Array List Implementation

Lists: array-based implementations, operations and shift algorithms, computational complexity, testing

What is a list?

- A list is a sequence of elements.

  -or-

  A list is either empty of consists of a single element followed by another list.
- For each definition, the position of an item in a list matters.
  - The list 7, 8, 4 is not the same as the list 8, 4, 7.
  - If you remove an item from a list, the remaining items should be in the same order as before.
  - If you add an item to a list, the other list items should retain the same ordering as before.

Creating your own ArrayList

- Sometimes ArrayList from the Java Collections Framework does not support all of the operations you need. Or the code is generic enough that it doesn't provide efficiency for the operations that your application demands the most.
- You need to understand what is going on under the covers.
  - Which operations are "expensive" – take lots of time?
  - Which operations are "cheap" – run efficiently/quickly?

Background code for examples

```
public class Student {
   private String name;
   private double gpa;

   public Student(String name, double gpa) {
      this.name = name;
      this.gpa = gpa;
   }
   public String getName() { return name; }
   public double getGpa() { return gpa; }
   public void setGpa(double gpa) {
       this.gpa = gpa;
   }
}
```

Creating a list without the Java Collections Framework

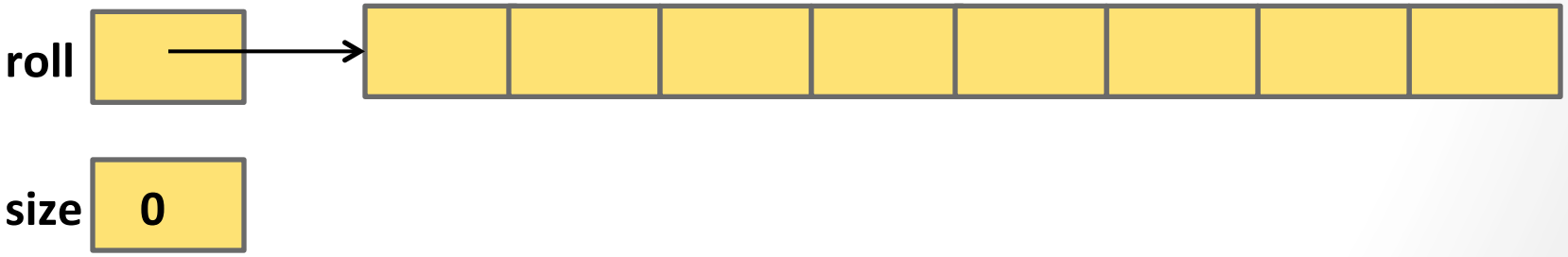
Steps to create an array-based data structure:

1. Decide what type of list elements are needed: **Student**
2. Decide the capacity.
   private int capacity = 100; // Make final if list cannot grow!
3. Create an array variable:
   private Student[ ] roll;
4. Create a simple counter to keep track of the number of actual list elements.
   private int size;
5. Initialize the array as a buffer to hold the list.
   roll = new Student[capacity];

Array- based data structure

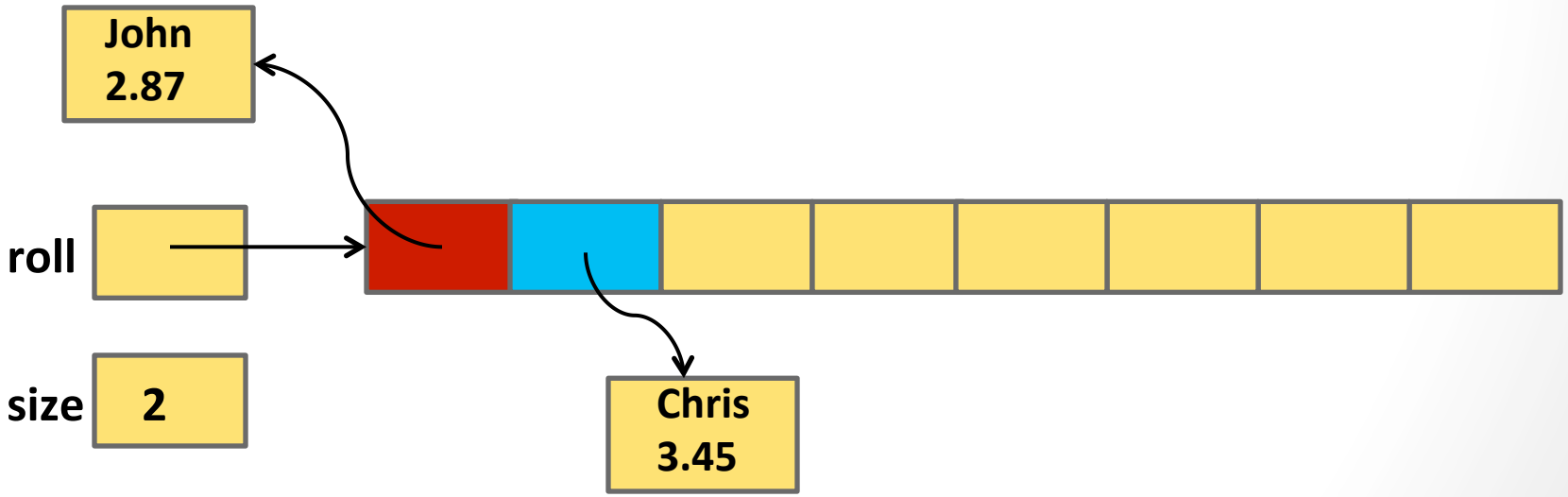
```
public class StudentList {
   private int capacity = 100;
   private Student[] roll = new Student[capacity];
   private int size = 0;
}
```

size: 1 + the index of the last element in the list (0 if the list is empty). Indexes of list elements range from 0 through size – 1. The rest of the array is just a buffer for new elements.



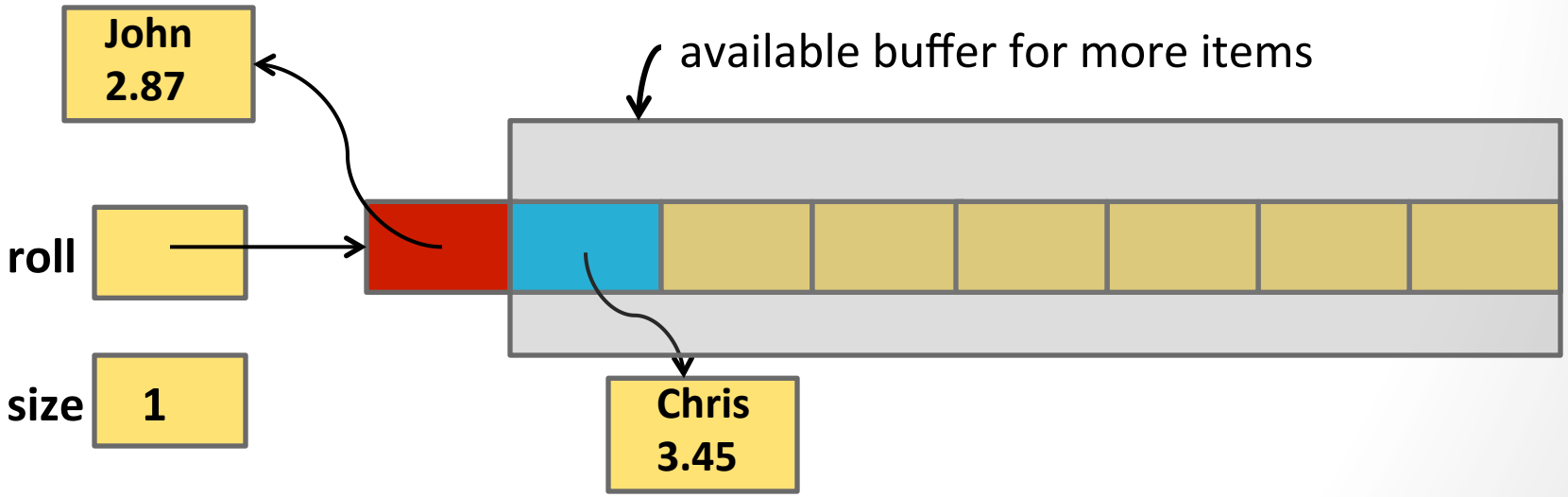
Adding elements to the end

```
public void addToEnd(Student s) {
    if (size < capacity) {
        roll[size] = s;
        size += 1;
    }
}
…
    addToEnd(new Student ("John", 2.87);
    addToEnd(new Student ("Chris", 3.45);
```



Removing the end element

```
public Student removeFromEnd( ) {
    if (size > 0) {
        Student s = roll[size - 1];
        size = size - 1;
        return s;
    }
    return null;
}
```



Ensuring capacity

ArrayLists in the Java collections framework automatically grow as needed whenever their capacities aren't sufficient. We can do the same:

1. Define a private method: growAsNeeded().
2. Change add methods: replace the if statement at the front by a call to growAsNeeded().

```
private void growAsNeeded() {
   if (size == capacity) {
      capacity = capacity * 2;
      Student[] buffer = new Student[capacity * 2];
      for (int k = 0; k < size; k++) {
         buffer[k] = roll[k];
      }
         roll = buffer;
   }
}
```

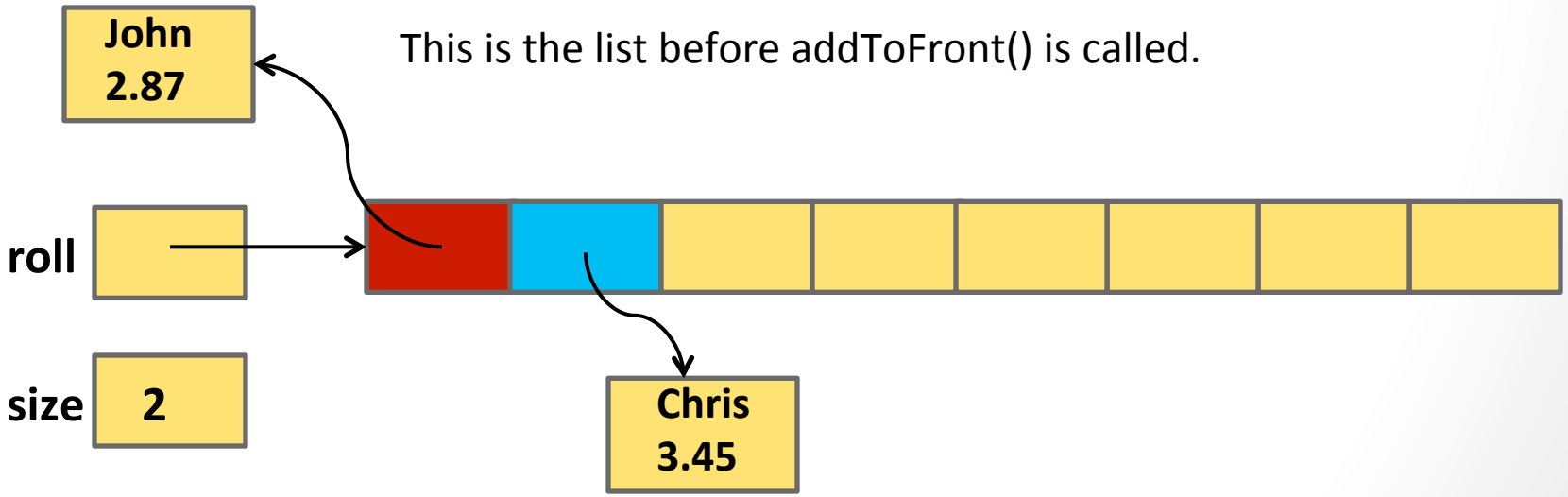
Add-to and remove-from end operation efficiencies

- Except when the list capacity is insufficient, adding a new element to the end of an array-based list with lots of elements takes no more time than adding to the end of a list with few elements.
- The time required to remove an item from the end of an array-based list is independent of the size of the list.

The “computational complexity” of the add to end (of a non-full list) and remove from end operations is constant: O(1).

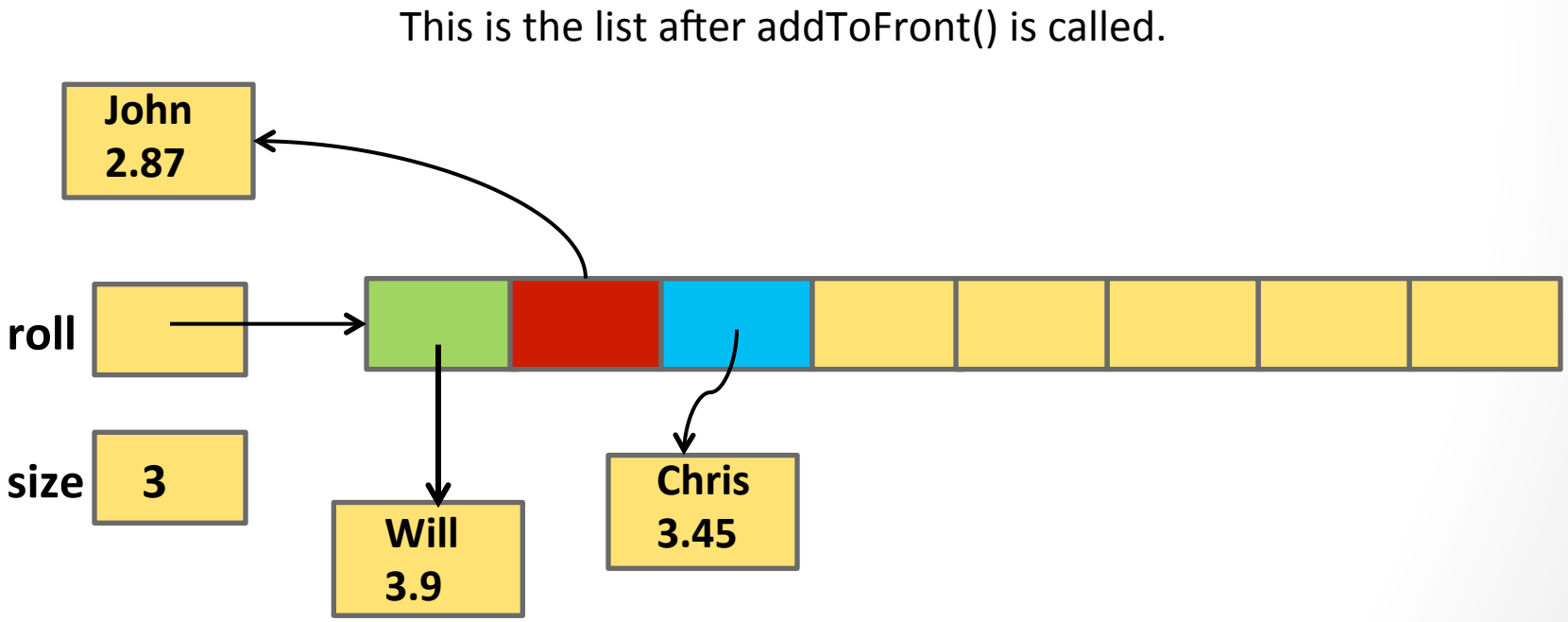
Adding an element to the front

```
public void addToFront(Student s) {
    growAsNeeded(); // Alternatively, check for size
    for (int k = size; k > 0; k--) {
        roll[k] = roll[k - 1];
    }
    size += 1;
    roll[0] = s;
}
```



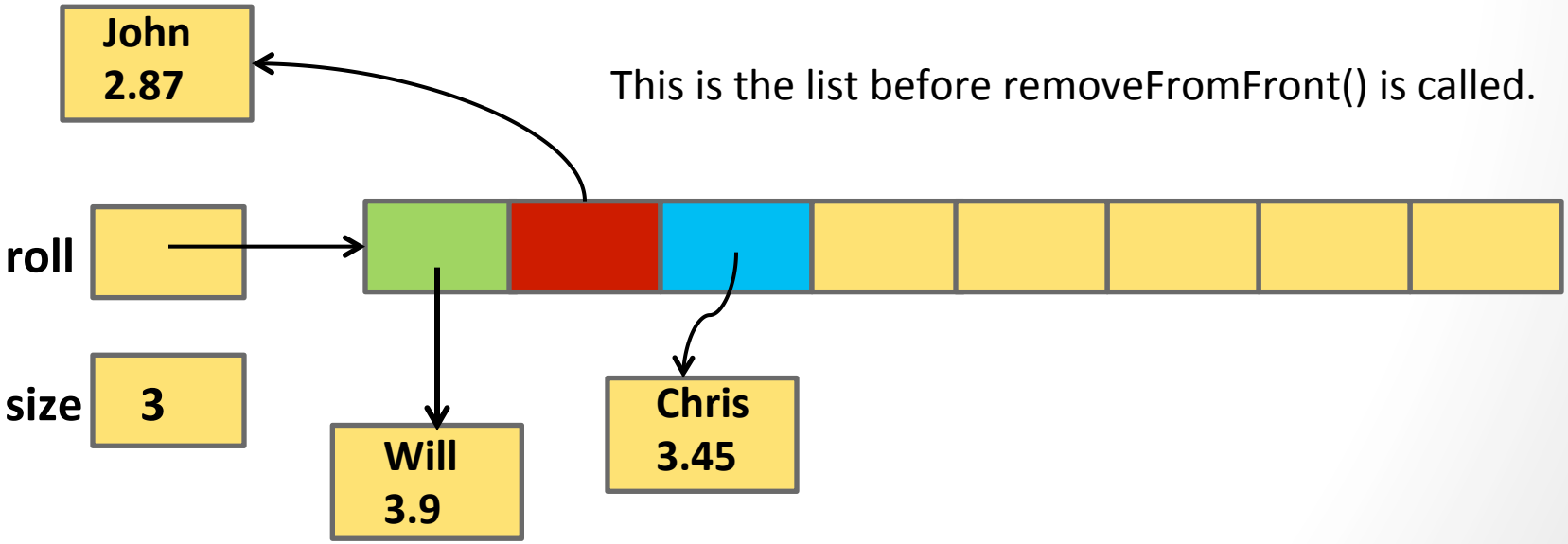
Shift right on front insertion

```
addToFront(new Student("Will",3.90));
```



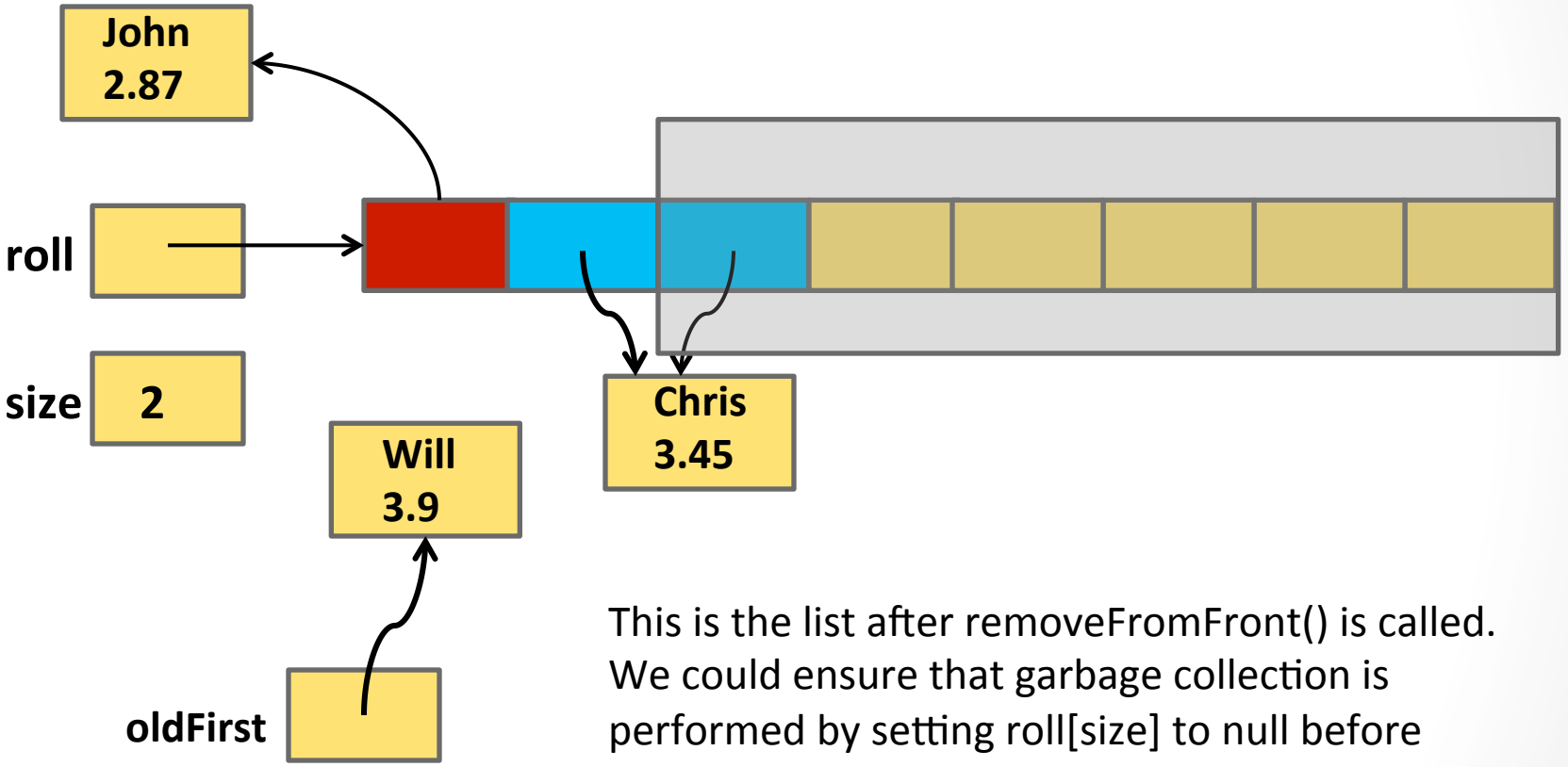
Removing the front element

```
public Student removeFromFront(){
    if (size > 0) {
        Student s = roll[0];
        for (int k = 0; k < size - 1; k++)
                roll[k] = roll[k + 1];
        size--;
        return s;
    }
    return null;
}
```



Shift left on front removal

```
Student oldFirst = this.removeFromFront();
```



This is the list after removeFromFront() is called. We could ensure that garbage collection is performed by setting roll[size] to null before returning.

Add-to and remove-from front/ middle operation efficiencies

- The amount of time require to add a new element to the front of an array-based list with lots of elements depends on the size of the list. Double the list size → double the time.
- The time required to remove an item from the front of an array-based list depends on the size of the list. Double the list size → double the time.

The "computational complexity" of the add to front and remove from front operations is linear in the number of list elements: O(n).

The "computational complexity" of the general add and remove operations (front, middle, or end) is also linear in the number of list elements: O(n). These are measured as "worst case" scenarios.

Testing list code

List operations should be coded correctly for each of the following 4 situations:

1. The operation changes/accesses the list at the front.
2. The operation changes/accesses the list at the end.
3. The operation changes/accesses the list in the middle.
4. The list is empty.

After each operation, make sure that:

1. The list has the correct number of elements.
2. The elements are correct and in the correct order.

Test combinations of operations such as multiple adds/removes, an add followed by a remove, etc.

Summary

To create an array-based list:

- Create the array.
- Create a size to count the list elements.
- Keep all list items contiguous – no holes in the array.
- Do not destroy the relative ordering of array elements on insertion or removal.

To test array-based list code (or any list code)

- Test at the front.
- Test in the middle.
- Test at the end.
- Test on an empty list.
- Test results should look at the size of the list and the list elements.