

# Subject Name (Gujarati)

4331601 -- Summer 2024

Semester 1 Study Material

*Detailed Solutions and Explanations*

## પ્રશ્ન 1(a) [3 ગુણ]

Array અને list નો તફાવત જણાવો.

### જવાબ

| Array                       | List                             |
|-----------------------------|----------------------------------|
| નિશ્ચિત કંડ બનાવતી વખતે     | ગતિશીલ કંડ - વધી/ઘટી શકે         |
| સમાન પ્રકારનો ડેટા          | મિશ્ર પ્રકારનો ડેટા              |
| મેમરી કાર્યક્ષમ - સતત ફળવણી | લવચીક પણ વધારે મેમરી             |
| કંપી access ગાણતરી માટે     | બિલ્ટ-ઇન methods operations માટે |

### મેમરી ટ્રીક

“Arrays નિશ્ચિત મિત્રો, Lists લવચીક નેતાઓ”

## પ્રશ્ન 1(b) [4 ગુણ]

Class અને object ના concept python program ની મદદથી સમજાવો.

### જવાબ

Class એ એક blueprint છે જે objects ના structure અને behavior વ્યાખ્યાયિત 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}")
```

```
\# Objects  
s1 = Student(" ", 20)  
s2 = Student(" ", 19)  
s1.display()
```

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

### મેમરી ટ્રીક

“Class Blueprint બનાવે Object Instances”

## પ્રશ્ન 1(c) [7 ગુણ]

Constructor ની વ્યાખ્યા આપો. વિવિધ પ્રકાર ના constructor python program સાથે સમજાવો.

## જવાબ

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

Constructor ના પ્રકારો:

| પ્રકાર        | વર્ણન                | ઉપયોગ                  |
|---------------|----------------------|------------------------|
| Default       | કોઈ parameters નહીં  | Object initialization  |
| Parameterized | Parameters સાથે      | ક્ષટ્રમ initialization |
| Copy          | Object ની copy બનાવે | Object duplication     |

## મેમરી ટ્રીક

``Default Parameters Copy Objects''

## પ્રશ્ન 1(c) OR [7 ગુણ]

Polymorphism ની વ્યાખ્યા આપો. Inheritance વડે Polymorphism નો python program લખો.

## જવાબ

Polymorphism એ સમાન interface વાપરીને અલગ અલગ objects પર અલગ અલગ operations કરવાની ક્ષમતા છે.

```
class Animal:  
    def sound(self):  
        pass  
  
class Dog(Animal):  
    def sound(self):  
        return "Woof!"  
  
class Cat(Animal):  
    def sound(self):  
        return "Meow!"  
  
# Polymorphic
```

```

animals = [Dog(), Cat()]
for animal in animals:
    print(animal.sound())

• Method Overriding: Child class मાં સમાન method name
• Dynamic Binding: Runtime પર method selection
• Code Reusability: સમાન interface, અલગ implementation

```

### મેમરી ટ્રીક

“ઘણા Objects, એક Interface”

### પ્રશ્ન 2(a) [3 ગુણ]

Python અંતર્ગત data structure List, Tuple અને Dictionary સમજાવો.

#### જવાબ

| Data Structure | ગુણધર્મો                               | ઉદાહરણ           |
|----------------|--|------------------|
| List           | Mutable, ordered, duplicates allowed   | [1, 2, 3, 2]     |
| Tuple          | Immutable, ordered, duplicates allowed | (1, 2, 3, 2)     |
| Dictionary     | Mutable, key-value pairs, unique keys  | {'a': 1, 'b': 2} |

### મેમરી ટ્રીક

“Lists બદલાય, Tuples રહે, Dictionaries નકશો”

### પ્રશ્ન 2(b) [4 ગુણ]

Stack ની એપ્લિકેશન જણાવો.

#### જવાબ

##### 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 |
+{-{-}{-}+}

```

### મેમરી ટ્રીક

“Functions Evaluate કરે Undo Parentheses”

## પ્રશ્ન 2(c) [7 ગુણ]

Stack ની વ્યાખ્યા આપો. PUSH અને POP operation ઉદાહરણ સાથે સમજાવો. Stack ના PUSH અને POP operation ની algorithm લખો.

### જવાબ

Stack એ LIFO (Last In First Out) સિક્ષાંત અનુસરતું linear data structure છે.

#### PUSH Algorithm:

1. Check stack
2. , print "Stack Overflow"
3. , top increment
4. Top position element add

#### POP Algorithm:

1. Check stack
2. , print "Stack Underflow"
3. , top element remove
4. Top decrement

### ઉદાહરણ:

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

### મેમરી ટ્રીક

"છેલ્લો અંદર, પહેલો બહાર - થાળીઓ જેણું"

## પ્રશ્ન 2(a) OR [3 ગુણ]

નીચેની વ્યાખ્યા આપો: I. Time Complexity II. Space Complexity III. Best case

### જવાબ

| શબ્દ             | વ્યાખ્યા                              | ઉદાહરણ              |
|------------------|---------------------------------------|---------------------|
| Time Complexity  | Algorithm execution time નું વિશ્લેષણ | O(n), O(log n)      |
| Space Complexity | Memory usage નું વિશ્લેષણ             | O(1), O(n)          |
| Best Case        | -યૂનતમ time/space જરૂરિયાત            | Sorted array search |

### મેમરી ટ્રીક

"Time Space Best Performance"

## પ્રશ્ન 2(b) OR [4 ગુણ]

નીચે આપેલા infix expression ને postfix માં ફરારવો. A -- (B / C + (D % E \* F) / G)\* H

### જવાબ

#### 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 परिणाम: A B C / D E % F \* G / + - H \*

### મेरી ટ્રીક

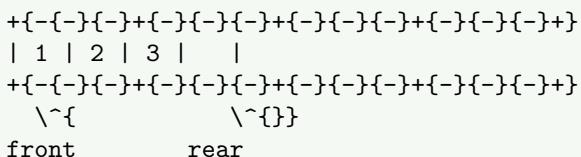
“Operands પહેલા, Operators પછી”

## પ્રશ્ન 2(c) OR [7 ગુણ]

Circular queue ની વ્યાખ્યા આપો. Circular queue ના INSERT અને DELETE operations આફુતિ સાથે સમજાવો.

### જવાબ

Circular Queue એ queue નું સુધારેલું સ્વરૂપ છે જ્યાં છેલ્લી સ્થિતિ પ્રથમ સ્થિતિ સાથે જોડાયેલી હોય છે.



### INSERT Algorithm:

1. Check queue
2. rear = (rear + 1) % size
3. queue[rear] = element
4. element , front = 0

### DELETE Algorithm:

1. Check queue
  2. element = queue[front]
  3. front = (front + 1) % size
  4. Element return
- ફાયદો: Memory efficiency
  - ઉપયોગ: CPU scheduling, buffering

### મેરી ટ્રીક

“ભરાઈ જાય તો પાછા ફરો”

## પ્રશ્ન 3(a) [3 ગુણ]

List નો ઉપયોગ કરી Stack નું Implementation સમજાવો.

### જવાબ

Stack operations Python List વડે:

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

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

## મેમરી ટ્રીક

“Append ધકેલે, Pop જોયે”

### પ્રશ્ન 3(b) [4 ગુણ]

Linked list ની વિવિધ એપ્લિકેશન વિશે ચર્ચા કરો.

#### જવાબ

##### Linked List Applications:

- Dynamic Memory:** Runtime પર size બદલાય
- Insertion/Deletion:** કોઈપણ સ્થાને કાર્યક્ષમ
- Implementation:** Stacks, queues, graphs
- Undo Functionality:** Browser history, text editors

| એપ્લિકેશન                 | ફંક્ષન્ઝ           | ઉપયોગ                   |
|---------------------------|--------------------|-------------------------|
| Music Playlist            | સરળ add/remove     | Media players           |
| Memory Management         | Dynamic allocation | Operating systems       |
| Polynomial Representation | કાર્યક્ષમ storage  | Mathematical operations |

## મેમરી ટ્રીક

“ગતિશીલ Implementation Undo Memory”

### પ્રશ્ન 3(c) [7 ગુણ]

Doubly linked list સમજાવો. Doubly linked list માં શરૂઆત ની node ને delete કરવા માટેનો algorithm લખો.

#### જવાબ

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

```
+{--{-} {-} {-} {-} {-}+{-} {-} {-} {-} {-}+{-} {-} {-} {-} {-}+}
| prev | data | next |
+{--{-} {-} {-} {-}+{-} {-} {-} {-} {-}+{-} {-} {-} {-} {-}+}
\^{} \^{} {}
NULL      points to next
```

##### શરૂઆતથી Delete કરવાનો Algorithm:

```
1. list , return
2. node :
   - head = NULL
3. :
   - temp = head
   - head = head.next
   - head.prev = NULL
   - temp delete

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
```

## મેમરી ટ્રીક

“એ દિશા નેવિગેશન”

### પ્રશ્ન 3(a) OR [3 ગુણ]

\*\*નીચે આપેલા infix expression ને postfix માં ફેરવો: A+B/C\*D-E/F-G\*\*

#### જવાબ

**Step-by-step conversion:**

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

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

Operator precedence: \*, / > +, -  
associativity

## મેમરી ટ્રીક

“ગુણા ભાગ પહેલા, સરવાળો બાદબાકી પછી”

### પ્રશ્ન 3(b) OR [4 ગુણ]

Circular Linked List તેના ગેરફાયદા સાથે સમજાવો.

#### જવાબ

**Circular Linked List** માં છેલ્લી node નો next pointer પ્રથમ node ને point કરે છે.

```
+{--}{-}{-}+    +{--}{-}{-}+    +{--}{-}{-}+
| 1 |{--}{-}{-}| 2 |{--}{-}{-}| 3 |
+{--}{-}{-}+    +{--}{-}{-}+    +{--}{-}{-}+
 \^{}           |}
 +{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+
```

#### ગેરફાયદાઓ:

- અનંત લૂપ જોખમ: ખોટા traversal
- જટિલ Implementation: વધારે સાવધાની જરૂરી
- Memory Overhead: વધારે pointer management
- Debugging મુશ્કેલી: Circular references

## મેમરી ટ્રીક

“વર્તુળો મૂંજવણ લાવી શકે”

### પ્રશ્ન 3(c) OR [7 ગુણ]

Doubly Linked List માં Insert operation ને perform કરવા માટેનો Python Program લખો. સ્વરચ આફ્ટિ સાથે સમજાવો.

#### જવાબ

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

```

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

    def insertBeginning(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

    : NULL {{-} [10] {-} [20] {-} NULL}

    : NULL {{-} [5] {-} [10] {-} [20] {-} NULL}
          \^{}  

          new head

```

#### Insert Operations:

- શરૂઆત: Head pointer update કરો
- અંત: છેલ્લા node સુધી traverse કરો
- મધ્ય: prev/next pointers update કરો

#### મેમરી ટ્રીક

“શરૂઆત અંત મધ્ય Insertions”

### પ્રશ્ન 4(a) [3 ગુણ]

Merge sort નો algorithm લખો.

#### જવાબ

#### Merge Sort Algorithm:

1. array size  $\leq 1$  , return
2. Array
3. recursively sort
4. Sorted merge

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

#### મેમરી ટ્રીક

“વહેંચો જીતો મેળવો”

### પ્રશ્ન 4(b) [4 ગુણ]

Singly Linked List અને Doubly Linked List નો તફાવત જણાવો.

#### જવાબ

| Singly Linked List  | Doubly Linked List       |
|---------------------|--------------------------|
| એક pointer (next)   | બે pointers (next, prev) |
| આગળ traversal માત્ર | બંને દિશામાં traversal   |
| ઓછી memory વપરાશ    | વધારે memory વપરાશ       |
| સરળ implementation  | જटિલ implementation      |

Singly: [data|next] {-} [data|next] {-} NULL

Doubly: NULL {{-} [prev|data|next] {-} [prev|data|next] {-} NULL}

### મેમરી ટ્રીક

"સિંગલ અાગળ, ડબલ બંને દિશા"

## પ્રશ્ન 4(c) [7 ગુણ]

Selection sort નો algorithm લખો. આપેલા ડેટા ને selection sort ની મદદથી ચડતા કર્માં ગોઠવી બતાવો. ડેટા: 13, 2, 6, 54, 18, 42, 11

### જવાબ

#### Selection Sort Algorithm:

1. i = 0 n-2 :
2. array[i...n-1] minimum
3. Minimum array[i] swap

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

| Pass | Array State                | Min મળ્યું | 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         | કોઈ swap નહીં |
| 5    | [2, 6, 11, 13, 18, 42, 54] | 42         | કોઈ swap નહીં |

અંતિમ પરિણામ: [2, 6, 11, 13, 18, 42, 54]

### મેમરી ટ્રીક

"ન્યૂનતમ પરસંદ કરો, સ્થાન બદલો"

## પ્રશ્ન 4(a) OR [3 ગુણ]

Insertion sort નો algorithm લખો.

### જવાબ

#### Insertion Sort Algorithm:

1. i = 1 n-1 :
2. key = array[i]
3. j = i-1
4. j >= 0 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 :  $O(n)$  sorted array

### મેમરી ટ્રીક

"ધોંય સ્થાને દાખલ કરો"

## પ્રશ્ન 4(b) OR [4 ગુણ]

Circular linked list માં અંત માં નવી node insert કરવા માટેનો algorithm લખો.

### જવાબ

#### Algorithm:

```

1. Data      new_node
2. list      :
   - head = new_node
   - new_node.next = new_node
3.      :
   - temp = head
   -     temp.next != head:
   -     temp = temp.next
   - temp.next = new_node
   - new_node.next = head

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

```

### મેમરી ટ્રીક

“Head પર પાછા વર્તુળ”

## પ્રશ્ન 4(c) OR [7 ગુણ]

Bubble sort નો algorithm લખો. આપેલા ડેટા ને bubble sort ની મદદથી ચડતા ક્રમમાં ગોઠવી બતાવો. ડેટા: 37, 22, 64, 84, 58, 52, 11

### જવાબ

#### Bubble Sort Algorithm:

```

1. i = 0      n-2      :
2. j = 0      n-2-i      :
3.     array[j] > array[j+1]:
4.         array[j]     array[j+1]     swap

```

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

| Pass | સરખામણી અને Swaps                           | પરિણામ                       |
|------|---|------------------------------|
| 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] |

અંતિમ પરિણામ: [11, 22, 37, 52, 58, 64, 84]

## પ્રશ્ન 5(a) [3 ગુણ]

Binary search tree અને તેની application સમજાવો.

## જવાબ

**Binary Search Tree (BST)** એ binary tree છે જ્યાં left subtree માં નાની values અને right subtree માં મોટી values હોય છે.

ગુણધર્મો:

- Left child < Parent < Right child
- Inorder traversal sorted sequence આપે છે
- Search time:  $O(\log n)$  average case

Applications:

| એપ્લિકેશન         | ફાયદો       | ઉપયોગ            |
|-------------------|-------------|------------------|
| Database Indexing | કડપી search | DBMS systems     |
| Expression Trees  | Evaluation  | Compilers        |
| Huffman Coding    | Compression | Data compression |

## મેમરી ટ્રીક

"Binary Search Trees કેવા ગોઠવે"

## પ્રશ્ન 5(b) [4 ગુણ]

Linear Search માટે Python Program લખો તથા ઉદાહરણ સાથે સમજાવો.

## જવાબ

```
def linear\_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i
    return {-}1

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

કાર્યક્રમાં:

- કષમિક તપાસ: Element દર element
- Time Complexity:  $O(n)$
- Space Complexity:  $O(1)$
- કામ કરે છે: Unsorted arrays પર

| Step | Element | મળ્યું? |
|------|---------|---------|
| 1    | 10      | ના      |
| 2    | 25      | ના      |
| 3    | 30      | હા!     |

## મેમરી ટ્રીક

"લીનિયર લાઇન દર લાઇન"

### પ્રશ્ન 5(c) [7 ગુણ]

આપેલી સાંખ્યઓ માટે Binary Search Tree બનાવો તથા તેના Preorder, Inorder અને Postorder traversals લખો: 45, 35, 12, 58, 5, 55, 58, 80, 35, 42

#### જવાબ

BST બાંધકામ (duplicates અવગણવામાં આવેલ):

```

    45
    /   {}
    35   58
   / {   / }
  12  42 55 80
 /
5
  
```

**Insertion ક્રમ:** 45(root), 35(left), 12(35 ની left), 58(right), 5(12 ની left), 55(58 ની left), 80(58 ની right), 42(12 ની right)

**Traversals:**

| Traversal | ક્રમ                          | નિયમ            |
|-----------|-------------------------------|-----------------|
| 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 |

#### મેમરી ટ્રીક

“Pre-Root પહેલા, In-Sorted, Post-Root છેદ્દે”

### પ્રશ્ન 5(a) OR [3 ગુણ]

નીચેની વ્યાખ્યા આપો: I. Binary tree II. level number III. Leaf-node

#### જવાબ

| શબ્દ         | વ્યાખ્યા                                 | ઉદાહરણ                         |
|--------------|--|--------------------------------|
| Binary tree  | દર node માં મહત્તમ 2 children વાળું tree | દેખ node માં $\leq 2$ children |
| Level number | Root થી અંતર (root = level 0)            | Root=0, children=1, વગેરે      |
| Leaf-node    | કોઈ children ન હોય તેવી node             | Terminal nodes                 |

```

      A   {{-} Level 0 (Root)}
      / {}
      B   C   {{-} Level 1}
      /
      D   {{-} Level 2 (Leaf)}
  
```

#### મેમરી ટ્રીક

“Binary Levels લીડ કરે Leaves તરફ”

### પ્રશ્ન 5(b) OR [4 ગુણ]

Linear Search અને Binary search વચ્ચેનો તફાવત જણાવો.

| Linear Search   | Binary Search             |
|---|---------------------------|
| <b>Unsorted arrays</b> પર કામ કરે   | <b>Sorted array</b> જરૂરી |
| કમિક તપાસ   | ભાગલા પાડીને જીતો         |
| Time: $O(n)$  | Time: $O(\log n)$         |
| સરળ implementation  | જટિલ implementation       |
| કોઈ preprocessing નહીં  | Sorting જરૂરી             |
| Linear: [1] [2] [3] [4] [5] {- }<br>Binary: [1] [2] [3] [4] [5] {- , } {- } |                           |

## મેમરી ટ્રીક

“Linear લાઇન, Binary વિભાજન”

## પ્રશ્ન 5(c) OR [7 ગુણ]

Binary search tree માં node ને insertion અને deletion માટેનો algorithm લખો.

## Insertion Algorithm:

1. root NULL , node root
2. data < root.data, left subtree insert
3. data > root.data, right subtree insert
4. data == root.data, insertion (duplicate)

## Deletion Algorithm:

```

1. node leaf : delete
2. node child : child
3. node children :
   - Inorder successor
   - Data successor data
   - Successor delete

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:
        # Delete node
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        # children node

```

```
temp = find\_min(root.right)
root.data = temp.data
root.right = delete(root.right, temp.data)
return root
```

કેસો:

- Leaf deletion: સીંગુ removal
- એક child: Child સાથે replace
- બે children: Successor સાથે replace

મેમરી ટ્રીક

“Insert સરખાવો, Delete બદલો”