પ્રશ્ન 1(અ) [3 ગુણ]

વ્યાખ્યાયિત કરો બિગ- ઓ નોટેશન, બિગ ઓમેગા નોટેશન, બિગ થીટા નોટેશન.

જવાબ:

સારણી: એસિમ્પ્ટોટિક નોટેશન સરખામણી

નોટેશન	પ્રતીક	વર્ણન	ઉપયોગ
બિગ-ઓ	O(f(n))	**ઉપલી હદ**	સૌથી ખરાબ કેસ
બિગ ઓમેગા	Ω(f(n))	**નીચલી હદ**	સૌથી સારો કેસ
બિગ થીટા	Θ(f(n))	**ચુસ્ત હદ**	સરેરાશ કેસ

- **બિગ-ઓ નોટેશન**: મહત્તમ સમય/સ્થળ જટિલતા વર્ણવે છે
- **બિગ ઓમેગા**: ન્યૂનતમ સમય/સ્થળ જટિલતા વર્ણવે છે
- **બિગ થીટા**: ચોક્કસ સમય/સ્થળ જટિલતા વર્ણવે છે

મેમરી ટ્રીક: "OWT - O ખરાબ માટે, Omega શ્રેષ્ઠ માટે, Theta ચુસ્ત માટે"

પ્રશ્ન 1(બ) [4 ગુણ]

સેટ વ્યાખ્યાયિત કરો. સેટ પર કરી શકાય તેવા વિવિધ ઓપરેશનો લખો.

જવાબ:

વ્યાખ્યા: સેટ એ અનન્ય તત્વોનો સંગ્રહ છે જેમાં કોઈ ડુપ્લિકેટ નથી.

સારણી: સેટ ઓપરેશનો

ઓપરેશન	પ્રતીક	વર્ણન	ઉદાહરણ
યુનિયન	A ∪ B	બધા તત્વો જોડે છે	{1,2} ∪ {2,3} = {1,2,3}
ઇન્ટરસેક્શન	A ∩ B	સામાન્ય તત્વો	{1,2} ∩ {2,3} = {2}
ડિફરન્સ	A - B	A માં છે પણ B માં નથી	{1,2} - {2,3} = {1}
સબસેટ	A ⊆ B	A ના બધા તત્વો B માં છે	{1} ⊆ {1,2} = સાચું

- **ઉમેરવું/દાખલ કરવું**: નવું તત્વ ઉમેરવું
- **દૂર કરવું/કાઢવું**: અસ્તિત્વમાં રહેલું તત્વ દૂર કરવું
- **સમાવેશ**: તત્વ અસ્તિત્વમાં છે કે નહીં તપાસવું

મેમરી ટ્રીક: "UIDS - યુનિયન, ઇન્ટરસેક્શન, ડિફરન્સ, સબસેટ"

પ્રશ્ન 1(ક) [7 ગુણ]

ક્રિકેટર માટે Python ક્લાસ લખો. ક્લાસ માં ક્રિકેટરનું નામ, ટીમનું નામ અને ડેટા સભ્યો તરીકે રનનો સમાવેશ થાય છે. ક્લાસ કાર્યો નીચે મુજબ છે: ડેટા સભ્યોને ઇનિશિયલાઇઝ કરવા, રન સેટ કરવા અને રન ડિસ્પ્લે કરવા.

જવાબ:

```
class Cricketer:
    def __init__(self, name="", team="", run=0):
        self.name = name
        self.team = team
        self.run = run

    def set_run(self, run):
        self.run = run

    def display_run(self):
        print(f"ખેલાડી: {self.name}")
        print(f"ટીમ: {self.team}")
        print(f"રન: {self.run}")

# ઉદાહરણ ઉપયોગ
player = Cricketer("વિરાટ કોહલી", "ભારત", 100)
player.display_run()
```

- **કન્સ્ટ્રક્ટર**: નામ, ટીમ અને રન ઇનિશિયલાઇઝ કરે છે
- **set_run()**: રન વેલ્યુ અપડેટ કરે છે
- **display_run()**: ખેલાડીની માહિતી બતાવે છે

મેમરી ટ્રીક: "CSD - કન્સ્ટ્રક્ટર, સેટ, ડિસ્પ્લે"

પ્રશ્ન 1(ક અથવા) [7 ગુણ]

વિદ્યાર્થીની માહિતી વાંચવા અને પ્રદર્શિત કરવા માટે વિદ્યાર્થી ક્લાસની રચના કરો, અને તેમાં getInfo() અને displayInfo() પદ્ધતિઓનો ઉપયોગ કરવામાં આવશે. જ્યાં getInfo() પ્રાઇવેટ પદ્ધતિ હશે.

જવાબ:

```
class Student:
    def __init__(self):
        self.name = ""
        self.roll_no = ""
        self.marks = 0
        self.__getInfo()  # પ્રાઇવેટ મેથડ કૉલ

    def __getInfo__(self):  # પ્રાઇવેટ મેથડ
        self.name = input("નામ દાખલ કરો: ")
        self.roll_no = input("રોલ નંબર દાખલ કરો: ")
        self.marks = int(input("માર્ક્સ દાખલ કરો: "))

    def displayInfo(self):
        print(f"નામ: {self.name}")
```

```
        print(f"રોલ નંબર: {self.roll_no}")
        print(f"માર્ક્સ: {self.marks}")

# ઉદાહરણ ઉપયોગ
student = Student()
student.displayInfo()
```

- **પ્રાઇવેટ મેથડ**: ડબલ અંડરસ્કોર (__getInfo) વાપરે છે
- **કન્સ્ટ્રક્ટર**: આપોઆપ પ્રાઇવેટ મેથડ કૉલ કરે છે
- **પબ્લિક મેથડ**: displayInfo() વિદ્યાર્થીનો ડેટા બતાવે છે

મેમરી ટ્રીક: "PCP - પ્રાઇવેટ, કન્સ્ટ્રક્ટર, પબ્લિક"

પ્રશ્ન 2(અ) [3 ગુણ]

સ્ટેક અને ક્યૂ વચ્ચે તફાવત કરો.

જવાબ:

સારણી: સ્ટેક વર્સસ ક્યૂ સરખામણી

લક્ષણ	સ્ટેક	ક્યૂ
ક્રમ	LIFO (છેલ્લું અંદર, પહેલું બહાર)	FIFO (પહેલું અંદર, પહેલું બહાર)
ઓપરેશનો	Push, Pop	Enqueue, Dequeue
એક્સેસ પોઇન્ટ	એક છેડો (ટોપ)	બે છેડા (ફ્રન્ટ અને રિયર)
ઉદાહરણ	પ્લેટનો સ્ટેક	બેંકની કતાર

- **સ્ટેક**: પુસ્તકોના ઢગલા જેવું - છેલ્લું ઉમેર્યું, પહેલું કાઢ્યું
- **ક્યૂ**: રાહ જોવાની લાઇન જેવું - પહેલું આવ્યું, પહેલું સેવા મળી

મેમરી ટ્રીક: "SLIF QFIF - સ્ટેક LIFO, ક્યૂ FIFO"

પ્રશ્ન 2(બ) [4 ગુણ]

રિકર્સન વ્યાખ્યાયિત કરો. ઉદાહરણ સાથે સમજાવો.

જવાબ:

વ્યાખ્યા: ફંક્શન પોતાને જ નાની સમસ્યા સાથે કૉલ કરવું જ્યાં સુધી બેઝ કંડિશન ન મળે.

```
def factorial(n):
    # બેઝ કેસ
    if n <= 1:
        return 1
    # રિકર્સિવ કેસ
    return n * factorial(n-1)

# ઉદાહરણ: factorial(3)
# 3 * factorial(2)
# 3 * 2 * factorial(1)
# 3 * 2 * 1 = 6
```

- **બેઝ કેસ**: રોકવાની શરત
- **રિકર્સિવ કેસ**: ફંક્શન પોતાને કૉલ કરે છે
- **સમસ્યા ઘટાડવી**: દરેક કૉલ નાની સમસ્યા હલ કરે છે

મેમરી ટ્રીક: "BRP - બેઝ, રિકર્સિવ, પ્રોબ્લેમ-રિડક્શન"

પ્રશ્ન 2(ક) [7 ગુણ]

સ્ટેકના સાઈઝ 5 તરીકે ધ્યાનમાં લો. સ્ટેક પર નીચેની કામગીરી લાગૂ કરો અને દરેક ઑપરેશન પછી સ્ટેટસ અને ટોપ પોઇન્ટર બતાવો. Push a,b,c pop

જવાબ:

સ્ટેક ઓપરેશનો ટ્રેસ:

```
શરૂઆતની સ્થિતિ:
સ્ટેક: [ _ _ _ _ _ ]  ટોપ: -1
       0 1 2 3 4

Push 'a' પછી:
સ્ટેક: [ a _ _ _ _ ]  ટોપ: 0
       0 1 2 3 4

Push 'b' પછી:
સ્ટેક: [ a b _ _ _ ]  ટોપ: 1
       0 1 2 3 4

Push 'c' પછી:
સ્ટેક: [ a b c _ _ ]  ટોપ: 2
       0 1 2 3 4

Pop પછી:
સ્ટેક: [ a b _ _ _ ]  ટોપ: 1
       0 1 2 3 4
કાઢેલું તત્વ: c
```

- **Push ઓપરેશનો**: ઇન્ડેક્સ 0 થી શરૂ કરીને તત્વો ઉમેરે છે

- **ટોપ પોઇન્ટર**: છેલ્લે દાખલ કરેલા તત્વ તરફ પોઇન્ટ કરે છે
- **Pop ઓપરેશન**: ટોપ તત્વ દૂર કરે છે, ટોપ પોઇન્ટર ઘટાડે છે

મેમરી ટ્રીક: "PTD - Push ટોપ ઘટાડવું"

પ્રશ્ન 2(અ અથવા) [3 ગુણ]

સ્ટેક અને ક્યૂની એપ્લિકેશનોની સૂચિ બનાવો.

જવાબ:

સારણી: સ્ટેક અને ક્યૂની એપ્લિકેશનો

ડેટા સ્ટ્રક્ચર	એપ્લિકેશનો
સ્ટેક	ફંક્શન કૉલ્સ, અન્ડુ ઓપરેશનો, એક્સપ્રેશન ઇવેલ્યુએશન, બ્રાઉઝર હિસ્ટરી
ક્યૂ	પ્રોસેસ શેડ્યુલિંગ, પ્રિન્ટર ક્યૂ, BFS ટ્રેવર્સલ, રિક્વેસ્ટ હેન્ડલિંગ

- **સ્ટેક એપ્લિકેશનો**: અન્ડુ-રિડુ, રિકર્સન, પાર્સિંગ
- **ક્યૂ એપ્લિકેશનો**: ટાસ્ક શેડ્યુલિંગ, બફરિંગ, બ્રેડથ-ફર્સ્ટ સર્ચ

મેમરી ટ્રીક: "સ્ટેક FUBE, ક્યૂ SPBH"

પ્રશ્ન 2(બ અથવા) [4 ગુણ]

સ્ટેકનો ઉપયોગ કરીને નીચેના સમીકરણને પોસ્ટફિક્સ નોટેશનમાં કન્વર્ટ કરો: i) (a*b)(c^d(d+e)-f) ii) a-b/(c*d/e)

જવાબ:

i) (a*b)(c^d(d+e)-f)

પ્રતીક	સ્ટેક	આઉટપુટ
(	(	
a	(	a
*	(*	a
b	(*	ab
)		ab*
*	*	ab*
(	*(	ab*
c	*(	ab*c
^	*(^	ab*c
d	*(^	ab*cd
(	*(^(	ab*cd
d	*(^(	ab*cdd
+	*(^(+	ab*cdd
e	*(^(+	ab*cdde
)	*(^	ab*cdde+
)	*	ab*cdde+^
-	*-	ab*cdde+^
f	*-	ab*cdde+^f
		ab*cdde+^f-*

પરિણામ: ab*cdde+^f-*

ii) a-b/(c*d/e)

પરિણામ: abcd*e/-

મેમરી ટ્રીક: "PEMDAS પોસ્ટફિક્સ માટે ઉલટું"

પ્રશ્ન 2(ક અથવા) [7 ગુણ]

લીસ્ટનો ઉપયોગ કરીને ક્યૂને અમલમાં મૂકવા માટે એક પ્રોગ્રામ ડેવલોપ કરો જે નીચેની કામગીરી કરે છે: enqueue, dequeue.

જવાબ:

```
class Queue:
```

```
    def __init__(self):
        self.queue = []
        self.front = 0
        self.rear = -1

    def enqueue(self, item):
        self.queue.append(item)
        self.rear += 1
        print(f"એનક્યૂ કર્યું: {item}")

    def dequeue(self):
        if self.front <= self.rear:
            item = self.queue[self.front]
            self.front += 1
            print(f"ડીક્યૂ કર્યું: {item}")
            return item
        else:
            print("ક્યૂ ખાલી છે")
            return None

    def display(self):
        if self.front <= self.rear:
            print("ક્યૂ:", self.queue[self.front:self.rear+1])
        else:
            print("ક્યૂ ખાલી છે")

# ઉદાહરણ ઉપયોગ
q = Queue()
q.enqueue('A')
q.enqueue('B')
q.dequeue()
q.display()
```

- **Enqueue**: રિયર પર તત્વ ઉમેરે છે
- **Dequeue**: ફ્રન્ટ પરથી તત્વ દૂર કરે છે
- **FIFO સિદ્ધાંત**: પહેલું અંદર, પહેલું બહાર

મેમરી ટ્રીક: "ERF - Enqueue રિયર, ફ્રન્ટ"

પ્રશ્ન 3(અ) [3 ગુણ]

લિંક લિસ્ટના પ્રકારોની સૂચિ બનાવો. દરેક પ્રકારનું ગ્રાફિકલ રજૂઆત આપો.

જવાબ:

સારણી: લિંક લિસ્ટના પ્રકારો

પ્રકાર	વર્ણન	ડાયાગ્રામ
સિંગલી	એક દિશા પોઇન્ટર	A→B→C→NULL
ડબલી	બે દિશા પોઇન્ટરો	NULL←A⇄B⇄C→NULL
સર્ક્યુલર	છેલ્લું પહેલા તરફ પોઇન્ટ કરે	A→B→C→A

```
સિંગલી લિંક લિસ્ટ:
[ડેટા|નેક્સ્ટ] -> [ડેટા|નેક્સ્ટ] -> [ડેટા|NULL]

ડબલી લિંક લિસ્ટ:
[પ્રિવ|ડેટા|નેક્સ્ટ] <-> [પ્રિવ|ડેટા|નેક્સ્ટ] <-> [પ્રિવ|ડેટા|નેક્સ્ટ]

સર્ક્યુલર લિંક લિસ્ટ:
[ડેટા|નેક્સ્ટ] -> [ડેટા|નેક્સ્ટ] -> [ડેટા|નેક્સ્ટ]
    ^                                |
    |________________________________|
```

મેમરી ટ્રીક: "SDC - સિંગલી, ડબલી, સર્ક્યુલર"

પ્રશ્ન 3(બ) [4 ગુણ]

સિંગલી લિંક લિસ્ટમાં આપેલ નોડ શોધવા માટે એક અલ્ગોરિધમ લખો.

જવાબ:

```python
def search_node(head, key):
    current = head
    position = 0

    while current is not None:
        if current.data == key:
            return position
        current = current.next
        position += 1

    return -1  # નહીં મળ્યું

# અલ્ગોરિધમ સ્ટેપ્સ:
# 1. હેડ થી શરૂ કરો
# 2. વર્તમાન ડેટાને કી સાથે સરખાવો
# 3. જો મળ્યું તો પોઝિશન રિટર્ન કરો
# 4. આગલા નોડ પર જાઓ
# 5. અંત સુધી પુનરાવર્તન કરો
```

- **લીનિયર સર્ચ**: હેડ થી ટેઇલ સુધી ટ્રાવર્સ કરો
- **ટાઇમ કોમ્પ્લેક્સિટી**: O(n)
- **રિટર્ન**: મળ્યું તો પોઝિશન, નહીં તો -1

મેમરી ટ્રીક: "SCMR - શરૂ, સરખાવો, આગળ વધો, રિટર્ન"

પ્રશ્ન 3(ક) [7 ગુણ]

સિંગલી લિંક લિસ્ટ પર પર નીચેની કામગીરી કરવા માટે પ્રોગ્રામનો અમલ કરો: 1)સિંગલી લિંક લિસ્ટ ની શરૂઆતમાં નોડ દાખલ કરો 2)સિંગલી લિંક લિસ્ટની શરૂઆતથી નોડ કાઢી નાખો

જવાબ:

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

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

    def insert_at_beginning(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
        print(f"શરૂઆતમાં {data} દાખલ કર્યું")

    def delete_from_beginning(self):
        if self.head is None:
            print("લિસ્ટ ખાલી છે")
            return None

        deleted_data = self.head.data
        self.head = self.head.next
        print(f"શરૂઆતથી {deleted_data} કાઢ્યું")
        return deleted_data

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("NULL")

# ઉદાહરણ ઉપયોગ
ll = SinglyLinkedList()
ll.insert_at_beginning(10)
ll.insert_at_beginning(20)
ll.delete_from_beginning()
ll.display()
```

- **ઇન્સર્ટ**: નોડ બનાવો, હેડ સાથે જોડો, હેડ અપડેટ કરો
- **ડિલીટ**: ડેટા સ્ટોર કરો, હેડને આગળ ખસેડો, ડેટા રિટર્ન કરો

મેમરી ટ્રીક: "CLU - બનાવો, જોડો, અપડેટ"

પ્રશ્ન 3(અ અથવા) [3 ગુણ]

સર્ક્યુલર લિંક લિસ્ટ અને સિંગલી લિંક લિસ્ટ વચ્ચે તફાવત કરો.

જવાબ:

સારણી: સર્ક્યુલર વર્સસ સિંગલી લિંક લિસ્ટ

લક્ષણ	સિંગલી લિંક લિસ્ટ	સર્ક્યુલર લિંક લિસ્ટ
છેલ્લો નોડ પોઇન્ટ કરે છે	NULL	પહેલા નોડ (હેડ)
ટ્રાવર્સલ	લીનિયર (એક દિશા)	સર્ક્યુલર (સતત)
અંત ડિટેક્શન	next == NULL	next == head
મેમરી	ઓછી (વધારાનું પોઇન્ટર નહીં)	સમાન સ્ટ્રક્ચર

- **સર્ક્યુલર ફાયદો**: NULL પોઇન્ટરો નહીં, સતત ટ્રાવર્સલ
- **સિંગલી ફાયદો**: સાદું અમલીકરણ, સ્પષ્ટ અંત

મેમરી ટ્રીક: "CNTE - સર્ક્યુલર કોઈ સમાપ્તિ અંત નહીં"

પ્રશ્ન 3(બ અથવા) [4 ગુણ]

સંક્ષિપ્તમાં લિંક લિસ્ટ સૂચિની ત્રણ એપ્લિકેશનો સમજાવો.

જવાબ:

સારણી: લિંક લિસ્ટ એપ્લિકેશનો

એપ્લિકેશન	વર્ણન	ફાયદો
ડાયનામિક મેમરી એલોકેશન	મેમરી બ્લોક્સ મેનેજ કરે છે	કાર્યક્ષમ મેમરી ઉપયોગ
સ્ટેક/ક્યૂનું અમલીકરણ	લિંક સ્ટ્રક્ચર ઉપયોગ કરે છે	ડાયનામિક સાઇઝ
પોલિનોમિયલ રજૂઆત	ગુણાંક અને પાવર સ્ટોર કરે છે	સરળ અંકગણિત ઓપરેશનો

- **મ્યુઝિક પ્લેલિસ્ટ**: ગીતો ડાયનામિક ઉમેરવા/દૂર કરવા
- **બ્રાઉઝર હિસ્ટરી**: પાછળ/આગળ નેવિગેટ કરવા
- **ઇમેજ વ્યૂઅર**: પહેલું/આગલું ઇમેજ નેવિગેશન

મેમરી ટ્રીક: "DIP - ડાયનામિક, અમલીકરણ, પોલિનોમિયલ"

પ્રશ્ન 3(ક અથવા) [7 ગુણ]

સર્ક્યુલર લિંક લિસ્ટ ને બનાવવા અને પ્રદર્શિત કરવા માટે એક પ્રોગ્રામ ડેવલોપ કરો.

જવાબ:

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

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

    def insert(self, data):
        new_node = Node(data)

        if self.head is None:
            self.head = new_node
            new_node.next = self.head
        else:
            current = self.head
            while current.next != self.head:
                current = current.next
            current.next = new_node
            new_node.next = self.head

    def display(self):
        if self.head is None:
            print("લિસ્ટ ખાલી છે")
            return

        current = self.head
        print("સર્ક્યુલર લિસ્ટ:")
        while True:
            print(current.data, end=" -> ")
            current = current.next
            if current == self.head:
                break
        print(f"{self.head.data} (હેડ પર પાછા)")

# ઉદાહરણ ઉપયોગ
cll = CircularLinkedList()
cll.insert(10)
cll.insert(20)
cll.insert(30)
cll.display()
```

- **બનાવટ**: છેલ્લા નોડને હેડ સાથે જોડવું
- **ડિસ્પ્લે**: ફરીથી હેડ પર પહોંચવા સુધી બંધ કરવું

મેમરી ટ્રીક: "CLH - બનાવો, જોડો, હેડ"

પ્રશ્ન 4(અ) [3 ગુણ]

સિલેક્શન સૉર્ટ પદ્ધતિનો પ્રોગ્રામ લખો.

જવાબ:

```
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

# ઉદાહરણ ઉપયોગ
data = [64, 34, 25, 12, 22]
sorted_data = selection_sort(data)
print("સૉર્ટેડ એરે:", sorted_data)
```

- **મિનિમમ શોધો**: અનસૉર્ટેડ ભાગમાં
- **સ્વેપ**: પ્રથમ અનસૉર્ટેડ એલિમેન્ટ સાથે
- **ટાઇમ કોમ્પ્લેક્સિટી**: O(n²)

મેમરી ટ્રીક: "FMS - શોધો, મિનિમમ, સ્વેપ"

પ્રશ્ન 4(બ) [4 ગુણ]

નીચેના ડેટાને ચડતા ક્રમમાં ગોઠવવા માટે ઇન્સર્શન સૉર્ટ લાગૂ કરો. 25 15 35 20 30 5 10

જવાબ:

ઇન્સર્શન સૉર્ટ સ્ટેપ્સ:

```
શરૂઆત: [25, 15, 35, 20, 30, 5, 10]

પાસ 1: [15, 25, 35, 20, 30, 5, 10]  (15 ઇન્સર્ટ કર્યું)
પાસ 2: [15, 25, 35, 20, 30, 5, 10]  (35 જગ્યાએ)
પાસ 3: [15, 20, 25, 35, 30, 5, 10]  (20 ઇન્સર્ટ કર્યું)
પાસ 4: [15, 20, 25, 30, 35, 5, 10]  (30 ઇન્સર્ટ કર્યું)
પાસ 5: [5, 15, 20, 25, 30, 35, 10]  (5 ઇન્સર્ટ કર્યું)
પાસ 6: [5, 10, 15, 20, 25, 30, 35]  (10 ઇન્સર્ટ કર્યું)

અંતિમ: [5, 10, 15, 20, 25, 30, 35]
```

- **પદ્ધતિ**: એલિમેન્ટ લો, સૉર્ટેડ ભાગમાં સ્થાન શોધો
- **સરખામણીઓ**: કુલ 15 સરખામણીઓ
- **શિફ્ટ્સ**: જગ્યા બનાવવા માટે એલિમેન્ટ્સ ખસેડવા

મેમરી ટ્રીક: "TFI - લેવું, શોધવું, ઇન્સર્ટ કરવું"

પ્રશ્ન 4(ક) [7 ગુણ]

લીનિયર સર્ચનો ઉપયોગ કરીને લિસ્ટમાંથી ચોક્કસ તત્વ શોધવા માટે પાયથોન પ્રોગ્રામનો અમલ કરો.

જવાબ:

```
def linear_search(arr, target):
    comparisons = 0

    for i in range(len(arr)):
        comparisons += 1
        if arr[i] == target:
            print(f"એલિમેન્ટ {target} ઇન્ડેક્સ {i} પર મળ્યું")
            print(f"સરખામણીઓની સંખ્યા: {comparisons}")
            return i

    print(f"એલિમેન્ટ {target} નહીં મળ્યું")
    print(f"સરખામણીઓની સંખ્યા: {comparisons}")
    return -1

def linear_search_all_positions(arr, target):
    positions = []
    for i in range(len(arr)):
        if arr[i] == target:
            positions.append(i)
    return positions

# ઉદાહરણ ઉપયોગ
data = [10, 25, 30, 15, 20, 30, 35]
target = 30

result = linear_search(data, target)
all_positions = linear_search_all_positions(data, target)
print(f"{target} ની બધી પોઝિશન: {all_positions}")
```

- **સિક્વન્શિયલ સર્ચ**: દરેક એલિમેન્ટ એક પછી એક તપાસવું
- **ટાઇમ કોમ્પ્લેક્સિટી**: O(n) સૌથી ખરાબ કેસ
- **બેસ્ટ કેસ**: O(1) જો પ્રથમ પોઝિશન પર મળે

મેમરી ટ્રીક: "CEO - દરેક એક તપાસો"

પ્રશ્ન 4(અ અથવા) [3 ગુણ]

ઇન્સર્શન સૉર્ટ પદ્ધતિનો પ્રોગ્રામ લખો.

જવાબ:

```
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

# ઉદાહરણ ઉપયોગ
data = [12, 11, 13, 5, 6]
print("મૂળ:", data)
sorted_data = insertion_sort(data.copy())
print("સૉર્ટેડ:", sorted_data)
```

- **કી એલિમેન્ટ**: વર્તમાન એલિમેન્ટ જે ઇન્સર્ટ કરવાનું છે
- **જમણી બાજુ શિફ્ટ**: મોટા એલિમેન્ટ્સ જમણી બાજુ ખસે છે
- **ઇન્સર્ટ**: યોગ્ય સ્થાને કી

મેમરી ટ્રીક: "KSI - કી, શિફ્ટ, ઇન્સર્ટ"

પ્રશ્ન 4(બ અથવા) [4 ગુણ]

નીચેના ડેટાને ક્વિક સૉર્ટ લાગૂ કરો અને તેમને યોગ્ય રીતે ગોઠવો. 5 6 1 8 2 9 10 15 7 13

જવાબ:

ક્વિક સૉર્ટ સ્ટેપ્સ:

```
શરૂઆત: [5, 6, 1, 8, 2, 9, 10, 15, 7, 13]
પિવોટ: 5 (પ્રથમ એલિમેન્ટ)

પાર્ટિશન 1: [1, 2] 5 [6, 8, 9, 10, 15, 7, 13]

ડાબો સબએરે [1, 2]:
પિવોટ: 1 → [] 1 [2]
પરિણામ: [1, 2]

જમણો સબએરે [6, 8, 9, 10, 15, 7, 13]:
પિવોટ: 6 → [] 6 [8, 9, 10, 15, 7, 13]

પાર્ટિશનિંગ ચાલુ...

અંતિમ: [1, 2, 5, 6, 7, 8, 9, 10, 13, 15]
```

- **વિભાજન**: પિવોટ પસંદ કરો, તેની આસપાસ પાર્ટિશન કરો
- **જીતો**: સબએરેને રીકર્સિવલી સૉર્ટ કરો

- **સરેરાશ સમય**: O(n log n)

મેમરી ટ્રીક: "DCC - વિભાજન, જીતો, જોડો"

પ્રશ્ન 4(ક અથવા) [7 ગુણ]

મર્જ સૉર્ટ અલ્ગોરિધમનો અમલ કરો.

જવાબ:

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])

    return merge(left, right)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result

# ઉદાહરણ ઉપયોગ
data = [38, 27, 43, 3, 9, 82, 10]
sorted_data = merge_sort(data)
print("સૉર્ટેડ એરે:", sorted_data)
```

- **વિભાજન**: એરેને અડધામાં વિભાજિત કરો
- **મર્જ**: સૉર્ટેડ સબએરેને જોડો
- **ટાઇમ કોમ્પ્લેક્સિટી**: હંમેશા O(n log n)

મેમરી ટ્રીક: "DSM - વિભાજન, સૉર્ટ, મર્જ"

પ્રશ્ન 5(અ) [3 ગુણ]

ટૂંકી નોંધ લખો: એપ્લિકેશન ઓફ ટ્રી.

જવાબ:

સારણી: ટ્રી એપ્લિકેશનો

એપ્લિકેશન	વર્ણન	ઉદાહરણ
ફાઇલ સિસ્ટમ	ડિરેક્ટરી સ્ટ્રક્ચર	ફોલ્ડર અને ફાઇલો
એક્સપ્રેશન પાર્સિંગ	ગાણિતિક સમીકરણો	(a+b)*c
ડેટાબેઝ ઇન્ડેક્સિંગ	ઝડપી ડેટા પુનઃપ્રાપ્તિ	ડેટાબેઝમાં B-ટ્રીઝ

- **ડિસિઝન ટ્રીઝ**: AI અને મશીન લર્નિંગ
- **હફમેન કોડિંગ**: ડેટા કોમ્પ્રેશન
- **ગેમ ટ્રીઝ**: ચેસ, ટિક-ટેક-ટો

મેમરી ટ્રીક: "FED - ફાઇલ, એક્સપ્રેશન, ડેટાબેઝ"

પ્રશ્ન 5(બ) [4 ગુણ]

વિવિધ ટ્રી ટ્રાવર્સલ પદ્ધતિઓ સમજાવો.

જવાબ:

સારણી: ટ્રી ટ્રાવર્સલ પદ્ધતિઓ

પદ્ધતિ	ક્રમ	પ્રક્રિયા
ઇનઓર્ડર	ડાબે-રૂટ-જમણે	LNR
પ્રીઓર્ડર	રૂટ-ડાબે-જમણે	NLR
પોસ્ટઓર્ડર	ડાબે-જમણે-રૂટ	LRN

```
ઉદાહરણ ટ્રી:
        A
       / \
      B   C
     / \
    D   E

ઇનઓર્ડર: D B E A C
પ્રીઓર્ડર: A B D E C
પોસ્ટઓર્ડર: D E B C A
```

- **ઇનઓર્ડર**: BST માટે સૉર્ટેડ સિક્વન્સ આપે છે
- **પ્રીઓર્ડર**: ટ્રી કોપી કરવા માટે વપરાય છે
- **પોસ્ટઓર્ડર**: ટ્રી ડિલીટ કરવા માટે વપરાય છે

મેમરી ટ્રીક: "LNR PNL LRN ઇન-પ્રી-પોસ્ટ માટે"

પ્રશ્ન 5(ક) [7 ગુણ]

બાઇનરી સર્ચ ટ્રી પર નીચેની કામગીરી કરવા માટે મેન્યૂ સંચાલિત પ્રોગ્રામ લખો: BST ટ્રી બનાવવા માટેનો પ્રોગ્રામ.

જવાબ:

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        if node is None:
            return TreeNode(data)

        if data < node.data:
            node.left = self._insert_recursive(node.left, data)
        elif data > node.data:
            node.right = self._insert_recursive(node.right, data)

        return node

    def inorder(self, node):
        if node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)

def main():
    bst = BST()

    while True:
        print("\n1. દાખલ કરો")
        print("2. દર્શાવો (ઇનઓર્ડર)")
        print("3. બહાર નીકળો")

        choice = int(input("પસંદગી દાખલ કરો: "))

        if choice == 1:
            data = int(input("ડેટા દાખલ કરો: "))
            bst.insert(data)
        elif choice == 2:
            print("BST (ઇનઓર્ડર):", end=" ")
```

```
            bst.inorder(bst.root)
            print()
        elif choice == 3:
            break

if __name__ == "__main__":
    main()
```

- **BST ગુણધર્મ**: ડાબે < રૂટ < જમણે
- **ઇન્સર્શન**: સરખાવો અને ડાબે/જમણે જાઓ
- **મેન્યૂ ડ્રિવન**: વપરાશકર્તા-મૈત્રીપૂર્ણ ઇન્ટરફેસ

મેમરી ટ્રીક: "CIM - સરખાવો, ઇન્સર્ટ, મેન્યૂ"

પ્રશ્ન 5(અ અથવા) [3 ગુણ]

વ્યાખ્યાયિત કરો અને ઉદાહરણો આપો: સ્ટ્રિક્ટ બાઇનરી ટ્રી અને કમ્પ્લીટ બાઇનરી ટ્રી.

જવાબ:

સારણી: બાઇનરી ટ્રી પ્રકારો

પ્રકાર	વ્યાખ્યા	ઉદાહરણ
સ્ટ્રિક્ટ બાઇનરી ટ્રી	દરેક નોડને 0 અથવા 2 બાળકો છે	દરેક આંતરિક નોડને બરાબર 2 બાળકો
કમ્પ્લીટ બાઇનરી ટ્રી	છેલ્લા સિવાય બધા લેવલ ભરેલા, ડાબેથી જમણે ભરેલા	બીજા છેલ્લા લેવલ સુધી પરફેક્ટ સ્ટ્રક્ચર

```
સ્ટ્રિક્ટ બાઇનરી ટ્રી:
        A
       / \
      B   C
         / \
        D   E

કમ્પ્લીટ બાઇનરી ટ્રી:
        A
       / \
      B   C
     / \ /
    D  E F
```

- **સ્ટ્રિક્ટ**: એક બાળક વાળો કોઈ નોડ નહીં
- **કમ્પ્લીટ**: શ્રેષ્ઠ સ્પેસ ઉપયોગ

મેમરી ટ્રીક: "SC - સ્ટ્રિક્ટ કમ્પ્લીટ"

પ્રશ્ન 5(બ અથવા) [4 ગુણ]

બાઇનરી ટ્રીની મૂળભૂત પરિભાષા સમજાવો: લેવલ નંબર, ડિગ્રી, ઇન-ડિગ્રી, આઉટ-ડિગ્રી, લીફ નોડ.

જવાબ:

```
બાઇનરી ટ્રી ઉદાહરણ:
    લેવલ 0:      A         (રૂટ)
                 / \
   લેવલ 1:     B   C
               / \   \
   લેવલ 2:   D   E   F    (લીવ્સ: D, E, F)
```

સારણી: બાઇનરી ટ્રી પરિભાષા

શબ્દ	વ્યાખ્યા	ઉદાહરણ
લેવલ નંબર	રૂટથી અંતર (રૂટ = 0)	A=0, B=1, D=2
ડિગ્રી	બાળકોની સંખ્યા	A=2, B=2, C=1
ઇન-ડિગ્રી	આવતા એજની સંખ્યા	બધા નોડ = 1 (સિવાય રૂટ = 0)
આઉટ-ડિગ્રી	જતા એજની સંખ્યા	ડિગ્રી સમાન
લીફ નોડ	બાળકો ન હોય તેવો નોડ	D, E, F

મેમરી ટ્રીક: "LDIOL - લેવલ, ડિગ્રી, ઇન-આઉટ, લીફ"

પ્રશ્ન 5(ક અથવા) [7 ગુણ]

બાઇનરી સર્ચ ટ્રી પર નીચેની કામગીરી કરવા માટે મેન્યૂ સંચાલિત પ્રોગ્રામ લખો: BST માં એક એલિમેન્ટ દાખલ કરો.

જવાબ:

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if self.root is None:
            self.root = TreeNode(data)
            print(f"રૂટ નોડ {data} બનાવ્યું")
        else:
            self._insert_helper(self.root, data)

    def _insert_helper(self, node, data):
        if data < node.data:
            if node.left is None:
```

```
                node.left = TreeNode(data)
                print(f"{data} ને {node.data} ની ડાબી બાજુએ દાખલ કર્યું")
            else:
                self._insert_helper(node.left, data)
        elif data > node.data:
            if node.right is None:
                node.right = TreeNode(data)
                print(f"{data} ને {node.data} ની જમણી બાજુએ દાખલ કર્યું")
            else:
                self._insert_helper(node.right, data)
        else:
            print(f"ડેટા {data} પહેલેથી અસ્તિત્વમાં છે")

    def display_inorder(self, node, result):
        if node:
            self.display_inorder(node.left, result)
            result.append(node.data)
            self.display_inorder(node.right, result)

def main():
    bst = BST()

    while True:
        print("\n--- BST ઓપરેશનો ---")
        print("1. એલિમેન્ટ દાખલ કરો")
        print("2. BST દર્શાવો (ઇનઓર્ડર)")
        print("3. બહાર નીકળો")

        choice = int(input("તમારી પસંદગી દાખલ કરો: "))

        if choice == 1:
            data = int(input("દાખલ કરવાનું એલિમેન્ટ દાખલ કરો: "))
            bst.insert(data)
        elif choice == 2:
            result = []
            bst.display_inorder(bst.root, result)
            print("BST એલિમેન્ટ્સ (સૉર્ટેડ):", result)
        elif choice == 3:
            print("બહાર નીકળી રહ્યા છીએ...")
            break
        else:
            print("અયોગ્ય પસંદગી!")

if __name__ == "__main__":
    main()
```

- **ઇન્સર્ટ લોજિક**: વર્તમાન નોડ સાથે સરખાવો, ડાબે/જમણે જાઓ
- **રિકર્સિવ એપ્રોચ**: સ્વચ્છ અને કાર્યક્ષમ અમલીકરણ
- **મેન્યૂ સિસ્ટમ**: ઇન્ટરેક્ટિવ વપરાશકર્તા ઇન્ટરફેસ

મેમરી ટ્રીક: "CRL - સરખાવો, રિકર્સિવ, ડાબે/જમણે"