

# Subject Name (Gujarati)

1333203 -- Winter 2023

Semester 1 Study Material

*Detailed Solutions and Explanations*

## પ્રશ્ન 1(અ) [3 ગુણ]

લિંકડ લિસ્ટની વ્યાખ્યા આપો. વિવિધ પ્રકારના લિંકડ લિસ્ટ ની ચારી આપો.

### જવાબ

વ્યાખ્યા	લિંકડ લિસ્ટના પ્રકાર
લિંકડ લિસ્ટ એ લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં એલિમેન્ટ્સ નોડ્સમાં સ્ટોર થાય છે, અને દરેક નોડ ક્રમમાં આગળના નોડને પોઇન્ટ કરે છે	1. સિંગલી લિંકડ લિસ્ટ 2. ડબલી લિંકડ લિસ્ટ 3. સક્ર્યુલર લિંકડ લિસ્ટ 4. સક્ર્યુલર ડબલી લિંકડ લિસ્ટ

### ડાયાગ્રામ:

Singly: [Data|Next] [Data|Next] [Data|Next] NULL  
Doubly: [Prev|Data|Next] [Prev|Data|Next] [Prev|Data|Next] NULL  
Circular: [Data|Next] [Data|Next] [Data|Next]

### મેમરી ટ્રીક

"એક, બે, ગોળ, બે-ગોળ"

## પ્રશ્ન 1(બ) [4 ગુણ]

પાયથનમાં લીનીયર અને નોન-લીનીયર ડેટા સ્ટર્ક્ચર ઉદાહરણ સાથે સમજાવો.

### જવાબ

ડેટા સ્ટ્રક્ચર	વર્ણન	પાયથન ઉદાહરણો
લીનીયર	એલિમેન્ટ્સ કમિક રીતે ગોઠવાયેલા હોય છે જેમાં દરેક એલિમેન્ટને એકદમ એક અગાઉનું અને એક પછીનું એલિમેન્ટ હોય છે (પ્રથમ અને છેલ્લા સિવાય)	Lists: [1, 2, 3] Tuples: (1, 2, 3) Strings: "abc" Queue: queue.Queue()
નોન-લીનીયર	એલિમેન્ટ્સ કમિક રીતે ગોઠવાયેલા નથી; એક એલિમેન્ટ અનેક એલિમેન્ટ્સ સાથે જોડાઈ શકે છે	Dictionary: {"a": 1, "b": 2} Set: {1, 2, 3} Tree: કસ્ટમ ઇમ્પ્લીમેન્ટેશન Graph: કસ્ટમ ઇમ્પ્લીમેન્ટેશન

## ડાયાગ્રામ:

### Mermaid Diagram (Code)

```

{Shaded}
{Highlighting} []
graph TD
    A[ ] --- B[ ]
    A --- C[ - ]
    A --- D[ ]
    B --- E[ ]
    B --- F[ ]
    B --- G[ ]
    C --- H[ ]
    C --- I[ ]
    C --- J[ ]
{Highlighting}
{Shaded}

```

## મેમરી ટ્રીક

“લીનીયર લાઈનમાં, નોન-લીનીયર ચારે બાજુ”

## પ્રશ્ન 1(ક) [7 ગુણ]

પાયથનમાં કલાસ, એટ્રીબ્યુટ, ઓફ્જિક્ટ અને કલાસ મેથડ યોગ્ય ઉદાહરણ સાથે સમજાવો.

## જવાબ

### ડાયાગ્રામ:

```

classDiagram
    class Student {
        {-roll_no}
        {-name}
        +__init__()
        +display()
    }

```

શબ્દ	વર્ણન
કલાસ	ઓફ્જિક્ટ્સ બનાવવા માટેનો બ્લૂપ્રિન્ટ, જેમાં શેર્ડ એટ્રીબ્યુટ્સ અને મેથડ્સ હોય છે
એટ્રીબ્યુટ્સ	કલાસની અંદર ડેટા સ્ટોર કરતા વેરિએબલ્સ
ઓફ્જિક્ટ	કલાસનું ઇન્સ્ટન્સ, જેમાં ચોક્કસ એટ્રીબ્યુટ વેલ્યુ હોય છે
કલાસ મેથડ	કલાસની અંદર ડિફાઇન થયેલા ફંક્શન્સ જે કલાસની સ્થિતિને એક્સેસ અને મોડિફાય કરી શકે છે

કોડ:

```
class Student:  
    #  
    school = "GTU"  
  
    #  
    def __init__(self, roll_no, name):  
        #  
        self.roll_no = roll_no  
        self.name = name  
  
    #  
    def display(self):  
        print(f"Roll No: {self.roll_no}, Name: {self.name}")  
  
    #  
    @classmethod  
    def change_school(cls, new_school):  
        cls.school = new_school  
  
#  
student1 = Student(101, "")  
student1.display() # : Roll No: 101, Name:
```

### મેમરી ટ્રીક

“કલાસ બનાવે, એટ્રિબ્યુટ સંગ્રહે, ઓફ્જેક્ટ વાપરે, મેથડ છિયા કરે”

### પ્રશ્ન 1(ક) OR [7 ગુણ]

કેટા એન્કેપ્સ્યુલેસન અને પોલી મોર્ફિસમની વ્યાખ્યા આપો. પોલી મોર્ફિસમ સમજાવવા માટેનો પાયથન કોડ વિકસાવો.

### જવાબ

કોન્સેપ્ટ	વ્યાખ્યા
કેટા એન્કેપ્સ્યુલેસન	કેટા અને મેથડ્સને એક એકમ (કલાસ)માં બંધ કરવા અને કેટલાક કોમ્પોનન્ટ્સને સીધી એકરોસથી પ્રતિબંધિત કરવા
પોલીમોર્ફિઝમ	વિવિધ કલાસને એક જ નામના મેથડનો પોતાનો અમલ પૂરો પાડવાની ક્ષમતા

## ડાયગ્રામ:

### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[ ] --- B[ ]
    A --- C[ ]
    A --- D[ ]
{Highlighting}
{Shaded}
```

## કોડ:

```
\#
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return "    !"

class Cat(Animal):
    def speak(self):
        return "    !"

class Duck(Animal):
    def speak(self):
        return "    !"

\#
def animal\_sound(animal):
    return animal.speak()

\#
dog = Dog()
cat = Cat()
duck = Duck()

\#
print(animal\_sound(dog))    \#      :      !
print(animal\_sound(cat))    \#      :      !
print(animal\_sound(duck))   \#      :      !
```

## મેમરી ટ્રીક

“એન્કેપ્સ્યુલેશન છુપાવે છે, પોલીમોર્ફિઝમ બદલાય છે”

## પ્રશ્ન 2(અ) [3 ગુણ]

સ્ટેક અને ક્યુ નો તફાવત આપો.

### જવાબ

ફીચર	સ્ટેક	ક્યુ
સિદ્ધાંત	LIFO (છેલ્લું આવે પહેલું જાય)	FIFO (પહેલું આવે પહેલું જાય)

ઓપરેશન  
એક્સેસ

પુશ, પોપ  
અલિમેન્ટ્સ ફક્ત એક છેડેથી ઉમેરાય/દૂર  
થાય છે (ટોપ)

એન્ક્યુ, ડિક્યુ  
અલિમેન્ટ્સ છેલ્વે ઉમેરાય છે અને  
આગળથી દૂર થાય છે

#### ડાયાગ્રામ:

Stack: [3] Queue: [1] [2] [3]  
[2] Front Rear  
[1]  
{-{-}{-}}

#### મેમરી ટ્રીક

"સ્ટેક ઉપરનું પહેલા, ક્યુ આગળનું પહેલા"

### પ્રશ્ન 2(બ) [4 ગુણ]

પુશ અને પોપ ઓપરેશન માટેનો અલ્ગોરિધમ લખો.

#### જવાબ

##### PUSH અલ્ગોરિધમ:

- 1.
2. , top 1
3. {top }

##### POP અલ્ગોરિધમ:

- 1.
2. , {top }
3. top 1
- 4.

#### કોડ:

```
class Stack:  
    def __init__(self, size):  
        self.stack = []  
        self.size = size  
        self.top = -1  
  
    def push(self, element):  
        if self.top == self.size - 1:  
            return "Stack Overflow"  
        else:  
            self.top += 1  
            self.stack.append(element)  
            return "Pushed " + str(element)  
  
    def pop(self):  
        if self.top == -1:  
            return "Stack Underflow"  
        else:  
            element = self.stack.pop()  
            self.top -= 1  
            return element
```

## મેમરી ટ્રીક

"ટોપ પર પુશ, ટોપથી પોપ"

### પ્રશ્ન 2(ક) [7 ગુણ]

નીચે આપેલ સમીકરણ ને ઇન્ફીક્સ માંથી પોસ્ટફિક્ષસ માં બદલો.  $A * (B + C) - D / (E + F)$

#### જવાબ

ડાયાગ્રામ:

Infix:  $A * (B + C) \{- D / (E + F)\}$

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

સ્ટેપ	સિમ્બોલ	સ્ટેક	આઉટપુટ
1	A	*	A
2	*	*	A
3	(	* (	A
4	B	* (	A B
5	+	* ( +	A B
6	C	* ( +	A B C
7	)	*	A B C +
8	-	-	A B C + *
9	D	-	A B C + * D
10	/	- /	A B C + * D
11	(	- / (	A B C + * D
12	E	- / (	A B C + * D E
13	+	- / ( +	A B C + * D E
14	F	- / ( +	A B C + * D E F
15	)	- /	A B C + * D E F +
16	end		A B C + * D E F + / -

#### જવાબ

A B C + \* D E F + / -

## મેમરી ટ્રીક

"ઓપરેટર સ્ટેક પર, ઓપરન્ડ સીધા પ્રિન્ટ"

### પ્રશ્ન 2(અ) OR [3 ગુણ]

સિમ્પલ ક્યુ અને સકર્યુલર ક્યુ નો તફાવત આપો.

#### જવાબ

ફીચર	સિમ્પલ ક્યુ	સકર્યુલર ક્યુ
સ્ટ્રેકચર	લીનિયર ડેટા સ્ટ્રેકચર	જોડાયેલા છેડાવાળો લીનિયર ડેટા સ્ટ્રેકચર
મેમરી	ડિક્યુ પછી ખાલી જગ્યાઓને કારણે અકાર્યક્ષમ મેમરી વપરાશ	ખાલી જગ્યાઓનો ફરીથી ઉપયોગ કરીને કાર્યક્ષમ મેમરી વપરાશ
ઇમ્પ્લિમેન્ટેશન	ફન્ટ હંમેશા ઇન્ડેક્સ 0 પર, રીયર વધે	ફન્ટ અને રીયર મોડ્યુલો ઓપરેશન સાથે સકર્યુલર રીતે ફરે

## ડાયાગ્રામ:

### Mermaid Diagram (Code)

```
{Shaded}  
{Highlighting} []  
graph LR  
    A[ ] --- B[ ]  
    B --- C[...]  
    C --- D[ ]  
  
    E[ ] --- F[ ]  
    F --- G[...]  
    G --- H[ ]  
    H --- F  
{Highlighting}  
{Shaded}
```

## મેમરી ટ્રીક

"સાદી લેડ્ડે, ગોળ ફ્રીથી વાપરે"

## પ્રશ્ન 2(બ) OR [4 ગુણ]

રીક્સીવ ફ્લેક્શનનો કોન્સેપ્ટ ચોગ્ય ઉદાહરણ સાથે સમજાવો.

### જવાબ

મુખ્ય પાસાઓ	વર્ણન
વ્યાખ્યા	એવું ફ્લેક્શન જે એક જ સમસ્યાના નાના ભાગને હલ કરવા માટે પોતાને જ કોલ કરે છે
બેઝ કેસ રિક્સીવ કેસ	એવી સ્થિતિ જ્યાં ફ્લેક્શન પોતાને કોલ કરવાનું બંધ કરે છે એવી સ્થિતિ જ્યાં ફ્લેક્શન સમસ્યાના સરળ સ્વરૂપ સાથે પોતાને કોલ કરે છે

### ડાયાગ્રામ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A["factorial(3)"] --> B["3 * factorial(2)"]
    B --> C["3 * 2 * factorial(1)"]
    C --> D["3 * 2 * 1 * factorial(0)"]
    D --> E["3 * 2 * 1 * 1"]
    E --> F["3 * 2 * 1"]
    F --> G["3 * 2"]
    G --> H[6]
{Highlighting}
{Shaded}
```

### કોડ:

```
def factorial(n):
    #
    if n == 0:
        return 1
    #
    else:
        return n * factorial(n-1)

#
result = factorial(5)  # 5! = 120
```

### મેમરી ટ્રીક

“બેઝ તોડે, રિકશન પાછું આપે”

## પ્રશ્ન 2(ક) OR [7 ગુણ]

Enqueue અને Dequeue ઓપરેશન માટેનો પાયથન કોડ વિકસાવો.

### જવાબ

#### ડાયાગ્રામ:

Enqueue:  
[1] [2] [3] [1] [2] [3] [4]

Dequeue:  
[1] [2] [3] [4] [2] [3] [4]

### કોડ:

```
class Queue:
    def __init__(self, size):
        self.queue = []
        self.size = size
        self.front = 0
        self.rear = -1
        self.count = 0

    def enqueue(self, item):
```

```

        if self.count == self.size:
            return ""
        else:
            self.rear += 1
            self.queue.append(item)
            self.count += 1
            return "Enqueued " + str(item)

    def dequeue(self):
        if self.count == 0:
            return ""
        else:
            item = self.queue.pop(0)
            self.count -= 1
            return item

    def display(self):
        return self.queue

    #
q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
print(q.display())  # [10, 20, 30]
print(q.dequeue())  # 10
print(q.display())  # [20, 30]

```

### મેમરી ટ્રીક

“છેડે ઉમરો, શરૂઆતથી કાઢો”

### પ્રશ્ન 3(અ) [3 ગુણ]

સીન્ગલી લિન્કડ લીસ્ટ અને સર્ક્રૂલર લિન્કડ લીસ્ટ નો તફાવત આપો.

#### જવાબ

ફીચર	સીન્ગલી લિન્કડ લિસ્ટ	સર્ક્રૂલર લિન્કડ લિસ્ટ
ઇન્લો નોડ	NULL તરફ પોઇન્ટ કરે છે	પહેલા નોડ તરફ પાછો પોઇન્ટ કરે છે
ટ્રાવર્સલ	ચોક્કસ અંત ધરાવે છે	સતત ટ્રાવર્સ કરી શકાય છે
મેમરી	દરેક નોડને એક પોઇન્ટર જોઈએ	દરેક નોડને એક પોઇન્ટર જોઈએ

#### ડાયાગ્રામ:

Singly: [1] [2] [3] NULL  
Circular: [1] [2] [3]

### મેમરી ટ્રીક

“સીન્ગલી અટકે, સર્ક્રૂલર ફરે”

### પ્રશ્ન 3(બ) [4 ગુણ]

દબલી લિન્કડ લીસ્ટ નો કો-સેપ્ટ સમજાવો.

## જવાબ

### ડાયગ્રામ:

```
NULL [Prev|1|Next] [Prev|2|Next] [Prev|3|Next] NULL
```

ફીલ્ડ	વર્ણન
નોડ સ્ટ્રક્ચર	દરેક નોડમાં ડેટા અને બે પોઇન્ટર્સ (previous અને next) હોય છે
નેવિગેશન	આગળ અને પાછળ એમ બંને દિશામાં ટ્રાવર્સ કરી શકાય છે
ઓપરેશન્સ	બંને છેડેથી ઇન્સર્શન અને ડિલીશન કરી શકાય છે
મેમરી વપરાશ	વધારાના પોઇન્ટરને કારણે સિંગલી લિન્કડ લિસ્ટ કરતા વધુ મેમરી જોઈએ

### કોડ:

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

### મેમરી ટ્રીક

“બે પોઇન્ટર, બે દિશા”

## પ્રશ્ન 3(ક) [7 ગુણ]

નીચે આપેલ ઓપરેશન માટે અલગોરિધમ લખો: ૧. લીસ્ટ ની શરૂઆતમાં નોડ દાખલ કરવા ૨. લીસ્ટ ના અંતમાં નોડ દાખલ કરવા

## જવાબ

### શરૂઆતમાં ઇન્સર્ટ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --{-{-}{}} B[ ]
    B --{-{-}{}} C[ ]
    C --{-{-}{}} D[ next ]]
    D --{-{-}{}} E[ ]
    E --{-{-}{}} F[ ]]

{Highlighting}
{Shaded}
```

### અંતે ઇન્સર્ટ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --{-{-}{}} B[ ]
    B --{-{-}{}} C[ ]
    C --{-{-}{}} D[ next NULL ]
    D --{-{-}{}} E{\ head NULL ?\}
    E --{-{-}{}} | F[head ]
    E --{-{-}{}} | G[ ]
    G --{-{-}{}} H[ next ]
    F --{-{-}{}} I[ ]
    H --{-{-}{}} I}

{Highlighting}
{Shaded}
```

કોડ:

```
def insert\_at\_beginning(head, data):
    new\_node = Node(data)
    new\_node.next = head
    return new\_node \#     head

def insert\_at\_end(head, data):
    new\_node = Node(data)
    new\_node.next = None

    \#
    if head is None:
        return new\_node

    \#
    temp = head
    while temp.next:
        temp = temp.next

    \#
    temp.next = new\_node
    return head
```

### મેમરી ટ્રીક

"શરૂઆત: નવો જૂનાને આગળ કરે, અંત: જૂનો નવાને આગળ કરે"

### પ્રશ્ન 3(અ) OR [3 ગુણ]

સી-ગલી લિન્કડ લિસ્ટ પરના વિવિધ ઓપરેશન ની ચાહી આપો.

#### જવાબ

##### સિંગલી લિન્કડ લિસ્ટ પરના ઓપરેશન

- ઇન્સર્શન (શરૂઆતમાં, મધ્યમાં, અંતે)
- ડિલીશન (શરૂઆતથી, મધ્યમાંથી, અંતથી)
- ટ્રાવર્સલ (દેક નોડની મુલાકાત)
- શોધ (યોક્કસ નોડ શોધવો)
- અપડેટિંગ (નોડ ડેટા બદલવો)

#### ડાયાગ્રામ:

##### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[ ] --- B[ ]
    A --- C[ ]
    A --- D[ ]
    A --- E[ ]
    A --- F[ ]
{Highlighting}
{Shaded}
```

## મેમરી ટ્રીક

"“ઉમેરો કાઢો ફરો શોધો બદલો”

### પ્રશ્ન 3(બ) OR [4 ગુણ]

સર્ક્યુલર લિન્કડ લિસ્ટ નો કોન્સેપ્ટ સમજાવો.

#### જવાબ

ડાયગ્રામ:

```

{--}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}
/           {}
↓           ↓
[1]   [2]   [3]   [4]
  
```

ફીચર	વર્ણન
સ્ટ્રક્ચર	છેલ્લો નોડ NULL ને બદલે પહેલા નોડને પોઇન્ટ કરે છે
ફાયદો	બધા નોડમાં સતત ટ્રાવર્સલની અનુમતિ આપે છે
એપ્લિકેશન	રાઉન્ડ રોબિન શેડ્યુલિંગ, સર્ક્યુલર બફર ઇમ્પ્લેમેન્ટેશન
ઓપરેશન	છેલ્લા નોડ માટે ખાસ હેન્ડલિંગ સાથે સિંગલી લિન્કડ લિસ્ટ જેવા ઇન્સર્શન અને ડિલીશન

#### કોડ:

```

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

# 3
head = Node(1)
node2 = Node(2)
node3 = Node(3)

head.next = node2
node2.next = node3
node3.next = head  #
  
```

## મેમરી ટ્રીક

"છેલ્લો પહેલાને જોડે"

### પ્રશ્ન 3(ક) OR [7 ગુણ]

લિન્કડ લિસ્ટની એપ્લિકેશનોની યાદી આપો. સીન્ગલી લિન્કડ લિસ્ટમાં કુલ નોડ ગણવા માટેનો અલગોરિદમ લખો.

#### જવાબ

##### લિન્કડ લિસ્ટની એપ્લિકેશન

- સ્ટેક અને ડ્યુનો અમલીકરણ
- ડાયનેમિક મેમરી એલોકેશન
- એપ્લિકેશનમાં અન્દું ફૂકશનાલિટી
- હેશ ટેબલ્સ (યેઈન્ઝિંગ)
- ગ્રાફ્સ માટે એડજસન્સી લિસ્ટ

## નોડ ગણવા માટેનો અલોરિધમ:

### Mermaid Diagram (Code)

```

{Shaded}
{Highlighting} []
graph LR
    A[ ] --> B{count = 0}
    B --> C{temp = head}
    C --> D{temp != NULL?}
    D --> E{count}
    D --> F{count}
    E --> G{temp = temp.next}
    G --> D
{Highlighting}
{Shaded}

```

કોડ:

```

def count\_nodes(head):
    count = 0
    temp = head

    while temp:
        count += 1
        temp = temp.next

    return count

# head
total\_nodes = count\_nodes(head)
print(f" : {total\_nodes}")

```

### મેમરી ટ્રીક

“ગણો ત્યારે ખસો”

## પ્રશ્ન 4(અ) [3 ગુણ]

લીનીયર સર્ચ અને બાયનરી સર્ચની સરખામણી કરો.

### જવાબ

ક્રીચર	લીનીયર સર્ચ	બાયનરી સર્ચ
ડેટા ગોઠવણા	સોર્ટ અને અનસોર્ટ બંને ડેટા પર કામ કરે છે	ફક્ત સોર્ટ ડેટા પર કામ કરે છે
ટાઇમ કોમ્પ્લેક્સટી	$O(n)$	$O(\log n)$
ઇમ્પ્લિમેન્ટેશન	સરળ	વધુ જટિલ
શેના માટે શ્રેષ્ઠ	નાના ડેટાસેટ અથવા અનસોર્ટ ડેટા	મોટા સોર્ટ ડેટાસેટ

### ડાયાગ્રામ:

Linear: [1] [2] [3] [4] [5] [6] [7] [8]  
 ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Binary: [1] [2] [3] [4] [5] [6] [7] [8]  
 ↓  
 /      {}  
 /      {}

### મેમરી ટ્રીક

“લીનીયર બધું જુઓ, બાઈનરી આધું કાપે”

## પ્રશ્ન 4(બ) [4 ગુણ]

સિલેક્શન સોર્ટ માટેનો અલગોરિધમ લખો.

### જવાબ

#### ડાયાગ્રામ:

```
Initial: [5, 3, 8, 1, 2]
Pass 1:  [1, 3, 8, 5, 2]  (min = 1      , 5      )
Pass 2:  [1, 2, 8, 5, 3]  (min = 2      , 3      )
Pass 3:  [1, 2, 3, 5, 8]  (min = 3      , 8      )
Pass 4:  [1, 2, 3, 5, 8]  (min = 5      ,       )
```

#### અલગોરિધમ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --> B[i = 0 n=1 ]
    B --> C[ ]
    C --> D[ ]
    D --> E[ ]
{Highlighting}
{Shaded}
```

#### કોડનો ફાંચો:

```
def selection\_sort(arr):
    n = len(arr)

    for i in range(n):
        min\_idx = i

        # for j in range(i+1, n):
        #     if arr[j] < arr[min\_idx]:
        #         min\_idx = j

        # arr[i], arr[min\_idx] = arr[min\_idx], arr[i]
```

## મેમરી ટ્રીક

“લઘુતમ શોધો, પોર્જિશન બદલો”

### પ્રશ્ન 4(ક) [7 ગુણ]

નીચે આપેલા લીસ્ટ ને બબલ સોર્ટ મેથડ વડે ચઢતા ક્રમમાં ગોઠવવા માટેનો પાયથન કોડ વિકસાવો. list1=[5,4,3,2,1,0]

#### જવાબ

##### ડાયગ્રામ:

```

Initial: [5, 4, 3, 2, 1, 0]
Pass 1: [4, 3, 2, 1, 0, 5]
Pass 2: [3, 2, 1, 0, 4, 5]
Pass 3: [2, 1, 0, 3, 4, 5]
Pass 4: [1, 0, 2, 3, 4, 5]
Pass 5: [0, 1, 2, 3, 4, 5]

```

##### કોડ:

```

def bubble\_sort(arr):
    n = len(arr)

    #
    for i in range(n):
        #
        for j in range(0, n{-}i{-}1):
            #
            if arr[j] {} arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

#
list1 = [5, 4, 3, 2, 1, 0]

#
sorted\_list = bubble\_sort(list1)

#
print("      : ", sorted\_list)
#:      : [0, 1, 2, 3, 4, 5]

```

## મેમરી ટ્રીક

“મોટા બબલ ઉપર જાય”

### પ્રશ્ન 4(અ) OR [3 ગુણ]

સોર્ટિંગ ની વ્યાખ્યા આપો. વિવિધ પ્રકારના સોર્ટિંગ ની યાદી આપો.

#### જવાબ

##### વ્યાખ્યા

સોર્ટિંગ એટલે ડેટાને ચોક્કસ કરું (ચઢતા અથવા ઉત્તરતા) ગોઠવવાની પ્રક્રિયા

##### સોર્ટિંગ મેથડ્સ

1. બબલ સોર્ટ 2. સિલેક્શન સોર્ટ 3. ઇન્સર્શન સોર્ટ 4. મર્જ સોર્ટ 5. ક્લિક સોર્ટ
6. હીપ સોર્ટ 7. રેડિક્સ સોર્ટ

ડાયાગ્રામ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[ ] --- B[ ]
    A --- C[ ]
    A --- D[ ]
    B --- E[ ]
    B --- F[ ]
    B --- G[ ]
    B --- H[ ]
    C --- I[ ]
    C --- J[ ]
    C --- K[ ]
{Highlighting}
{Shaded}
```

મેમરી ટ્રીક

“સારા સોટિથી શોધવાનું સરળ બને”

### પ્રશ્ન 4(બ) OR [4 ગુણ]

Insertion sort method નો અલગોરિદમ લખો.

જવાબ

ડાયાગ્રામ:

```
Initial: [5, 2, 4, 6, 1, 3]
Pass 1: [2, 5, 4, 6, 1, 3] (2 5 )
Pass 2: [2, 4, 5, 6, 1, 3] (4 5 )
Pass 3: [2, 4, 5, 6, 1, 3] (6 )
Pass 4: [1, 2, 4, 5, 6, 3] (1 )
Pass 5: [1, 2, 3, 4, 5, 6] (3 2 )
```

અલગોરિદમ:

#### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --- B[i = 1 n-1 ]
    B --- C["key = arr[i] "]
    C --- D[j = i-1 ]
    D --- E\{"j = 0 arr[j] key?\}
    E --- F[ ]
    F --- G[j ]
    G --- E
    E --- H[key ]
    H --- I[ ]
{Highlighting}
{Shaded}
```

કોડનો ડાંચો:

```
def insertion\_sort(arr):
    for i in range(1, len(arr)):
```

```

key = arr[i]
j = i {-} 1

\# key
while j {=} 0 and arr[j] {} key:
    arr[j + 1] = arr[j]
    j {-=} 1

arr[j + 1] = key

```

### મેમરી ટ્રીક

“કાડ લો, યોગ્ય ક્રમમાં મૂકો”

### પ્રશ્ન 4(ક) OR [7 ગુણ]

નીચે આપેલા લીસ્ટ ને સિલેક્શન સોર્ટ મેથડ વડે ચઢતા ક્રમમાં ગોઠવવા માટેનો પાયથન કોડ વિકસાવો. list1=[6,3,25,8,-1,55,0]

### જવાબ

#### ડાયાગ્રામ:

```

Initial: [6, 3, 25, 8, {-1, 55, 0}]
Pass 1:  [{-1, 3, 25, 8, 6, 55, 0} (min = {-}1 , 6      )]
Pass 2:  [{-1, 0, 25, 8, 6, 55, 3} (min = 0   , 3      )]
Pass 3:  [{-1, 0, 3, 8, 6, 55, 25} (min = 3   , 25     )]
Pass 4:  [{-1, 0, 3, 6, 8, 55, 25} (min = 6   , 8      )]
Pass 5:  [{-1, 0, 3, 6, 8, 55, 25} (min = 8   ,       )]
Pass 6:  [{-1, 0, 3, 6, 8, 25, 55} (min = 25  , 55     )]

```

#### કોડ:

```

def selection\_sort(arr):
    n = len(arr)

    for i in range(n):
        \#
        min\_idx = i
        for j in range(i+1, n):
            if arr[j] {} arr[min\_idx]:
                min\_idx = j

        \#
        arr[i], arr[min\_idx] = arr[min\_idx], arr[i]

    return arr

\#
list1 = [6, 3, 25, 8, {-}1, 55, 0]

\#
sorted\_list = selection\_sort(list1)

\#
print("      : ", sorted\_list)
\#      :  [-1, 0, 3, 6, 8, 25, 55]

```

### મેમરી ટ્રીક

“નાનામાં નાનું શોધો, આગળ મૂકો”

## પ્રશ્ન 5(અ) [3 ગુણ]

Tree data structure ને લગતા નીચે આપેલ પદોની વ્યાખ્યા આપો. 1. Forest 2. Root node 3. Leaf node

### જવાબ

પદ	વ્યાખ્યા
Forest	અલગ-અલગ ટ્રીઓનો સમૂહ (ટ્રીઓ વચ્ચે કોઈ જોડાણ નથી)
Root Node	ટ્રીનો સૌથી ઉપરનો નોડ જેનો કોઈ પેરેન્ટ નથી, જેનાથી બધા બીજા નોડ્સ ઉત્તરે છે
Leaf Node	એવો નોડ જેને કોઈ ચિલ્ડ્રન નથી (ટ્રીના તમિયે આવેલો ટર્મિનલ નોડ)

**ડાયગ્રામ:**

```

Forest:      Tree1      Tree2      Tree3
            / {         /         |}
            /   {       /           |
                    /   {}        |}
                    /   {}        |
Leaf:        [R]        [A]      [B]      [L]      [L]
    
```

Forest: Tree1 Tree2 Tree3

/ { / |

/ {} |

/ {} |

Root: [R]

/ {}

/ {}

Leaf: [A] [B] [L] [L]

### મેમરી ટ્રીક

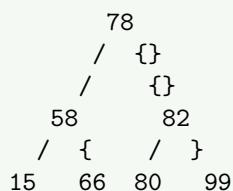
"ફોરેસ્ટમાં ઘણા રૂટ, રૂટથી બધું શરૂ, લીફ્સ બધું પૂરું"

## પ્રશ્ન 5(બ) [4 ગુણ]

78,58,82,15,66,80,99 માટે Binary search tree દોરો અને તે tree માટેનું In-order traversal લખો.

### જવાબ

#### બાઈનરી સર્ચ ટ્રી:



#### ઇન-ઓર્ડર ટ્રાવર્સલ:

સ્ટેપ	વિઝિટ કરું
1	78 ના ડાબા સબટ્રી પર જાઓ
2	58 ના ડાબા સબટ્રી પર જાઓ
3	15 ને વિઝિટ કરો
4	58 ને વિઝિટ કરો
5	66 ને વિઝિટ કરો
6	78 ને વિઝિટ કરો
7	82 ના ડાબા સબટ્રી પર જાઓ
8	80 ને વિઝિટ કરો
9	82 ને વિઝિટ કરો
10	99 ને વિઝિટ કરો

ઇન-ઓર્ડર ટ્રાવર્સલ રિઝલ્ટ: 15, 58, 66, 78, 80, 82, 99

## પ્રશ્ન 5(ક) [7 ગુણ]

નીચે આપેલ ઓપરેશન માટે અલગોરિધમ લખો: ૧. Binary Tree માં નોડ દાખલ કરવા ૨. Binary Tree માંથી નોડ કાઢવા માટે

## જવાબ

## ઇન્સર્શન અલગોરિધમ:

## Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --{-{-}{}} B[ ]
    B --{-{-}{}} C{\ root NULL ?\}
    C --{-{-}{}} | | D[root ]
    C --{-{-}{}} | | E[ ]
    E --{-{-}{}} F[ ]
    D --{-{-}{}} G[ ]
    F --{-{-}{}} G}
{Highlighting}
{Shaded}
```

## ડિલીશન અલગોરિધમ:

## Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
    A[ ] --{-{-}{}} B{\ ?\}
    B --{-{-}{}} | | C[ ]
    B --{-{-}{}} | | D[ ]
    D --{-{-}{}} E[ ]
    E --{-{-}{}} F[ ]
    F --{-{-}{}} G[ ]
    C --{-{-}{}} H[ ]
    G --{-{-}{}} H}
{Highlighting}
{Shaded}
```

## કોડ:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

    # Binary Tree
def insert(root, data):
    if root is None:
        return Node(data)

    #
queue = []
queue.append(root)

    while queue:
```

```

temp = queue.pop(0)

if temp.left is None:
    temp.left = Node(data)
    break
else:
    queue.append(temp.left)

if temp.right is None:
    temp.right = Node(data)
    break
else:
    queue.append(temp.right)

return root

# Binary Tree
def delete\_node(root, key):
    if root is None:
        return None

    if root.left is None and root.right is None:
        if root.data == key:
            return None
        else:
            return root

    # key\_node = None
    #
    last = None
    parent = None

    #
    queue = []
    queue.append(root)

    while queue:
        temp = queue.pop(0)

        if temp.data == key:
            key\_node = temp

        if temp.left:
            parent = temp
            queue.append(temp.left)
            last = temp.left

        if temp.right:
            parent = temp
            queue.append(temp.right)
            last = temp.right

    if key\_node:
        #
        key\_node.data = last.data

        #
        if parent.right == last:
            parent.right = None
        else:

```

```

parent.left = None

return root

```

### મેમરી ટ્રીક

"ખાલી જગ્યાએ ઉમેરો, બદલીને કાઢો"

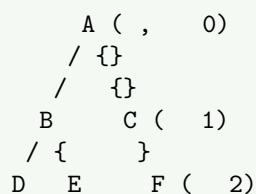
## પ્રશ્ન 5(અ) OR [3 ગુણ]

Tree data structure ને લગતા નીચે આપેલ પદોની વ્યાખ્યા આપો. 1. In-degree 2. Out-degree 3. Depth

### જવાબ

પદ	વ્યાખ્યા
In-degree	નોડમાં આવતી એજઞ્ચસની સંખ્યા (ટ્રીમાં પ્રત્યેક નોડ માટે (રૂટ સિવાય) હંમેશા 1 હોય છે)
Out-degree	નોડમાંથી બહાર જતી એજઞ્ચસની સંખ્યા (નોડના ચિલ્ડનની સંખ્યા)
Depth	રૂટથી નોડ સુધીના પાથની લંਬાઈ (પાથમાં એજઞ્ચસની સંખ્યા)

### ડાયાગ્રામ:



નોડ	In-degree	Out-degree
A	0	2
B	1	2
C	1	1
D	1	0
E	1	0
F	1	0

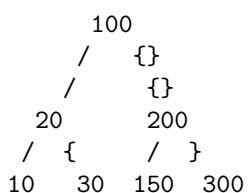
### મેમરી ટ્રીક

"ઈન કાઉન્ટ્સ પેરેન્ટ્સ, આઉટ કાઉન્ટ્સ ચિલ્ડન, ડેપ્થ કાઉન્ટ્સ એજઞ્ચસ ફોર્મ રૂટ"

## પ્રશ્ન 5(બ) OR [4 ગુણ]

નીચે દર્શાવેલા Binary tree માટે Preorder and postorder traversal લખો.

### બાઈનરી ટ્રી:



## જવાબ

ટ્રાવર્સલ	ક્રમ	રિઝલ્ટ
Preorder	રૂટ, ડાયું, જમણું	100, 20, 10, 30, 200, 150, 300
Postorder	ડાયું, જમણું, રૂટ	10, 30, 20, 150, 300, 200, 100

પ્રીઓર્ડર વિન્યુઅલાઈઝન:

### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[100] --> B[100]
    B --> C[ ]
    C --> D[20]
    D --> E[10]
    E --> F[30]
    F --> G[ ]
    G --> H[200]
    H --> I[150]
    I --> J[300]
{Highlighting}
{Shaded}
```

પોસ્ટઓર્ડર વિન્યુઅલાઈઝન:

### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph TD
    A[100] --> B[ ]
    B --> C[10]
    C --> D[30]
    D --> E[20]
    E --> F[ ]
    F --> G[150]
    G --> H[300]
    H --> I[200]
    I --> J[100]
{Highlighting}
{Shaded}
```

## મેમરી ટ્રીક

- પ્રીઓર્ડર: "રૂટ પહેલા, પછી બાળકો"
- પોસ્ટઓર્ડર: "બાળકો પહેલા, પછી રૂટ"

## પ્રશ્ન 5(ક) OR [7 ગુણ]

Binary Search Tree રચવા માટેનો પાયથન કોડ વિકસાવો.

## જવાબ

ડાયાગ્રામ:

### Mermaid Diagram (Code)

```
{Shaded}
{Highlighting} []
graph LR
```

```

A[ ] {-{-}{}|50    | B[50]}
B {-{-}{}|30    | C[50]}
C {-.-{-}{} D[30]}
C {-{-}{}|70    | E[50]}
E {-.-{-}{} F[30]}
E {-{-}{} G[70]}
G {-{-}{}|20    | H[50]}
H {-.-{-}{} I[30]}
I {-.-{-}{} J[20]}
H {-{-}{} K[70]}

{Highlighting}
{Shaded}

s1s:

class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(root, key):
    # ,
    if root is None:
        return Node(key)

    # ,
    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)

    #
    return root

def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)

def preorder(root):
    if root:
        print(root.key, end=" ")
        preorder(root.left)
        preorder(root.right)

def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.key, end=" ")

    #

def main():
    #          BST      : 50, 30, 20, 40, 70, 60, 80
    root = None
    elements = [50, 30, 20, 40, 70, 60, 80]

    for element in elements:
        root = insert(root, element)

```

```

\#
print("      : ", end="")
inorder(root)
print("{n}      : ", end="")
preorder(root)
print("{n}      : ", end="")
postorder(root)

\#
main()

ઉદાહરણ આઉટપુટ:
      : 20 30 40 50 60 70 80
      : 50 30 20 40 70 60 80
      : 20 40 30 60 80 70 50

```

### મેમરી ટ્રીક

“નાના ડાલે, મોટા જમણો”