Lecture 10 - February 6

Arrays and Linked Lists

SLL: List Constructions
SLL: getSize and getTail
Trading Space for Time: tail and size
SLL: addFirst

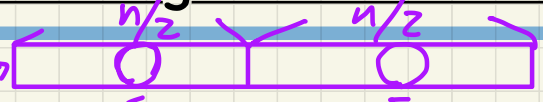
Announcements/Reminders

- **Assignment 2** (on **SLL**) released
  + **Required** studies: Generics in Java (Slides 33 – 36)
  + **Recommended** studies: extra SLL problems
- **Assignment 1** solution released
- ***splitArrayHarder***: an **extended** version released
- Lecture notes template available
- Office Hours: 3pm to 4pm, Mon/Tue/Wed/Thu
- Contact Information of TAs on common eClass site

* Precondition: $\forall i, j \cdot 0 \le i < \frac{n}{2} \wedge \frac{n}{2} \le j < n \Rightarrow input[i] \le input[j]$

Exercise: Mixing the "Best" from both Sorts?

Recall:



$input[i] \le input[j]$

- In insertion sort, costs of insertions are increasing.
- In selection sort, costs of selections are decreasing.

Idea:

- Perform insertion sort until half of the input is sorted.
- Perform selection sort to finish sorting the remaining half.

Q: Will this "new" algorithm perform better than $O(n^2)$?

Not working

<2, 1, 7, 3, 4, 0>

insertions

<1, 2, 7, 3, 4, 0>

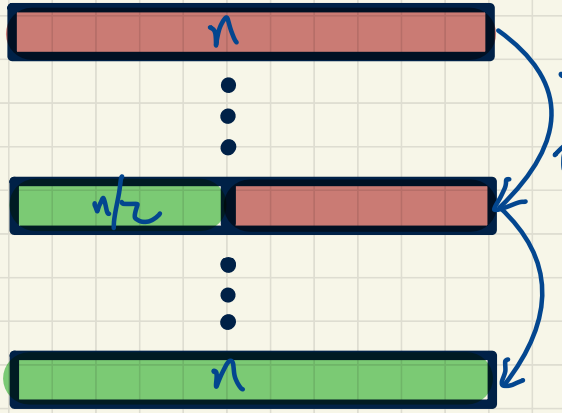
selections ✗

<1, 2, 7, 0, 3, 4>

Working

<2, 0, 1, 4, 7, 3>

$O(n^2 + n^2) = O(n^2)$



$O(\underbrace{1 + 2 + \cdots + n/2}_{\frac{(1 + n/2) \cdot n/2}{2}}) = O(n^2)$

1st insertion, $n/2$th insertion

$O(\underbrace{n/2 + (n/2 - 1) + \cdots + 1}_{((n/2 + 1) \cdot n/2)/2}) = O(n^2)$

SLL: Constructing a Chain of Nodes

Alan → Mark → Tom

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

Aliasing

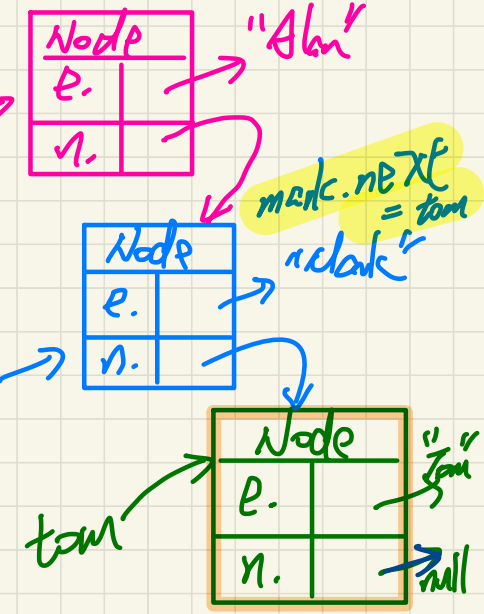
1. tom
2. mark.next
3. alan.next.next

Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
```

unknown variable, compilation error

```
Node alan = new Node("Alan", mark);
Node mark = new Node("Mark", tom);
Node tom = new Node("Tom", null);
```

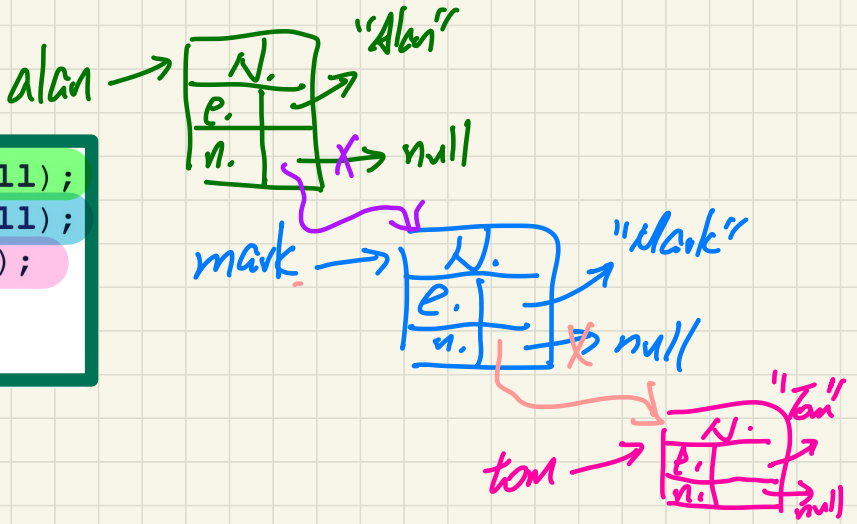


SLL: Constructing a Chain of Nodes

```
public class Node {
   private String element;
   private Node next;
   public Node(String e, Node n) { element = e; next = n; }
   public String getElement() { return element; }
   public void setElement(String e) { element = e; }
   public Node getNext() { return next; }
   public void setNext(Node n) { next = n; }
}
```

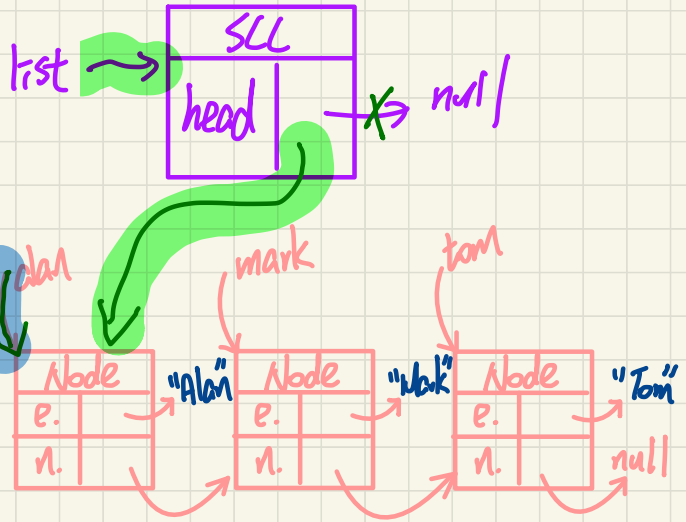
Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
```



SLL: Setting a List's Head to a Chain of Nodes

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```



Approach 1

```
Node tom = new Node("Tom", null);
Node mark = new Node("Mark", tom);
Node alan = new Node("Alan", mark);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

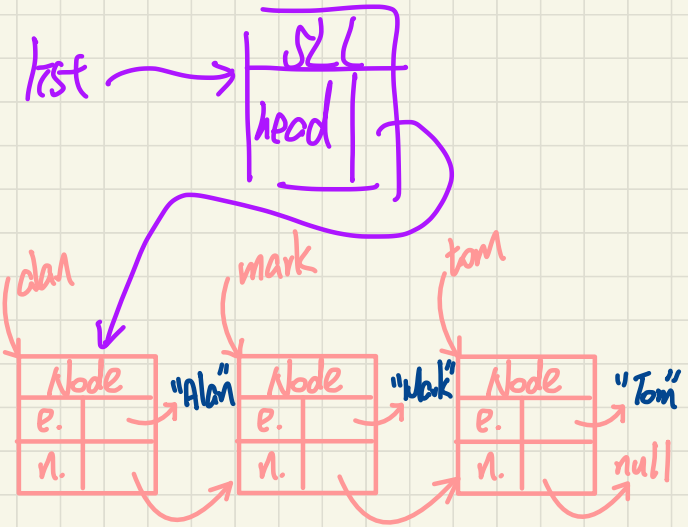
default const.

list.head == alan

T.

SLL: Setting a List's Head to a Chain of Nodes

```
public class SinglyLinkedList {
   private Node head = null;
   public void setHead(Node n) { head = n; }
   public int getSize() { ... }
   public Node getTail() { ... }
   public void addFirst(String e) { ... }
   public Node getNodeAt(int i) { ... }
   public void addAt(int i, String e) { ... }
   public void removeLast() { ... }
}
```

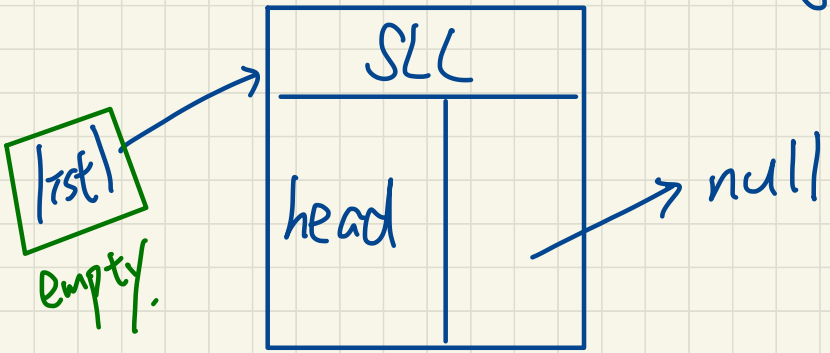


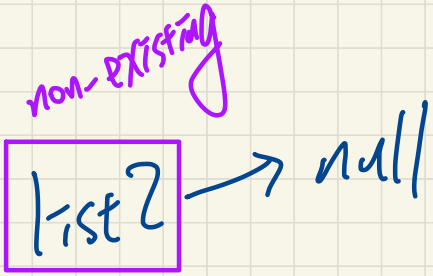
Approach 2

```
Node alan = new Node("Alan", null);
Node mark = new Node("Mark", null);
Node tom = new Node("Tom", null);
alan.setNext(mark);
mark.setNext(tom);
SinglyLinkedList list = new SinglyLinkedList();
list.setHead(alan);
```

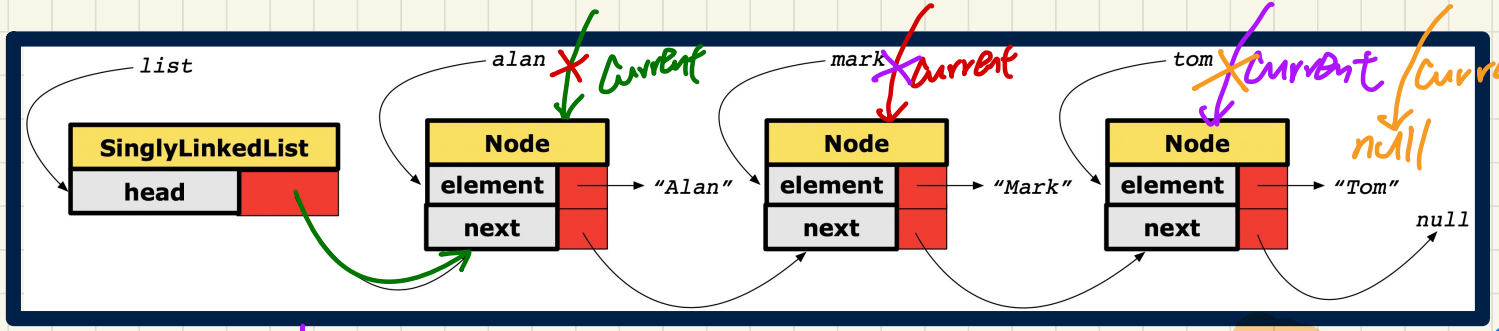
Empty SLL

```
SLL list1 = new SLL();
SLL list2 = null;
```





SLL Operation: Counting the Number of Nodes



a SLL method.

```
1  int getSize() {
2    int size = 0;
3    Node current = head;
4    while (current != null) {
5      current = current.getNext();
6      size ++;
7    }
8    return size;
9  }
```

≈ loop counter

exit when current == null

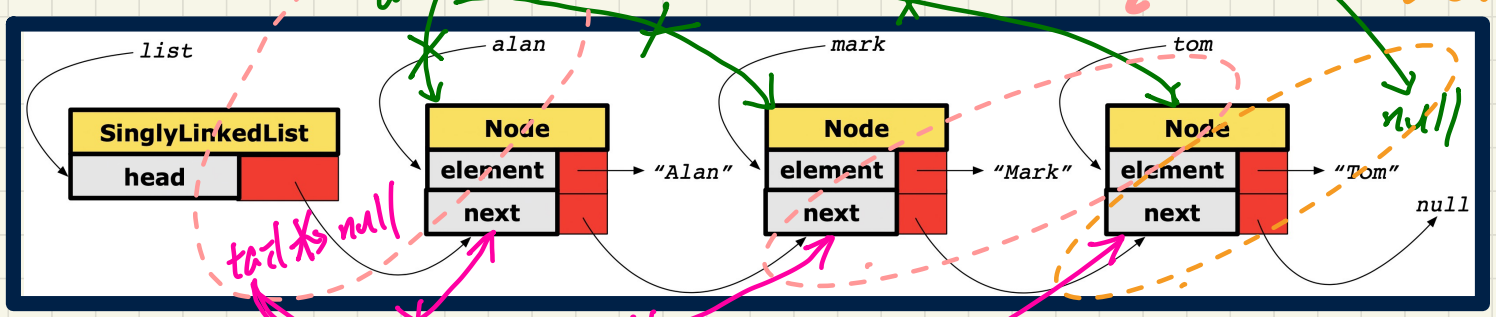
O(n) may be improved.

Trace: list.getSize()

O(n) # iterations = # nodes in chain

current	current != null	End of Iteration	size
alan	alan != null	current == mark	1
mark	mark != null	current == tom	2
tom	tom != null	current == null	3
null	null != null		
	F		

SLL Operation: Finding the Tail of the List




```
1  Node getTail() {
2      Node current = head;
3      Node tail = null;
4      while (current != null) {
5          tail = current;
6          current = current.getNext();
7      }
8      return tail;
9  }
```

Trace: list.getTail()

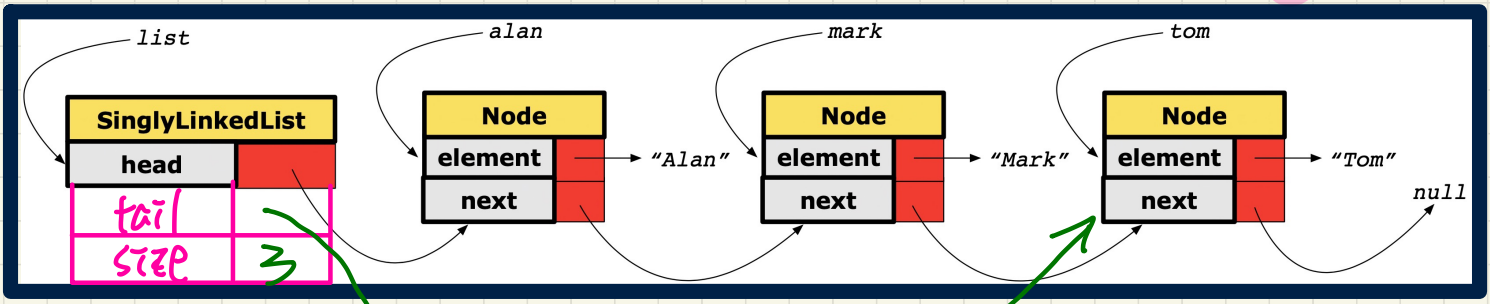
current	current != null	End of Iteration	tail

iterations = size of list

O(n)

SLL: Trading Space for Time

waste more memory space to make subsequent computations cheaper



Catch: For methods that might impact head, tail, or size of a SLL, the bodies of these methods must update them properly.

SLL class

attributes ⇓ more cost of space
- head
- tail
- size

SLL list = new SLL();

attribute access ⇓ less cost for running time
- list.tail
- list.size

O(1)

Q. Does tail or size need to be updated?

SLL Operation: Inserting to the Front of the List

```java
@Test
public void testSLL_02() {
    SinglyLinkedList list = new SinglyLinkedList();
    assertTrue(list.getSize() == 0);
    assertTrue(list.getFirst() == null);

    list.addFirst("Tom");
    list.addFirst("Mark");
    list.addFirst("Alan");
    assertTrue(list.getSize() == 3);
    assertEquals("Alan", list.getFirst().getElement());
    assertEquals("Mark", list.getFirst().getNext().getElement());
    assertEquals("Tom", list.getFirst().getNext().getNext().getElement());
}
```

```java
1  void addFirst (String e) {
2    head = new Node(e, head);
3    if (size == 0) {
4      tail = head;
5    }
6    size ++;
7  }
```