Lecture 9 - February 4

Arrays and Linked Lists

Q: Mixing Insertion & Selection Sorts
SLL: Visual Introduction & Operations
SLL in Java: Node vs. SinglyLinkedList

Announcements/Reminders

- Assignment 2 (on SLL) to be released soon
- 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

Exercise: Mixing the "Best" from both Sorts?

(selections)

Recall:

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

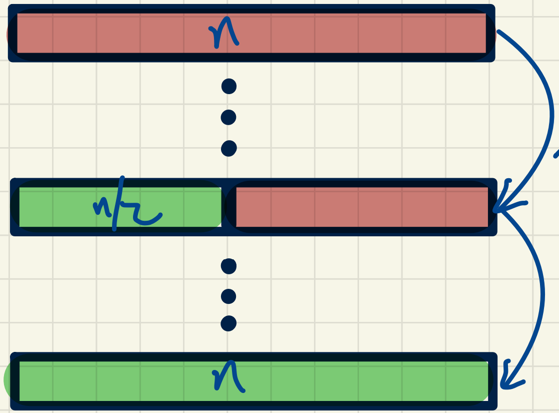
Idea:

- 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(n²)?

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

$= \boxed{O(n^2)}$



$\frac{(1+n/2)\cdot n/2}{2}$

n/2 insertions $O(\boxed{1 + 2 + \cdots + n/2}) = O(n^2)$

1st insertion

n/2 th insertion

$((n/2+1)\cdot n/2)/2$

n/2 selections $O(\boxed{n/2 + (n/2-1) + \cdots + 1}) = O(n^2)$

Singly-Linked Lists (SLL): Visual Introduction

- A chain of connected nodes (via aliasing)
- Each node contains:
  + reference to a data object
  + reference to the next node
- Head vs. Tail
- Accessing a position in a linear collection:
  + Array uses absolute indexing: O(1)
  + SLL uses relative positioning: O(n)
- The chain may grow or shrink dynamically.

e.g. m == n.next

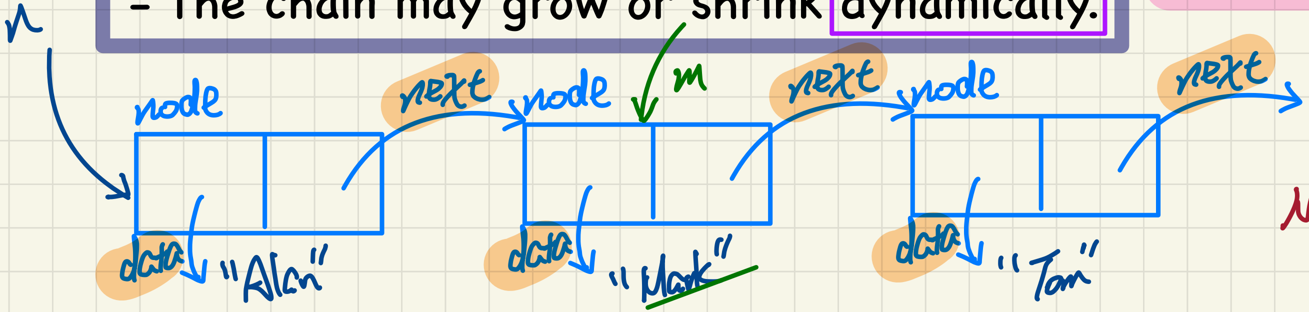
n: 1st node
n.data == "Alan"

n.next != null
n.next.data == "Mark"

n.next.next != null
n.next.next.data == "Tom"

n.next.next.next.data

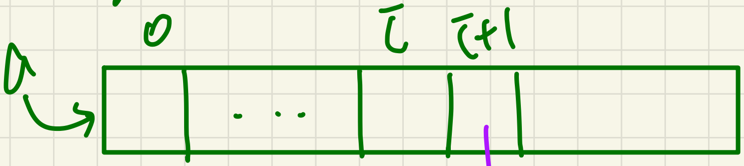
NullPointerException



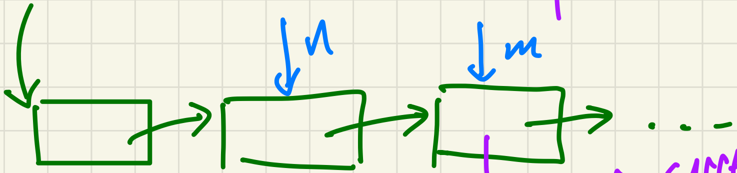
Linear vs. Non-linear Collections

each position in the collection has a unique successor.

each position has multiple successors

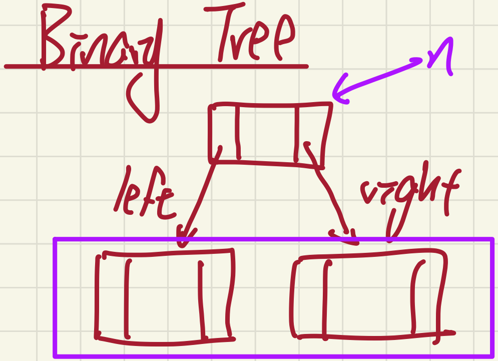


unique successor of a[i]



unique successor of n ($\because$ m == n.next)

Binary Tree



multiple successors of n

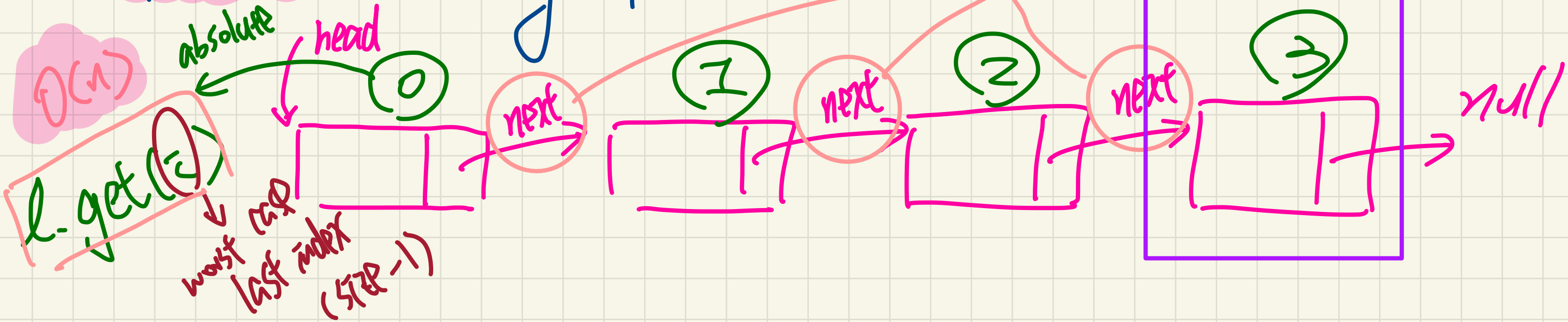
Absolute Indexing of Arrays

$a[i] \rightarrow O(1)$

Int: calculate the address offset from a_i beginning of array.

Relative Positioning of LL

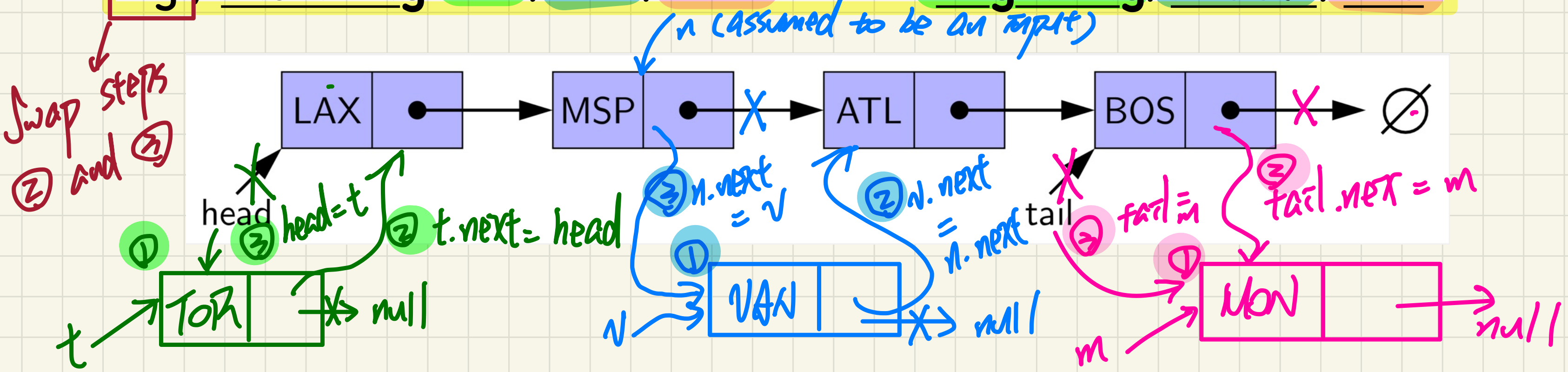
last node:
next == null



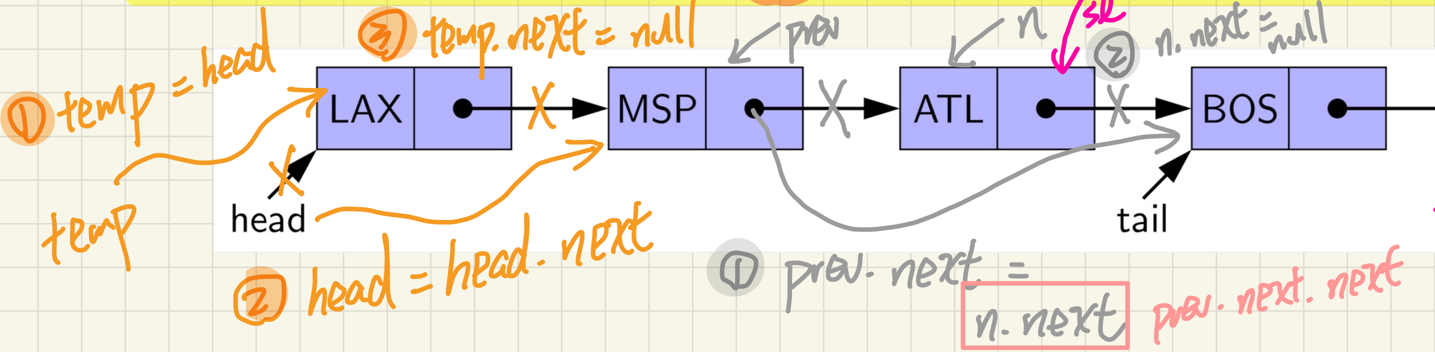
A SLL Grows or Shrinks Dynamically

e.g., Inserting TOR/VAN/MON to the beginning/middle/end.

swap steps ② and ③



e.g., Removing LAX/ATL/BOS from the beginning/middle/end.



Exercise

given sl

2nd last node

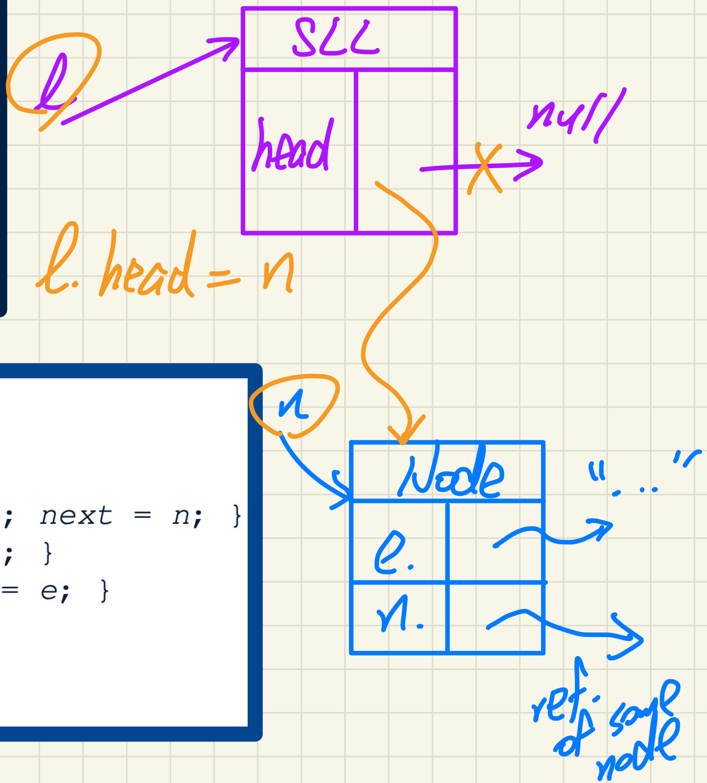
Implementing SLL in Java: SinglyLinkedList vs. Node

```java
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() { ... }
}
```

recursive type

```java
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; }
}
```

Runtime



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; }
}
```

tom

~~this.~~ mark

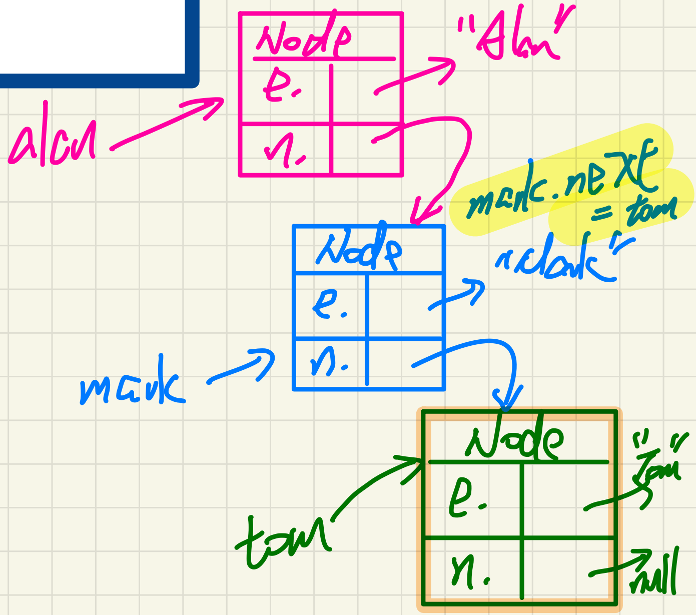
tom

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);
```



Node
e.
n.
"Alan"
alan
mark.next
= tom
Node
e.
n.
"Mark"
mark
Node
e.
n.
"Tom"
null
tom