

# Data Structure with Python (4331601) - Summer 2025 Solution

Milav Dabgar

May 09, 2025

## પ્રશ્ન 1(a) [3 ગુણ]

લીનીઅર અને નોન લીનીઅર ડેટા સ્ટ્રક્ચર નો તફાવત લખો.

જવાબ

જવાબ:

કોષ્ટક 1. લીનીઅર vs નોન લીનીઅર ડેટા સ્ટ્રક્ચર

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

જવાબ:

OOP કોન્સેપ્ટ્સ કોષ્ટક:

કોષ્ટક 2. OOP કોન્સેપ્ટ્સ

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

જવાબ:

**Polymorphism** એટલે "અનેક સ્વરૂપો" - એજ મેથડ નામ અલગ અલગ ક્લાસોમાં અલગ વર્તન દર્શાવે.

કોડ:

Listing 1. Polymorphism ઉદાહરણ

```

1 class Animal:
2     def sound(self):
3         pass
4
5 class Dog(Animal):
6     def sound(self):
7         return "Bark"
8
9 class Cat(Animal):
10    def sound(self):
11        return "Meow"
12
13 # Polymorphism ની કુરંધિ
14 animals = [Dog(), Cat()]
15 for animal in animals:
16    print(animal.sound())

```

- **Polymorphism:** સેમ ઇન્ટરફેસ, અલગ ઇમ્પ્લિમેન્ટેશન
- **Runtime binding:** ઓબ્જેક્ટ ટાઇપ પર આધારિત મેથડ કોલ
- **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

જવાબ:

**Abstraction** ઇમ્પ્લિમેન્ટેશન વિગતો છુપાવે છે અને ફક્ત જરૂરી ફીચર્સ બતાવે છે.

કોડ:

Listing 2. Abstraction ઉદાહરણ

```

1 from abc import ABC, abstractmethod
2
3 class Shape(ABC):
4     @abstractmethod

```

```

5  def area(self):
6      pass
7
8  class Rectangle(Shape):
9      def __init__(self, length, width):
10         self.length = length
11         self.width = width
12
13     def area(self):
14         return self.length * self.width
15
16 # ઉપયોગ
17 rect = Rectangle(5, 3)
18 print(f"Area: {rect.area()}")

```

- **Abstract class:** સીધી રીતે instantiate કરી શકાતી નથી
- **Abstract method:** ચાઇલ્ડ ક્લાસોમાં ઇમ્પ્લિમેન્ટ કરવું આવશ્યક
- **ઇન્ટરફેસ ડેફિનિશન:** સબકલાસ માટે ટેમ્પલેટ 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

જવાબ:

કોષ્ટક 3. Time Complexity Cases

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

જવાબ:

કોષ્ટક 4. Expression Types

એક્સપ્રેશન	ઓપરેટર પોઝિશન	ઉદાહરણ
Infix	ઓપરેન્ડ્સ વચ્ચે	A + B
Prefix	ઓપરેન્ડ્સ પહેલાં	+ A B
Postfix	ઓપરેન્ડ્સ 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 = ઓપરેટરની સ્થિતિ

## પ્રશ્ન 2(c) [7 ગુણ]

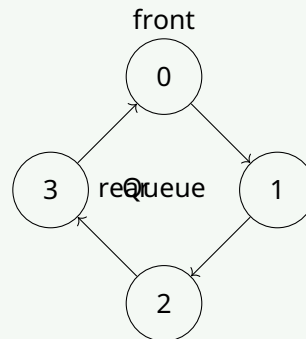
Circular queue ની વ્યાખ્યા આપો. Circular queue ના INSERT અને DELETE operations આકૃતિ સાથે સમજાવો.

### જવાબ

**જવાબ:**

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

**આકૃતિ:**



આકૃતિ 1. Circular Queue

### INSERT ઓપરેશન:

1. ચકાસો કે queue ભરાઈ ગઈ છે કે નહીં
2. જો ભરાઈ નથી, rear વધારો
3. જો rear size કરતાં વધે, rear = 0 કરો
4. rear પોઝિશન એલિમેન્ટ insert કરો

### DELETE ઓપરેશન:

1. ચકાસો કે queue ખાલી છે કે નહીં
2. જો ખાલી નથી, front માંથી એલિમેન્ટ કાઢો
3. front વધારો
4. જો front size કરતાં વધે, front = 0 કરો
  - **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

જવાબ:

કોષ્ટક 5. Data Structure Types

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

જવાબ:

કોષ્ટક 6. Simple vs Circular Queue

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

**જવાબ:**

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

**PUSH Algorithm:**

1. ચકાસો કે stack ભરાઈ ગઈ છે કે નહીં
2. જો ભરાઈ નથી, top વધારો
3. top પોઝિશન એલિમેન્ટ insert કરો
4. top pointer અપડેટ કરો

**POP Algorithm:**

1. ચકાસો કે stack ખાલી છે કે નહીં
2. જો ખાલી નથી, top એલિમેન્ટ સ્ટોર કરો
3. top pointer ઘટાડો
4. સ્ટોર કરેલું એલિમેન્ટ return કરો

**ઉદાહરણ:**

### Listing 3. Stack ઉદાહરણ

```
1 Stack: [10, 20, 30] <- top
2 PUSH 40: [10, 20, 30, 40] <- top
3 POP: returns 40, stack: [10, 20, 30] <- top
```

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

**જવાબ:**

**પગલાબદ્ધ રૂપાંતર:**

### કોષ્ટક 7. Infix to Postfix Conversion

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 ઓપરેટર્સ ઓપરેન્ડ્સ પછી મૂકે

## પ્રશ્ન 3(b) [4 ગુણ]

Doubly linked list વિશે ટૂંકનોંધ લખો.

### જવાબ

જવાબ:

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

**સ્ટ્રક્ચર:**

NULL 



































 NULL

આકૃતિ 2. Doubly Linked List

**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 ટ્રિફ્શીય ફાયદાઓ આપે

## પ્રશ્ન 3(c) [7 ગુણ]

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

## જવાબ

જવાબ:

કોડ:

Listing 4. Singly Linked List Deletion

```

1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5
6 class LinkedList:
7     def __init__(self):
8         self.head = None
9
10    def delete_first(self):
11        if self.head is None:
12            return "List is empty"
13        self.head = self.head.next
14        return "First node deleted"
15
16    def delete_last(self):
17        if self.head is None:
18            return "List is empty"
19        if self.head.next is None:
20            self.head = None
21            return "Last node deleted"
22
23        current = self.head
24        while current.next.next:
25            current = current.next
26        current.next = None
27        return "Last node deleted"
28
29    def display(self):
30        elements = []
31        current = self.head
32        while current:
33            elements.append(current.data)
34            current = current.next
35        return elements
36
37 # ઉપયોગ
38 ll = LinkedList()
39 # nodes ઉમેરો અને deletion ટેસ્ટ કરો

```

- 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

જવાબ:  
Queue એપ્લિકેશન-સ:

કોષ્ટક 8. Queue Applications

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 કમ્પ્લેક્ષ operations શાંતિથી handle કરે

## પ્રશ્ન 3(b OR) [4 ગુણ]

Singly linked list પર આપણે કરી શકીએ તેવા વિવિધ ઓપરેશન-સ સમજાવો.

## જવાબ

જવાબ:  
Singly Linked List ઓપરેશન-સ:

કોષ્ટક 9. Singly Linked List Operations

ઓપરેશન	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 સાઈઝ: 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

જવાબ:

અંતે insertion માટે Algorithm:

1. આપેલા ડેટા સાથે નવું node બનાવો
2. નવા node નું next = NULL કરો
3. જો list ખાલી છે:
  - head = નવું node કરો
  - નવા node નું prev = NULL કરો
4. નહીં તો:
  - છેલ્લા node સુધી traverse કરો
  - છેલ્લા node નું next = નવું node કરો
  - નવા node નું prev = છેલ્લું node કરો
5. success return કરો

કોડ:

Listing 5. Insert at End Doubly Linked List

```

1 def insert_at_end(self, data):
2     new_node = Node(data)
3     if self.head is None:
4         self.head = new_node
5         return
6
7     current = self.head
8     while current.next:
9         current = current.next
10
11     current.next = new_node
12     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

જવાબ:

કોડ:

## Listing 6. Linear Search

```

1 def linear_search(arr, target):
2     for i in range(len(arr)):
3         if arr[i] == target:
4             return i
5     return -1
6
7 # ઉદાહરણ ઉપયોગ
8 data = [10, 20, 30, 40, 50]
9 result = linear_search(data, 30)
10 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]

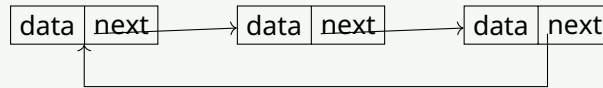
Circular linked list વિશે ટૂંકનોંધ લખો.

## જવાબ

જવાબ:

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

**આકૃતિ:**



આકૃતિ 3. Circular Linked List

**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

**કાયદાઓ:**

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

1. Pivot element પસંદ કરો
2. Pivot આસપાસ array partition કરો
3. બાઈ subarray ને recursively sort કરો
4. જમણી subarray ને recursively sort કરો

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

1. left = 0, right = array length - 1 સેટ કરો
2. જ્યાં સુધી left <= right:
  - mid = (left + right) / 2 calculate કરો
  - જો target = array[mid], mid return કરો
  - જો target < array[mid], right = mid - 1
  - જો target > array[mid], left = mid + 1
3. ન મળે તો -1 return કરો

**ઉદાહરણ:** [11, 12, 22, 25, 34, 64, 90] માં 22 શોધો**કોષ્ટક 10. Binary Search Trace**

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**

**જવાબ:**

Linked List એપ્લિકેશન-સ:

**કોષ્ટક 11. Linked List Applications**

એપ્લિકેશન	ઉપયોગ
Dynamic Arrays	Resizable 32b 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 એપ્લિકેશન-સમાં રહે છે

## પ્રશ્ન 4(c OR) [7 ગુણ]

ઉદાહરણ સાથે Insertion sort માટેનો python program લખો.

**જવાબ**

**જવાબ:**

**કોડ:**

**Listing 7. Insertion Sort**

```

1 def insertion_sort(arr):
2     for i in range(1, len(arr)):
3         key = arr[i]
4         j = i - 1
5
6         while j >= 0 and arr[j] > key:
7             arr[j + 1] = arr[j]
8             j -= 1
9
10        arr[j + 1] = key
11
12    return arr
13
14    # ઉદાહરણ
15    data = [64, 34, 25, 12, 22, 11, 90]
16    sorted_data = insertion_sort(data)
17    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 સોર્ટિંગ 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

જવાબ:

કોષ્ટક 12. Binary Tree Terms

શબ્દ	વ્યાખ્યા
<b>Complete Binary Tree</b>	છેલ્લા level સિવાય બધા levels ડાબેથી ભરાયેલા
<b>In-degree</b>	Node માં આવતા edges ની સંખ્યા
<b>Out-degree</b>	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

જવાબ:

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

**Algorithm:**

1. દરેક pass માટે (0 થી n-1):
2. દરેક element માટે (0 થી n-pass-1):
3. જો  $arr[j] > arr[j+1]$ :
4.  $arr[j]$  અને  $arr[j+1]$  swap કરો

ઉદાહરણ: [64, 34, 25, 12]

કોષ્ટક 13. Bubble Sort Trace

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

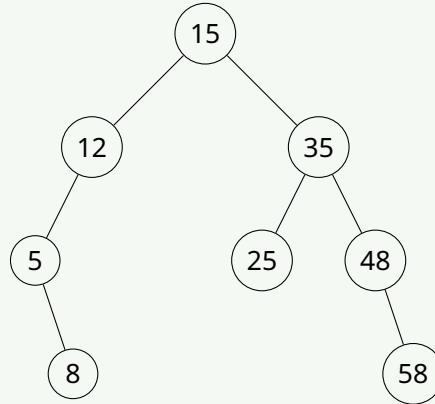
Bubble સૌથી મોટાને પાછળ લાવે છે

## 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

જવાબ:  
BST Construction:



આકૃતિ 4. Binary Search Tree

Traversal Sequences:

કોષ્ટક 14. Traversals

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 → Left → Right
- **Inorder:** Left → Root → Right (sorted order આપે)
- **Postorder:** Left → Right → 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

જવાબ:

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

1. Root થી શરૂઆત કરો
2. જો target = current node, found return કરો
3. જો target < current node, ડાબે જાઓ
4. જો target > current node, જમણે જાઓ
5. મળે કે NULL સુધી પહોંચે ત્યાં સુધી repeat કરો
  - **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

જવાબ:

**Bubble Sort Trace:**

કોષ્ટક 15. 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.

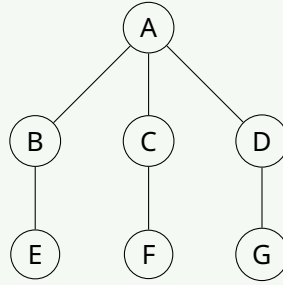
જવાબ

જવાબ:  
Tree એપ્લિકેશન્સ:

કોષ્ટક 16. Tree Applications

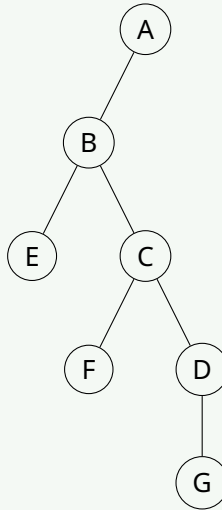
એપ્લિકેશન	ઉપયોગ
File System	Directory hierarchy
Expression Trees	ગાણિતિક expressions
Decision Trees	AI અને machine learning
Heap	Priority queues

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



આકૃતિ 5. General Tree

Binary Tree માં રૂપાંતર:



આકૃતિ 6. Converted Binary Tree

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:** માત્રક binary tree operations લાગુ પડે

#### મેમરી ટ્રીક

First-child સભે, Next-sibling જમણે