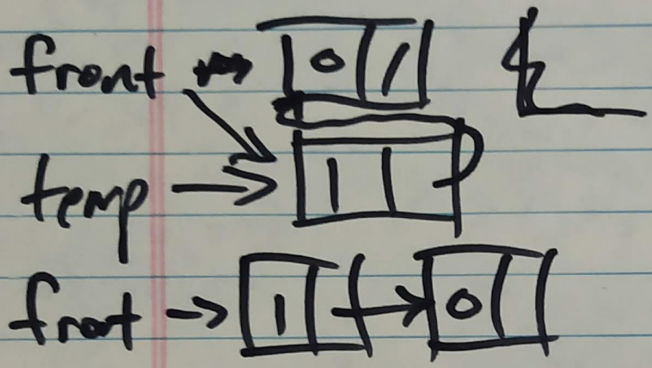
Linked Lists 3

n = 10 [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

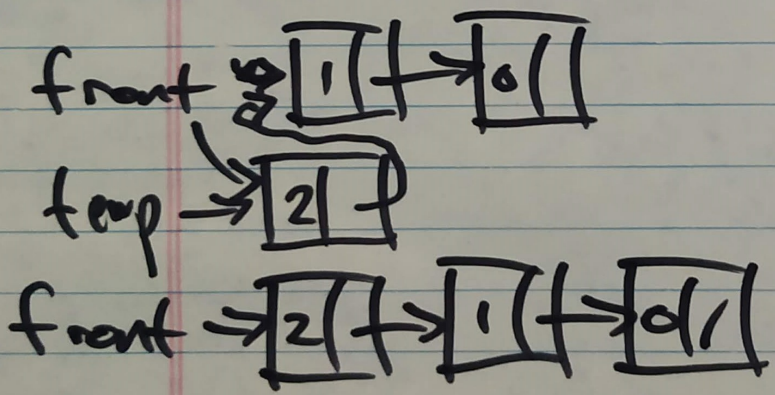
```
public LinkedIntList (int n) {
    front = null;
    for(int i = 0; i <= n; i++) {
        front = new ListNode(i, front);
    }
}
```

```
front = null;
front = new ListNode(0);
```

(0, front)




```
ListNode temp = new ListNode(1, front);
front = temp;
```

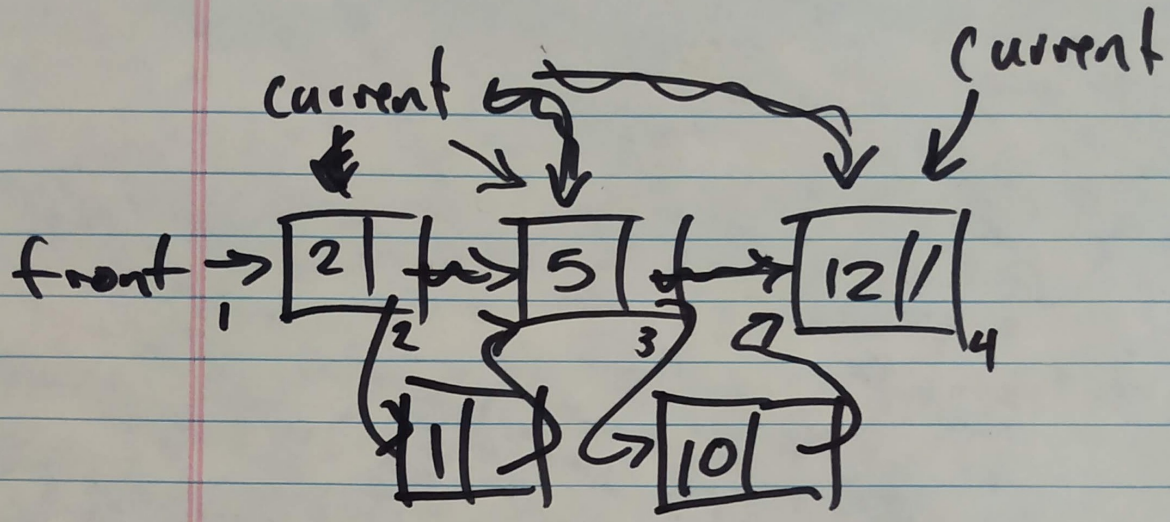


X
```
ListNode temp = new ListNode(2, front);
front = temp;
```

```
front = new ListNode(2, front);
```

```
// pre: list is already sorted
// post: insert given value to preserve order
public void addSorted(int value) {
    ...
}
```

Start with the "middle case" -- what happens usually



10

13

X

```
ListNode current = front;
while( current.data < value ) {
    current = current.next;
}
current = ....
```

```
ListNode current = front;
while(current.next.data < value) {
    current = current.next;
}
current.next = new ListNode(value, current.next)
```

arr[i+1]

x = t + 1

while
if(current.next.data < value &&
current.next != null)

"short-circuited evaluation"

robust false && ——— sensitive

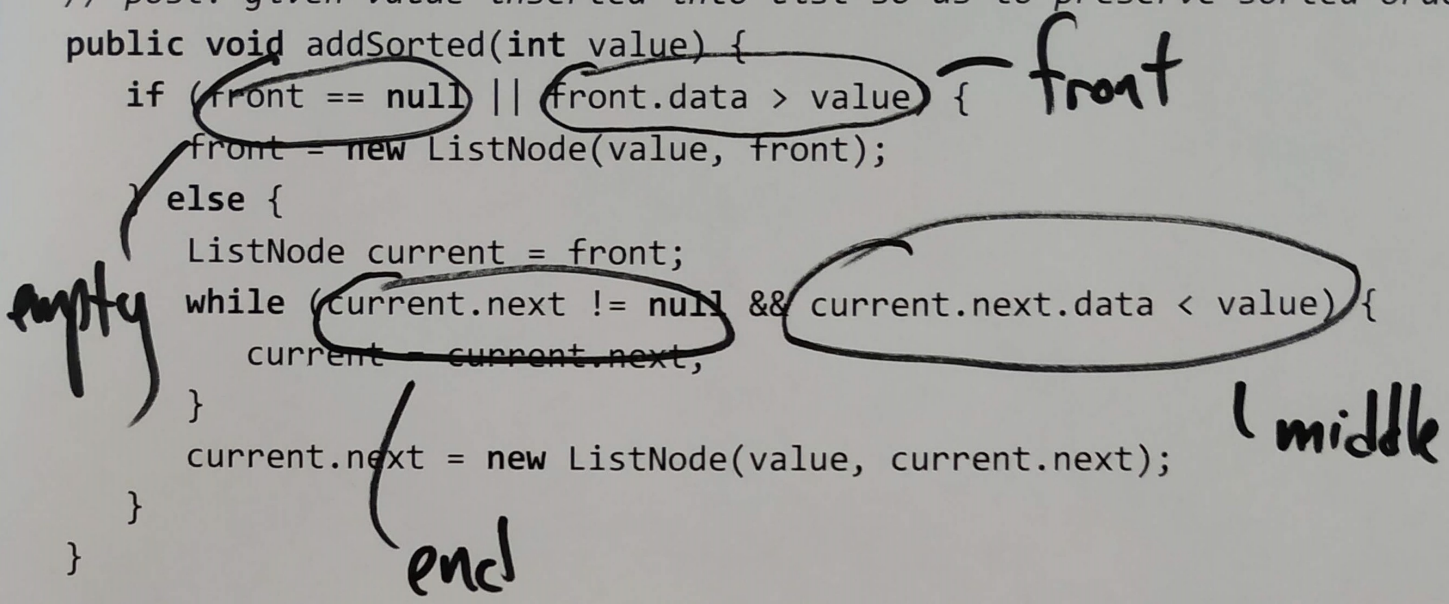
true || ———

while(current.next != null &&
current.next.data < value)

```
if(front.data >= value) {
    front = new ListNode(value, front);
}
if(front.data >= value || front == null)

if(front == null || front.data >= value){
    ...
}
```

```
// pre : list is in sorted (non-decreasing) order
// post: given value inserted into list so as to preserve sorted order
public void addSorted(int value) {
   if (front == null || front.data > value) {
      front = new ListNode(value, front);
   } else {
      ListNode current = front;
      while (current.next != null && current.next.data < value) {
         current = current.next;
      }
      current.next = new ListNode(value, current.next);
   }
}
```



```java
// pre : list is in sorted (non-decreasing) order
// post: given value inserted into list so as to preserve sorted order
public void addSorted(int value) {
   if (front == null || front.data > value) {
      front = new ListNode(value, front);
   } else {
      ListNode prev = front;
      ListNode current = front.next;
      while (current != null && current.data < value) {
         prev = prev.next;
         current = current.next;
      }
      prev.next = new ListNode(value, current);
   }
}
```