# ડેટા સ્ટ્રક્ચર એન્ડ એપ્લિકેશન (1333203) - સમર 2024 સોલ્યુશન

Milav Dabgar

June 12, 2024

## પ્રશ્ન 1(a) [3 ગુણ]

### રેખીય ડેટા સ્ટ્રક્ચર વ્યાખ્યાયિત કરો અને તેના ઉદાહરણો આપો.

**જવાબ**

રેખીય ડેટા સ્ટ્રક્ચર એ એલિમેન્ટ્સનો એવો સંગ્રહ છે કે જેમાં દરેક એલિમેન્ટની પહેલાં અને પછી એક જ એલિમેન્ટ હોય છે (સિવાય કે પ્રથમ અને છેલ્લા એલિમેન્ટ).

**કોષ્ટક 1.** રેખીય ડેટા સ્ટ્રક્ચરના ઉદાહરણો

| ડેટા સ્ટ્રક્ચર | વર્ણન |
|---|---|
| Array | નિશ્ચિત સાઇઝનો એલિમેન્ટ્સનો સંગ્રહ જે ઇન્ડેક્સ દ્વારા ઍક્સેસ થાય છે |
| Linked List | નોડ્સની શ્રેણી જેમાં ડેટા અને આગળના નોડનો રેફરન્સ હોય છે |
| Stack | LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ) સ્ટ્રક્ચર |
| Queue | FIFO (ફર્સ્ટ ઇન ફર્સ્ટ આઉટ) સ્ટ્રક્ચર |

**મેમરી ટ્રીક**

``ALSQ are in a Line''

## પ્રશ્ન 1(b) [4 ગુણ]

### ટાઇમ અને સ્પેસ કોમ્પ્લેક્ષીટી વ્યાખ્યાયિત કરો.

**જવાબ**

ટાઇમ અને સ્પેસ કોમ્પ્લેક્સિટી એલ્ગોરિધમની કાર્યક્ષમતાને એક્ઝિક્યુશન ટાઇમ અને મેમરી વપરાશના સંદર્ભમાં માપે છે, જેમ ઇનપુટ સાઇઝ વધે છે.

**કોષ્ટક 2.** કોમ્પ્લેક્સિટી કમ્પેરિઝન

| કોમ્પ્લેક્સિટી પ્રકાર | વ્યાખ્યા | માપન | મહત્વ |
|---|---|---|---|
| **ટાઇમ કોમ્પ્લેક્સિટી** | એલ્ગોરિધમના એક્ઝિક્યુશન ટાઇમને ઇનપુટ સાઇઝના ફંક્શન તરીકે માપે છે | બિગ O નોટેશન (O(n), O(1), $O(n^2)$) | એલ્ગોરિધમ કેટલી ઝડપથી ચાલે છે તે નક્કી કરે છે |
| **સ્પેસ કોમ્પ્લેક્સિટી** | એલ્ગોરિધમને જરૂરી મેમરી સ્પેસને ઇનપુટ સાઇઝના ફંક્શન તરીકે માપે છે | બિગ O નોટેશન (O(n), O(1), $O(n^2)$) | એલ્ગોરિધમને કેટલી મેમરી જોઈએ છે તે નક્કી કરે છે |

**મેમરી ટ્રીક**

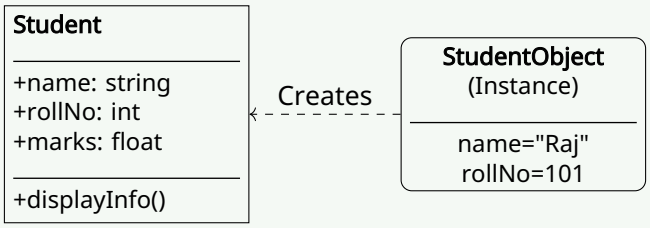
``TS: Time-Speed and Space-Storage''

## પ્રશ્ન 1(c) [7 ગુણ]

### ક્લાસ અને ઓબ્જેક્ટ ઉદાહરણ સાથે સમજાવો.

**જવાબ**

ક્લાસ અને ઓબ્જેક્ટ એ OOP ના મૂળભૂત કોન્સેપ્ટ છે જ્યાં ક્લાસ એ એટ્રિબ્યુટ્સ અને બિહેવિયર્સ ધરાવતા ઓબ્જેક્ટ બનાવવા માટેના બ્લુપ્રિન્ટ છે.




**આકૃતિ 1.** ક્લાસ અને ઓબ્જેક્ટ રિલેશનશિપ

**Listing 1.** કોડ ઉદાહરણ

```python
class Student:
    def __init__(self, name, rollNo, marks):
        self.name = name
        self.rollNo = rollNo
        self.marks = marks

    def displayInfo(self):
        print(f"Name: {self.name}, Roll No: {self.rollNo}, Marks: {self.marks}")

# ઓબ્જેક્ટ બનાવવા
student1 = Student("Raj", 101, 85.5)
student1.displayInfo()
```

- **ક્લાસ**: એટ્રિબ્યુટ્સ (name, rollNo, marks) અને મેથડ્સ (displayInfo) વ્યાખ્યાયિત કરતા બ્લુપ્રિન્ટ
- **ઓબ્જેક્ટ**: ક્લાસથી બનાવેલ ઇન્સ્ટન્સ (student1) જેમાં ચોક્કસ વેલ્યુઝ હોય છે

**મેમરી ટ્રીક**

``CAR''

## પ્રશ્ન 1(c) OR [7 ગુણ]

### ઇંસ્ટટંસ મેથડ, ક્લાસ મેથડ અને સ્ટેટિક મેથડ ઉદાહરણ સાથે સમજાવો.

**જવાબ**

Python ત્રણ પ્રકારની મેથડ્સને સપોર્ટ કરે છે: ઇન્સ્ટન્સ, ક્લાસ અને સ્ટેટિક મેથડ, દરેક અલગ હેતુ માટે વપરાય છે.

**કોષ્ટક 3.** મેથડ પ્રકારોની તુલના

| મેથડ પ્રકાર | ડેકોરેટર | પ્રથમ પેરામીટર | હેતુ | એક્સેસ |
|---|---|---|---|---|
| **ઇન્સ્ટન્સ મેથડ** | કોઈ નહીં | self | ઇન્સ્ટન્સ ડેટા પર કામ કરે | ઇન્સ્ટન્સ સ્ટેટને એક્સેસ/મોડિફાઇ કરી શકે |
| **ક્લાસ મેથડ** | @classmethod | cls | ક્લાસ ડેટા પર કામ કરે | ક્લાસ સ્ટેટને એક્સેસ/મોડિફાઇ કરી શકે |
| **સ્ટેટિક મેથડ** | @staticmethod | કોઈ નહીં | યુટિલિટી ફંક્શન્સ | ઇન્સ્ટન્સ કે ક્લાસ સ્ટેટને એક્સેસ કરી શકતી નથી |

**Listing 2.** કોડ ઉદાહરણ

```
class Student:
    school = "ABC School"
    def __init__(self, name):
        self.name = name
    def instance_method(self):
        return f"Hi {self.name} from {self.school}"
    @classmethod
    def class_method(cls):
        return f"School is {cls.school}"
    @staticmethod
    def static_method():
        return "This is a utility function"
```

**મેમરી ટ્રીક**

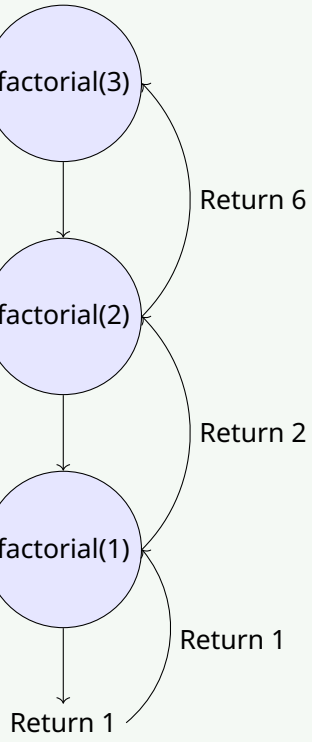
``ICS''

## પ્રશ્ન 2(a) [3 ગુણ]

### રીકર્ઝીવ ફંકશન નો કોંસેપ્ટ સમજાવો.

**જવાબ**

રિકર્સિવ ફંક્શન એ એવું ફંક્શન છે જે પોતાની એક્ઝિક્યુશન દરમિયાન સમાન સમસ્યાના નાના ઉદાહરણોને હલ કરવા માટે પોતાને જ કૉલ કરે છે.




**આકૃતિ 2.** રિકર્સિવ ફંક્શન એક્ઝિક્યુશન

**મેમરી ટ્રીક**

``BASE and RECURSE''

## પ્રશ્ન 2(b) [4 ગુણ]

**સ્ટેક અને ક્યુ વ્યાખ્યાયિત કરો.**

**જવાબ**

સ્ટેક અને ક્યુ એ લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં ડેટા ઇન્સર્શન અને રિમૂવલ માટે અલગ એક્સેસ પેટર્ન છે.

**કોષ્ટક 4.** સ્ટેક વિ. ક્યુ

| ફીચર | સ્ટેક | ક્યુ |
|---|---|---|
| **એક્સેસ પેટર્ન** | LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ) | FIFO (ફર્સ્ટ ઇન ફર્સ્ટ આઉટ) |
| **ઓપરેશન્સ** | પુશ (ઇન્સર્ટ), પૉપ (રિમૂવ) | એનક્યુ (ઇન્સર્ટ), ડિક્યુ (રિમૂવ) |
| **એક્સેસ પોઇન્ટ્સ** | સિંગલ એન્ડ (ટોપ) | ટુ એન્ડ્સ (ફ્રન્ટ, રિયર) |
| **વિઝ્યુઅલાઇઝેશન** | ઊભા થાંભલામાં ગોઠવેલી થાળીઓ જેવું | લાઇનમાં ઊભેલા લોકો જેવું |
| **એપ્લિકેશન્સ** | ફંક્શન કૉલ્સ, અનડુ ઓપરેશન્સ | પ્રિન્ટ જોબ્સ, પ્રોસેસ શેડ્યુલિંગ |

**મેમરી ટ્રીક**

``SLIFF vs QFIFF''

## પ્રશ્ન 2(c) [7 ગુણ]

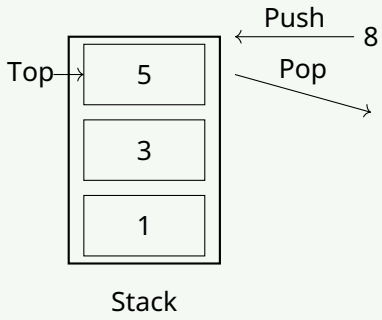
**સ્ટેક ના બેઝિક ઓપરેશન સમજાવો.**

**જવાબ**

સ્ટેક ઓપરેશન્સ LIFO (લાસ્ટ ઇન ફર્સ્ટ આઉટ) સિદ્ધાંતને અનુસરે છે.

**કોષ્ટક 5.** સ્ટેક ઓપરેશન્સ

| ઓપરેશન | વર્ણન | ટાઇમ કોમ્પ્લેક્સિટી |
|---|---|---|
| **પુશ** | ટોપ પર એલિમેન્ટ ઇન્સર્ટ કરવું | O(1) |
| **પૉપ** | ટોપથી એલિમેન્ટ રિમૂવ કરવું | O(1) |
| **પીક/ટોપ** | રિમૂવ કર્યા વિના ટોપ એલિમેન્ટ જોવું | O(1) |
| **isEmpty** | ચેક કરવું કે સ્ટેક ખાલી છે કે નહીં | O(1) |
| **isFull** | ચેક કરવું કે સ્ટેક ભરેલો છે કે નહીં | O(1) |




**આકૃતિ 3.** સ્ટેક ઓપરેશન્સ

**Listing 3.** Stack Implementation

```
class Stack:
    def __init__(self):
        self.items = []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        if not self.isEmpty():
            return self.items.pop()
    def isEmpty(self):
        return len(self.items) == 0
```

**મેમરી ટ્રીક**

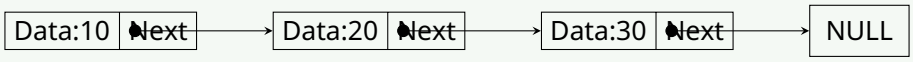
``PIPES''

## પ્રશ્ન 2(a) OR [3 ગુણ]

**સિંગલી લિંક્ડ લિસ્ટ વ્યાખ્યાયિત કરો.**

**જવાબ**

સિંગલી લિંક્ડ લિસ્ટ એ એક લીનિયર ડેટા સ્ટ્રક્ચર છે જેમાં નોડ્સનો કલેક્શન હોય છે જ્યાં દરેક નોડમાં ડેટા અને આગળના નોડનો રેફરન્સ હોય છે.




**આકૃતિ 4.** સિંગલી લિંક્ડ લિસ્ટ

**મેમરી ટ્રીક**

``DNL''

## પ્રશ્ન 2(b) OR [4 ગુણ]

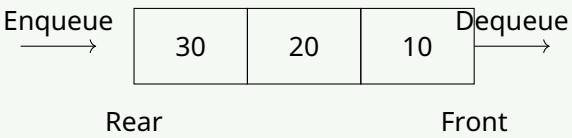
### ક્યુ ઉપર એનક્યુ ડીક્યુ ઓપરેશન સમજાવો.

**જવાબ**

એનક્યુ અને ડિક્યુ ક્યુ ડેટા સ્ટ્રક્ચરમાં એલિમેન્ટ્સ ઉમેરવા અને કાઢવા માટેના મુખ્ય ઓપરેશન્સ છે.

**કોષ્ટક 6.** ક્યુ ઓપરેશન્સ

| ઓપરેશન | વર્ણન | ઇમ્પ્લિમેન્ટેશન | ટાઇમ કોમ્પ્લેક્સિટી |
|---|---|---|---|
| **એનક્યુ** | રિયર એન્ડ પર એલિમેન્ટ ઉમેરવું | queue.append(element) | O(1) |
| **ડિક્યુ** | ફ્રન્ટ એન્ડથી એલિમેન્ટ કાઢવું | element = queue.pop(0) | O(1) linked list, O(n) array |




**આકૃતિ 5.** ક્યુ ઓપરેશન્સ

**મેમરી ટ્રીક**

``ERfDFr''

## પ્રશ્ન 2(c) OR [7 ગુણ]

### A+B/C+D પદ ને પોસ્ટફિક્સ મા ફેરવો અને STACK નો ઉપયોગ કરીને A,B,C અને D ની કોઇ યકમત ધારીને એનુ મુલ્ય શોધો.

**જવાબ**

"A+B/C+D" એક્સપ્રેશનને પોસ્ટફિક્સમાં કન્વર્ટ કરીને સ્ટેકનો ઉપયોગ કરીને તેનું મૂલ્યાંકન કરવું:
**સ્ટેપ 1: પોસ્ટફિક્સમાં કન્વર્ટ કરવું**

**કોષ્ટક 7.** ઇનફિક્સથી પોસ્ટફિક્સ કન્વર્ઝન

| સિમ્બોલ | સ્ટેક | આઉટપુટ | એક્શન |
|---|---|---|---|
| A | | A | આઉટપુટમાં ઉમેરો |
| + | + | A | સ્ટેકમાં પુશ કરો |
| B | + | A B | આઉટપુટમાં ઉમેરો |
| / | + / | A B | સ્ટેકમાં પુશ કરો (ઉચ્ચ પ્રિસિડન્સ) |
| C | + / | A B C | આઉટપુટમાં ઉમેરો |
| + | + | A B C / | પૉપ કરો, પુશ કરો |
| D | + | A B C / + D | આઉટપુટમાં ઉમેરો |
| End | | A B C / + D + | બાકીના પૉપ કરો |

**ફાઇનલ પોસ્ટફિક્સ:** $ABC/+D+$
**સ્ટેપ 2: વેલ્યુઝ A=5, B=10, C=2, D=3 સાથે મૂલ્યાંકન કરવું**

**કોષ્ટક 8.** પોસ્ટફિક્સ ઇવેલ્યુએશન

| સિમ્બોલ | સ્ટેક | કેલ્ક્યુલેશન |
|---|---|---|
| 5 (A) | 5 | વેલ્યુ પુશ કરો |
| 10 (B) | 5, 10 | વેલ્યુ પુશ કરો |
| 2 (C) | 5, 10, 2 | વેલ્યુ પુશ કરો |
| / | 5, 5 | $10/2 = 5$ |
| + | 10 | $5 + 5 = 10$ |
| 3 (D) | 10, 3 | વેલ્યુ પુશ કરો |
| + | 13 | $10 + 3 = 13$ |

**રિઝલ્ટ:** 13

**મેમરી ટ્રીક**

``PC-SE''

## પ્રશ્ન 3(a) [3 ગુણ]

**લિંક્ડ લિસ્ટ ના ઉપયોગો લખો.**

**જવાબ**

**કોષ્ટક 9.** લિંક્ડ લિસ્ટના ઉપયોગો

| એપ્લિકેશન | શા માટે લિંક્ડ લિસ્ટ વપરાય છે |
|---|---|
| **ડાયનેમિક મેમરી એલોકેશન** | રિએલોકેશન વિના કાર્યક્ષમ ઇન્સર્શન |
| **સ્ટેક અને ક્યુ** | જરૂરિયાત મુજબ વધી અને ઘટી શકે છે |
| **અનડુ ફંક્શનાલિટી** | હિસ્ટરી મેનેજમેન્ટ સરળ છે |
| **હેશ ટેબલ્સ** | કોલિઝન હેન્ડલિંગ માટે |
| **મ્યુઝિક પ્લેલિસ્ટ** | ગીતો વચ્ચે સરળ નેવિગેશન |

**મેમરી ટ્રીક**

``DSUHM''

## પ્રશ્ન 3(b) [4 ગુણ]

**પાયથનમા સિંગલી લિંક્ડ લિસ્ટ કેવી રીતે બનાવી શકાય એ સમજાવો.**

**જવાબ**

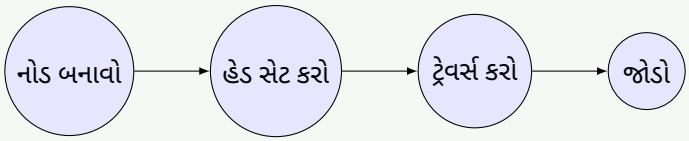
પાયથનમાં સિંગલી લિંક્ડ લિસ્ટ બનાવવા માટે નોડ ક્લાસ ડિફાઇન કરવી અને બેઝિક ઓપરેશન્સ ઇમ્પ્લિમેન્ટ કરવા પડે છે.

**Listing 4.** Creating Linked List

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

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

    def append(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node
```




**આકૃતિ 6.** લિસ્ટ બનાવવી

**મેમરી ટ્રીક**

``CHEN''

## પ્રશ્ન 3(c) [7 ગુણ]

**સિંગલી લિંક્ડ લિસ્ટ ની શરૂઆતમાં અને અંતમાં નવા નોડ ઉમેરવાનો કોડ લખો.**

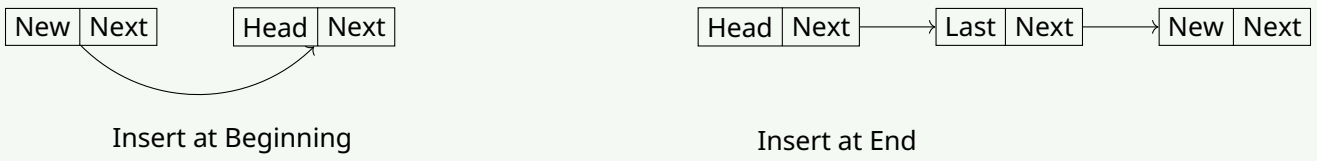
**જવાબ**

**Listing 5.** Insertion Code

```
def insert_at_beginning(self, data):
    new_node = Node(data)
    new_node.next = self.head
    self.head = new_node

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
```




**આકૃતિ 7.** ઇન્સર્શન ઓપરેશન્સ

**મેમરી ટ્રીક**

``BEN''

## પ્રશ્ન 3(a) OR [3 ગુણ]

**સિંગલી લિંક્ડ મા રહેલ નોડ ની સંખ્યા ગણવા માટેનો કોડ લખો.**

**જવાબ**

**Listing 6.** Count Nodes

```
def count_nodes(self):
    count = 0
    current = self.head
    while current:
        count += 1
        current = current.next
    return count
```

**મેમરી ટ્રીક**

``CIT''

## પ્રશ્ન 3(b) OR [4 ગુણ]

**કોલમ એ અને કોલમ બી ના યોગ્ય વિકલ્પ જોડો.**

**જવાબ**

**કોષ્ટક 10.** મેચ કરો

| કોલમ એ | કોલમ બી | મેચ |
|---|---|---|
| 1. સિંગલી લિંક્ડ લિસ્ટ | c. નોડ્સમાં ડેટા અને આગામી નોડનો સંદર્ભ હોય છે | 1-c |
| 2. ડબલી લિંક્ડ લિસ્ટ | d. નોડ્સમાં આગામી અને પાછલા બંને નોડ્સનો ડેટા અને સંદર્ભો હોય છે | 2-d |
| 3. સર્ક્યુલર લિંક્ડ લિસ્ટ | b. નોડ્સ એક લૂપ બનાવે જેમા છેલ્લો નોડ પ્રથમ નોડ તરફ નિર્દેશ કરે | 3-b |
| 4. નોડ | a. મૂળભૂત એકમ કે જેમા ડેટા અને સંદર્ભ હોઇ | 4-a |

Singly: $A \rightarrow B \rightarrow NULL$

Doubly: $A \leftrightarrow B \leftrightarrow NULL$

Circular: $A \rightarrow B \rightarrow A$

**આકૃતિ 8.** લિંક્ડ લિસ્ટ પ્રકારો

**મેમરી ટ્રીક**

``SDCN''

## પ્રશ્ન 3(c) OR [7 ગુણ]

**સિંગલી લિંક્ડ લિસ્ટ મા પ્રથમ અને છેલ્લો નોડ ને કાઢી નાખવાનુ સમજાવો.**

**જવાબ**

સિંગલી લિંક્ડ લિસ્ટમાંથી નોડ કાઢવાની જટિલતા પોઝિશન (પ્રથમ વિ. છેલ્લો) પર આધારિત હોય છે.

**કોષ્ટક 11.** ડિલીશન કંપેરિઝન

| પોઝિશન | અભિગમ | કોમ્પ્લેક્સિટી |
|---|---|---|
| પ્રથમ નોડ | હેડ પોઇન્ટર બદલો | O(1) |
| છેલ્લો નોડ | બીજા છેલ્લા નોડ સુધી ટ્રેવર્સ કરો | O(n) |

**Listing 7.** Deletion Code

```
def delete_first(self):
    if self.head is None: return
    self.head = self.head.next

def delete_last(self):
    if self.head is None: return
    if self.head.next is None:
        self.head = None
        return
    current = self.head
    while current.next.next:
        current = current.next
    current.next = None
```

**મેમરી ટ્રીક**

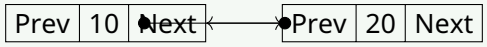
``FELO''

## પ્રશ્ન 4(a) [3 ગુણ]

**ડબ્લી લિંક્ડ લિસ્ટ નો કોન્સૈપ્ટ સમજાવો.**

**જવાબ**

ડબલી લિંક્ડ લિસ્ટ એ બાયડાયરેક્શનલ લીનિયર ડેટા સ્ટ્રક્ચર છે.




**આકૃતિ 9.** ડબલી લિંક્ડ લિસ્ટ

**મેમરી ટ્રીક**

``PDN''

## પ્રશ્ન 4(b) [4 ગુણ]

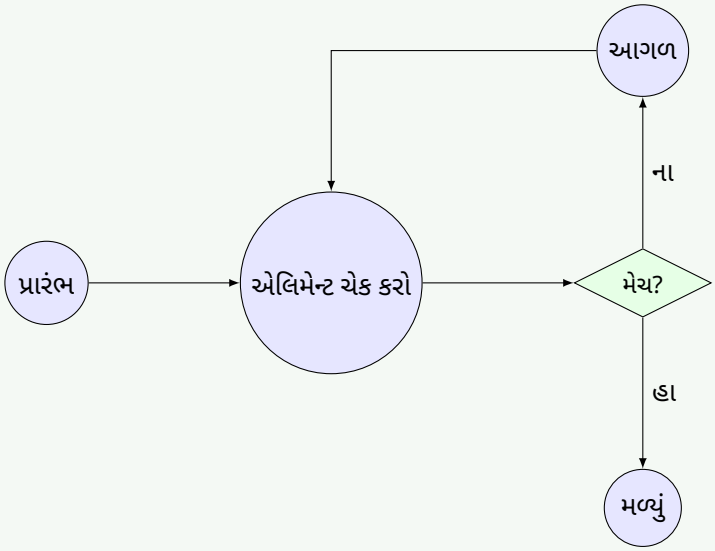
**લિનિયર સર્ચ નો કોન્સેપ્ટ સમજાવો.**

**જવાબ**

લિનિયર સર્ચ એ સરળ સિક્વેન્શિયલ સર્ચ અલ્ગોરિધમ છે.

**કોષ્ટક 12.** લિનિયર સર્ચ

| | |
|---|---|
| **કાર્યપ્રણાલી** | શરૂઆતથી અંત સુધી ક્રમશઃ દરેક એલિમેન્ટ ચેક કરો |
| **ટાઇમ કોમ્પ્લેક્સિટી** | O(n) - વર્સ્ટ અને એવરેજ કેસ |
| **બેસ્ટ કેસ** | O(1) |




**આકૃતિ 10.** લિનિયર સર્ચ પ્રોસેસ

**મેમરી ટ્રીક**

``SCENT''

## પ્રશ્ન 4(c) [7 ગુણ]

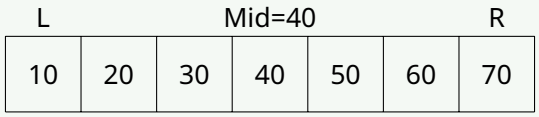
**બાયનરી સર્ચ અલ્ગોરિધમ ઇમ્પ્લીમેંટ કરવા માટેનો કોડ લખો.**

**જવાબ**

બાયનરી સર્ચ એક કાર્યક્ષમ અલ્ગોરિધમ છે જે સર્ચ ઇન્ટરવલને વારંવાર અડધા ભાગમાં વિભાજિત કરે છે.

**Listing 8.** Binary Search

```
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```




**આકૃતિ 11.** બાયનરી સર્ચ

**મેમરી ટ્રીક**

``MCLR''

## પ્રશ્ન 4(a) OR [3 ગુણ]

### સિલેક્શન સોર્ટ અલ્ગોરીધમ નો કોન્સૈપ્ટ સમજાવો.

**જવાબ**

સિલેક્શન સોર્ટ અનસોર્ટેડ ભાગમાંથી મિનિમમ એલિમેન્ટ શોધીને શરૂઆતમાં મૂકે છે.

- **ટાઇમ કોમ્પ્લેક્સિટી**: $O(n^2)$
- **સ્પેસ કોમ્પ્લેક્સિટી**: $O(1)$

**મેમરી ટ્રીક**

``FSMR''

## પ્રશ્ન 4(b) OR [4 ગુણ]

### બબલ સોર્ટ મેથડ સમજાવો.

**જવાબ**

બબલ સોર્ટ આસપાસના એલિમેન્ટ્સની તુલના કરે છે અને જો તેઓ ખોટા ક્રમમાં હોય તો તેમને સ્વેપ કરે છે.

**કોષ્ટક 13.** બબલ સોર્ટ

| પાસ | $n-1$ પાસ |
|---|---|
| કોમ્પ્લેક્સિટી | $O(n^2)$ |

[5, 3, 8, 4, 2]
↓
[3, 5, 4, 2, 8] (પાસ 1)
↓
[3, 4, 2, 5, 8] (પાસ 2)

**આકૃતિ 12.** બબલ સોર્ટ

**મેમરી ટ્રીક**

``CABS''

## પ્રશ્ન 4(c) OR [7 ગુણ]

**ઉદાહરણ સાથે ક્વીક સોર્ટ મેથડનુ વર્કિંગ સમજાવો.**

**જવાબ**

ક્વિક સોર્ટ એ ડિવાઇડ-એન્ડ-કોન્કર અલ્ગોરિધમ છે.

**કોષ્ટક 14.** ક્વિક સોર્ટ સ્ટેપ્સ

| | |
|---|---|
| 1 | પિવોટ એલિમેન્ટ પસંદ કરો |
| 2 | પાર્ટિશન: નાના ડાબી બાજુ, મોટા જમણી બાજુ |
| 3 | રિકર્સિવલી સોર્ટ કરો |

Pivot: 4
Left: [2, 1, 3] Right: [7, 6, 8, 5]

**આકૃતિ 13.** ક્વિક સોર્ટ પાર્ટિશન

**મેમરી ટ્રીક**

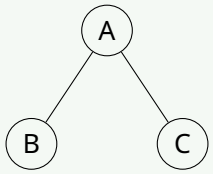
``PPR''

## પ્રશ્ન 5(a) [3 ગુણ]

**બાયનરી ટ્રી સમજાવો.**

**જવાબ**

બાયનરી ટ્રી એ એક હાયરાર્કિકલ ડેટા સ્ટ્રક્ચર છે જેમાં દરેક નોડને વધુમાં વધુ બે ચિલ્ડ્રન હોય છે.




**આકૃતિ 14.** બાયનરી ટ્રી

**મેમરી ટ્રીક**

``RLTM''

## પ્રશ્ન 5(b) [4 ગુણ]

**ટ્રી ના સંદભભ મા રૂટ, પાથ, પેરંટ અને ચિલ્ડ્રન પદો વ્યાખ્યાયિત કરો.**

**જવાબ**

**કોષ્ટક 15.** ટ્રી શબ્દાવલી

| પદ | વ્યાખ્યા |
|---|---|
| **રૂટ** | સૌથી ઉપરનો નોડ જેને કોઈ પેરેન્ટ નથી |
| **પાથ** | નોડ્સનો સિક્વન્સ |
| **પેરેન્ટ** | નોડ જે ચાઇલ્ડ ધરાવે છે |
| **ચિલ્ડ્રન** | પેરેન્ટ નોડથી સીધા જોડાયેલા નોડ્સ |

**મેમરી ટ્રીક**

``RPPC''

## પ્રશ્ન 5(c) [7 ગુણ]

**નીચે આપેલા ટ્રી માટે પ્રીઓર્ડર અને પોસ્ટઓર્ડર ટ્રાવર્સલ લાગુ કરો.**

**જવાબ**

આપેલ ટ્રી: 40(Root), Left:30, Right:50...

**કોષ્ટક 16.** ટ્રાવર્સલ

| ટ્રાવર્સલ | ઓર્ડર | રિઝલ્ટ |
|---|---|---|
| **પ્રીઓર્ડર** | રૂટ, લેફ્ટ, રાઇટ | 40, 30, 25, 15, 28, 35, 50, 45, 60, 55, 70 |
| **પોસ્ટઓર્ડર** | લેફ્ટ, રાઇટ, રૂટ | 15, 28, 25, 35, 30, 45, 55, 70, 60, 50, 40 |

**મેમરી ટ્રીક**

``PRE-NLR, POST-LRN''

## પ્રશ્ન 5(a) OR [3 ગુણ]

**બાયનરી ટ્રી ની એપ્લિકેશન્સ લખો.**

**જવાબ**

- બાયનરી સર્ચ ટ્રી (સર્ચિંગ)
- એક્સપ્રેશન ટ્રી (મેથેમેટિકલ)
- હફમેન કોડિંગ (કમ્પ્રેશન)
- પ્રાયોરિટી ક્યુ (હીપ)
- ડિસિઝન ટ્રી (મશીન લર્નિંગ)

**મેમરી ટ્રીક**

``BEHPD''

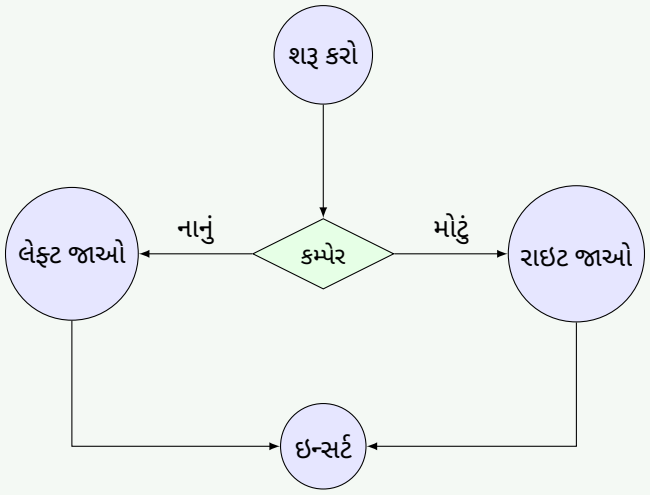
## પ્રશ્ન 5(b) OR [4 ગુણ]

**બાયનરી સર્ચ ટ્રી મા નોડ કેવી રીતે ઉમેરી શકાય તે સમજાવો.**

**જવાબ**

BST પ્રોપર્ટી: લેફ્ટ $<$ નોડ $<$ રાઇટ.

1. રૂટથી શરૂ કરો.
2. જો નવી $<$ કરંટ, લેફ્ટ જાઓ.
3. જો નવી $>$ કરંટ, રાઇટ જાઓ.
4. ખાલી પોઝિશન પર ઇન્સર્ટ કરો.




**આકૃતિ 15.** BST ઇન્સર્શન

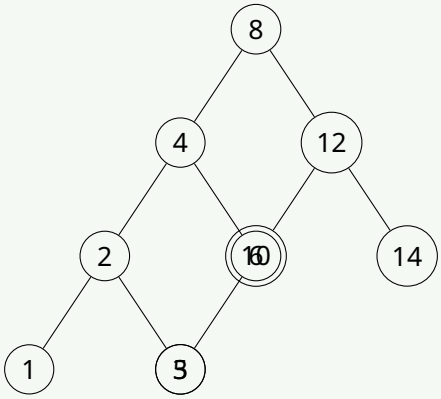
**મેમરી ટ્રીક**

``LSRG''

## પ્રશ્ન 5(c) OR [7 ગુણ]

**8, 4, 12, 2, 6, 10, 14, 1, 3, 5 નમ્બર માટે બાયનરી સર્ચ ટ્રી દોરો અને ટ્રી માટે ઇન ઓર્ડર ટ્રાવર્સલ લખો.**

**જવાબ**




**આકૃતિ 16.** બનાવેલ BST

**ઇન-ઓર્ડર ટ્રાવર્સલ (લેફ્ટ, રૂટ, રાઇટ):**
1, 2, 3, 4, 5, 6, 8, 10, 12, 14

**મેમરી ટ્રીક**

``LNR''