[1] Singly linked list | Doubly linked list

[a]

Singly linked list:

```
InsertTail (Node head, Node n) {
    Node h ← head
    do {
        if (h.Next() = Null) {
            h.Next() ← n
            n.Next() = Null  return }
        h ← h.Next()
    } while (True) }
```

Doubly linked list:

```
InsertTail (Node head, Node n) {
    Node h ← head
    do {
        if (h.Next() = Null) {
            h.Next() ← n
            n.Prev() ← h
            n.Next() ← Null
            return }
        h ← h.Next
    } While (True)
```

[b]

Singly linked list:

```
DeleteTail (Node head) {
    Node h ← head.Next()
    Node Prev ← head
    do {
        if (h.Next() = Null) {
            Prev.Next() ← Null
            delete (h)
            return }
        Prev ← h
        h ← h.Next()
    } While (True) }
```

Doubly linked list:

```
DeleteTail (Node head) {
    Node h ← head
    do {
        if (h.Next() = Null) {
            PrevNode ← h.Prev()
            PrevNode.Next() ← Null
            h.Prev() ← Null
            delete (h)
            return }
        h ← h.Next()
    } While (True) }
```

[c]

Singly linked list:

```
InsertHead (Node head, Node n) {
    temp ← head
    n.Next() ← head
    head ← n }
```

Doubly linked list:

```
InsertTail (Node head, Node n) {
    n.Next() ← head
    head.Prev() ← n
    head ← head.Prev() }
```

1]
```
DeleteHead( Node head){
    temp <- head
    head <- head.Next()
    delete (temp) }
```

```
DeleteHead( Node Head) {
    temp <- head
    head <- head.Next()
    head.Prev() <- Null
    temp.Next() <- Null
    delete (temp)
```

2]

Recursive

a]
```
Search (Node n, data) {
    if (n = Null) {
        return Null}
    if (n.Data()= data) {
        return n}
    Search(n.Next(), data)}
```

Iterative

b]
```
Search (Node head, data) {
    Node h <- head
    do {
        if (h.Data() = data) {
            return h }
        h <- h.Next()
    }While (h != Null)
    return Null }
```

3] a]
```
Insert Head( Node head , Node y) {
    y.Next() <- head
    head <- y }
```

b]
```
InsertAt( Node head, data) {
    Node newNode.Data() <- data
    temp <- head
    do {
        if (newNode.Data() < temp.Data()) {
            newNode.Next() <- temp.Next()
            temp.Next() <- newNode
            return }
        temp <- temp.Next()
    }While ( temp.Next() != Null) }
```

```
[c] insertAt(Node head, Node newNode, int k) {
        temp ← head
        for i ← 0 → i < k {
            temp ← temp.Next()
        newNode.Next() ← temp.Next()
        temp.Next() ← newNode }

[d] insertAtEnd(Node head, Node newNode) {
        temp ← head
        While(temp.Next() != Null) {
                temp ← temp.Next() }
        temp.Next() ← newNode }

[e] deleteVal ( Val) {
        at ← search(Node head, Val)
        if(at = Null) {return}
        temp ← at.Prev()
        temp.Next() ← at.Next()
        at.Next().Prev() ← temp
        delete(at) ← Null
```

```
[f]         Recursive
 deleteALL(Val, Node temp) {
    if(temp = Null) {return}
    if (temP.Next().Data()
            = Val) {
        temP.Next() ← temp.Next()
                          .Next()
        deleteAll(Val, temp) }
 else{
        deleteALL(Val, temP.
                          Next())
   }
}
```

```
            Iterative
deleteALL(Node head, Val)
        temp ← head
        while(temp.Next() != Null) {
          if(temp.Next().Data() = Val) {
             curr ← temp.Next()
             temp.Next() ← curr.Next()
             delete (curr)}
        else {
             temP ← temP.Next()
        }
}
```

g)

```
deleteAt(Node head, int k) {
        temp ← head
        for i ← 0 → i < k { prev ← temp
            temp ← temp.Next() }
        prev.Next() ← temp.Next()
        delete (temp) }
```

h)

Iterative

```
copy(Node head, Node F2) {
    temp1 ← head
    temp2 ← F2
    while(temp1 != Null) {
        temp2.Data() ←
            temp1.Data()
        temp2 ← temp2.Next()
        temp1 ← temp1.Next()
}}
```

Recursive

```
Copy(Node i, Node j) {

    j.Next() ← newNode
    if (i = Null) {return}
    newNode.Data() ← i.Data()
    copy(i.Next(), j.Next())
```

i)

```
reverse (Node head) {
    a ← head
    b ← head.Next()
    c ← b
    do {
        c ← c.Next()
        b.Next() ← a
        a ← b
        b ← c
    }while(b.Next() != Null)

    head.Next() ← Null
    head ← b
```

[J] testorder(Node head){

```
    temp ← head
    while(temp != Null){
        if(temp.Data() > temp.Next().Data()){
            return false}
        temp ← temp.Next()}
    return true}
```

[K] interchange(Node head){

```
    last ← head
    while(last.Next() != Null){
        last ← last.Next()}
    temp ← last.Data()
    last.Data() ← head.Data()
    head.Data() ← temp}}
```

[L] delDuplicates(Node head){

```
    temp ← head
    while(temp.Next().Next() != Null){
        if(temp.Data()== temp.Next().Data()){
            temp.Next() ← temp.Next().Next()}
        temp ← temp.Next()}}
```

[4] [a] areEqual(Node head1, Node head2){

```
        if(F1.Size() != F2.Size()){
            return False}
        a ← head1        b ← head2
        while(a.Next() != Null){
            if(a.Data() != b.Data()){
                return false}
            a ← a.Next()
            b ← b.Next()}

        return true}
```

```
[b] ConCat (head 1, head2) {
        temp ← head 1
        While (temp. Next() != Null) {
              temp ← temp. Next() }
        temp. Next() ← head1 }
```

```
[c] Copy (head 1, head2) {
     temp1 ← head 1          temp2 ← head2
     While (temp1 != Null) {
          temp2. Data() ← temp1. Data()
          temp1 ← temp1. Next()
          temp2 ← temp2. Next() } }
```

```
[5] [a] delete End( Node F, Node R) {
              prev ← R. Prev()
              Prev. Next() ← Null
              R. Prev() ← Null
              delete (R)
              R ← Prev }
```

```
[b] insert End( Node F, Node R, Node NewNode) {
        R. Next() ← NewNode
        NewNode. Prev() ← R
        R ← R. Next() }
```