

# ડેટા સ્ટ્રકচર વિથ પાયથન (4331601) - સમર 2024 સોલ્યુશન

Milav Dabgar

June 06, 2024

## પ્રશ્ન 1(a) [3 ગુણ]

Array અને list નો તફાવત જણાવો.

### જવાબ

જવાબ:

કોષ્ટક 1. Array vs List

Array	List
નિશ્ચિત કદ બનાવતી વખતે	ગતિશીલ કદ - વધી/ઘટી શકે
સમાન પ્રકારનો ડેટા	મિશ્ર પ્રકારનો ડેટા
મેમરી કાર્યક્ષમ - સતત ફાળવણી	લવચીક પણ વધારે મેમરી
જડપી access ગણતરી 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

જવાબ:

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

Listing 1. Class and Object Example

```
1 class Student:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     def display(self):
7         print(f"Name: {self.name}, Age: {self.age}")
8
9 # Objects બનાવવા
10 s1 = Student("રામ", 20)
11 s2 = Student("સીતા", 19)
12 s1.display()
```

- **Class:** Template બનાવે છે
- **Object:** વાર્સટિક 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

#### જવાબ:

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

**Listing 2.** Constructor Types

```

1 class Demo:
2     # Default Constructor
3     def __init__(self):
4         self.value = 0
5
6     # Parameterized Constructor
7     def __init__(self, x, y=10):
8         self.x = x
9         self.y = y
10
11    # ઉપયોગ
12 d1 = Demo(5)    # x=5, y=10 (default)
13 d2 = Demo(3, 7) # x=3, y=7

```

#### Constructor ના પ્રકારો:

**કોષ્ટક 2.** Types of Constructors

પ્રકાર	વર્ણન	ઉપયોગ
Default	કોઈ parameters નહીં	Object initialization
Parameterized	Parameters સાથે	કર્ટમ 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 ની python program લખો.

ଜ୍ଵାବ

## ଜୀବାବ୍:

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

### **Listing 3. Polymorphism Example**

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

- **Method Overriding:** Child class मां समान method name
  - **Dynamic Binding:** Runtime पर method selection
  - **Code Reusability:** समान interface, अलग implementation

ਮੇਮਰੀ ਟੀਕ

## ଘણા Objects, એક Interface

### પ્રશ્ન 2(a) [3 ગુણ]

Python અંતર્ગત data structure List, Tuple અને Dictionary સમજાવો.

ଜ୍ଵାବ

ଜୀବାବୁ

### ક્રિકેટ 3. List vs Tuple vs Dictionary

Data Structure	ગુણાધર્મો	દેખરેણ
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}

ਮੇਮਰੀ ਡੀਕ

## Lists બદલાય, Tuples રહે, Dictionaries નકશો

## પ્રશ્ન 2(b) [4 ગુણ]

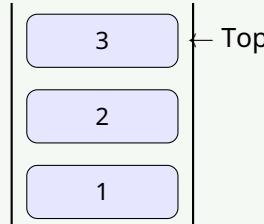
## Stack ની એપ્લિકેશન જગતાવો.

## જવાબ

### જવાબ:

#### Stack Applications:

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



આકૃતિ 1. Stack Structure

## 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 ના algorithm લખો.

## જવાબ

### જવાબ:

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

#### PUSH Algorithm:

1. Check કરો કે stack ભરેલો છે કે નહીં
2. જો ભરેલો હોય, print "Stack Overflow"
3. અન્યથા, top increment કરો
4. Top position પર element add કરો

#### POP Algorithm:

1. Check કરો કે stack ખાલી છે કે નહીં
2. જો ખાલી હોય, print "Stack Underflow"
3. અન્યથા, top થી element remove કરો
4. Top decrement કરો

#### ઉદાહરણ:

Listing 4. Stack Operations

```

1 stack = []
2 stack.append(10) # PUSH
3 stack.append(20) # PUSH
4 item = stack.pop() # POP returns 20

```

## મેમરી ટ્રીક

છેલ્લો અંદર, પહેલો બહાર - થાળીઓ જેવું

## પ્રશ્ન 2(a OR) [3 ગુણ]

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

### જવાબ

જવાબ:

કોષ્ટક 4. Complexity Terms

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	n્યૂનતમ 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 ફેરવો.  $A - (B / C + (D \% E * F) / G)^* H$

### જવાબ

જવાબ:

#### Step-by-step conversion:

- Infix:  $A - (B / C + (D \% E * F) / G)^* H$
- Result:  $A B C / D E \% F * G / + - H ^*$

#### Stack operations:

- Operators:  $-$ ,  $($ ,  $/$ ,  $+$ ,  $($ ,  $\%$ ,  $*$ ,  $)$ ,  $/$ ,  $), ^*$
- Final:  $A B C / D E \% F * G / + - H ^*$

Postfix પરિણામ:  $A B C / D E \% F * G / + - H ^*$

### મેમરી ટ્રીક

Operands પહેલા, Operators પછી

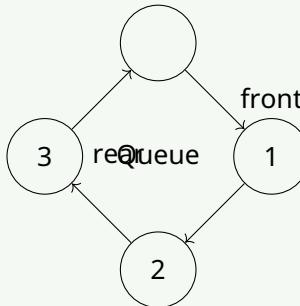
## પ્રશ્ન 2(c OR) [7 ગુણ]

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

### જવાબ

જવાબ:

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



આકૃતિ 2. Circular Queue

**INSERT Algorithm:**

1. Check કરો કે queue ભરેલો છે કે નહીં
2.  $\text{rear} = (\text{rear} + 1) \% \text{size}$
3.  $\text{queue}[\text{rear}] = \text{element}$
4. જો પ્રથમ element હોય, તો  $\text{front} = 0$  સેટ કરો

**DELETE Algorithm:**

1. Check કરો કે queue ખાલી છે કે નહીં
2.  $\text{element} = \text{queue}[\text{front}]$
3.  $\text{front} = (\text{front} + 1) \% \text{size}$
4. Element return કરો
  - ફાયદો: Memory efficiency
  - ઉપયોગ: CPU scheduling, buffering

**મેમરી ટ્રીક**

ભરાઈ જાય તો પાછા ફરો

**પ્રશ્ન 3(a) [3 ગુણા]**

List નો ઉપયોગ કરી Stack નું Implementation સમજાવો.

**જવાબ****જવાબ:**

Stack operations Python List વડે:

**Listing 5. Stack Implementation using List**

```

1 stack = [] # ખાલી stack
2 stack.append(10) # PUSH
3 stack.append(20) # PUSH
4 top = stack.pop() # POP

```

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

**મેમરી ટ્રીક**

Append ધકેલે, Pop ખેચે

## પ્રશ્ન 3(b) [4 ગુણ]

Linked list ની વિવિધ એપ્લિકેશન વિશે ચર્ચા કરો.

### જવાબ

#### જવાબ:

##### Linked List Applications:

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

કાંઈ 5. Linked List Applications

એપ્લિકેશન	ફાયદો	ઉપયોગ
Music Playlist	સરળ add/remove	Media players
Memory Management	Dynamic allocation	Operating systems
Polynomial Representation	કાર્યક્રમ storage	Mathematical operations

### મેમરી ટ્રીક

ગતિશીલ Implementation Undo Memory

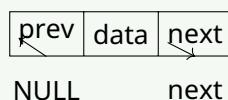
## પ્રશ્ન 3(c) [7 ગુણ]

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

### જવાબ

#### જવાબ:

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



આકૃતિ 3. Doubly Linked List Node

#### શરૂઆતથી Delete કરવાનો Algorithm:

1. જો list ખાલી હોય, તો return
2. જો માત્ર એક node હોય:
  - head = NULL
3. અન્યથાઃ
  - temp = head
  - head = head.next
  - head.prev = NULL
  - temp ને delete કરો

Listing 6. Delete from Beginning

```

1 def delete_beginning(self):
2     if self.head is None:
3         return
4     if self.head.next is None:
5         self.head = None
  
```

```

6     else:
7         self.head = self.head.next
8         self.head.prev = None

```

**મેમરી ટ્રીક**

બે દિશા નેવિગેશન

**પ્રશ્ન 3(a OR) [3 ગુણ]**

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

**જવાબ****જવાબ:****Step-by-step conversion:**

- Infix:  $A + B / C * D - E / F - G$
- Postfix:  $A B C / D * + E F / - G -$
- Operator precedence:  $*, / > +, -$
- ડાબેથી જમણો associativity

**મેમરી ટ્રીક**

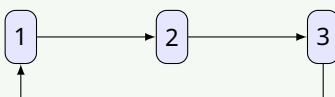
ગુણ ભાગ પહેલા, સરવાળો બાદબાકી પછી

**પ્રશ્ન 3(b OR) [4 ગુણ]**

Circular Linked List તેના ગેરફાયદા સાથે સમજાવો.

**જવાબ****જવાબ:**

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

**આકૃતિ 4. Circular Linked List****ગેરફાયદાઓ:**

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

**મેમરી ટ્રીક**

વર્તુળો મુંજવણ લાવી શકે

## પ્રશ્ન 3(c OR) [7 ગુણ]

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

### જવાબ

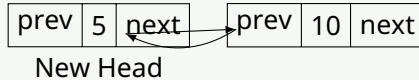
#### જવાબ:

**Listing 7. Insert in Doubly Linked List**

```

1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5         self.prev = None
6
7 class DoublyLinkedList:
8     def __init__(self):
9         self.head = None
10
11    def insert_beginning(self, data):
12        new_node = Node(data)
13        if self.head is None:
14            self.head = new_node
15        else:
16            new_node.next = self.head
17            self.head.prev = new_node
18            self.head = new_node

```



**આકૃતિ 5. Inserting 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

#### જવાબ:

#### Merge Sort Algorithm:

- જો array size  $\leq 1$  હોય, તો return
- Array ને બે ભાગમાં વહેંચો
- બને ભાગોને recursively sort કરો
- Sorted ભાગોને merge કરો

**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

જવાબ:

કોષ્ટક 6. Singly vs Doubly Linked List

Singly Linked List	Doubly Linked List
એક pointer (next)	બે pointers (next, prev)
આગામ ટ્રેવરસલ માત્ર	બંને દિશામાં ટ્રેવરસલ
ઓછી memory વપરાશ	વધારે memory વપરાશ
સરળ implementation	જાટેલ implementation

Singly: [data | next] → NULL

Doubly: NULL ← [prev | data | next] ↔ [prev | data | next] → NULL

આકૃતિ 6. Linked List Types

### મેમરી ટ્રીક

સિંગલ આગામ, ડબલ બંને દિશા

## પ્રશ્ન 4(c) [7 ગુણ]

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

### જવાબ

જવાબ:

**Selection Sort Algorithm:**

1. i = 0 થી n-2 સુધી:
  2. array[i...n-1] માં minimum શોધો
  3. Minimum ને array[i] સાથે swap કરો
- [13, 2, 6, 54, 18, 42, 11] માટે Trace:

કોષ્ટક 7. Selection Sort Trace

Pass	Array State	Min મળું	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 પસંદ કરો, સ્થાન બદલો

## પ્રશ્ન 4(a OR) [3 ગુણ]

Insertion sort નો algorithm લખો.

#### જવાબ

##### જવાબ:

##### Insertion Sort Algorithm:

1. i = 1 થી n-1 સુધી:
2. key = array[i]
3. j = i-1
4. જ્યાં સુધી j >= 0 અને 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 માટે

#### મેમરી ટ્રીક

યોગ્ય સ્થાને દાખલ કરો

## પ્રશ્ન 4(b OR) [4 ગુણ]

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

#### જવાબ

##### જવાબ:

##### Algorithm:

1. Data સાથે new\_node બનાવો
2. જો list ખાલી હોય:
  - head = new\_node
  - new\_node.next = new\_node
3. અન્યથાઃ
  - temp = head
  - જ્યાં સુધી temp.next != head:

- temp = temp.next
- temp.next = new\_node
- new\_node.next = head

Listing 8. Insert at End Circular Linked List

```

1 def insert_end(self, data):
2     new_node = Node(data)
3     if self.head is None:
4         self.head = new_node
5         new_node.next = new_node
6     else:
7         temp = self.head
8         while temp.next != self.head:
9             temp = temp.next
10        temp.next = new_node
11        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****જવાબ:****Bubble Sort Algorithm:**

- i = 0 થી n-2 સુધી:
  - j = 0 થી n-2-i સુધી:
  - જો array[j] > array[j+1]:
  - array[j] અને array[j+1] ને swap કરો
- [37, 22, 64, 84, 58, 52, 11] માટે Trace:

કોષ્ટક 8. Bubble Sort 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]

**મેમરી ટ્રીક**

સૌથી મોટા બબલ ઉપર

## પ્રશ્ન 5(a) [3 ગુણ]

Binary search tree અને તેની application સમજાવો.

### જવાબ

#### 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:

કોષ્ટક 9. BST Applications

એપ્લિકેશન	ફાયદો	ઉપયોગ
Database Indexing	ઝડપી 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

#### જવાબ:

Listing 9. 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
8 numbers = [10, 25, 30, 45, 60]
9 result = linear_search(numbers, 30)
10 print(f"Element found at index: {result}") # Output: 2

```

#### કામગીરી:

- ક્રમિક તપાસ: Element એ એલેમેન્ટ
- Time Complexity:  $O(n)$
- Space Complexity:  $O(1)$
- કામ કરે છે: Unsorted arrays પર

કોષ્ટક 10. Linear Search Trace

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

## મેમરી ટ્રીક

લીનિયર લાઇન દર લાઇન

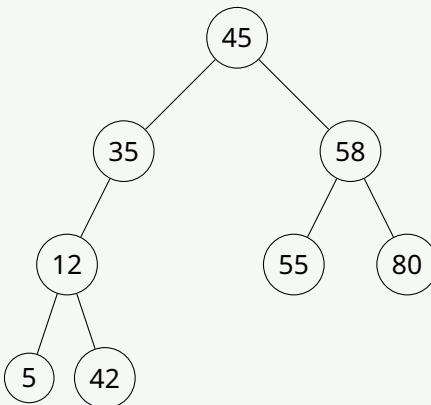
## પ્રશ્ન 5(c) [7 ગુણ]

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

## જવાબ

જવાબ:

BST બાંધકામ (duplicates અવગાળવામાં આવેલ):



અકૃતિ 7. Binary Search Tree

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:

## કોષ્ટક 11. 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

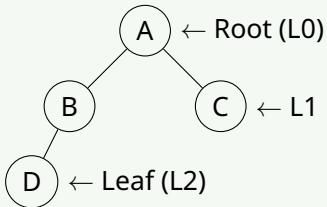
Pre-Root પહેલા, In-Sorted, Post-Root છેછે

## પ્રશ્ન 5(a OR) [3 ગુણ]

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

**જવાબ****જવાબ:****કોષ્ટક 12. Binary Tree Terms**

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

**આકૃતિ 8. Binary Tree Levels****મેમરી ટ્રીક**

Binary Levels લીડ કરે Leaves તરફ

**પ્રશ્ન 5(b OR) [4 ગુણ]**

Linear Search અને Binary search વચ્ચેનો તફાવત જણાવો.

**જવાબ****જવાબ:****કોષ્ટક 13. Linear vs Binary Search**

Linear Search	Binary Search
Unsorted arrays પર કામ કરે	Sorted array જરૂરી
ક્ષમિક તપાસ	ભાગલા પાડીને જીતો
Time: $O(n)$	Time: $O(\log n)$
સરળ implementation	જાટિયાં implementation
કોઈ preprocessing નહીં	Sorting જરૂરી

Linear: દરેકની તપાસ

Binary: મધ્ય તપાસો, ભાગલા પાડો

**આકૃતિ 9. Search Comparison****મેમરી ટ્રીક**

Linear લાઇન, Binary વિભાજન

## પ્રશ્ન 5(c OR) [7 ગુણ]

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

### જવાબ

#### જવાબ:

##### Insertion Algorithm:

1. જો root NULL છે, તો નવી node ને root બનાવો
2. જો data < root.data, તો left subtree માં insert કરો
3. જો data > root.data, તો right subtree માં insert કરો
4. જો data == root.data, તો insertion નહીં (duplicate)

##### Deletion Algorithm:

1. જો node leaf છે: સીધું delete કરો
2. જો node માં એક child છે: child સાથે બદલો
3. જો node માં બે children છે:
  - Inorder successor શોધો
  - Data ને successor ના data સાથે બદલો
  - Successor ને delete કરો

**Listing 10. BST Operations**

```

1 def insert(root, data):
2     if root is None:
3         return Node(data)
4     if data < root.data:
5         root.left = insert(root.left, data)
6     elif data > root.data:
7         root.right = insert(root.right, data)
8     return root
9
10 def delete(root, data):
11     if root is None:
12         return root
13     if data < root.data:
14         root.left = delete(root.left, data)
15     elif data > root.data:
16         root.right = delete(root.right, data)
17     else:
18         # Delete કરવાની node મળી
19         if root.left is None:
20             return root.right
21         elif root.right is None:
22             return root.left
23         # બે children સાથેની node
24         temp = find_min(root.right)
25         root.data = temp.data
26         root.right = delete(root.right, temp.data)
27     return root

```

#### કેસો:

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

### મેમરી ટ્રીક

Insert સરખાવો, Delete બદલો