Assignment 13 (Linked List)

> **Name: Mohammed Varaliya**

> **Roll No: 54**

Questions

1. **Concatenating 2 linked list.**
2. **Merging 2 sorted linked list.**

---

1. **Concatenating 2 linked list.**
    a. **Code:**

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

class SinglyLinkedList:
    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_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def concatenating_2_ll(self, head1, head2):

        if head1 is None:
            return head2

        cur_node = head1
        while cur_node.next is not None:
            cur_node = cur_node.next
```

```
        cur_node.next = head2

        return head1

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")


if __name__ == "__main__":

    llist1 = SinglyLinkedList()
    llist1.append(1)
    llist1.append(5)
    llist1.append(10)
    llist1.append(15)

    llist2 = SinglyLinkedList()
    llist2.append(2)
    llist2.append(8)
    llist2.append(13)
    llist2.append(18)
    llist2.append(20)

    llist1.display_list_iterative()
    print("\n")
    llist2.display_list_iterative()

    llist1.concatenating_2_ll(llist1.head, llist2.head)
    print("\n")
    llist1.display_list_iterative()
```

```
# Concatenating 2 linked list.

def concatenating_2_ll(self, head1, head2):

    if head1 is None:
        return head2

    cur_node = head1
    while cur_node.next is not None:
        cur_node = cur_node.next

    cur_node.next = head2

    return head1
```

1. **Explanation:**

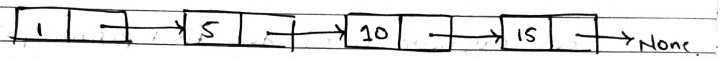
PAGE No.
DATE / /

Assignment 13 21-Nov-2024.

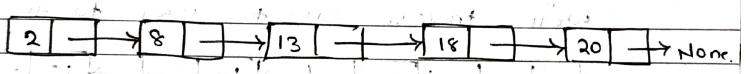
1. Concatenating 2 linked list.

⇒ let's assume we have these following linked list.

List 1 :



list 2:



we have to concatenate these two linked list and return one single list as a output.

Step 1:

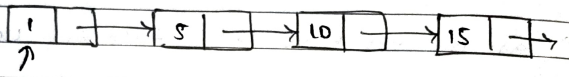
we have head of both the linked list,

head 1 = 1.
head 2 = 2.

we'll have to traverse the entire first linked list, all the way to the end, and change the 'next' pointer of last node to the head 2 node.

Step 2:

· list 1.

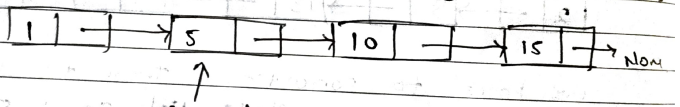


cur_node = self.head

our cur_node is at the first position pointing to the head of the list.

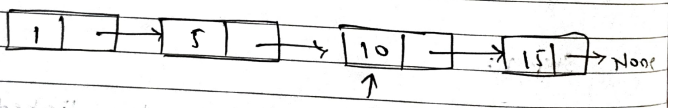
we'll traverse all the way to the end.

⇒



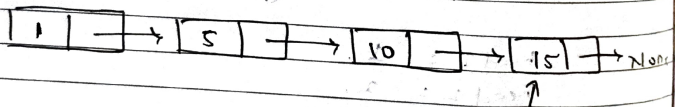
cur_node

⇒



cur_node

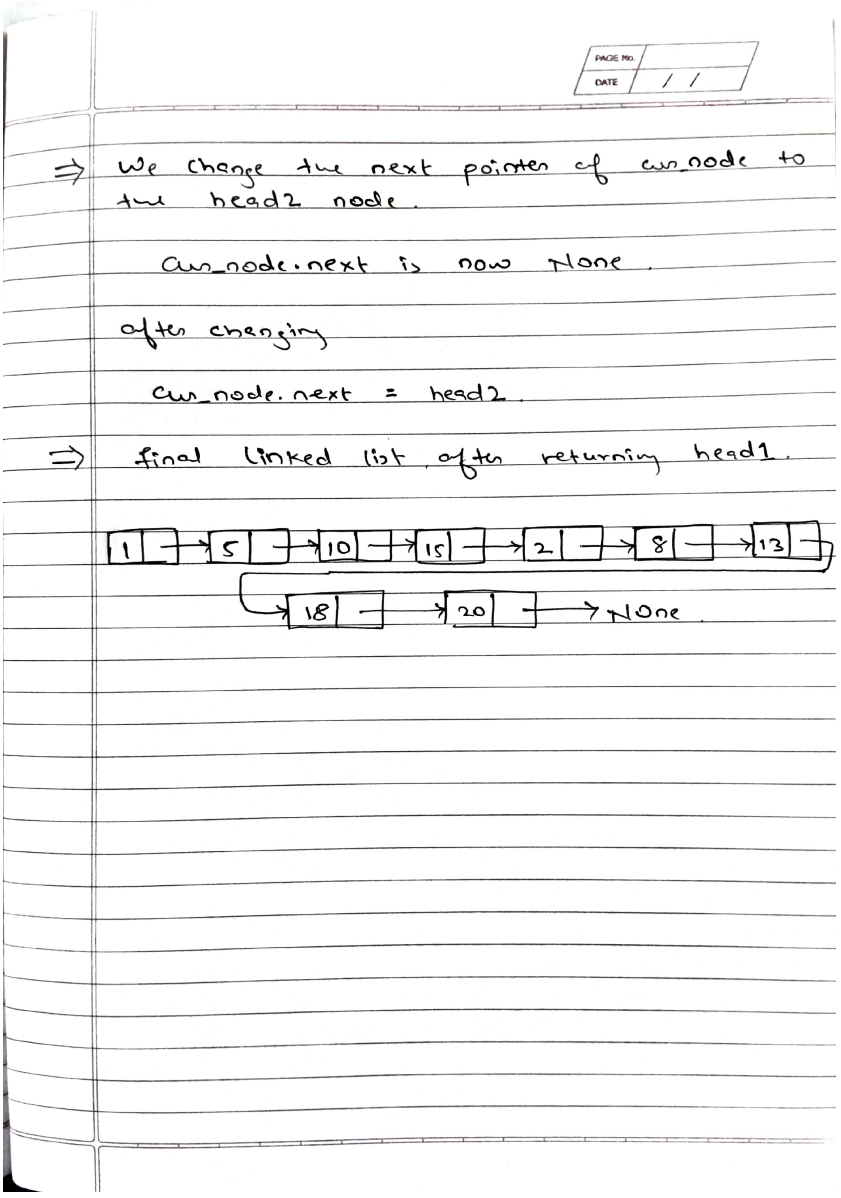
⇒



cur_node.

⇒ Now, we reached to the end of the list and the cur_node.next pointer is pointing to None,

⇒



b. **Output:**

```
1 -> 5 -> 10 -> 15 -> None


2 -> 8 -> 13 -> 18 -> 20 -> None


1 -> 5 -> 10 -> 15 -> 2 -> 8 -> 13 -> 18 -> 20 -> None
```

---

2. **Merging 2 sorted linked list.**

a. **Code:**

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

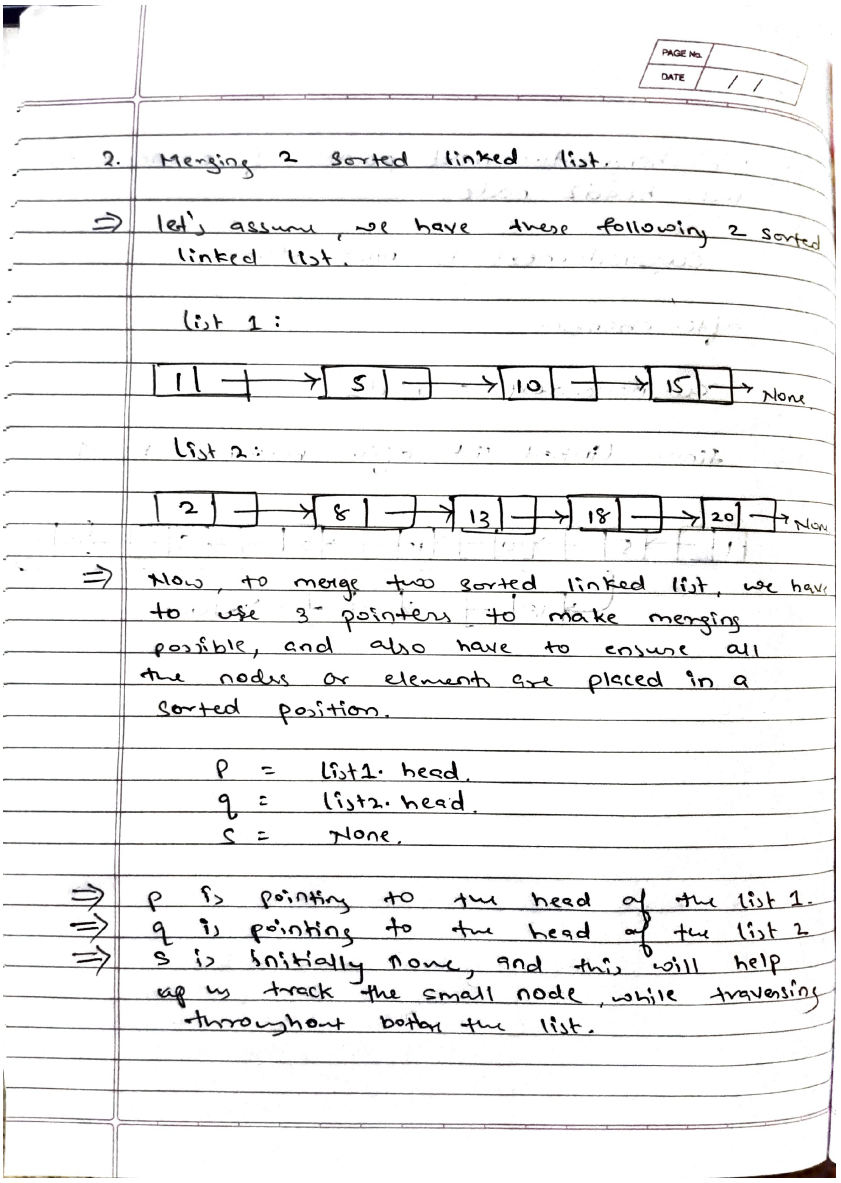
```
class SinglyLinkedList:
    def __init__(self):
        self.head = None

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

        if self.head is None:
            self.head = new_node
            return

        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def merge_sorted(self, llist):
        p = self.head
        q = llist.head
        s = None

        if not p:
            return q
        if not q:
            return p

        if p and q:
            if p.data <= q.data:
                s = p
                p = s.next
            else:
                s = q
                q = s.next
            new_head = s

        while p and q:
            if p.data <= q.data:
                s.next = p
                s = p
                p = s.next
            else:
                s.next = q
                s = q
                q = s.next

        if not p:
            s.next = q
        if not q:
            s.next = p

        return new_head

    def display_list_iterative(self):
        cur_node = self.head
        while cur_node:
            print(cur_node.data, end=" -> ")
            cur_node = cur_node.next
        print("None")
```

```
if __name__ == "__main__":

    llist1 = SinglyLinkedList()
    llist1.append(1)
    llist1.append(5)
    llist1.append(10)
    llist1.append(15)

    llist2 = SinglyLinkedList()
    llist2.append(2)
    llist2.append(8)
    llist2.append(13)
    llist2.append(18)
    llist2.append(20)

    llist1.display_list_iterative()
    print("\n")
    llist2.display_list_iterative()

    llist1.merge_sorted(llist2)
    print("\n")
    llist1.display_list_iterative()
```

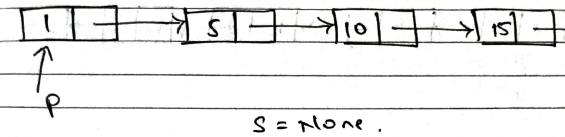
```
# Merging 2 sorted linked list.

def merge_sorted(self, llist):
    p = self.head
    q = llist.head
    s = None

    if not p:
        return q
    if not q:
        return p

    if p and q:
        if p.data <= q.data:
            s = p
            p = s.next
        else:
            s = q
            q = s.next
        new_head = s

    while p and q:
        if p.data <= q.data:
            s.next = p
            s = p
            p = s.next
        else:
            s.next = q
            s = q
            q = s.next

    if not p:
```

```
        s.next = q
    if not q:
        s.next = p

    return new_head
```

1. **Explanation:**

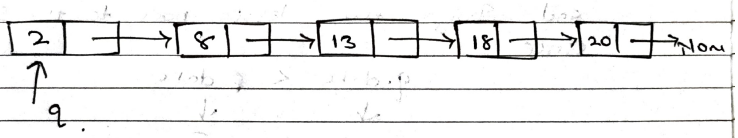


⇒ List 1:



s = None.

List 2:



⇒ Now, we compare both p and q node values and assign the lesser node to the s variable.

since,

p.data < q.data.

1 < 2.

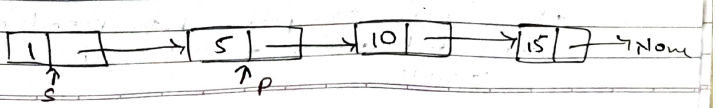
we, update.

s = p. and

p = s.next,

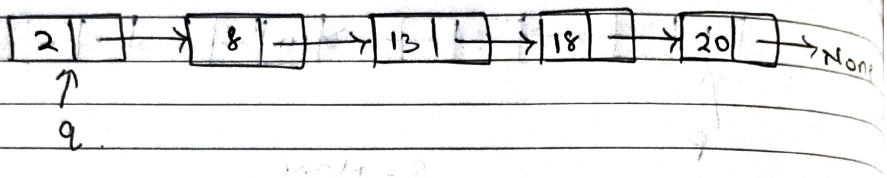
p = s.next, because we want to check all the next nodes of list 1 for the further merging process.

⇒ List 1:



PAGE No.
DATE / /

List 2:



⇒ Now, we compare both p and q node values and assign the lesser node to the s variable. Since,

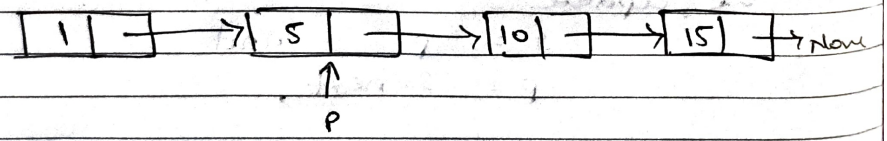
q.data < p.data
↓ ↓
2 < 5.

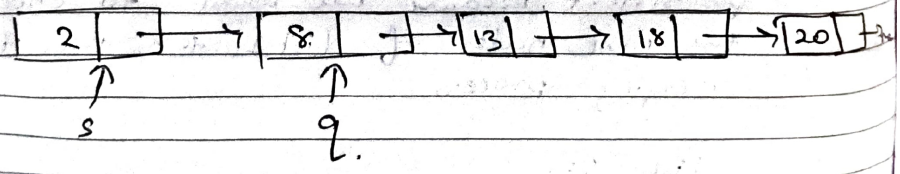
we update,

s = q
q = s.next.

q.next for further merging process.

⇒ List 1.





⇒ Now we compare both p and q node values and assign the lesser node to the s variable.

PAGE No.
DATE / /

since,

p.data < q.data

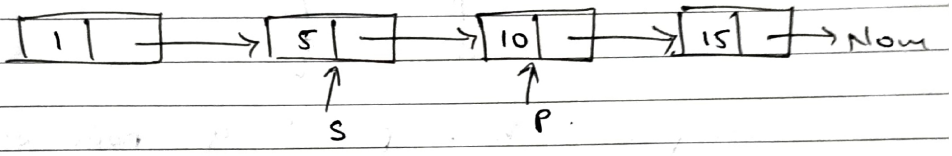
5 < 8.

we update,

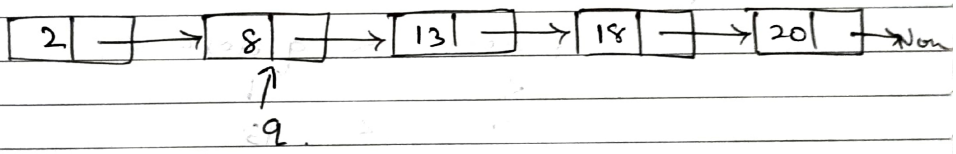
s = p.

q = s.next.

⇒ list 1



list2:



⇒ Now we compare both p and q node values and assign the lesser node to the s variable.

since,

q.data < p.data

8 < 10

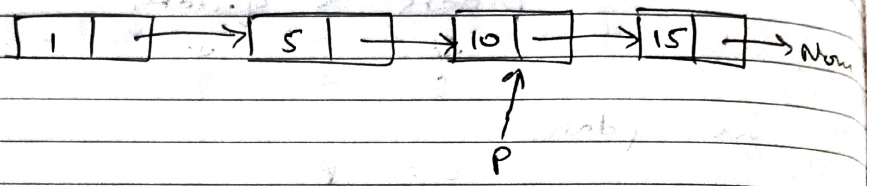
we update,

s = q

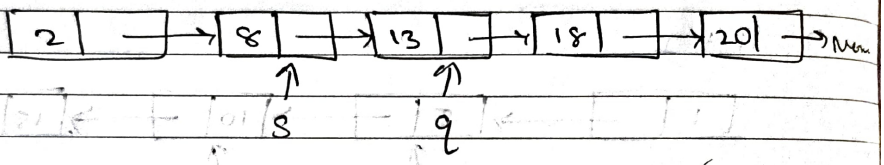
q = s.next.

PAGE No.
DATE 1 1

⇒ list 1:



list 2:



⇒ compare both p and q node values and assign lesser node values to the s variable.

since

p.data < q.data

↓ ↓

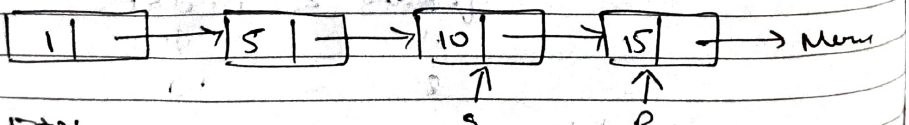
10 < 13

we update,

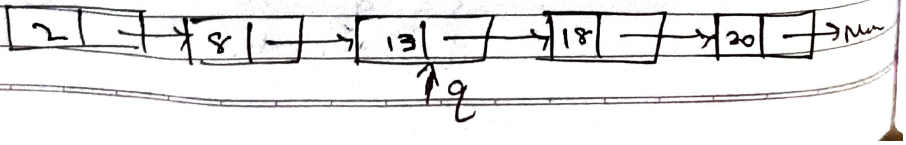
s = p

p = s.next.

⇒ list 1:



list 2:



PAGE No.
DATE / /

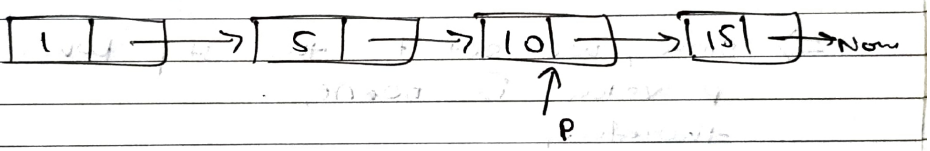
⇒ compare both p and q node values and assign lesser node to the s variable.
Since,

q.data < p.data.
↓ ↓
13 < 15

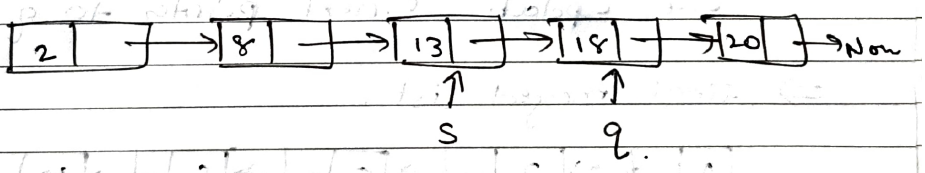
we update,

s = q
q = s.next

⇒ list 1:



list2:

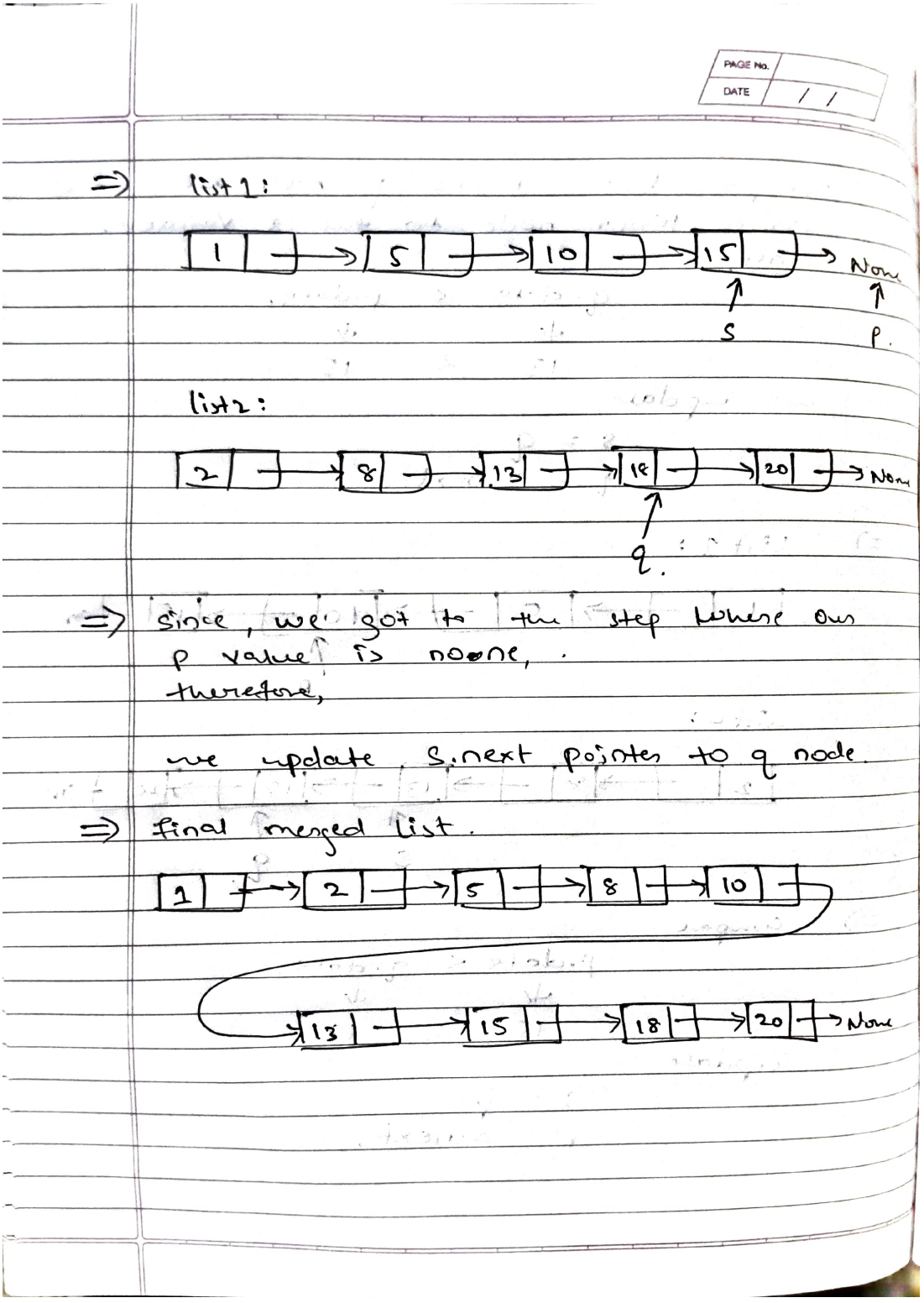


⇒ compare

p.data < q.data.
↓ ↓
15 < 18

update

s = p
p = s.next.



b. **Output:**

```
1 -> 5 -> 10 -> 15 -> None

2 -> 8 -> 13 -> 18 -> 20 -> None

1 -> 2 -> 5 -> 8 -> 10 -> 13 -> 15 -> 18 -> 20 -> None
```

1. **GitHub Code Explanation with example walkthrough.**

MCA-Coursework/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-13 at main · Mohammedvaraliya/MCA-Coursework
Contribute to Mohammedvaraliya/MCA-Coursework development by creating an account on GitHub.

https://github.com/Mohammedvaraliya/MCA-Coursework/tree/main/SEM-1/Data-Structure-Using-Python/Assignments/Assignment-13