Principles of Computer Science

Sorting Linked Lists

Sorting Linked Lists

- Assumes that contents are objects that can be compared.

```
class Student {
  ...

  int compareByName(Student another) {
    return this.name.compareTo(another.name);
  }

  int compareByGrade(Student another) {
    if(this.grade == another.grade)
      return 0;      // this is equal to the other
    else if( this.grade < another.grade )
      return -1;     // this less than the other
    else
      return 1;      // this greater than the other
  }
}
```

Sorting Linked Lists

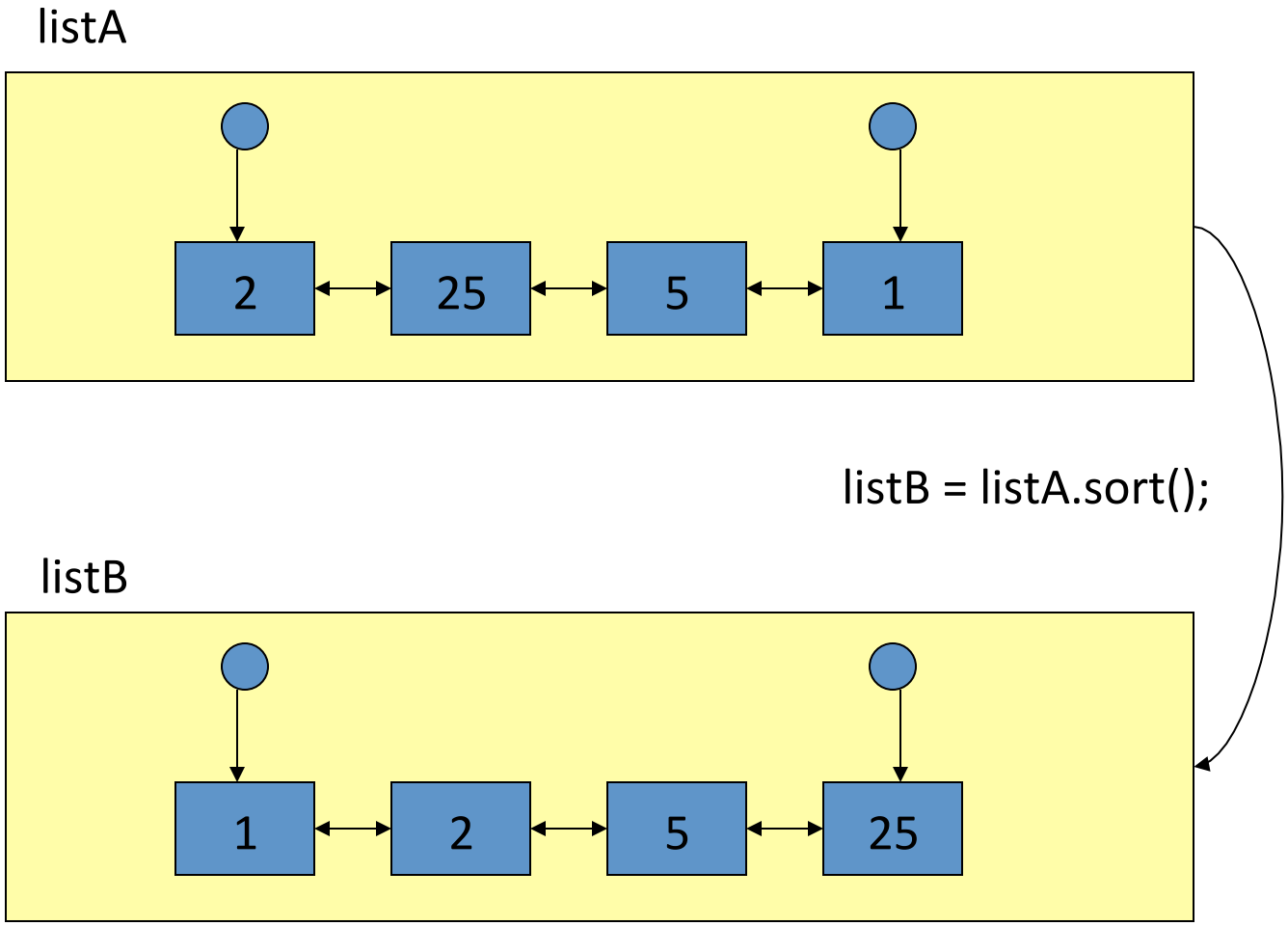
- Out of Place Sorting
  - Produces a new linked list whose elements are sorted
  - The current list is unchanged

```
class DoubleLinkedList {
  ...
  DoubleLinkedList sort() {
    ...
  }
}
```

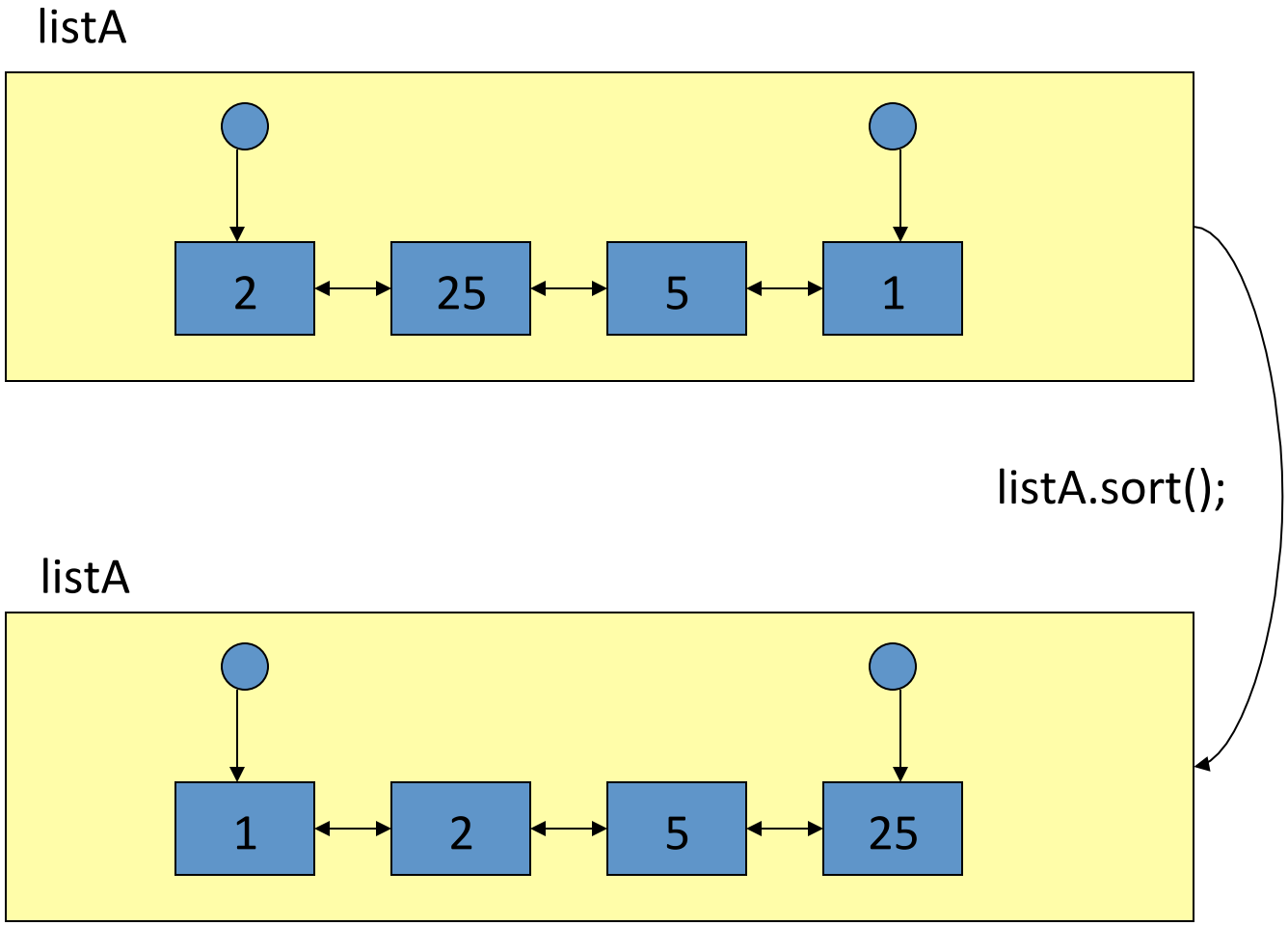
- In Place Sorting
  - The current list is modified and its elements are now sorted

```
class DoubleLinkedList {
  ...
  void sort() {
    ...
  }
}
```

Out of Place Sorting



In Place Sorting



Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  private ListElement nextGreater(ListElement elem) {
    ListElement cursor = this.firstElement;
    ListElement result = null;
    while( cursor != null ) {
      // process only elements bigger than elem
      if(cursor.compareTo(elem) > 0) {
        // check if cursor is smaller than result
        if(result == null)
          result = cursor;
        else
          if(cursor.compareTo(result) < 0) result = cursor;
      }
      cursor = cursor.next;
    }
    return result;
  }
}
```

Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  private ListElement smallest {
    ListElement cursor = this.firstElement;
    ListElement result = null;
    while( cursor != null ) {
      if( result == null )
        result = cursor;
      else {
        if(cursor.compareTo(result) < 0)
          result = cursor;
      }
      cursor = cursor.next;
    }
    return result;
  }
}
```

Implementing Out of Place Sorting

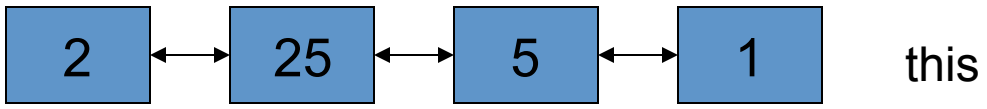
```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```

Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() { ... }
  private ListElement smallest() { ... }
  private ListElement nextBigger() { ... }
}
```

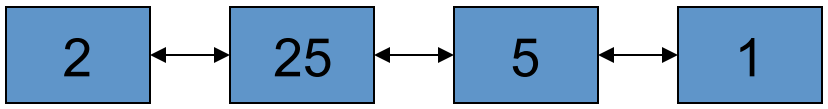
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```

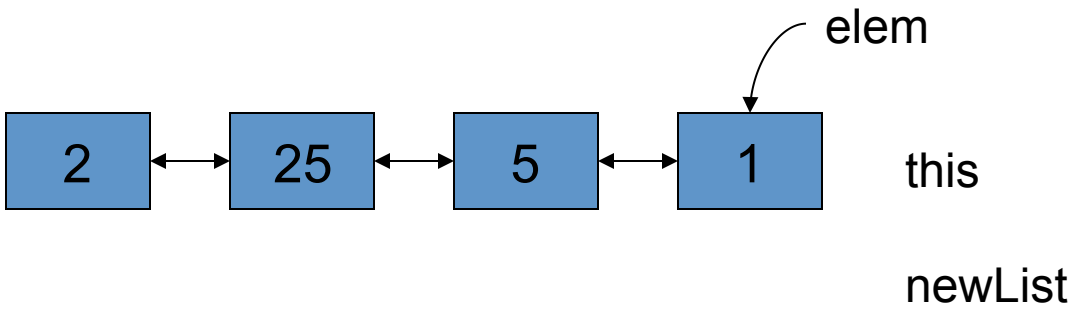


this

newList

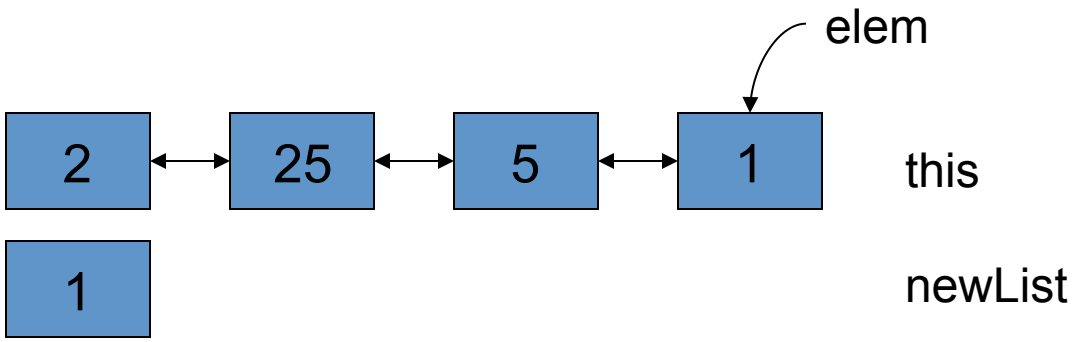
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



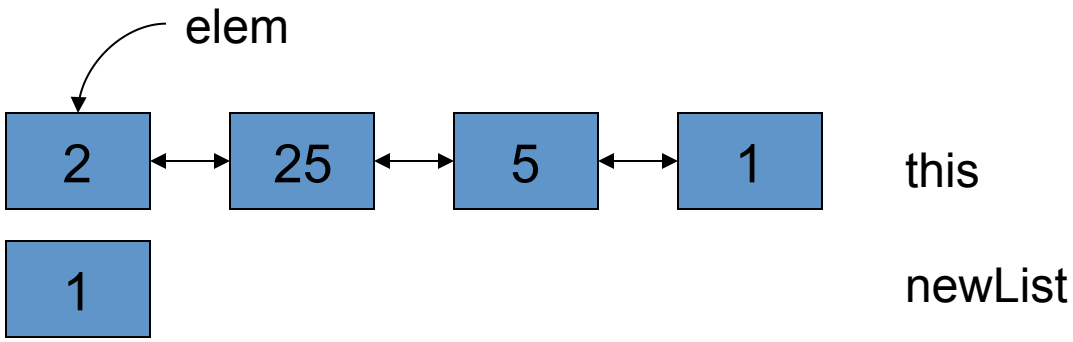
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



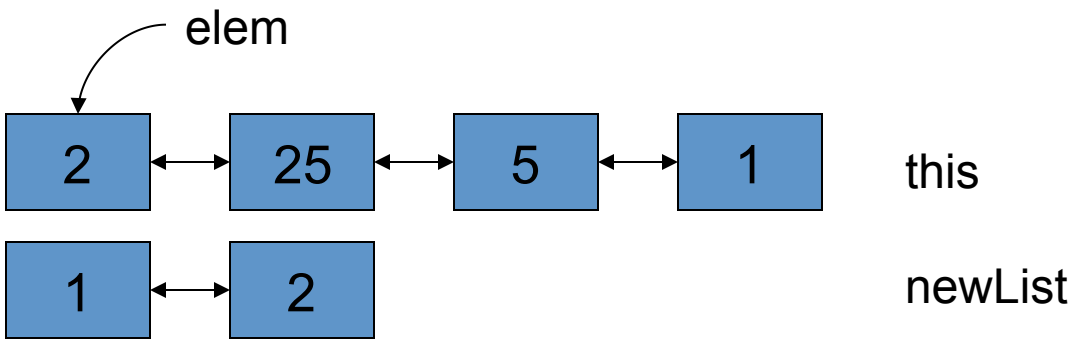
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



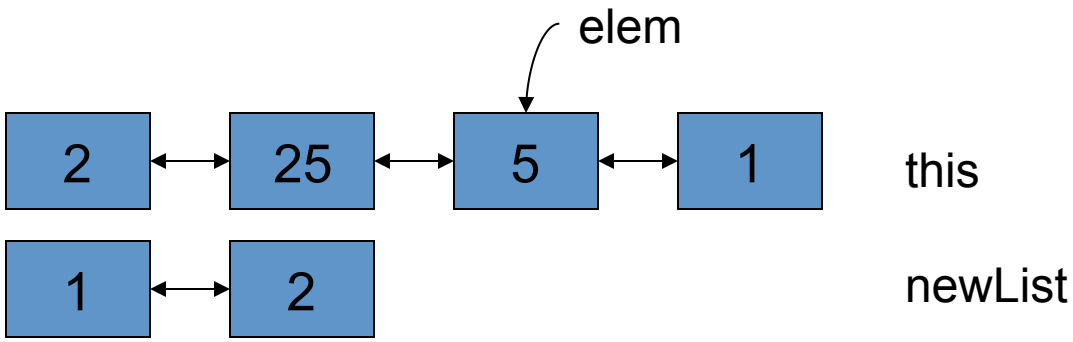
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



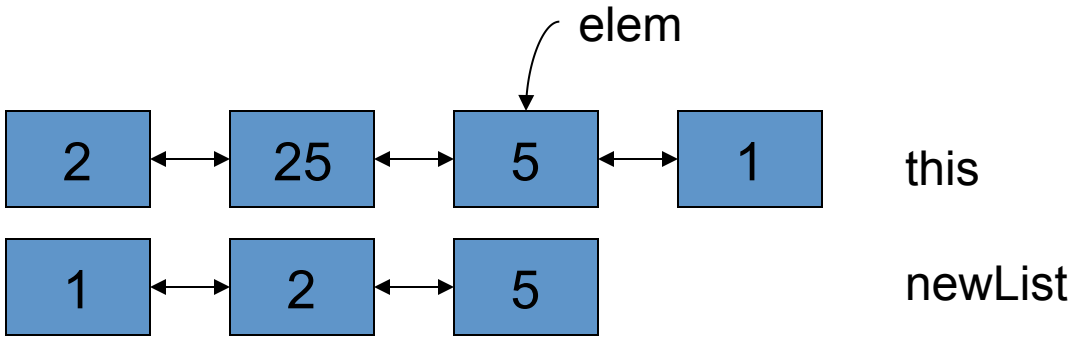
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



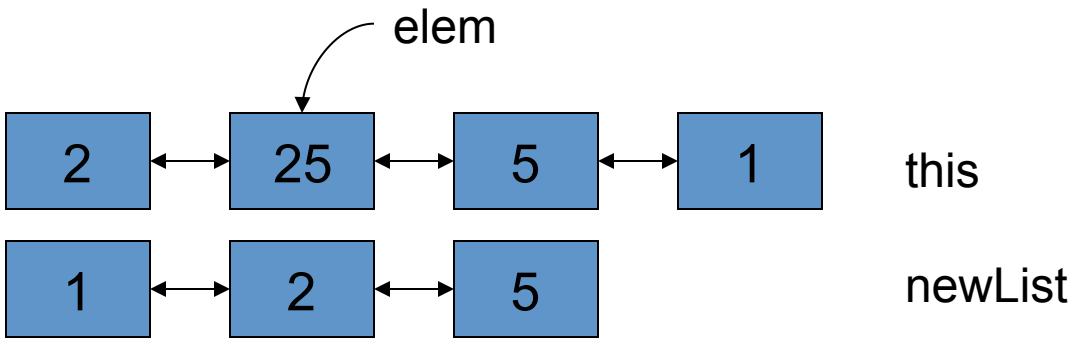
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



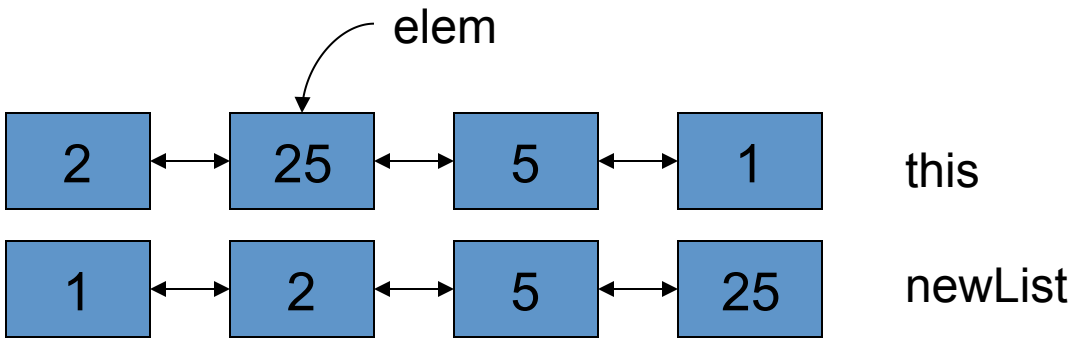
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



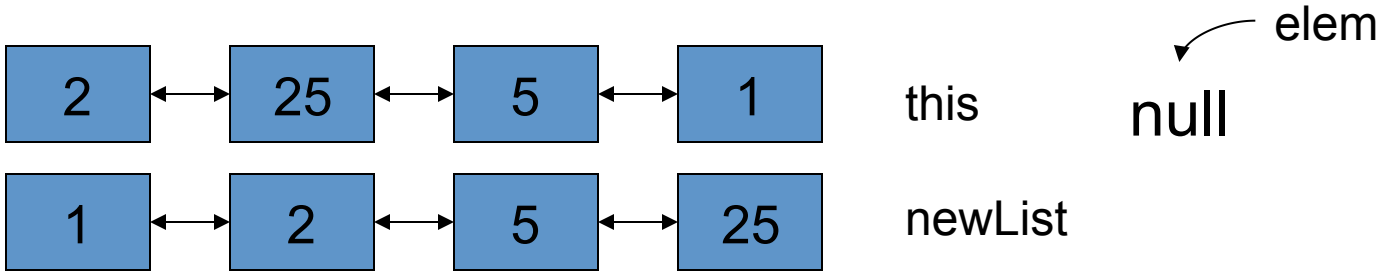
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```



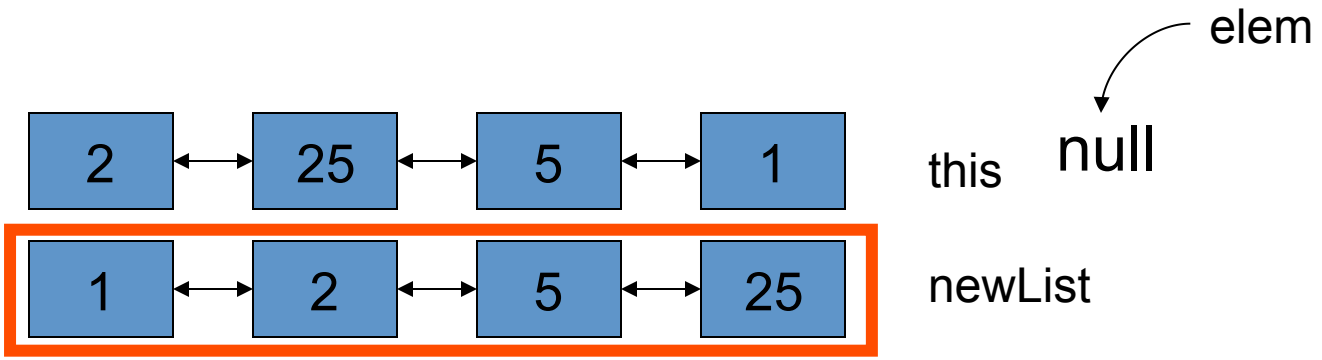
Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```

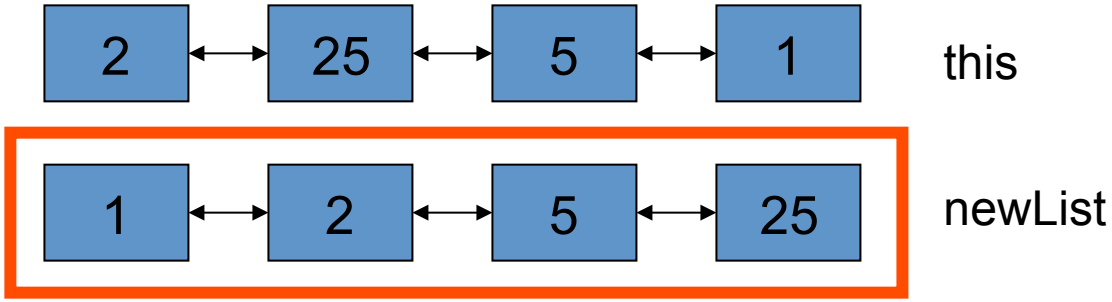


Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  public DoubleLinkedList sort() {
    DoubeLinkedList newList = new DoubleLinkedList();
    ListElement elem = this.smallest();
    while(elem != null) {
      newList.add(elem);
      elem = this.nextBigger(elem);
    }
    return newList;
  }
}
```

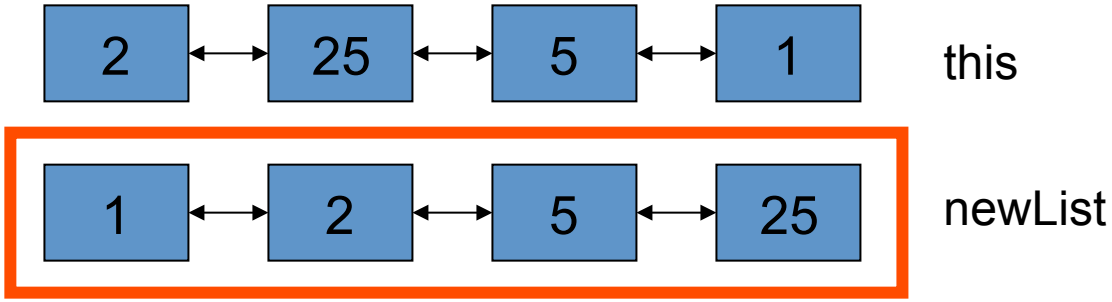


Out of Place Sorting



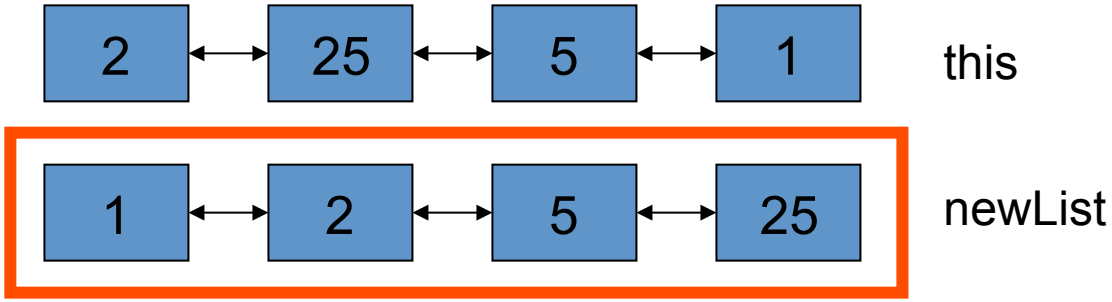
- Is this algorithm correct? Can you think of a case where this algorithm will fail?

Out of Place Sorting



- Is this algorithm correct? Can you think of a case where this algorithm will fail?
- What happens when there are duplicate items in the list?

Out-of-Place Sorting



- Is this algorithm correct? Can you think of a case where this algorithm will fail?
- What happens when there are duplicate items in the list?
- **The algorithm *cannot* handle duplicate items**

Out of Place Insert Sort

- Complexity Analysis
  - Time complexity
  - Space complexity

Out of Place Insert Sort

- Time Complexity
  - There are **N** elements in the original list.
  - Assuming NO DUPLICATES
  - ```
    while (elem != null) {
        . . .
      elem = this.nextGreater(elem)
    }
    ```
    iterates at most **N** times.
  - In each iterator, we call this.nextGreater(elem) once.
  - $\mathbf{T_{sort}(N) = N * T_{nextGreater}(N)}$
    ```

Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  private ListElement nextGreater(ListElement elem) {
    ListElement cursor = this.firstElement;
    ListElement result = null;
    while( cursor != null ) {
      // process only elements bigger than elem
      if(cursor.compareTo(elem) > elem) {
        // check if cursor is smaller than result
        if(result = null)
          result = cursor;
        else
          if(cursor.compareTo(result) < 0) result = cursor;
      }
      cursor = cursor.next;
    }
    return result;
  }
}
```

Implementing Out of Place Sorting

```
class DoubleLinkedList {
  ...
  private ListElement nextGreater(ListElement elem) {
    ListElement cursor = this.firstElement;
    ListElement result = null;
    while( cursor != null ) {
      // process only elements bigger than elem
      if(cursor.compareTo(elem) > elem) {
        // check if cursor is smaller than result
        if(result = null)
          result = cursor;
        else
          if(cursor.compareTo(result) < 0) result = cursor;
      }
      cursor = cursor.next;
    }
    return result;
  }
}
```

$T_{nextGreater}(N) = N$

Out of Place Insert Sort

- Time Complexity
  - There are **N** elements in the original list.
  - Assume NO DUPLICATES
  - while (elem != null) {
    . . .
    elem = this.nextGreater(elem)
    }
    iterates at most **N** times.
  - In each iterator, we call this.nextGreater(elem) once.

  - $\mathbf{T_{sort}(N) = N * T_{nextGreater}(N)}$

  - $\mathbf{T_{sort}(N) = N * N = N^2}$

Out of Place Sorting

- Space complexity
  - how much space is needed if the original list contains **N** elements?
  - The newList contains N elements
- Space complexity of sort() is **N.**
- Requires linear space to sort the original list.

In Place sorting

- Modifies the linked list without creating a new list.
- The original ordering of elements is **lost.**
- Saves space.
- class DoubleLinkedList {
  public void sortInPlace();
  }

In Place Sorting: Bubble sort

```
class DoubleLinkedList {
  ...
  private void swapWithNext(ListElement elem) { ... }
  public int size() { ... }
  public void bubbleSort() {
    while( true ) {
      boolean outOfOrder = false;
      ListElement cursor = this.firstElement;
      for(int I=0; I < this.size() - 1; I ++) {
        if(cursor.compareTo(cursor.next) > 0) {
          this.swapWithNext(cursor);
          outOfOrder = true;
        } else
          cursor = cursor.next;
      }
      if(! outOfOrder) break;
    }
  }
}
```

Exercise: run through the code

```
class DoubleLinkedList {
  ...
  private void swapWithNext(ListElement elem) { ... }
  public int size() { ... }
  public void bubbleSort() {
    while( true ) {
      boolean outOfOrder = false;
      ListElement cursor = this.firstElement;
      for(int I=0; I < this.size() - 1; I ++) {
        if(cursor.compareTo(cursor.next) > 0) {
          this.swapWithNext(cursor);
          outOfOrder = true;
        } else
          cursor = cursor.next;
      }
      if(! outOfOrder) break;
    }
  }
}
```

Summary

- Look at an implementation of doubly linked list
- Out-place sorting
- In-place sorting
- More study about sorting algorithms