Same Question, Different Choices

In LinkedList, both `addAt()` and `hasDuplicates()` can be dependent on the number of items in the list. If the list gets really big, will there be a noticeable difference in their running times?

A. Yes, `addAt()` will in the **worst case** be much slower than `hasDuplicates()`

B. Yes, `hasDuplicates()` will in the **worst case** be much slower than `addAt()`

C. No, in the **worst case** they will run in about the same time
Administrivia

- A1 scores are released. Regrades open till Wednesday
- Makeups will hopefully be graded by tomorrow?
- A3 is due at the end of this week
- We’ll try to get A2 graded soon (ETA hopefully Thursday)
- Test2 is next week
- We’re almost at the halfway point!!!
Test1 debrief

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Median</th>
<th>Maximum</th>
<th>Mean</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.5</td>
<td>40.5</td>
<td>45.5</td>
<td>39.4</td>
<td>4.67</td>
</tr>
</tbody>
</table>
Comparison

Let’s run some experiments!
Efficiency

Dependent vs independent of size isn’t the whole story
Another list data structure

Singly:

Doubly:

What is an example of an operation that would be more efficient with doubly-linked lists?
Ordered Collections

List

Stack

Queue
Stack and Queue Operations

• Add item

• Remove and return next item

• Return next item without removing
(one possible design)

<table>
<thead>
<tr>
<th>Stack&lt;&lt;T&gt;</th>
<th>Operation</th>
<th>Stack</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(newItem: \text{T}): \text{void}</td>
<td>Add item</td>
<td>push</td>
<td>enqueue</td>
</tr>
<tr>
<td>pop(): \text{T}</td>
<td>Remove and return next item</td>
<td>pop</td>
<td>dequeue</td>
</tr>
<tr>
<td>peek(): \text{T}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isEmpty(): \text{boolean}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue&lt;&lt;T&gt;</th>
</tr>
</thead>
</table>

| enqueue(newItem: \text{T}): \text{void} | Return next item without removing | peek | peek |
| dequeue(): \text{T} | | | |
| peek(): \text{T} | | | |
| isEmpty(): \text{boolean} | | | |
Stack and Queue Implementations
As Linked List

Stack
Queue

push and pop from head
enqueue at tail
Code Demo: LinkedStack and LinkedQueue
Array Bag (fixed capacity)

- add: independent of size
- frequencyOf: dependent on size

```java
void add(T item) {
    // omitted: if no room, throw exception
    items[size] = item;
    size++;
}

int frequencyOf(T item) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (items[i].equals(item))
            count++;
    }
    return count;
}
```
Array Bag (resizable)

- add: worst case, dependent on size; best case, independent
- frequencyOf: always dependent on size

```java
void add(T item) {
    // omitted: if no room, copy into bigger array
    items[size] = item;
    size++;
}
```

```java
int frequencyOf(T item) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (items[i].equals(item))
            count++;
    }
    return count;
}
```
Linked Bag

- `add`: always independent of size
- `frequency0f`: always dependent on size

Efficiency + unbounded capacity was the great win of linked bags over array bags
Linked List

- **addAt**: best case independent of size; worst case dependent
- **hasDuplicates**: not only dependent, but runs considerably slower than addAt as size increases [experiment]
How is stair climbing like `addAt` and `hasDuplicates`?
Linked List

addAt, worst case, size $n$

hasDuplicates, worst case, size $n$

Keep this in mind, we’ll return to this when we do searching and sorting
 Complexity Notes for Dummies

- A mathematical notation of how your code scales given a particular input.

- Example: $O(n)$

- $O(n^3) > O(n^2) > O(n) > O(1)$

- Question is $n^2 + n + 2 < O(n^3)$?

```java
for (int i = 0; i < N; i++) {
    //do something in constant time...
}
```
Complexity Notes for Dummies

- Question is $n^2 + n + 2 < O(n^3)$?
- Yes!!

- Note: We only care about how it scales as the input goes to infinity. It gives us a strict upper bound.
  - As long as there some constant $k$, such that for all $n \geq k$ and a $c$, $n^2 + n + 2 < cn^3$
  - Then $n^2 + n + 2 < O(n^3)$
  - What’s the lowest possible $k$ here, for $c = 3$?

Practice on this on Wednesday