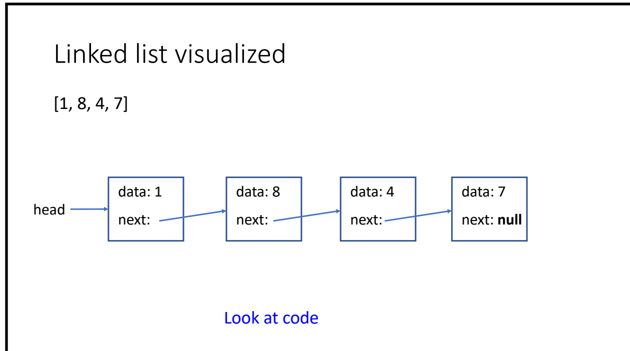
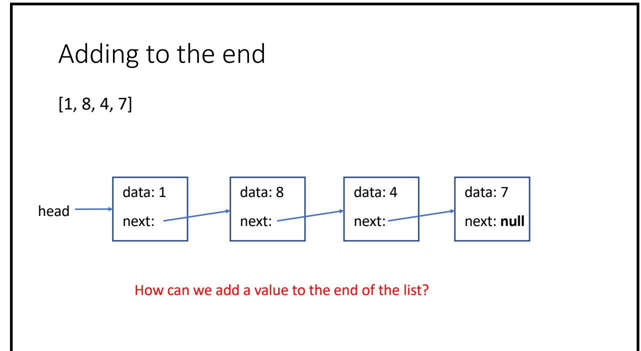
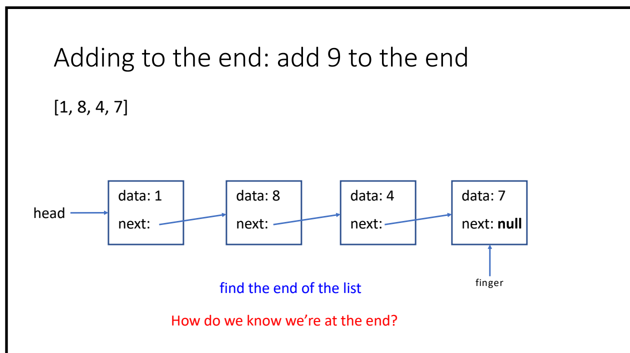


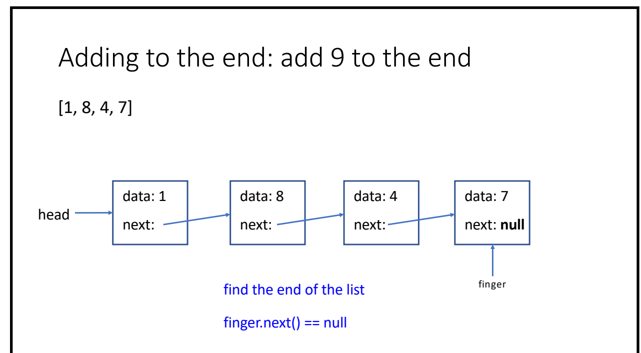
1



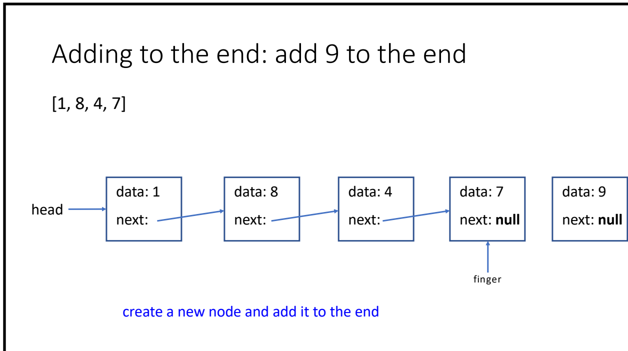
2



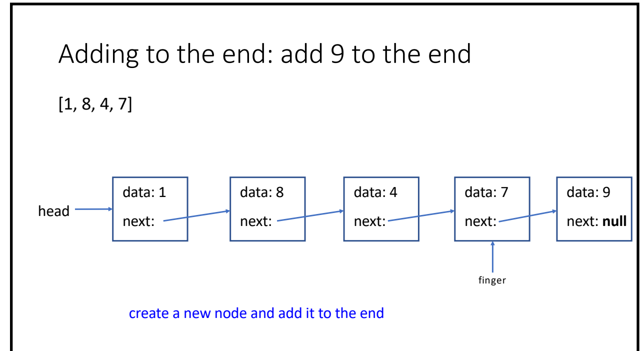
3



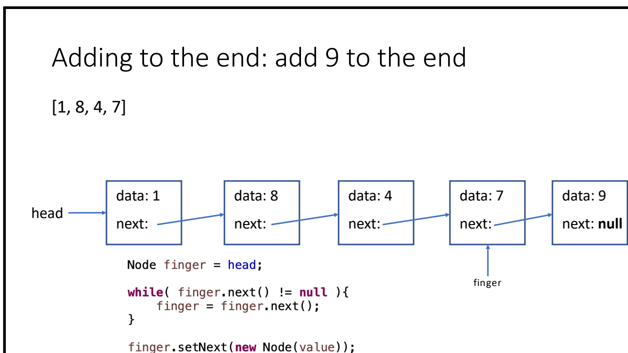
4



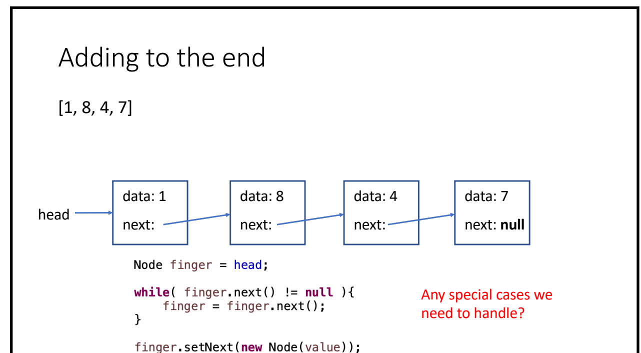
5



6



7



8

Adding to the end

[1, 8, 4, 7]

head: **null**

What would happen here?

```
Node finger = head;

while( finger.next() != null ){
    finger = finger.next();
}

finger.setNext(new Node(value));
```

9

Adding to the end

[1, 8, 4, 7]

head: **null**

```
public void addLast(E value){
    if( head == null ){
        head = new Node(value);
    }else{
        Node finger = head;

        while( finger.next() != null ){
            finger = finger.next();
        }

        finger.setNext(new Node(value));
    }
}
```

10

Removing a value

[1, 8, 4, 7]

head → [data: 1, next:] → [data: 8, next:] → [data: 4, next:] → [data: 7, next: **null**]

How can we we remove a value from the list?

11

Removing a value: remove 4

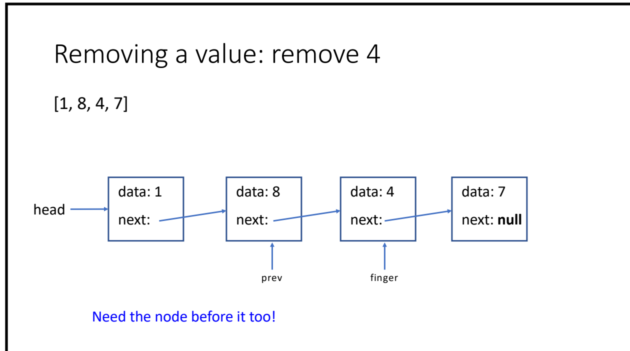
[1, 8, 4, 7]

head → [data: 1, next:] → [data: 8, next:] → [data: 4, next:] → [data: 7, next: **null**]

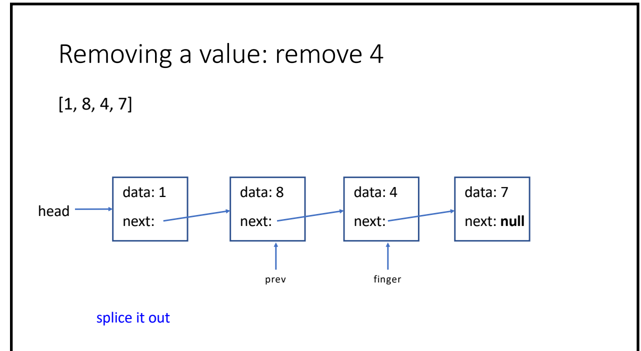
finger

Find it

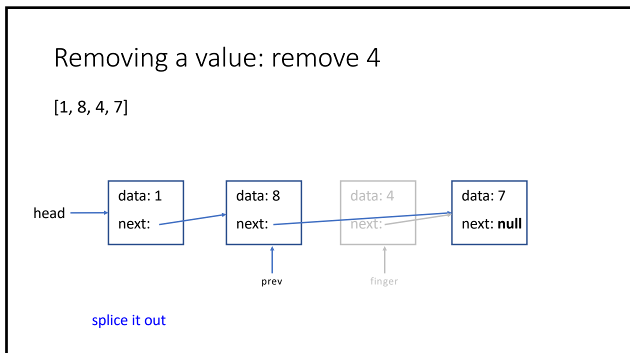
12



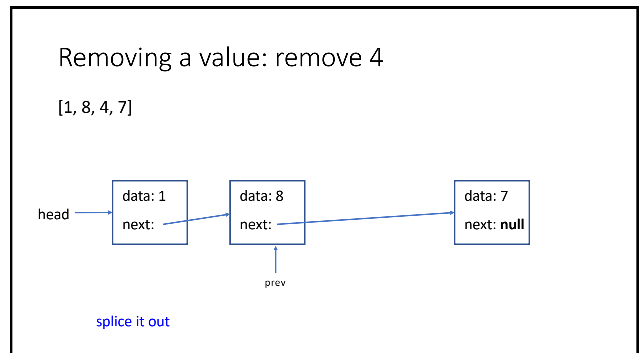
13



14

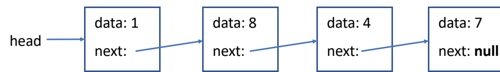


15



16

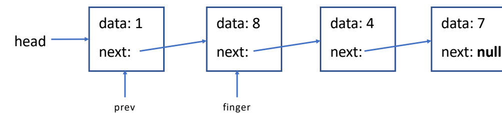
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

17

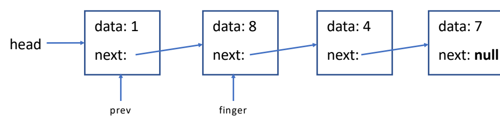
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

18

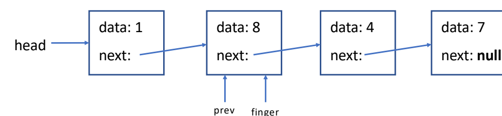
Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

19

Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

20

Removing a value

head → data: 1 next: → data: 8 next: → data: 4 next: → data: 7 next: **null**

prev → data: 8; finger → data: 4

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

21

Removing a value

head → data: 1 next: → data: 8 next: → data: 4 next: → data: 7 next: **null**

prev → data: 8; finger → data: 4

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

22

Removing a value

head → data: 1 next: → data: 8 next: → data: 4 next: → data: 7 next: **null**

prev → data: 8; finger → data: 4

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

When would finger be null?

23

Removing a value

head → data: 1 next: → data: 8 next: → data: 7 next: **null**

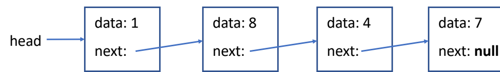
data: 4 next: → data: 7

prev → data: 8; finger → data: 4

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

24

Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Any special cases we need to handle?

25

Removing a value

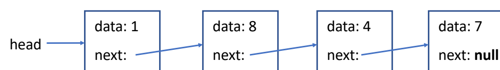
head: **null**

What would happen here?

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

26

Removing a value

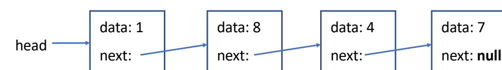



```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Any *other* special cases we need to handle?

27

Removing a value




```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
    prev = finger;
    finger = finger.next();
}

if( finger != null ){
    prev.setNext(finger.next());
}
```

Delete 1. Does it work?

28

Removing a value

head → data: 1 next: → data: 8 next: → data: 4 next: → data: 7 next: **null**

prev (points to data: 1), finger (points to data: 8)

```
Node finger = head.next();
Node prev = head;

while (finger != null && !finger.value().equals(value)){
     prev = finger;
     finger = finger.next();
}

if( finger != null ){
     prev.setNext(finger.next());
}
```

Delete 1. Does it work?

29

Removing a value

```
public void remove(E value){
    if( head != null ) {
        if( head.value().equals(value) ){
            head = head.next();
        }else{
            Node finger = head.next();
            Node prev = head;

            while (finger != null && !finger.value().equals(value)){
                prev = finger;
                finger = finger.next();
            }

            if( finger != null ){
                prev.setNext(finger.next());
            }
        }
    }
}
```

30

Removing a value

```
public void remove(E value){
    if( head != null ) {
        if( head.value().equals(value) ){
            head = head.next();
        }else{
            Node finger = head.next();
            Node prev = head;

            while (finger != null && !finger.value().equals(value)){
                prev = finger;
                finger = finger.next();
            }

            if( finger != null ){
                prev.setNext(finger.next());
            }
        }
    }
}
```

handle empty list

removing first element

all other elements

31

Linked lists: fast or slow?

add to the end
add to the front
contains
get
insert at an index
remove an element
set the value of an existing element
size

32

Linked lists: fast or slow?

add to the end slow
add to the front: fast
contains slow
get slow
insert at an index slow
remove an element slow
remove from the front fast
set the value of an existing element slow
size fast

33

Linked lists: fast or slow?

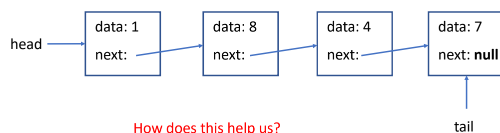
add to the end slow
add to the front: fast
contains slow
get slow
insert at an index slow
remove an element slow
remove from the front fast
set the value of an existing element slow
size fast

Can we make any of these faster?

34

Linked list visualized

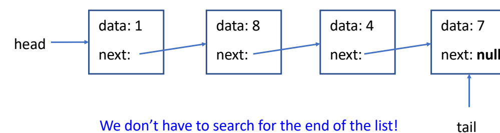
[1, 8, 4, 7]



How does this help us?

35

Linked list visualized



We don't have to search for the end of the list!

```
tail.setNext(new Node(value));
tail = tail.next();
```

36

Linked list visualized


```
public void addLast(E value){
    if( head == null ){
        head = new Node(value);
        tail = head;
    }else{
        tail.setNext(new Node(value));
        tail = tail.next();
    }
}
```

37

Linked list visualized

[1, 8, 4, 7]

Any downsides?

38

Circularly linked list visualized

[1, 8, 4, 7]

Where is head?

39

Circularly linked list visualized

[1, 8, 4, 7]

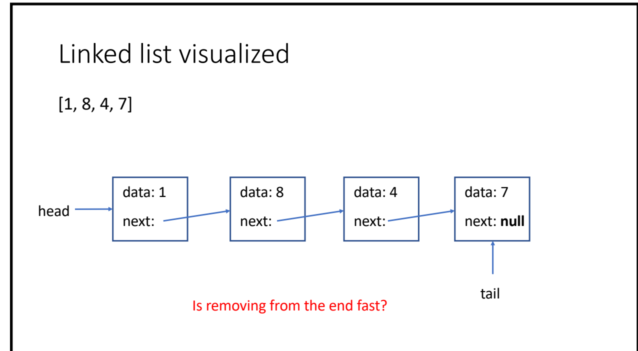
tail.next()

40

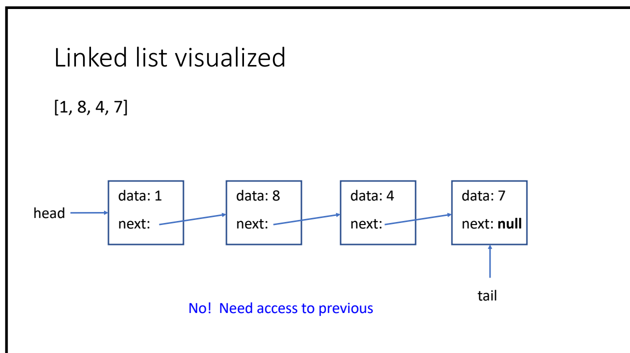
Linked lists: fast or slow?

add to the end	fast
add to the front:	fast
contains	slow
get	slow
insert at an index	slow
remove an element	slow
remove from the front	fast
set the value of an existing element	slow
size	fast

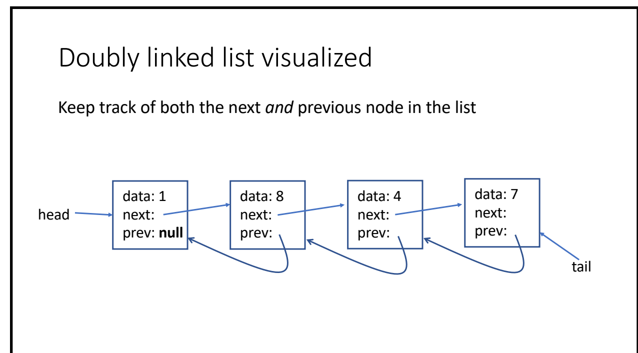
41



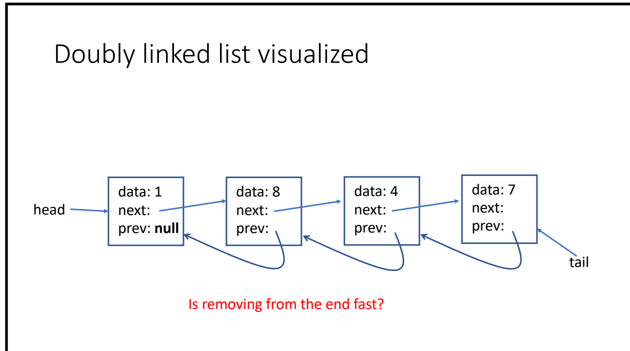
42



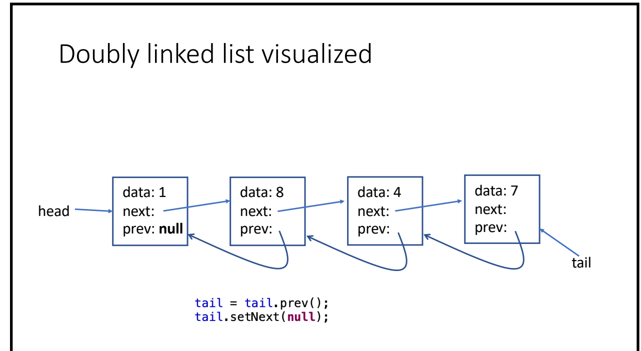
43



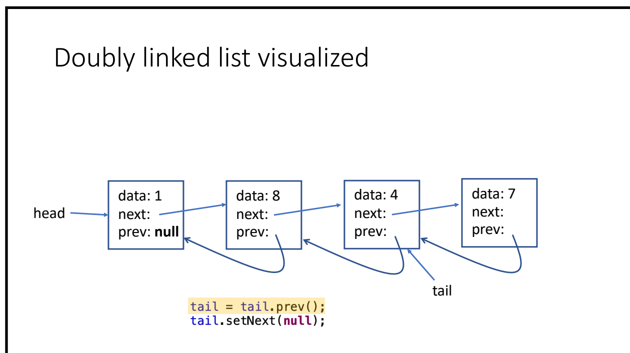
44



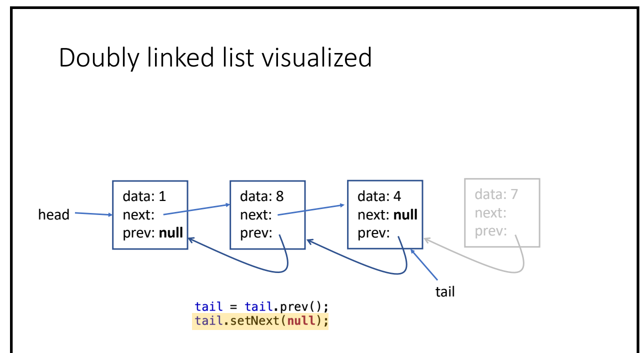
45



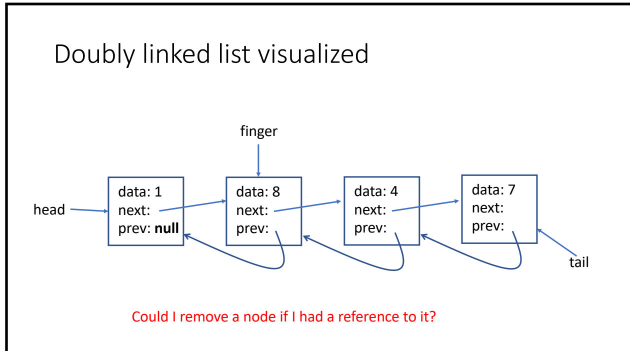
46



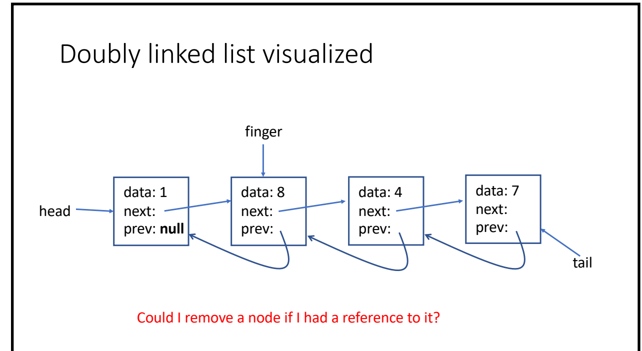
47



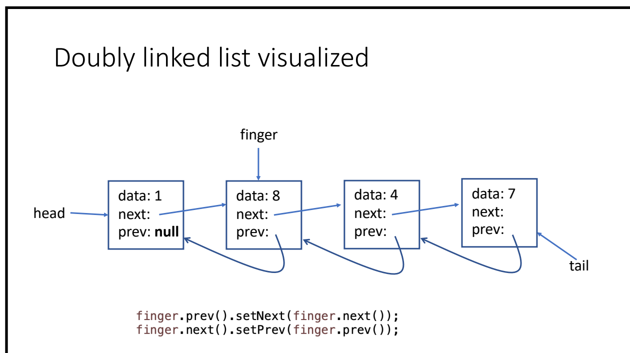
48



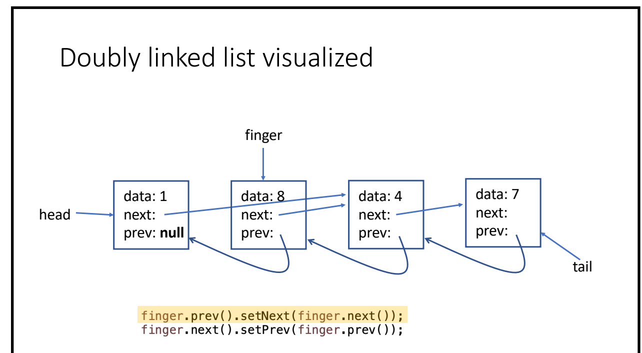
49



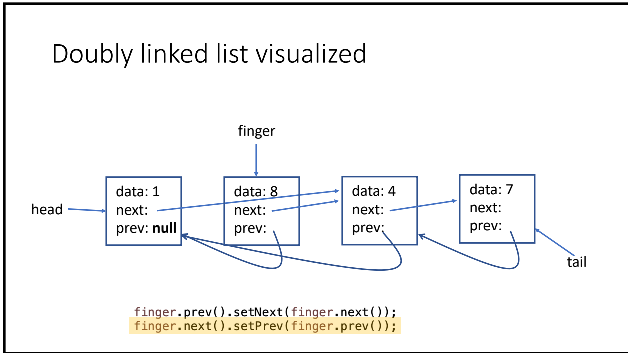
50



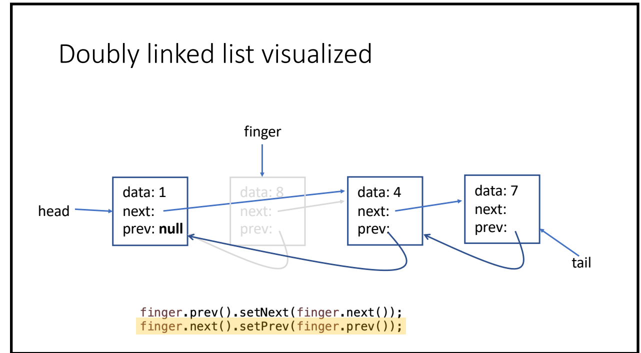
51



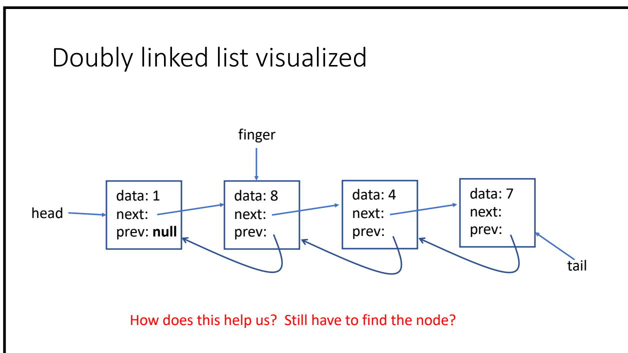
52



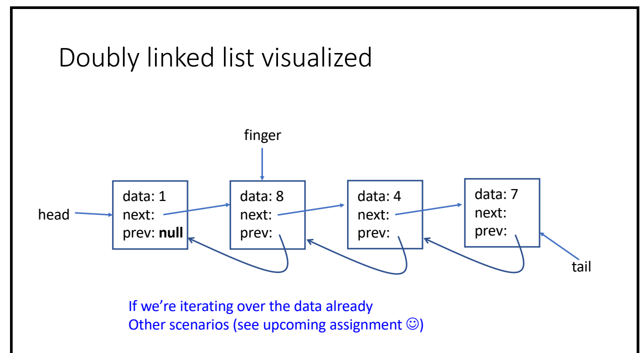
53



54



55



56

List performance

	ArrayList	Singly linked list	Singly linked list (tail)	Doubly linked list
add to end	fast (amortized)	slow	fast	fast
add to front	slow	fast	fast	fast
insert at index	slow	slow	slow	slow
contains	slow	slow	slow	slow
get	fast	slow	slow	slow
set	fast	slow	slow	slow
remove	slow	slow	slow	slow
remove from end	fast	slow	slow	fast
remove from front	slow	fast	fast	fast
size	fast	fast	fast	fast

57