Data Structures

S Abhishek

AM.EN.U4CSE19147

1. Find the merge point of two linked lists.

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


def display(n):
    temp = n

    while temp:
        print(temp.data, end=" ")
        temp = temp.next
    print()


def intersection(n1, n2):
    temp1 = n1

    while temp1 is not None:
        temp2 = n2
        while temp2 is not None:
            if temp1 == temp2:
                print("\nIntersection Point is : {}".format(temp1.data))
                return
            else:
                temp2 = temp2.next

        temp1 = temp1.next

    print("\nNo Intersection Point")
```

```
head1 = Node(1)

nod1 = Node(2)
nod2 = Node(3)
nod3 = Node(4)
nod4 = Node(5)
nod5 = Node(6)
nod6 = Node(7)
nod7 = Node(8)

head1.next = nod1
head1.next.next = nod2
head1.next.next.next = nod3
head1.next.next.next.next = nod4
head1.next.next.next.next.next = nod5
head1.next.next.next.next.next.next = nod6

head2 = Node(9)
head2.next = nod7
head2.next.next = nod4
head2.next.next.next = nod5
head2.next.next.next.next = nod6

print("Linked List 1 : ", end=" ")
display(head1)
print("Linked List 2 : ", end=" ")
display(head2)

intersection(head1, head2)
```

```
Linked List 1 :  1 2 3 4 5 6 7
Linked List 2 :  9 8 5 6 7

Intersection Point is : 5
```

```
head2 = Node(9)
head2.next = nod7
#head2.next.next = nod4
#head2.next.next.next = nod5
#head2.next.next.next.next = nod6
```

```
Linked List 1 :  1 2 3 4 5 6 7
Linked List 2 :  9 8

No Intersection Point
```

2. Create an n-disc towers of Hanoi. Move all the discs from tower A to tower C.

```
class Hanoi:

  def __init__(self, label):
    self.data = list()
    self.label = label

  def is_empty(self):
    return len(self.data) == 0

  def push(self, e):
    self.data.append(e)

  def top(self):
    return self.data[len(self.data) - 1]

  def pop(self):
    return self.data.pop()


def traversal(rod_A, rod_B):
  if rod_A.is_empty():
    rod_A.push(rod_B.pop())
    print(rod_B.label, "-->", rod_A.label)
  elif rod_B.is_empty():
```

```
        rod_B.push(rod_A.pop())
        print(rod_A.label, "-->", rod_B.label)
    elif rod_B.top() < rod_A.top():
        rod_A.push(rod_B.pop())
        print(rod_B.label, "-->", rod_A.label)
    else:
        rod_B.push(rod_A.pop())
        print(rod_A.label, "-->", rod_B.label)


def HanoiTower(n):
    rod_A = Hanoi("A")
    rod_B = Hanoi("B")
    rod_C = Hanoi("C")

    min = (2 ** n) - 1

    for obj in range(n, 0, -1):
        rod_A.push(obj)

    if n % 2 == 0:
        rod_B, rod_C = rod_C, rod_B

    for i in range(min):
        if i % 3 == 0:
            traversal(rod_A, rod_C)
        if i % 3 == 1:
            traversal(rod_A, rod_B)
        if i % 3 == 2:
            traversal(rod_B, rod_C)


temp = int(input("Number of Disks: "))
if temp == 0:
    print("No Disk")
else:
    HanoiTower(temp)
```

```
Number of Disks: 3
A --> C
A --> B
C --> B
A --> C
B --> A
B --> C
A --> C
```

```
Number of Disks: 4
A --> B
A --> C
B --> C
A --> B
C --> A
C --> B
A --> B
A --> C
B --> C
B --> A
C --> A
B --> C
A --> B
A --> C
B --> C
```

3. Solve towers of hanoi using recursion

```python
def hanoi(n, source, destination, dummy):
    if n == 1:
        print("Move Disk 1 from Rod", source, "to Rod", destination)
        return
    hanoi(n - 1, source, dummy, destination)
    print("Move Disk", n, "from Rod", source, "to Rod", destination)
```

```
  hanoi(n - 1, dummy, destination, source)


n_rod = int(input("Enter the Number of Disk : "))
print()
hanoi(n_rod, 'A', 'C', 'B')
```

```
Enter the Number of Disk : 4

Move Disk 1 from Rod A to Rod B
Move Disk 2 from Rod A to Rod C
Move Disk 1 from Rod B to Rod C
Move Disk 3 from Rod A to Rod B
Move Disk 1 from Rod C to Rod A
Move Disk 2 from Rod C to Rod B
Move Disk 1 from Rod A to Rod B
Move Disk 4 from Rod A to Rod C
Move Disk 1 from Rod B to Rod C
Move Disk 2 from Rod B to Rod A
Move Disk 1 from Rod C to Rod A
Move Disk 3 from Rod B to Rod C
Move Disk 1 from Rod A to Rod B
Move Disk 2 from Rod A to Rod C
Move Disk 1 from Rod B to Rod C
```

```
Enter the Number of Disk : 3

Move Disk 1 from Rod A to Rod C
Move Disk 2 from Rod A to Rod B
Move Disk 1 from Rod C to Rod B
Move Disk 3 from Rod A to Rod C
Move Disk 1 from Rod B to Rod A
Move Disk 2 from Rod B to Rod C
Move Disk 1 from Rod A to Rod C
```

4. Reverse a stack using recursion.

```
def push(s, item):
    s.append(item)
    return s


def display(s1):
    print("\nStack is ", end="")
    for i in range(len(s1) - 1, -1, -1):
        print(s1[i], end=' ')
    print()


def isempty(s2):
    return len(s2) == 0


def pop(s3):
    if isempty(s3):
        print("Stack Underflow")
        return
    return s3.pop()


def insertAtBottom(s5, item):
    if isempty(s5):
        push(s5, item)
    else:
        temp = pop(s5)
        insertAtBottom(s5, item)
        push(s5, temp)


def reverse(s4):
    if not isempty(s4):
        temp = pop(s4)
        reverse(s4)
```

```
        insertAtBottom(s4, temp)


stack = []

n = int(input("Enter the Number of Elements : "))

for j in range(n):
    m = int(input("Enter the Element {} : ".format(j + 1)))
    push(stack, m)

display(stack)

# pop(stack)
# pop(stack)

reverse(stack)

display(stack)
```

```
Enter the Number of Elements : 5
Enter the Element 1 : 1
Enter the Element 2 : 2
Enter the Element 3 : 3
Enter the Element 4 : 4
Enter the Element 5 : 5

Stack is 5 4 3 2 1

Stack is 1 2 3 4 5
```

```
Enter the Number of Elements : 7
Enter the Element 1 : 1
Enter the Element 2 : 5
Enter the Element 3 : 6
Enter the Element 4 : 9
Enter the Element 5 : 4
Enter the Element 6 : 3
Enter the Element 7 : 2

Stack is 2 3 4 9 6 5 1

Stack is 1 5 6 9 4 3 2
```