# [TAP:XACKV] SLL

- What's the run time of getLast() in SLL?
  - A. O(1) if SLL keeps track of "tail"
  - B. O(log n)
  - [✓ C.] O(n) default SLL
  - D. O(n^2)
  - E. Whatever

# Administrative Details

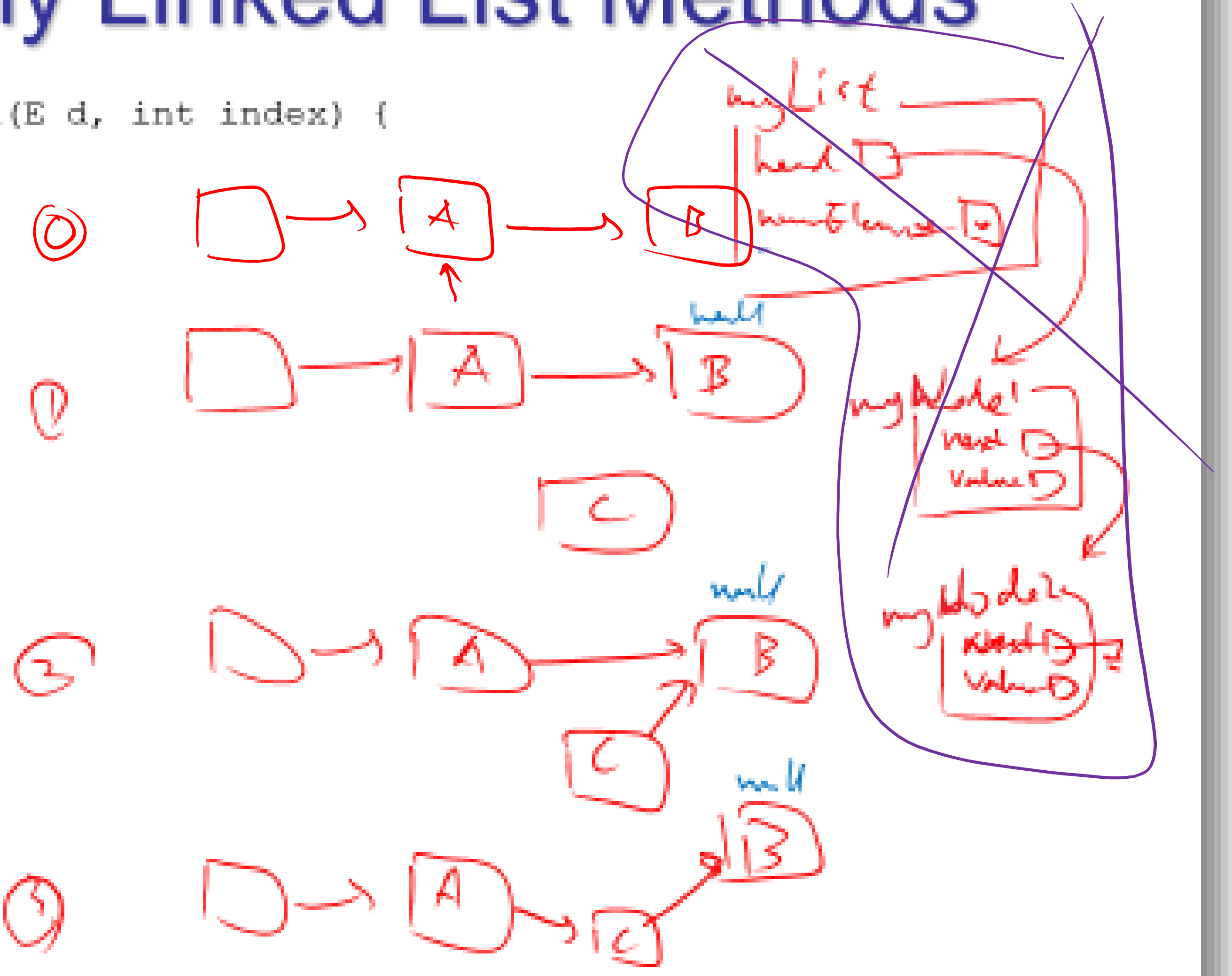
- Lab 4
  - A wrong version of LinkedList.java was posted on the course website.
    - If you downloaded the file, please delete it.
    - If you have it open on your browser, please refresh your browser.
    - (The honor code applies here.)
  - Your starter repo contains the correct version, so you don't have to do anything if you haven't checked out the file on the website

# Agenda

- Doubly Linked List (DLL)
- (Linear and Binary) Search

# Singly Linked List Methods

```
public void add(E d, int index) {
```

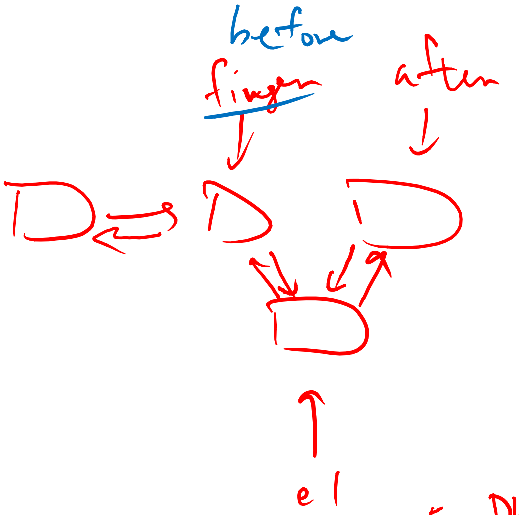


# Singly Linked List Methods

```
public void add(E d, int index) {

    if (index == 0)
        addFirst(d);
    else if (index == numElements)
        addLast(d);
    else {
        Node finger = head;

        for (int i=0; i<index-1; i++) {      ⓪
            finger = finger.next();
        }

        Node el = new Node(d, finger.next()); //①②  b
        finger.setNext(el); //③
        numElements++;
    }
}
```

# Doubly Linked List Methods

```
public void add(E d, int index) {
    if (index == 0)
        addFirst(d);
    else (index == numElements)
        addLast(d);
    else {
        DoublyLinkedNode ~~finger~~ before = head;
        for (int i = 0; i < index-1; i++)
            ~~finger~~ before = ~~finger~~ before.next();
        DoublyLinkedNode after = ~~finger~~ before.next();
        DoublyLinkedNode el = new DoublyLinkedNode(d, after, ~~finger~~ before);
        finger.setNext(el);
        after.setPrevious(el);
        numElements++;
```



```
        E   DLN    DNN
public DLN(d, after, before)
    data = d;
    nextElement = after;
    previousElement = before;
```

# Agenda

- Doubly Linked List (DLL)
- ⊙ (Linear and Binary) Search

# Searching in sorted list vs unsorted list

- Search in ***unsorted*** list

  ~~7~~, ~~3~~, ~~12~~, ~~6~~, ~~9~~, ~~1~~, ~~15~~

  - Linear Search: O( n ) , O(1) , O(n) $= O(\frac{n}{2})$

    worst best ave

- Search in ***sorted*** list

  ~~1, 3, 6~~, ~~7~~, ~~9~~, ~~12~~, ~~15~~

  - Linear Search: O( n ), O(1), O(n)
  - Binary Search: O( log n ) ← log n

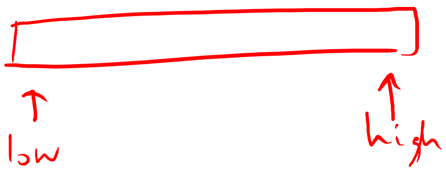
$\frac{n}{2^k} = 1$

$n = 2^k$

$\log n = k$

| # of elements checked | 0 | 1 | 2 | . . . | k | k+1 |
|---|---|---|---|---|---|---|
| # " to be checked | n | $\frac{n}{2}$ | $\frac{n}{2^2}$ | . . . | 1 | 0 |

# Binary Search



```
public static int bs(int a[], int v)
    return bsHelper(a, v, 0, a.length-1);

private static int bsHelper(int a[], int v, int low, int high)
    // base case
    if (low > high)
        return -1;

    int mid = (low+high)/2;

    if (a[mid] == v)
        return mid;
    // recursive case
    if (a[mid] < v)
        return bsHelper(a, v, mid+1, high);
    return bsHelper(a, v, low, mid-1);
```

# Binary Search

- Why does it work?
  - Because items can be ordered (they are *comparable*)
  - So they can be sorted then searched based on ordering
- Why is it fast?
  - Cut search space in half with each comparison!
- Challenges:
  - Requires items to be *comparable*
  - If items are not comparable, we typically need to do a *linear search*