Subject Name (Gujarati)

4331601 -- Summer 2025

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(a) [3 ગુણ]

લીનીઅર અને નોન લીનીઅર ડેટા સ્ટ્રક્ચર નો તફાવત લખો.

જવાબ

લીનીઅર ડેટા સ્ટ્રક્ચર	નોન લીનીઅર ડેટા સ્ટ્રક્ચર
એલિમેન્ટ્સ ક્રમિક રીતે સ્ટોર કરાય છે	એલિમેન્ટ્સ હાયરાર્કિકલ રીતે સ્ટોર કરાય છે
સિંગલ લેવલ ગોઠવણી	મલ્ટિ લેવલ ગોઠવણી
સરળ ટ્રાવર્સલ	જટિલ ટ્રાવર્સલ
ઉદાહરણ: Array, Stack, Queue	ઉદાહરણ: Tree, Graph

મેમરી ટ્રીક

``લીનીઅર પાણીની જેમ વહે, નોન-લીનીઅર નેટવર્ક જેવું નેવિગેટ કરે''

પ્રશ્ન 1(b) [4 ગુણ]

Object Oriented programming ના વિવિધ concepts સમજાવો.

જવાબ

OOP કોન્સેપ્ટ્સ કોષ્ટક:

કોન્સેપ્ટ	વર્ણન
Encapsulation	ડેટા અને મેથડ્સ એકસાથે બાંધવું
Inheritance	પેરેન્ટ ક્લાસથી પ્રોપર્ટીઝ મેળવવી
Polymorphism	એક નામ, અનેક સ્વરૂપો
Abstraction	ઇમ્પ્લિમેન્ટેશન વિગતો છુપાવવી

- **Encapsulation**: ડેટા હાઇડિંગ અને બન્ડલિંગ
- **Inheritance**: પેરેન્ટ-ચાઇલ્ડ સંબંધ દ્વારા કોડ પુનઃઉપયોગ
- **Polymorphism**: મેથડ ઓવરરાઇડિંગ અને ઓવરલોડિંગ
- **Abstraction**: ઇમ્પ્લિમેન્ટેશન વગરનું ઇન્ટરફેસ

મેમરી ટ્રીક

``દરેક હોશિયાર પ્રોગ્રામર Abstracts કરે છે''

પ્રશ્ન 1(c) [7 ગુણ]

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

જવાબ

Polymorphism એટલે ``અનેક સ્વરૂપો'' - એજ મેથડ નામ અલગ અલગ ક્લાસોમાં અલગ વર્તન દર્શાવે.
કોડ:

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

class Dog(Animal):
    def sound(self):
        return "Bark"

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

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

- **Polymorphism**: સેમ ઇન્ટરફેસ, અલગ ઇમ્પ્લિમેન્ટેશન
- **Runtime binding**: ઑબ્જેક્ટ ટાઇપ પર આધારિત મેથડ કૉલ
- **કોડ લવચીકતા**: નવી ક્લાસો સાથે સરળતાથી વિસ્તાર

મેમરી ટ્રીક

``Polymorphism પરફેક્ટ પ્રોગ્રામિંગ પ્રદાન કરે''

પ્રશ્ન 1(c) OR [7 ગુણ]

Abstraction ની વ્યાખ્યા આપો. Abstract class નો concept સમજવા માટેનો python program લખો.

જવાબ

Abstraction ઇમ્પ્લિમેન્ટેશન વિગતો છુપાવે છે અને ફક્ત જરૂરી ફીચર્સ બતાવે છે.
કોડ:

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(Shape):
    def \_\_init\_\_(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

\#
rect = Rectangle(5, 3)
print(f"Area: \{rect.area()\}")
```

- **Abstract class**: સીધી રીતે instantiate કરી શકાતી નથી
- **Abstract method**: ચાઇલ્ડ ક્લાસોમાં ઇમ્પ્લિમેન્ટ કરવું આવશ્યક
- **ઇન્ટરફેસ ડેફિનિશન**: સબક્લાસ માટે ટેમ્પ્લેટ પ્રદાન કરે

મેમરી ટ્રીક

``Abstraction વાસ્તવિક ઇમ્પ્લિમેન્ટેશન ટાળે''

પ્રશ્ન 2(a) [3 ગુણ]

નીચેની વ્યાખ્યા આપો: I. Best case II. Worst case III. Average case

જવાબ

કેસ	વ્યાખ્યા
Best case	અલ્ગોરિધમ માટે લઘુત્તમ સમય જરૂરી
Worst case	અલ્ગોરિધમ માટે મહત્તમ સમય જરૂરી
Average case	રેન્ડમ ઇનપુટ માટે અપેક્ષિત સમય

મેમરી ટ્રીક

``Best-Worst-Average = પરફોર્મન્સ એનાલિસિસ''

પ્રશ્ન 2(b) [4 ગુણ]

Infix, postfix અને prefix એક્સપ્રેશન સમજાવો.

જવાબ

એક્સપ્રેશન	ઑપરેટર પોઝિશન	ઉદાહરણ
Infix	ઑપરેન્ડ્સ વચ્ચે	A + B
Prefix	ઑપરેન્ડ્સ પહેલાં	+ A B
Postfix	ઑપરેન્ડ્સ પછી	A B +

- **Infix**: પ્રાકૃતિક ગાણિતિક સંકેત
- **Prefix**: Polish notation
- **Postfix**: Reverse Polish notation
- **Stack ઉપયોગ**: Postfix કૌંસ દૂર કરે છે

મેમરી ટ્રીક

``In-Pre-Post = ઑપરેટરની સ્થિતિ''

પ્રશ્ન 2(c) [7 ગુણ]

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

જવાબ

Circular Queue: લીનીઅર ડેટા સ્ટ્રક્ચર જેમાં છેલ્લી સ્થિતિ પ્રથમ સ્થિતિ સાથે જોડાય છે.
આકૃતિ:

```
    [0] [1] [2] [3]
     ↑           ↑
  front        rear
```

INSERT ઑપરેશન:

```
1.      queue
2.        , rear
3.   rear size      , rear = 0
4. rear          insert
```

DELETE ઑપરેશન:

```
1.      queue
```

```
2.        , front
3. front
4.   front size      , front = 0
```

- **ગોળ પ્રકૃતિ**: કાર્યક્ષમ મેમરી ઉપયોગ
- **કોઈ શિફ્ટિંગ નહીં**: એલિમેન્ટ્સ જગ્યામાં રહે
- **Front-rear pointers**: queue બાઉન્ડરીઝ ટ્રેક કરે

મેમરી ટ્રીક

``Circular સ્પેસ બચાવે છે''

પ્રશ્ન 2(a) OR [3 ગુણ]

ઉદાહરણ સાથે વિવિધ Data Structure જણાવો.

જવાબ

પ્રકાર	ડેટા સ્ટ્રક્ચર	ઉદાહરણ
લીનીઅર	Array	[1,2,3,4]
લીનીઅર	Stack	Function calls
લીનીઅર	Queue	Printer queue
નોન-લીનીઅર	Tree	File system
નોન-લીનીઅર	Graph	Social network

મેમરી ટ્રીક

``Arrays-Stacks-Queues = લીનીઅર, Trees-Graphs = નોન-લીનીઅર''

પ્રશ્ન 2(b) OR [4 ગુણ]

Circular queue એ simple queue કરતાં કેવી રીતે અલગ છે તે જણાવો.

જવાબ

Simple Queue	Circular Queue
લીનીઅર ગોઠવણી	ગોળ ગોઠવણી
મેમરી બગાડ	કાર્યક્ષમ મેમરી ઉપયોગ
ફિક્સ્ડ front અને rear	Wraparound pointers
False overflow	True overflow detection

- **મેમરી કાર્યક્ષમતા**: Circular ડિલીટ કરેલી જગ્યાઓ ફરી વાપરે
- **Pointer મેનેજમેન્ટ**: Wraparound માટે મોડ્યુલો અંકગણિત
- **પરફોર્મન્સ**: બહેતર સ્પેસ ઉપયોગ

મેમરી ટ્રીક

``Circular મેમરી સમસ્યાઓ જીતે છે''

પ્રશ્ન 2(c) OR [7 ગુણ]

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

જવાબ

Stack: LIFO (Last In First Out) ડેટા સ્ટ્રક્ચર.
PUSH Algorithm:

```
1.      stack
2.        , top
3. top          insert
4. top pointer
```

POP Algorithm:

```
1.      stack
2.        , top
3. top pointer
4.           return
```

ઉદાહરણ:

```
Stack: [10, 20, 30] \leftarrow top
PUSH 40: [10, 20, 30, 40] \leftarrow top
POP: returns 40, stack: [10, 20, 30] \leftarrow top
```

- **LIFO સિદ્ધાંત**: છેલ્લું ઉમેરેલું એલિમેન્ટ પ્રથમ કાઢવામાં આવે
- **Top pointer**: વર્તમાન stack પોઝિશન ટ્રેક કરે
- **Overflow/Underflow**: ઑપરેશન પહેલાં ચકાસણી

મેમરી ટ્રીક

``Stack છેલ્લા-અંદર-પ્રથમ-બહાર સ્ટોર કરે''

પ્રશ્ન 3(a) [3 ગુણ]

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

જવાબ

પગલાબદ્ધ રૂપાંતર:

પગલું	Scanned	Stack	Postfix
1	(	(	
2	(	((	
3	(	(((	
4	A	(((	A
5	-	(((-	A
6	B	(((-	AB
7	)	((	AB-
8	*	((*	AB-
9	C	((*	AB-C
10	)	(	AB-C*
11	+	(+	AB-C*
12	(	(+(	AB-C*
13	(	(+((	AB-C*
14	D	(+((	AB-C*D
15	-	(+((-	AB-C*D
16	E	(+((-	AB-C*DE
17	)	(+	AB-C*DE-
18	/	(+(/	AB-C*DE-
19	F	(+(/	AB-C*DE-F
20	)	(+	AB-C*DE-F/
21	)		AB-C*DE-F/+

અંતિમ જવાબ: AB-C*DE-F/+

મેમરી ટ્રીક

``Postfix ઑપરેટર્સ ઑપરેન્ડ્સ પછી મૂકે''

પ્રશ્ન 3(b) [4 ગુણ]

Doubly linked list વિશે ટૂંકનોંધ લખો.

જવાબ

Doubly Linked List: દ્વિદિશીય લિંક્સ સાથેની લીનીઅર ડેટા સ્ટ્રક્ચર.
સ્ટ્રક્ચર:

```
NULL  [prev|data|next]  [prev|data|next]  [prev|data|next]  NULL
```

ફાયદાઓ:
- **દ્વિદિશીય traversal**: આગળ અને પાછળ navigation
- **કાર્યક્ષમ deletion**: પાછલા node ના reference ની જરૂર નહીં
- **બહેતર insertion**: આપેલા node પહેલાં સરળતાથી insert કરી શકાય

ગેરફાયદાઓ:
- **વધારાની મેમરી**: વધારાના pointer storage
- **જટિલ operations**: વધુ pointer manipulations

મેમરી ટ્રીક

``Doubly દ્વિદિશીય ફાયદાઓ આપે''

પ્રશ્ન 3(c) [7 ગુણ]

Singly linked list માં પ્રથમ અને અંતિમ node કાઢવા માટેનો Python Program લખો.

જવાબ

કોડ:

```
class Node:
    def \_\_init\_\_(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def \_\_init\_\_(self):
        self.head = None

    def delete\_first(self):
        if self.head is None:
            return "List is empty"
        self.head = self.head.next
        return "First node deleted"

    def delete\_last(self):
        if self.head is None:
            return "List is empty"
        if self.head.next is None:
            self.head = None
            return "Last node deleted"

        current = self.head
```

```
        while current.next.next:
            current = current.next
        current.next = None
        return "Last node deleted"

    def display(self):
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        return elements

\#
ll = LinkedList()
\# nodes      deletion
```

- **પ્રથમ ડિલીટ**: head pointer અપડેટ કરો
- **છેલ્લું ડિલીટ**: બીજા છેલ્લા node સુધી traverse કરો
- **Edge cases**: ખાલી list અને સિંગલ node

મેમરી ટ્રીક

``Delete pointer અપડેટ દ્વારા આપે''

પ્રશ્ન 3(a) OR [3 ગુણ]

Queue ની વિવિધ એપ્લિકેશન જણાવો.

જવાબ

Queue એપ્લિકેશન્સ:

એપ્લિકેશન	ઉપયોગ
CPU Scheduling	Process management
Print Queue	Document printing
BFS Algorithm	Graph traversal
Buffer	Data streaming

- **FIFO પ્રકૃતિ**: પ્રથમ આવ્યો પ્રથમ સેવા
- **Real-time systems**: ઓર્ડરમાં requests handle કરે
- **Resource sharing**: વાજબી ફાળવણી

મેમરી ટ્રીક

``Queues ક્રમબદ્ધ operations શાંતિથી handle કરે''

પ્રશ્ન 3(b) OR [4 ગુણ]

Singly linked list પર આપણે કરી શકીએ તેવા વિવિધ ઑપરેશન્સ સમજાવો.

જવાબ

Singly Linked List ઑપરેશન્સ:

ઑપરેશન	વર્ણન
Insertion	શરૂઆત/અંત/મધ્યમાં node ઉમેરવું

Deletion	કોઈપણ પોઝિશનથી node કાઢવું
Traversal	બધા nodes ને ક્રમિક રીતે visit કરવા
Search	list માં ચોક્કસ ડેટા શોધવું
Count	કુલ nodes ની ગિનતી કરવી

- **ડાયનામિક સાઇઝ**: runtime દરમિયાન વધે/ઘટે
- **મેમરી કાર્યક્ષમતા**: જરૂર મુજબ allocate કરે
- **Sequential access**: કોઈ random access નથી

મેમરી ટ્રીક

``Insert-Delete-Traverse-Search-Count''

પ્રશ્ન 3(c) OR [7 ગુણ]

Doubly linked list માં અંતે નવી node insert કરવા માટેનો algorithm લખો.

જવાબ

અંતે insertion માટે Algorithm:

```
1.            node
2.    node   next = NULL
3.   list     :
   - head =   node
   -    node   prev = NULL
4.     :
   -    node    traverse
   -    node   next =   node
   -    node   prev =   node
5. success return
```

કોડ:

```
def insert\_at\_end(self, data):
    new\_node = Node(data)
    if self.head is None:
        self.head = new\_node
        return

    current = self.head
    while current.next:
        current = current.next

    current.next = new\_node
    new\_node.prev = current
```

- **દ્વિદિશીય લિંકિંગ**: next અને prev બંને pointers અપડેટ કરો
- **અંત insertion**: છેલ્લું node શોધવા traverse કરો
- **દ્વિદિશીય કનેક્શન**: list integrity જાળવો

મેમરી ટ્રીક

``દ્વિદિશીય લિંક્સ સાથે હોશિયારીથી Insert કરો''

પ્રશ્ન 4(a) [3 ગુણ]

Linear search માટેનો Python Program લખો.

જવાબ

કોડ:

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

\#
data = [10, 20, 30, 40, 50]
result = linear\_search(data, 30)
print(f"Element found at index: \{result\}")
```

- **Sequential search**: દરેક element એક પછી એક ચકાસો
- **Time complexity**: O(n)
- **સાદું implementation**: સમજવામાં આસાન

મેમરી ટ્રીક

``Linear દરેક element દ્વારા જુએ છે''

પ્રશ્ન 4(b) [4 ગુણ]

Circular linked list વિશે ટૂંકનોંધ લખો.

જવાબ

Circular Linked List: છેલ્લું node પ્રથમ node તરફ પાછું point કરે છે અને ગોળ બનાવે છે.
આકૃતિ:

```
[data|next]  [data|next]  [data|next]
     ↑                          ↓
```

લક્ષણો:

- **કોઈ NULL pointers નથી**: છેલ્લું node પ્રથમ સાથે જોડાય
- **સતત traversal**: અનંત traversal શક્ય
- **મેમરી કાર્યક્ષમતા**: બહેતર cache performance
- **એપ્લિકેશન્સ**: Round-robin scheduling, multiplayer games

ફાયદાઓ:

- **કાર્યક્ષમ insertion**: કોઈપણ પોઝિશને
- **કોઈ બગડેલા pointers નથી**: બધા nodes જોડાયેલા

મેમરી ટ્રીક

``Circular બધું loop માં જોડે છે''

પ્રશ્ન 4(c) [7 ગુણ]

Quick sort algorithm ઉદાહરણ સાથે સમજાવો.

જવાબ

Quick Sort: Pivot element વાપરીને divide અને conquer sorting algorithm.
Algorithm:

```
1. Pivot element
2. Pivot     array partition
3.    subarray   recursively sort
```

```
4.    subarray   recursively sort
```

ઉદાહરણ: [64, 34, 25, 12, 22, 11, 90] સોર્ટ કરો
પગલું 1: Pivot = 64

```
[34, 25, 12, 22, 11] 64 [90]
```

પગલું 2: બાઈ partition [34, 25, 12, 22, 11] સોર્ટ કરો Pivot = 34

```
[25, 12, 22, 11] 34 []
```

અંતિમ sorted: [11, 12, 22, 25, 34, 64, 90]

- **Divide અને conquer**: સમસ્યાને નાના ભાગોમાં વહેંચો
- **In-place sorting**: ન્યૂનતમ વધારાની મેમરી
- **Average complexity**: O(n log n)

મેમરી ટ્રીક

``Quick Partitions પછી જીતે છે''

પ્રશ્ન 4(a) OR [3 ગુણ]

Binary search algorithm ઉદાહરણ સાથે સમજાવો.

જવાબ

Binary Search: Divide અને conquer વાપરીને sorted arrays માટે search algorithm.
Algorithm:

```
1. left = 0, right = array length - 1
2.      left <= right:
   - mid = (left + right) / 2 calculate
   -   target = array[mid], mid return
   -   target < array[mid], right = mid - 1
   -   target > array[mid], left = mid + 1
3.       -1 return
```

ઉદાહરણ: [11, 12, 22, 25, 34, 64, 90] માં 22 શોધો

પગલું	Left	Right	Mid	Value	Action
1	0	6	3	25	22 < 25, right = 2
2	0	2	1	12	22 > 12, left = 2
3	2	2	2	22	મળ્યું!

મેમરી ટ્રીક

``Binary ઝડપથી શોધવા બે ભાગ કરે છે''

પ્રશ્ન 4(b) OR [4 ગુણ]

Linked list ની વિવિધ એપ્લિકેશન જણાવો.

જવાબ

Linked List એપ્લિકેશન્સ:

એપ્લિકેશન	ઉપયોગ
Dynamic Arrays	Resizable ડેટા storage
Stack/Queue Implementation	LIFO/FIFO structures

Graph Representation	Adjacency lists
Memory Management	Free memory blocks
Music Playlist	Next/previous song navigation

- **ડાયનામિક મેમરી**: જરૂર મુજબ allocate કરો
- **કાર્યક્ષમ insertion/deletion**: કોઈ shifting જરૂરી નથી
- **લવચીક structure**: બદલાતી જરૂરિયાતોને અનુકૂળ

મેમરી ટ્રીક

``Linked Lists ડાયનામિક એપ્લિકેશન્સમાં રહે છે''

પ્રશ્ન 4(c) OR [7 ગુણ]

ઉદાહરણ સાથે Insertion sort માટેનો python program લખો.

જવાબ

કોડ:

```
def insertion\_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i {-} 1

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

        arr[j + 1] = key

    return arr

\#
data = [64, 34, 25, 12, 22, 11, 90]
sorted\_data = insertion\_sort(data)
print(f"Sorted array: \{sorted\_data\}")
```

પગલાબદ્ધ ઉદાહરણ:

```
Initial: [64, 34, 25, 12, 22, 11, 90]
Pass 1:  [34, 64, 25, 12, 22, 11, 90]
Pass 2:  [25, 34, 64, 12, 22, 11, 90]
Pass 3:  [12, 25, 34, 64, 22, 11, 90]
Pass 4:  [12, 22, 25, 34, 64, 11, 90]
Pass 5:  [11, 12, 22, 25, 34, 64, 90]
Pass 6:  [11, 12, 22, 25, 34, 64, 90]
```

- **કાર્ડ sorting analogy**: playing cards ગોઠવવા જેવું
- **Stable sort**: સમાન elements નો relative order જાળવે
- **Online algorithm**: ડેટા મળતા જ list sort કરી શકે

મેમરી ટ્રીક

``Insertion યોગ્ય જગ્યામાં Insert કરે છે''

પ્રશ્ન 5(a) [3 ગુણ]

નીચેની વ્યાખ્યા આપો: I. Complete Binary tree II. In-degree III. Out-degree.

જવાબ

શબ્દ	વ્યાખ્યા
Complete Binary Tree	છેલ્લા level સિવાય બધા levels ડાબેથી ભરાયેલા
In-degree	Node માં આવતા edges ની સંખ્યા
Out-degree	Node માંથી જતા edges ની સંખ્યા

મેમરી ટ્રીક

``Complete-In-Out = Tree terminology''

પ્રશ્ન 5(b) [4 ગુણ]

Bubble sort algorithm ઉદાહરણ સાથે સમજાવો.

જવાબ

Bubble Sort: અડીખમ elements ની તુલના કરો અને ખોટા ક્રમમાં હોય તો swap કરો.
Algorithm:

```
1.    pass    (0    n-1):
   2.    element    (0    n-pass-1):
      3.    arr[j] > arr[j+1]:
         4. arr[j]    arr[j+1] swap
```

ઉદાહરણ: [64, 34, 25, 12]

Pass	Comparisons	Result
1	64>34(swap), 64>25(swap), 64>12(swap)	[34,25,12,64]
2	34>25(swap), 34>12(swap)	[25,12,34,64]
3	25>12(swap)	[12,25,34,64]

- **Bubble up**: સૌથી મોટું element અંતે bubble થાય
- **Multiple passes**: દરેક pass એક element સાચી જગ્યામાં મૂકે
- **સાદું implementation**: સમજવામાં આસાન

મેમરી ટ્રીક

``Bubble સૌથી મોટાને પાછળ લાવે છે''

પ્રશ્ન 5(c) [7 ગુણ]

આપેલી સંખ્યાઓ માટે Binary Search Tree બનાવો તથા તેના Preorder, Inorder અને Postorder traversals લખો: 15, 35, 12, 48, 5, 25, 58, 8

જવાબ

BST Construction:

```
        15
       /  {}
      12    35
     /     /  {}
    5     25    48
     {            }
      8            58
```

Traversal Sequences:

Traversal	Sequence
Preorder	15, 12, 5, 8, 35, 25, 48, 58
Inorder	5, 8, 12, 15, 25, 35, 48, 58
Postorder	8, 5, 12, 25, 58, 48, 35, 15

Traversal નિયમો:
- **Preorder**: Root $\rightarrow Left \rightarrow Right$
- **Inorder**: Left $\rightarrow Root \rightarrow Right(sortedorder)$
- **Postorder**: Left $\rightarrow Right \rightarrow Root$

મેમરી ટ્રીક

``Pre-In-Post = Root ની સ્થિતિ''

પ્રશ્ન 5(a) OR [3 ગુણ]

Binary tree ની વ્યાખ્યા આપો. Binary tree માં node searching વિશે સમજાવો.

જવાબ

Binary Tree: Hierarchical ડેટા structure જેમાં દરેક node ને મહત્તમ બે children હોય.
Search Algorithm:

```
1. Root
2.   target = current node, found return
3.   target < current node,
4.   target > current node,
5.    NULL            repeat
```

- **Hierarchical structure**: Parent-child સંબંધ
- **Binary property**: node દીઠ મહત્તમ બે children
- **Search કાર્યક્ષમતા**: Balanced trees માટે O(log n)

મેમરી ટ્રીક

``Binary બે પાથમાં branch કરે છે''

પ્રશ્ન 5(b) OR [4 ગુણ]

આપેલા ડેટા ને bubble sort ની મદદથી ચડતા ક્રમમાં ગોઠવી બતાવો. ડેટા: 44, 72, 94, 28, 18, 442, 41

જવાબ

Bubble Sort Trace:

Pass	Array State	Swaps
Initial	[44, 72, 94, 28, 18, 442, 41]	-
Pass 1	[44, 72, 28, 18, 94, 41, 442]	94>28, 94>18, 442>41
Pass 2	[44, 28, 18, 72, 41, 94, 442]	72>28, 72>18, 94>41
Pass 3	[28, 18, 44, 41, 72, 94, 442]	44>28, 44>18, 72>41
Pass 4	[18, 28, 41, 44, 72, 94, 442]	28>18, 44>41
Pass 5	[18, 28, 41, 44, 72, 94, 442]	કોઈ swaps નથી

અંતિમ sorted array: [18, 28, 41, 44, 72, 94, 442]

મેમરી ટ્રીક

``Bubble sort દરેક pass સૌથી મોટાને અંતે bubbles કરે''

પ્રશ્ન 5(c) OR [7 ગુણ]

Trees ની વિવિધ એપ્લિકેશન જણાવો. General tree ને Binary Search Tree માં રૂપાંતર કરવા માટેની technique ઉદાહરણ સાથે સમજાવો.

જવાબ

Tree એપ્લિકેશન્સ:

એપ્લિકેશન	ઉપયોગ
File System	Directory hierarchy
Expression Trees	ગાણિતિક expressions
Decision Trees	AI અને machine learning
Heap	Priority queues

General Tree થી BST રૂપાંતર:
Technique: First Child - Next Sibling Representation
મૂળ General Tree:

```
    A
   /|\
  B C D
 /| |
E F G
```

Binary Tree માં રૂપાંતર:

```
    A
   /
  B
   {}
    C
   / {}
  E   D
   {}
    F
     {}
      G
```

પગલાં:

1. **First child**: Binary tree માં left child બને
2. **Next sibling**: Binary tree માં right child બને
3. **Recursive application**: બધા nodes પર લાગુ કરો

- **વ્યવસ્થિત રૂપાંતર**: Tree structure જાળવે
- **Binary representation**: node દીઠ ફક્ત બે pointers વાપરે
- **Space કાર્યક્ષમતા**: મ□□ક binary tree operations લાગુ પડે

મેમરી ટ્રીક

``First-child ડાબે, Next-sibling જમણે''