II ①

Singly

```
Node {
     data
     next ← null
     }
Singly Linkedlist {
    Node head ← None
    Size ← 0
    AddToEnd (data) {
      new_node ← Node (data)
      if (head == None)
           head ← new_node
     else
        P ← head
        while (P.next)
             P ← P.next
        P.next ← new_node
     Size ← Size + 1
    }
    DeletFromEnd {
          Node P ← head
          While (P.next)
                P ← P.next
          P ← null
    } Size ← Size - 1

InsertAT Head (data) {
  newNode ← Node(data)
  newNode.next ← head
   head ← newNode
  Size ← Size + 1
  }
DeletFromHead() {
   P ← head
   head ← P.next
  }
}
```

DoublyLinked

```
Node {
    data
    next
    prev
}
DoublyLinkedList {
    Node head
    Node tail
    InsertToEnd(data){
        NewNode = Nod(data)
        P ← head
        while P.next
            P ← P.next
        P.next ← NewNode
        NewNode.prev ← P
    }
    DeletEnd( ){
        tail ← tail.prev
        tail.next ← null
    }
    InsertHead(data){
        NewNode ← Node(data)
        NewNode.next ← head
        head ← NewNode
    }
```

```
DeletHead( ){
    P ← head
    head ← P.next
    head.prev ← null
}
```

Q2

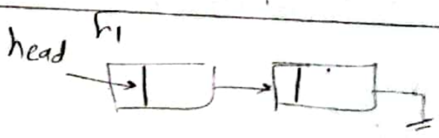
Recursive

```
Node getElement(data,crr){
    if Curr.data == data
            return crr
    else if   Curr == null
            return null
    return getElement(data, curr.next)
}
```

Iterative

```
Node getElement(data){
    Node crr ← head
    while (curr)
        if curr.data == data
                return Curr
        else
                curr ← curr.next
    return null.
```

Q3



```
① Insert(~~[illegible]~~ data){
    y ← Node(data)
    y.next ← head
    head ← y
  }
② Insert(data){
    NewNode ← Node(data)
    ~~P~~ P ← head
    P ← P.next
    while (P.next){
     if NewNode.data < P.next.data
            NewNode.next ← P.next
              P.next ← NewNode
              return
    }
}
```

③

```
Insert (data, index) {
  NewNode ← Node (data)
  if index == 0
      NewNode.next ← head
         head ← NewNode
  else
      P ← head
      for i = 0 → i = k-1
          P ← P.next
      U ← P.next
      NewNode.next ← U
      P.next ← NewNode
  Size ← Size + 1
}
```

④

```
AddLast (data) {
  NewNode ← Node (data)
   if head == Node
        head ← NewNode
   else
      P ← head
      while (P.next)
          P ← P.next
      P.next ← NewNode
   Size ← Size + 1
}
```

⑤

```
DeletData (data) {
   NewNode ← Node (data)
   ~~While~~ P ← head
   while (P.next) {
        ~~P ← P.next~~
   }
   ~~P.data ← data~~
   Tail
    if P.next.data == data
          P.next ← P.next.next
          return
   else
          P ← P.next
   }
}
```

⑥ Recursive

```
DeletAll (Val, *list heas) {
   if head == NULL)
      return head
   if head.data == Val) {
     *node ← head
      head ← head.next
      return DeletAll (Val, head)
   }
   head.next ← DeletAll (Val, head.next)
   return head
}
```

Iterative

```
DeletAll (*head, Val){
  if (head= =null)
    return head
  while (head and head.data = = Val)
    head ← head.next
  Node curr ← head
  Node *prev ← nullptr
  While (curr){
    if (Curr.data = = Val)
        Prev.next ← curr.next
    else
        Prev ← curr
    Curr ← curr.next
  }
  return head
}
```

⑦

```
DeletElement (index){
    if index = = 0
    P ← head
    head ← P.next
    else
        cur ← head
        Prev ← none
        for i=0 → i = index
            Prev ← cur
            Cur ← cur.next
      Prev.next ← cur.next
    Size ← Size +1
}
```

⑧ Recursion $F_2 \rightarrow$ head

```
Copy (head){
    if head == None
        return head
    NewNode = Node(Val)
    NewNod.next = Copy(head.next)
    return NewNode
}
```

Iterative

```
Copy (F2){
    X ← F1
    y ← F2
  While (i){
    i ← i.next
  NodeNew ← Node ~~data~~ (data)
    j.next ← NodeNew
    j ← j.next
    j.data = i.data
  }
}
```

⑨

```
Reverse (Node node){
    Node prev ← null
    Node curr ← node
    Node next ← null
    While (curr){
        next ← curr.next
        Curr.next ← prev
        Prev ← curr
        Curr ← next
    }
    node ← prev
    return Node
}
```

(10)

```
isSortedDesc(Node head){
  if (head==null)
     return true
  for Note t = head → t.next
  if(t.data <= t.next.data)
       return false
  return true
```

(11)

```
Exchange(Node head){
     if(head.next.next==head){
         head ←head.next
         return head
     }
     Node P←head
     While (P.next.next != head)
          P←P.next
     P.next.next ← head.next
     head.next ←P.next
     P.next ←head
     head ←head.next
     return head
}
```

(12)

```
RemoveDuplicated() {
   Node curr←head
   While (curr){
     Node temp ←curr
     while(temp and temp.
              data.equal(curr.data)
       {
        temp ←tem.next
       }
     curr.next ← temp
     curr ←curr.next
   }
}
```

Q4

(1)

```
CheckEquality(Node node1,
                Node node2)
  while(node1 !=null and
          node2 !=null){
     if (node1.val != node2.val)
         return false
     node1= node1.next
     node2=node2.next
   }
  if(node1)
     return false
  if (node2)
     return false
  return true
}
```

②

```
ConCat(Node F1, Node F2) {
    Curr ← F1.head
    while(Curr.next)
        Curr ← Curr.next
    Curr.next ← F2.head.next
}
```

---

③

```
Copy(Node F1, Node F2) {
    F2.head ← F1.head
}
```

Q5

①

```
DeleteLast() {
    R ← R.Prev
    R.next ← null
```

②

```
Insert(data) {
    NewNode ← Node(data)
    P ← head
    while(P.next)
        P ← P.next
    P.next ← NewNode
    NewNode.Prev ← P
}
```