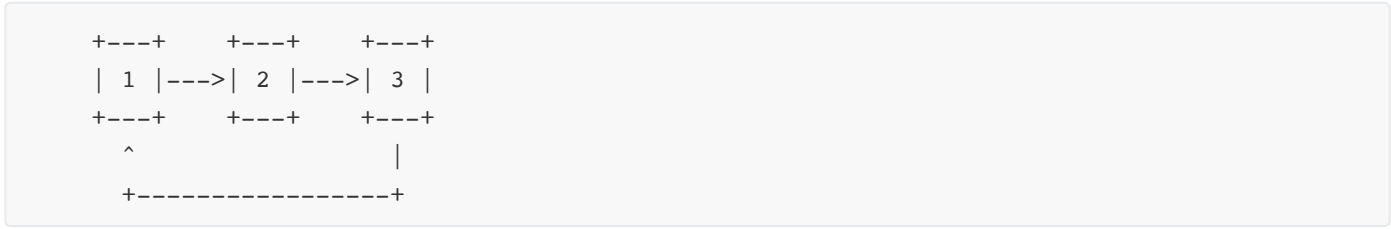
Mnemonic: "Multiply Divide Before Add Subtract"

Question 3(b) OR [4 marks]

Explain Circular Linked List with its disadvantages.

Answer:

Circular Linked List માં last node નો next pointer first node ને point કરે છે.



Disadvantages:

- **Infinite Loop Risk**: Improper traversal
- **Complex Implementation**: Extra care needed
- **Memory Overhead**: Additional pointer management
- **Debugging Difficulty**: Circular references

Mnemonic: "Circles Can Cause Confusion"

Question 3(c) OR [7 marks]

Write a Python Program to perform Insert operation in doubly Linked List. Explain with neat diagrams.

Answer:

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

class DoublyLinkedList:
    def __init__(self):
        self.head = None

    def insert_beginning(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
```

```
Before:  NULL <- [10] <-> [20] -> NULL

After:   NULL <- [5] <-> [10] <-> [20] -> NULL
                 ^
               new head
```

Insert Operations:

- **Beginning**: Update head pointer
- **End**: Traverse to last node
- **Middle**: Update prev/next pointers

Mnemonic: "Begin End Middle Insertions"

Question 4(a) [3 marks]

Write an algorithm for Merge sort.

Answer:

Merge Sort Algorithm:

```
1. If array size <= 1, return
2. Divide array into two halves
3. Recursively sort both halves
4. Merge sorted halves
```

Time Complexity: O(n log n)
Space Complexity: O(n)

Mnemonic: "Divide Conquer Merge"

Question 4(b) [4 marks]

Differentiate between Singly Linked List and Doubly Linked List.

Answer:

Singly Linked List	Doubly Linked List
One pointer (next)	**Two pointers** (next, prev)
Forward traversal only	**Bidirectional traversal**
Less memory usage	**More memory** usage
Simple implementation	**Complex implementation**

```
Singly:  [data|next] -> [data|next] -> NULL

Doubly:  NULL <- [prev|data|next] <-> [prev|data|next] -> NULL
```

Mnemonic: "Single Forward, Double Bidirectional"

Question 4(c) [7 marks]

Write an algorithm for selection sort. Give the trace to sort the given data using selection sort method. Data are: 13, 2, 6, 54, 18, 42, 11

Answer:

Selection Sort Algorithm:

```
1. For i = 0 to n-2:
   2. Find minimum in array[i...n-1]
   3. Swap minimum with array[i]
```

Trace for [13, 2, 6, 54, 18, 42, 11]:

Pass	Array State	Min Found	Swap
0	[13, 2, 6, 54, 18, 42, 11]	2	13↔2
1	[2, 13, 6, 54, 18, 42, 11]	6	13↔6
2	[2, 6, 13, 54, 18, 42, 11]	11	13↔11
3	[2, 6, 11, 54, 18, 42, 13]	13	54↔13
4	[2, 6, 11, 13, 18, 42, 54]	18	No swap
5	[2, 6, 11, 13, 18, 42, 54]	42	No swap

Final Result: [2, 6, 11, 13, 18, 42, 54]

Mnemonic: "Select Minimum, Swap Position"

Question 4(a) OR [3 marks]

Write an algorithm for Insertion sort.

Answer:

Insertion Sort Algorithm:

```
1. For i = 1 to n-1:
   2. key = array[i]
   3. j = i-1
   4. While j >= 0 and array[j] > key:
      5. array[j+1] = array[j]
      6. j = j-1
   7. array[j+1] = key
```

Time Complexity: $O(n^2)$
Best Case: O(n) for sorted array

Mnemonic: "Insert In Right Position"

Question 4(b) OR [4 marks]

Write an algorithm to insert a new node at the end of circular linked list.

Answer:

Algorithm:

```
1. Create new_node with data
2. If list is empty:
   - head = new_node
   - new_node.next = new_node
3. Else:
   - temp = head
   - While temp.next != head:
     - temp = temp.next
   - temp.next = new_node
   - new_node.next = head
```

```python
def insert_end(self, data):
    new_node = Node(data)
    if self.head is None:
        self.head = new_node
        new_node.next = new_node
    else:
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        temp.next = new_node
        new_node.next = self.head
```

Mnemonic: "Circle Back To Head"

Question 4(c) OR [7 marks]

Write an algorithm for bubble sort. Give the trace to sort the given data using bubble sort method. Data are: 37, 22, 64, 84, 58, 52, 11

Answer:

Bubble Sort Algorithm:

```
1. For i = 0 to n-2:
   2. For j = 0 to n-2-i:
      3. If array[j] > array[j+1]:
         4. Swap array[j] and array[j+1]
```

Trace for [37, 22, 64, 84, 58, 52, 11]:

Pass	Comparisons & Swaps	Result
1	37↔22, 64↔84, 84↔58, 84↔52, 84↔11	[22, 37, 64, 58, 52, 11, 84]
2	37↔64, 64↔58, 64↔52, 64↔11	[22, 37, 58, 52, 11, 64, 84]
3	37↔58, 58↔52, 58↔11	[22, 37, 52, 11, 58, 64, 84]
4	37↔52, 52↔11	[22, 37, 11, 52, 58, 64, 84]
5	37↔11	[22, 11, 37, 52, 58, 64, 84]
6	22↔11	[11, 22, 37, 52, 58, 64, 84]

Final Result: [11, 22, 37, 52, 58, 64, 84]

Mnemonic: "Bubble Up The Largest"

Question 5(a) [3 marks]

Explain Binary search tree and application of it.

Answer:

Binary Search Tree (BST) એ binary tree છે જ્યાં left subtree માં smaller values અને right subtree માં larger values હોય છે.

Properties:

- **Left child** < **Parent** < **Right child**
- **Inorder traversal** gives sorted sequence
- **Search time**: O(log n) average case

Applications:

Application	Benefit	Use Case
Database Indexing	Fast search	DBMS systems
Expression Trees	Evaluation	Compilers
Huffman Coding	Compression	Data compression

Mnemonic: "Binary Search Trees Organize Data"

Question 5(b) [4 marks]

Write Python Program for Linear Search and explain it with an example

Answer:

```python
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return -1

# Example
numbers = [10, 25, 30, 45, 60]
result = linear_search(numbers, 30)
print(f"Element found at index: {result}")  # Output: 2
```

Working:

- **Sequential check**: Element by element
- **Time Complexity**: O(n)
- **Space Complexity**: O(1)
- **Works on**: Unsorted arrays

Step	Element	Found?
1	10	No
2	25	No
3	30	Yes!

Mnemonic: "Linear Line By Line"

Question 5(c) [7 marks]

Create a Binary Search Tree for the keys 45, 35, 12, 58, 5, 55, 58, 80, 35, 42 and write the Preorder, Inorder and Postorder traversal sequences.

Answer:

BST Construction (duplicates ignored):

```
         45
        /  \
      35    58
     / \   / \
   12  42 55 80
   /
  5
```

Insertion Order: 45(root), 35(left), 12(left of 35), 58(right), 5(left of 12), 55(left of 58), 80(right of 58), 42(right of 12)

Traversals:

Traversal	Sequence	Rule
Preorder	45, 35, 12, 5, 42, 58, 55, 80	Root-Left-Right
Inorder	5, 12, 35, 42, 45, 55, 58, 80	Left-Root-Right
Postorder	5, 42, 12, 35, 55, 80, 58, 45	Left-Right-Root

Mnemonic: "Pre-Root First, In-Sorted, Post-Root Last"

Question 5(a) OR [3 marks]

Define following terms: I. Binary tree II. level number III. Leaf-node

Answer:

Term	Definition	Example
Binary tree	Tree with max 2 children per node	Each node has ≤ 2 children
Level number	Distance from root (root = level 0)	Root=0, children=1, etc.
Leaf-node	Node with no children	Terminal nodes

```
    A     <- Level 0 (Root)
   / \
  B   C   <- Level 1
 /
D        <- Level 2 (Leaf)
```

Mnemonic: "Binary Levels Lead To Leaves"

Question 5(b) OR [4 marks]

Differentiate between Linear Search and Binary search.

Answer:

Linear Search	Binary Search
Works on unsorted arrays	**Requires sorted** array
Sequential checking	**Divide and conquer**
Time: O(n)	**Time**: O(log n)
Simple implementation	**Complex implementation**
No preprocessing needed	**Sorting required**

```
Linear:  [1][2][3][4][5] -> Check each
Binary:  [1][2][3][4][5] -> Check middle, divide
```

Mnemonic: "Linear Line, Binary Bisect"

Question 5(c) OR [7 marks]

Write an algorithm for insertion and deletion a node in Binary search tree.

Answer:

Insertion Algorithm:

```
1. If root is NULL, create new node as root
2. If data < root.data, insert in left subtree
3. If data > root.data, insert in right subtree
4. If data == root.data, no insertion (duplicate)
```

Deletion Algorithm:

```
1. If node is leaf: Simply delete
2. If node has one child: Replace with child
3. If node has two children:
   - Find inorder successor
   - Replace data with successor's data
   - Delete successor
```

```python
def insert(root, data):
    if root is None:
        return Node(data)
    if data < root.data:
        root.left = insert(root.left, data)
    elif data > root.data:
        root.right = insert(root.right, data)
    return root

def delete(root, data):
    if root is None:
        return root
    if data < root.data:
        root.left = delete(root.left, data)
    elif data > root.data:
        root.right = delete(root.right, data)
    else:
        # Node to be deleted found
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        # Node with two children
```

```
            temp = find_min(root.right)
            root.data = temp.data
            root.right = delete(root.right, temp.data)
        return root
```

Cases:

- **Leaf deletion**: Direct removal
- **One child**: Replace with child
- **Two children**: Replace with successor

Mnemonic: "Insert Compare, Delete Replace"