AUBURN

UNIVERSITY

SAMUEL GINN
COLLEGE OF ENGINEERING

Lists

List collections

A **list** is a collection that keeps its elements in some particular **order**.

Many different types of order…

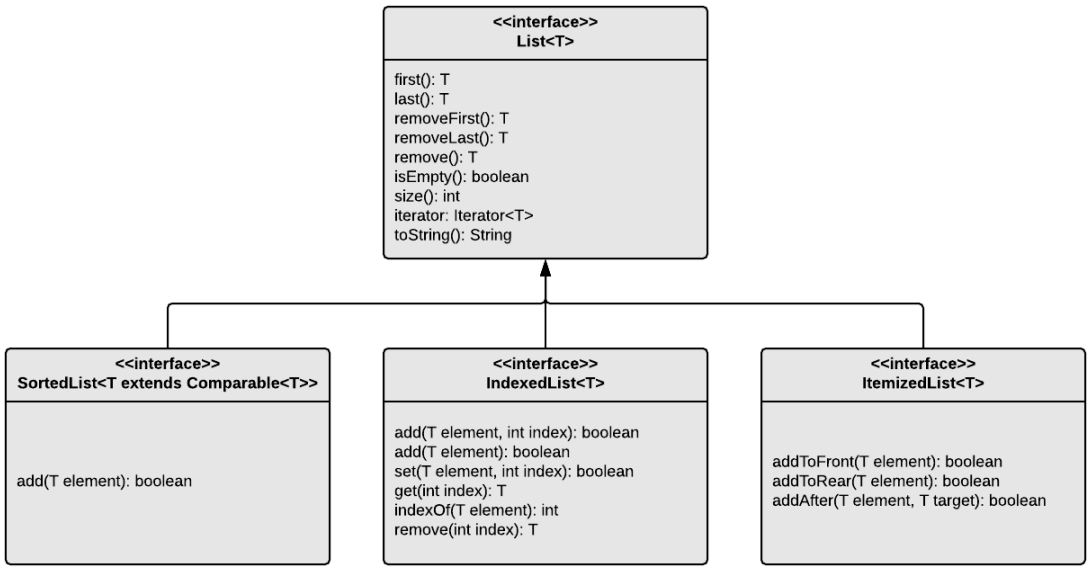
- Total order on element value (sorted lists)
- Absolute positional order (indexed lists)
- Relative positional order (bullet lists)
- Time-based order (first-in, first-out)
- Priority order

… so there could be many different list interfaces defined.

A List interface hierarchy

Methods common to all list types.

The add method, plus methods specific to certain list types.



ant
bird
cat
dog

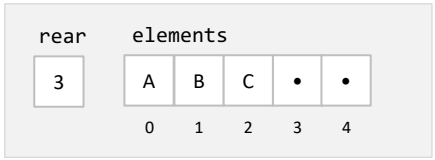
1. cat
2. dog
3. bird
4. ant

- bird
- dog
- ant
- cat

Implementing IndexedList

Array-Based

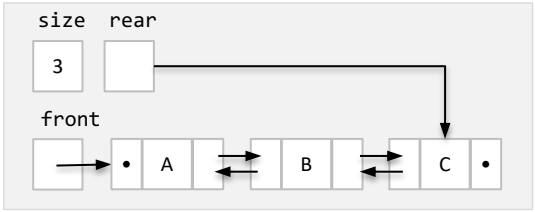
- Keep elements left-justified (anchored at 0, no gaps)
- Keep a size counter (can serve as a rear marker)




```
public class ArrayIndexedList<T> implements IndexedList<T> {

       private T[] elements;
       private int rear;
```

Node-Based

- Doubly-linked
- Keep both a front and rear pointer
- Keep a size counter



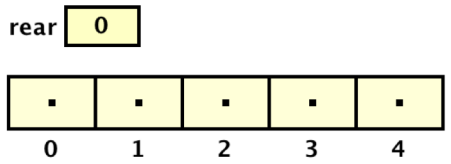

```
public class LinkedIndexedList<T> implements IndexedList<T> {

       private Node front;
       private Node rear;
       private int size;
```

Array-based implementation

ArrayIndexedList – add(element)

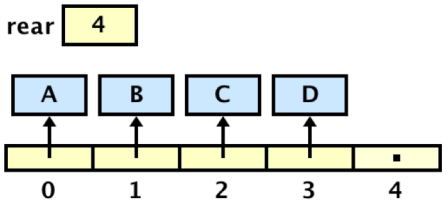
```
public class ArrayIndexedList<T>
                  implements IndexedList<T> {
    private T[] elements;
    private int rear;

    public boolean add(T element) {









    }
```

```
ArrayIndexedList list = new ArrayIndexedList(5);
```




```
list.add("A");
list.add("B");
list.add("C");
list.add("D");
```

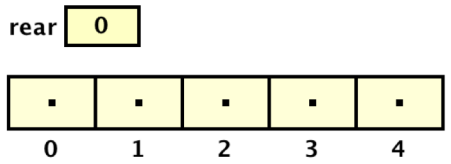
We can use the exact same implementation as the add method of the ArrayBag.



ArrayIndexedList – add(element)

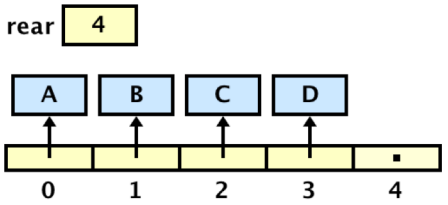
```
public class ArrayIndexedList<T>
                implements IndexedList<T> {
    private T[] elements;
    private int rear;

    public boolean add(T element) {
        if (isFull()) {
            resize(elements.length * 2);
        }
        elements[rear] = element;
        rear++;
        return true;
    }
```

```
ArrayIndexedList list = new ArrayIndexedList(5);
```




```
list.add("A");
list.add("B");
list.add("C");
list.add("D");
```

We can use the exact same implementation as the add method of the ArrayBag.



ArrayIndexedList – add(element)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {
    private T[] elements;
    private int rear;

    public boolean add(T element) {
        if (isFull()) {
            resize(elements.length * 2);
        }
        elements[rear] = element;
        rear++;
        return true;
    }
```

Time Complexity: **O(1)** *amortized*

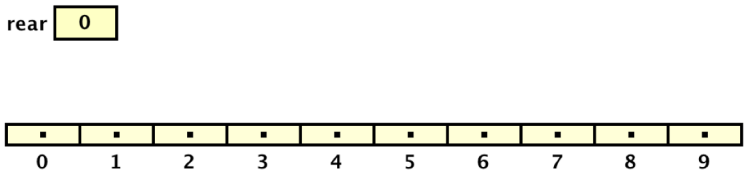
O(N) – but we can amortize this

O(1)

ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {









    }
```

```
ArrayIndexedList list = new ArrayIndexedList(10);
```

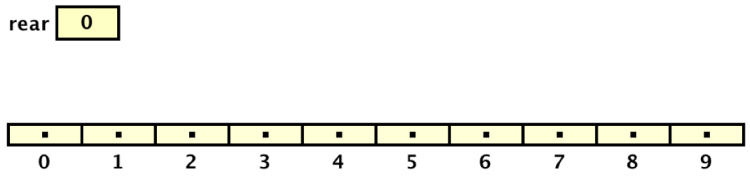



```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```

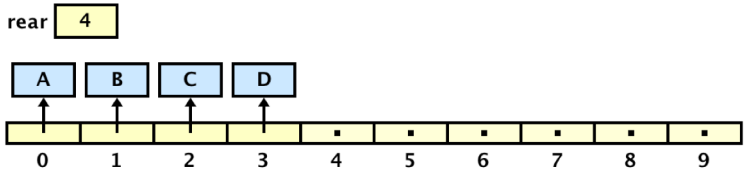
ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }









    }
```

```
ArrayIndexedList list = new ArrayIndexedList(10);
```



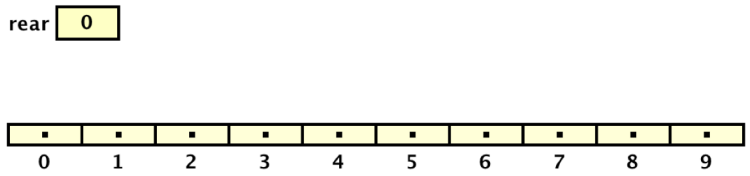

```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```



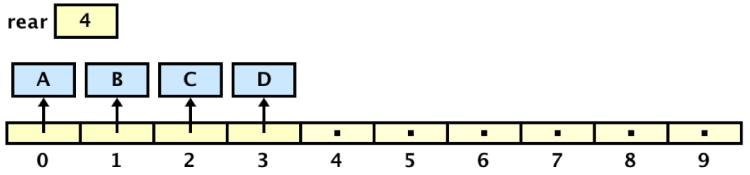
ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }









    }
```

```
ArrayIndexedList list = new ArrayIndexedList(10);
```




```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```

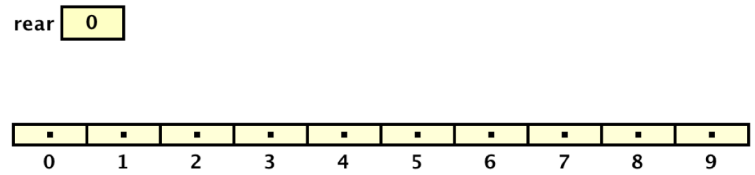



```
list.add("E", 5);
```

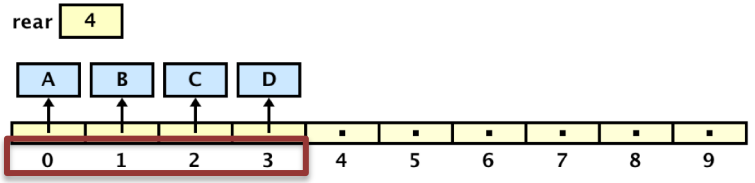
ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }









    }
```

```
ArrayIndexedList list = new ArrayIndexedList(10);
```




```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```

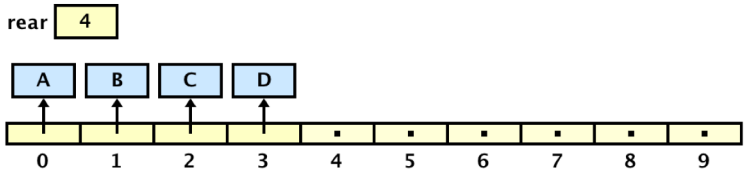



```
list.add("E", 5);
```

must be >= 0 and < rear

ArrayIndexedList – add(element, index)

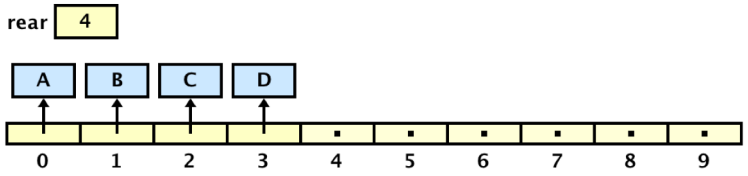
```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }









    }
```




```
list.add("E", 1);
```

ArrayIndexedList – add(element, index)

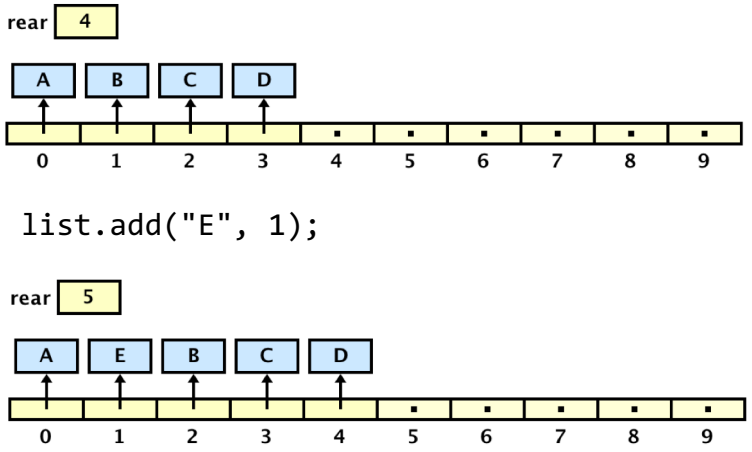
```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }




    }
```




```
list.add("E", 1);
```

ArrayIndexedList – add(element, index)

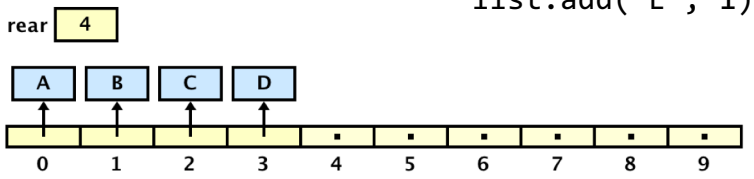
```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }




    }
```



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }




    }
```

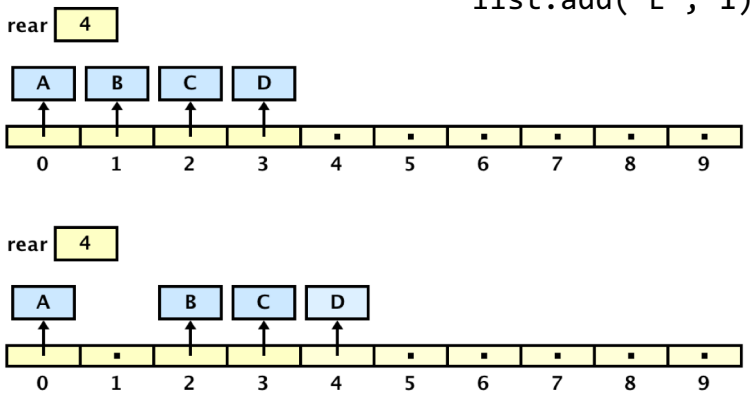
```
list.add("E", 1);
```



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }




    }
```

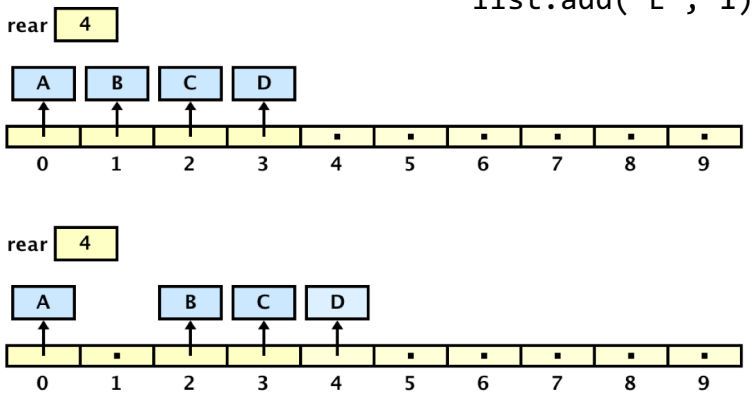
list.add("E", 1);



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);



    }
```

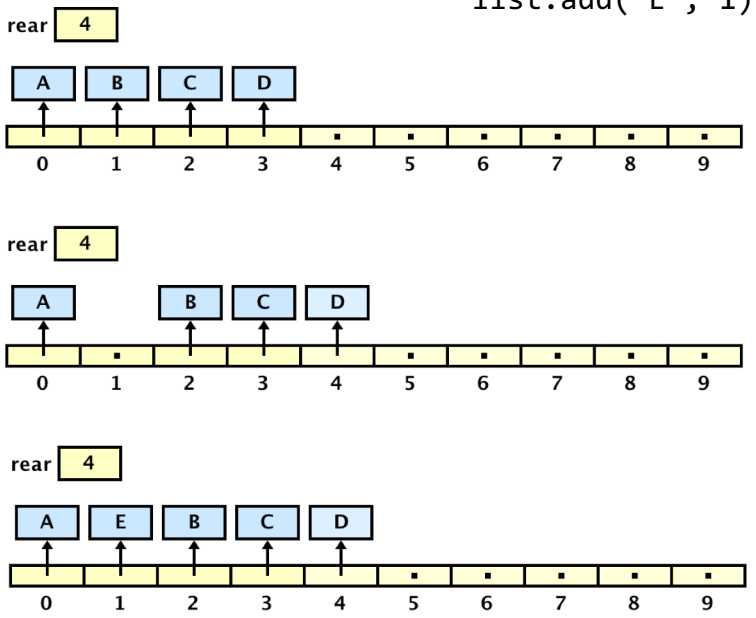
list.add("E", 1);



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;


    }
```

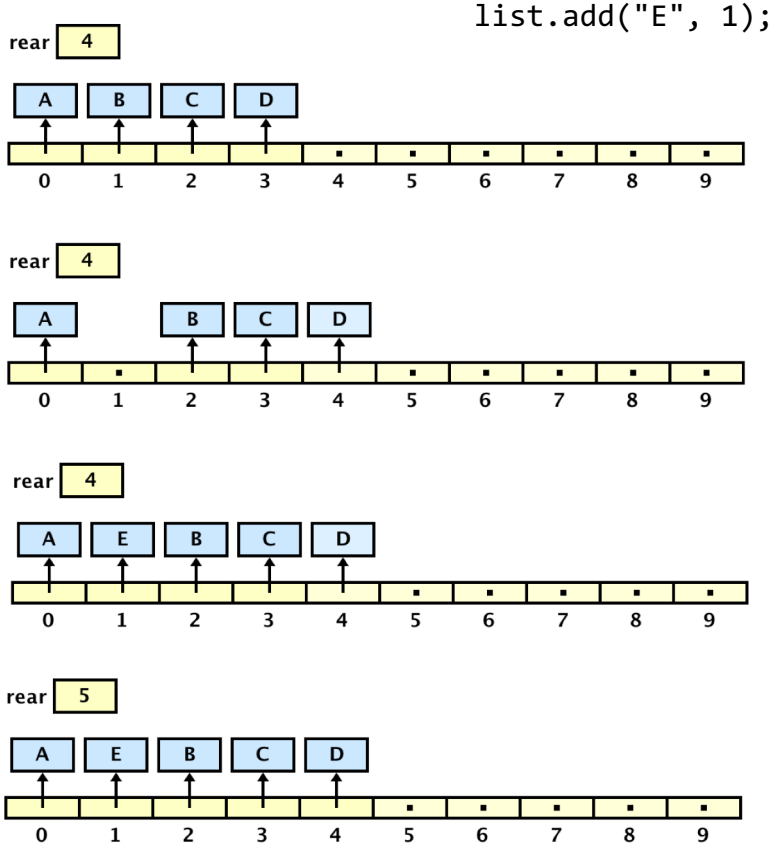
list.add("E", 1);



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;
        rear++;

    }
```

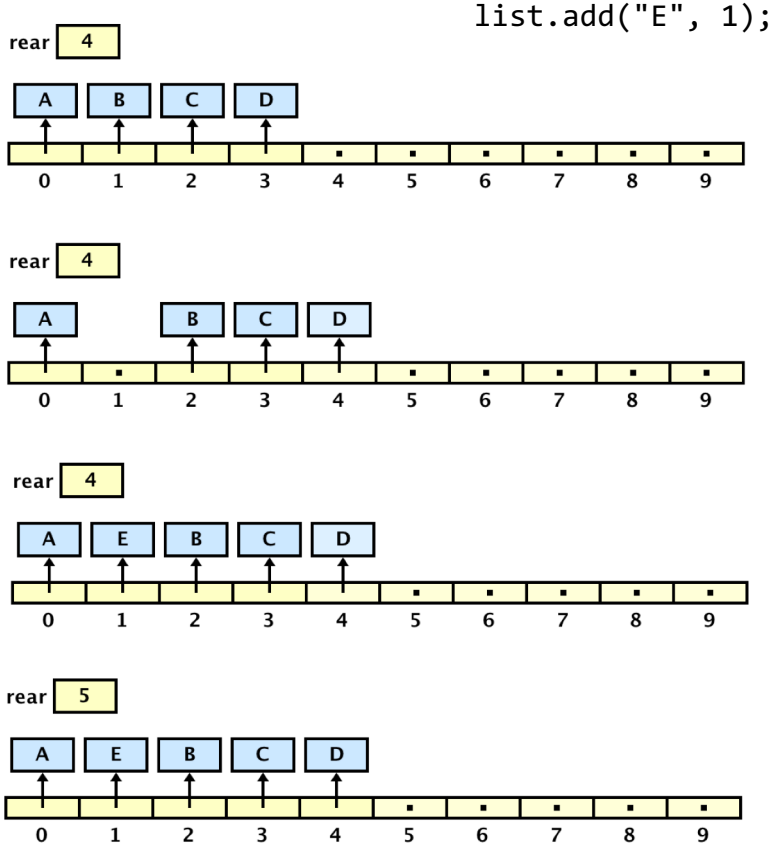
list.add("E", 1);



ArrayIndexedList – add(element, index)

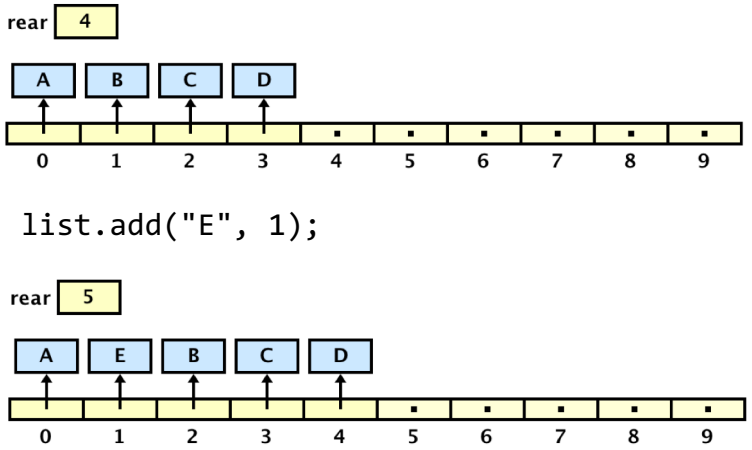
```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;
        rear++;
        return true;
    }
```

list.add("E", 1);



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;
        rear++;
        return true;
    }
```



ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;
        rear++;
        return true;
    }
```

Time Complexity:

- O(1)
- O(1)
- O(N) – but we can amortize this
- O(1)

ArrayIndexedList – add(element, index)

```
public class ArrayIndexedList<T>
                implements IndexedList<T> {

    public boolean add(T element, int index) {
        if (index == rear) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        if (isFull()) {
            resize(elements.length * 2);
        }
        shiftRight(index);
        elements[index] = element;
        rear++;
        return true;
    }
```

Time Complexity: **O(N)**

O(1)

O(1)

O(N) – but we can amortize this

O(N)

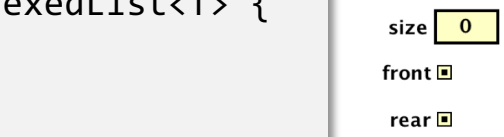
O(1)

Node-based implementation

LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {









    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```

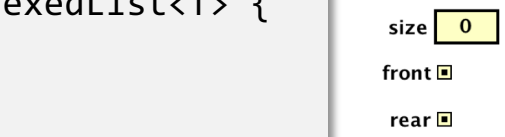



```
list.add("A");
```

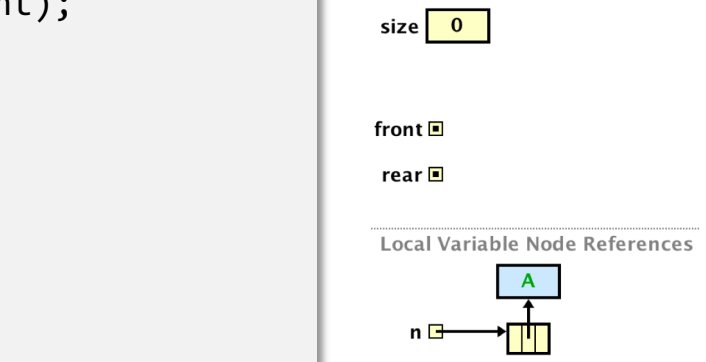
LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                  implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);









    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```



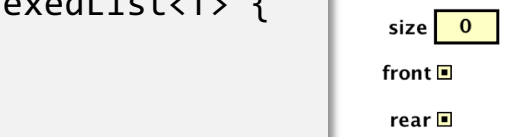

```
list.add("A");
```



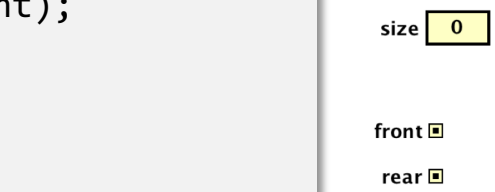
LinkedIndexedList – add(element)

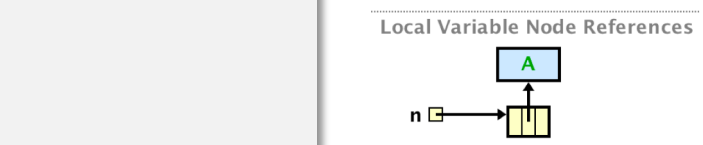
```
public class LinkedIndexedList<T>
                  implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {


        } else {




        }


    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```




```
list.add("A");
```

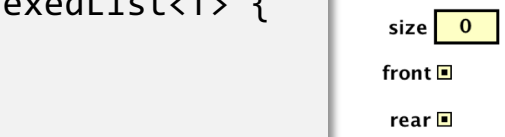




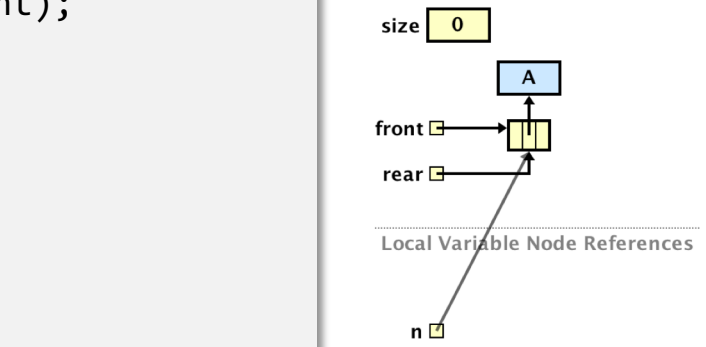
LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {



        }


    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```



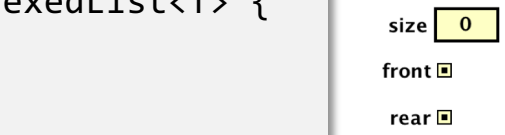

```
list.add("A");
```



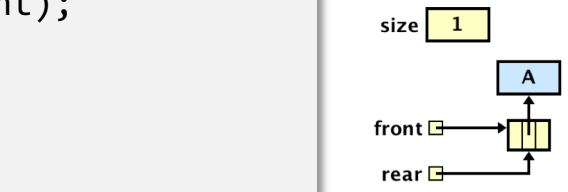
LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {



        }
        size++;
        return true;
    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```

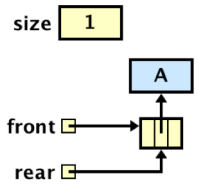



```
list.add("A");
```

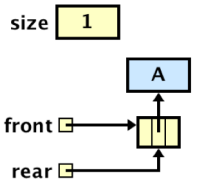


LinkedIndexedList – add(element)

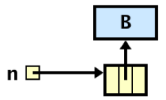
```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {



        }
        size++;
        return true;
    }
```




```
list.add("B");
```

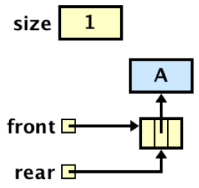


Local Variable Node References

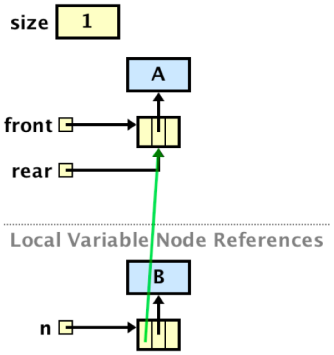


LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {
            n.prev = rear;



        }
        size++;
        return true;
    }
```

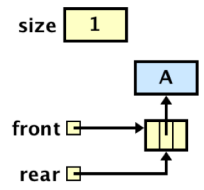



```
list.add("B");
```

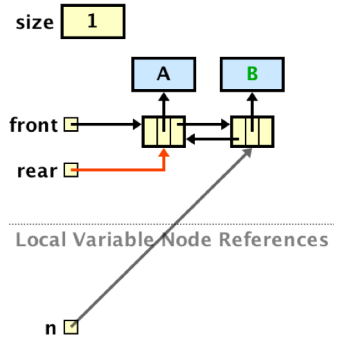


LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {
            n.prev = rear;
            rear.next = n;

        }
        size++;
        return true;
    }
```

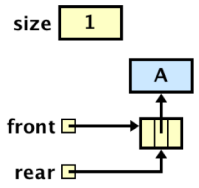



```
list.add("B");
```

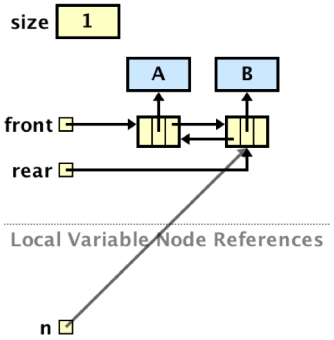


LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {
            n.prev = rear;
            rear.next = n;
            rear = n;
        }
        size++;
        return true;
    }
```

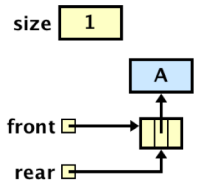



```
list.add("B");
```

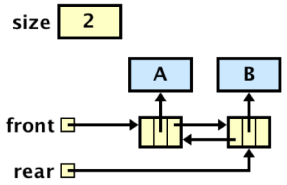


LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {
            n.prev = rear;
            rear.next = n;
            rear = n;
        }
        size++;
        return true;
    }
```




```
list.add("B");
```



LinkedIndexedList – add(element)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    private Node front, rear;
    private int size;

    public boolean add(T element) {
        Node n = new Node(element);
        if (isEmpty()) {
            front = n;
            rear = n;
        } else {
            n.prev = rear;
            rear.next = n;
            rear = n;
        }
        size++;
        return true;
    }
```

Time Complexity: **O(1)**

O(1)

O(1)

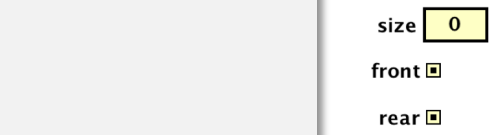
O(1)

O(1)

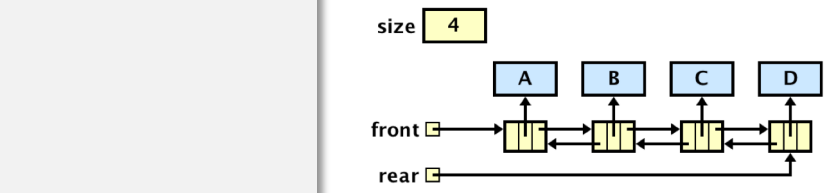
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                  implements IndexedList<T> {
    public boolean add(T element, int index) {









    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```



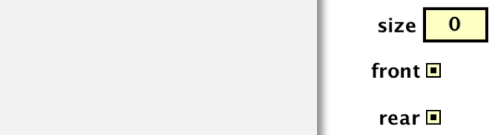

```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```



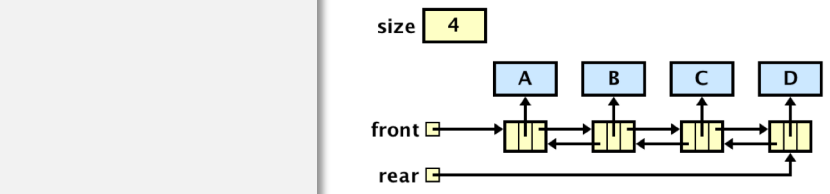
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }









    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```




```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```

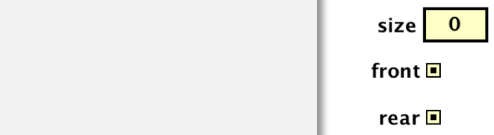



```
list.add("E", 5);
```

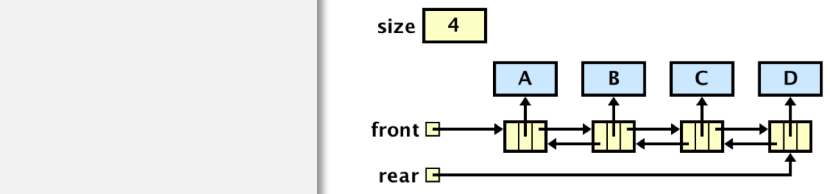
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }









    }
```

```
LinkedIndexedList list = new LinkedIndexedList();
```




```
list.add("A", 0);
list.add("B", 1);
list.add("C", 2);
list.add("D", 3);
```

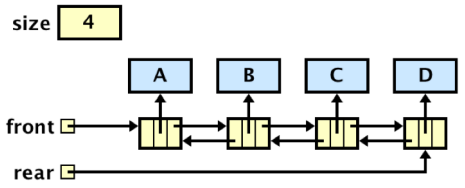



```
list.add("E", 5);
```

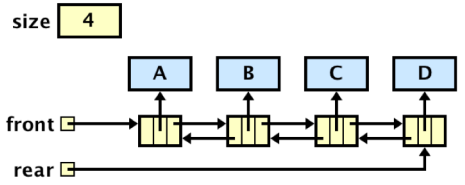
must be >= 0 and < size

LinkedIndexedList – add(element, index)

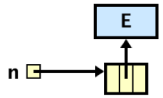
```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);









    }
```




```
list.add("E", 0);
```

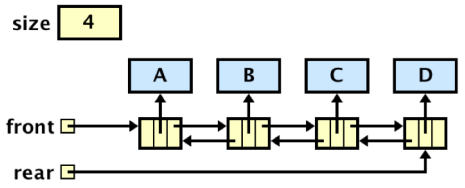


Local Variable Node References

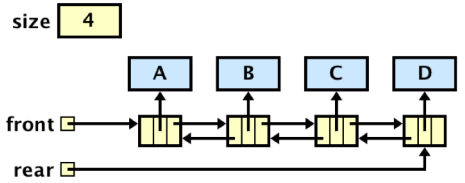


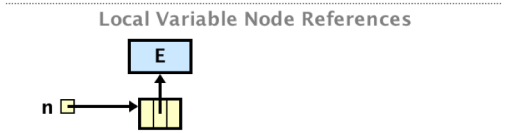
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {



        } else {









        }


    }
```



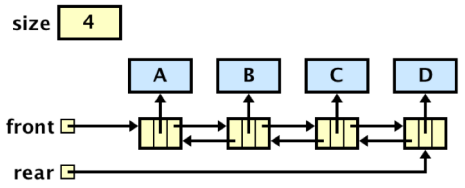

```
list.add("E", 0);
```



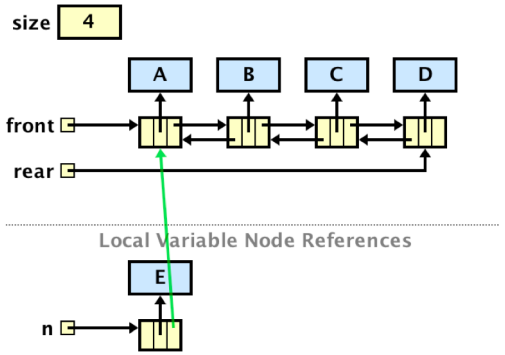


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;


        } else {









        }


    }
```

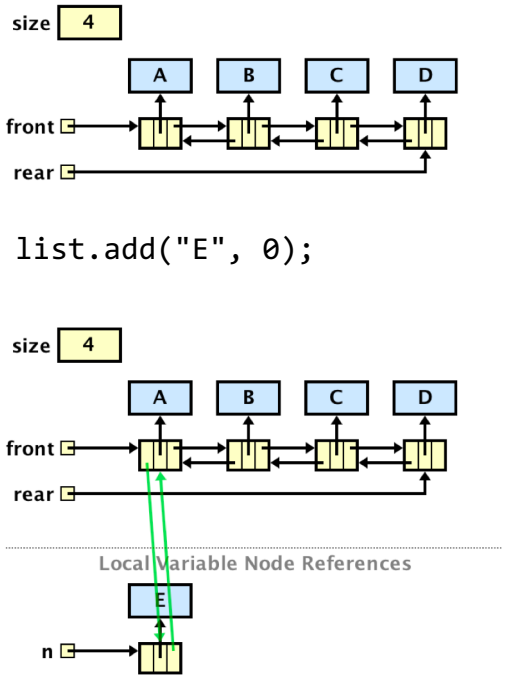



```
list.add("E", 0);
```



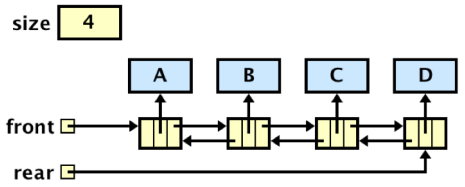
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;

        } else {









        }


    }
```

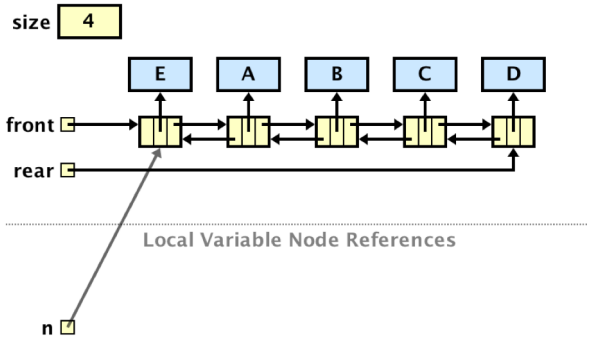


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {









        }
        size++;
        return true;
    }
```

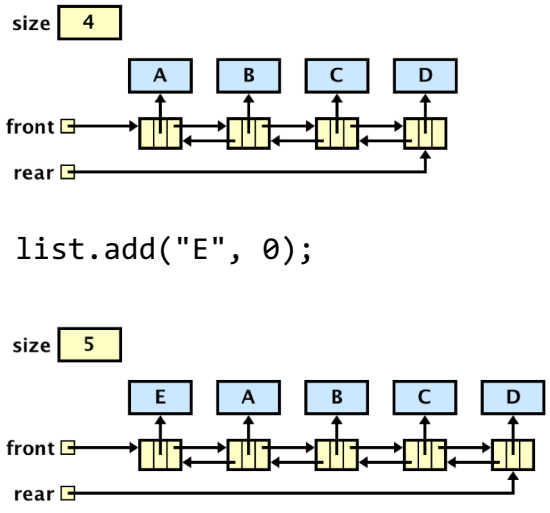



```
list.add("E", 0);
```



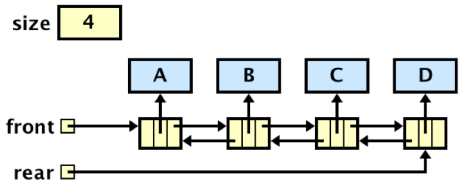
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {









        }
        size++;
        return true;
    }
```

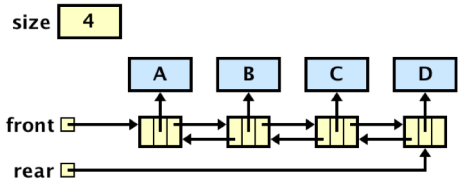


LinkedIndexedList – add(element, index)

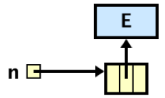
```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {









        }
        size++;
        return true;
    }
```




```
list.add("E", 2);
```

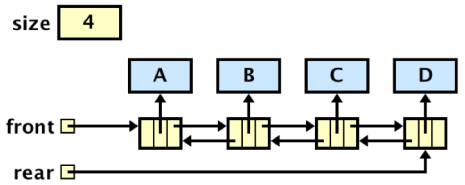


Local Variable Node References

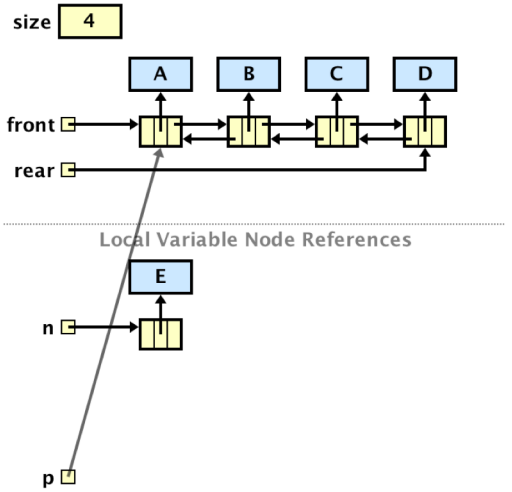


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;









        }
        size++;
        return true;
    }
```

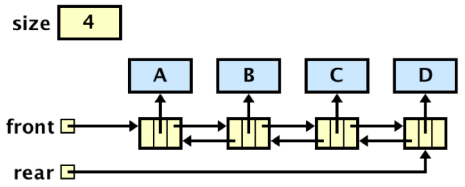



```
list.add("E", 2);
```

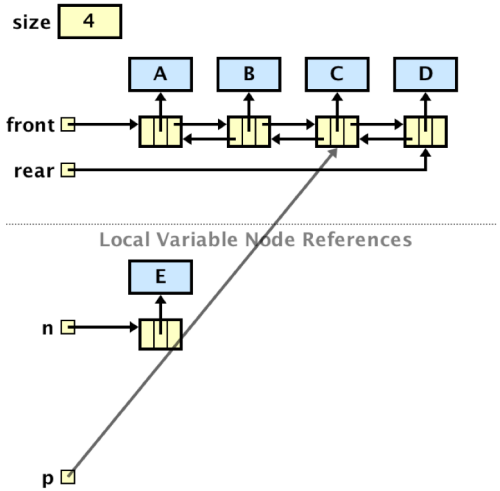


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
        }
        size++;
        return true;
    }
```

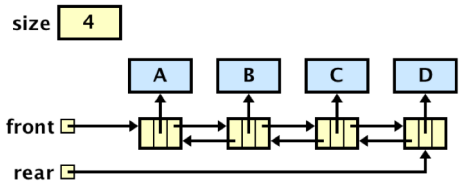



```
list.add("E", 2);
```

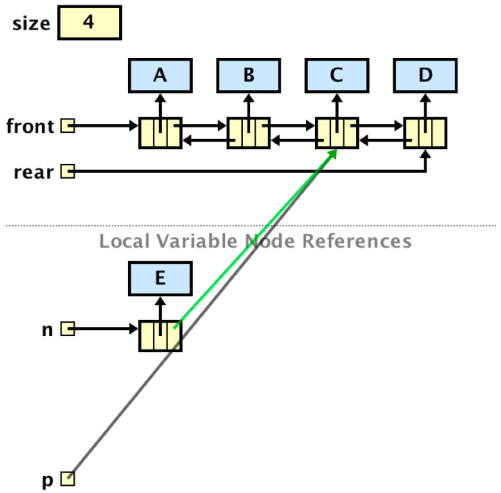


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;



        }
        size++;
        return true;
    }
```

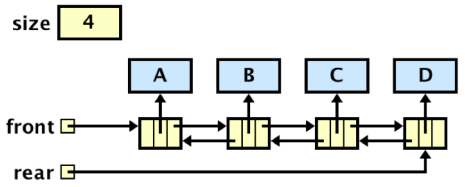



```
list.add("E", 2);
```

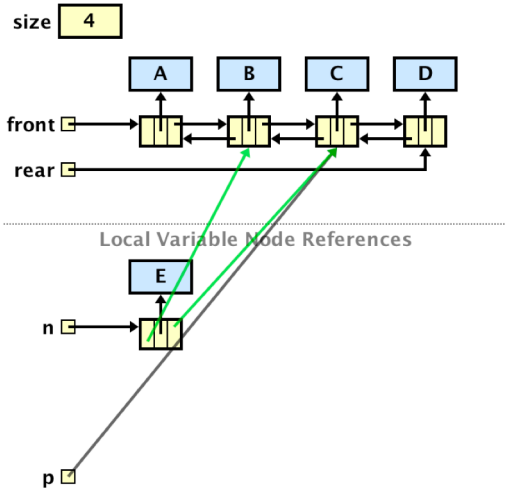


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;


        }
        size++;
        return true;
    }
```

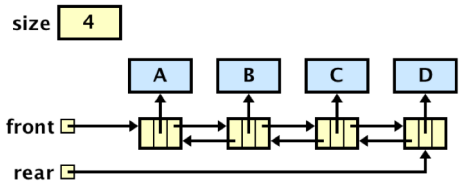



```
list.add("E", 2);
```

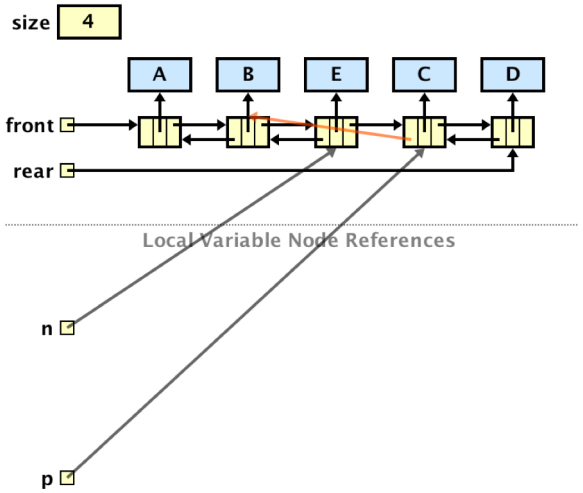


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;
            p.prev.next = n;

        }
        size++;
        return true;
    }
```

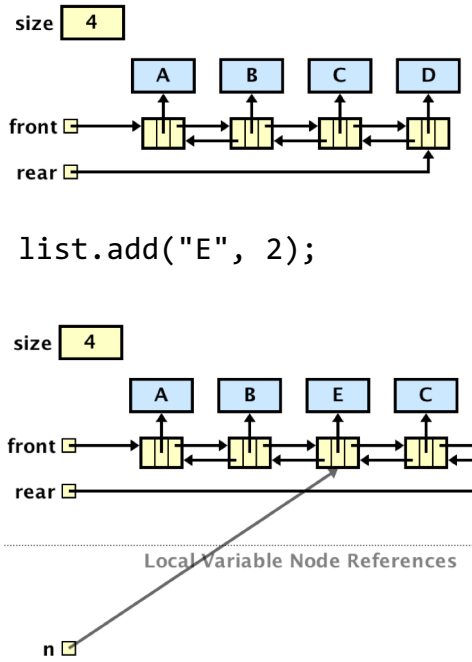



```
list.add("E", 2);
```



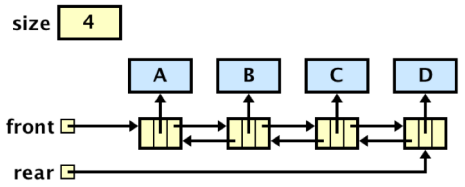
LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;
            p.prev.next = n;
            p.prev = n;
        }
        size++;
        return true;
    }
```

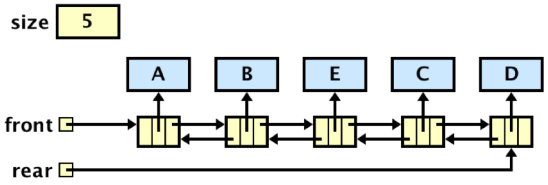


LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;
            p.prev.next = n;
            p.prev = n;
        }
        size++;
        return true;
    }
```




```
list.add("E", 2);
```



LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;
            p.prev.next = n;
            p.prev = n;
        }
        size++;
        return true;
    }
```

Time Complexity:

- O(1)
- O(1)
- O(1)
- O(1)
- O(1)

LinkedIndexedList – add(element, index)

```
public class LinkedIndexedList<T>
                implements IndexedList<T> {
    public boolean add(T element, int index) {
        if (index == size) {
            return add(element);
        }
        if (!validIndex(index)) {
            return false;
        }
        Node n = new Node(element);
        if (index == 0) {
            n.next = front;
            front.prev = n;
            front = n;
        } else {
            Node p = front;
            for (int i = 0; i < index; i++) {
                p = p.next;
            }
            n.next = p;
            n.prev = p.prev;
            p.prev.next = n;
            p.prev = n;
        }
        size++;
        return true;
    }
```

Time Complexity: **O(N)**

O(1)

O(1)

O(1)

O(N)

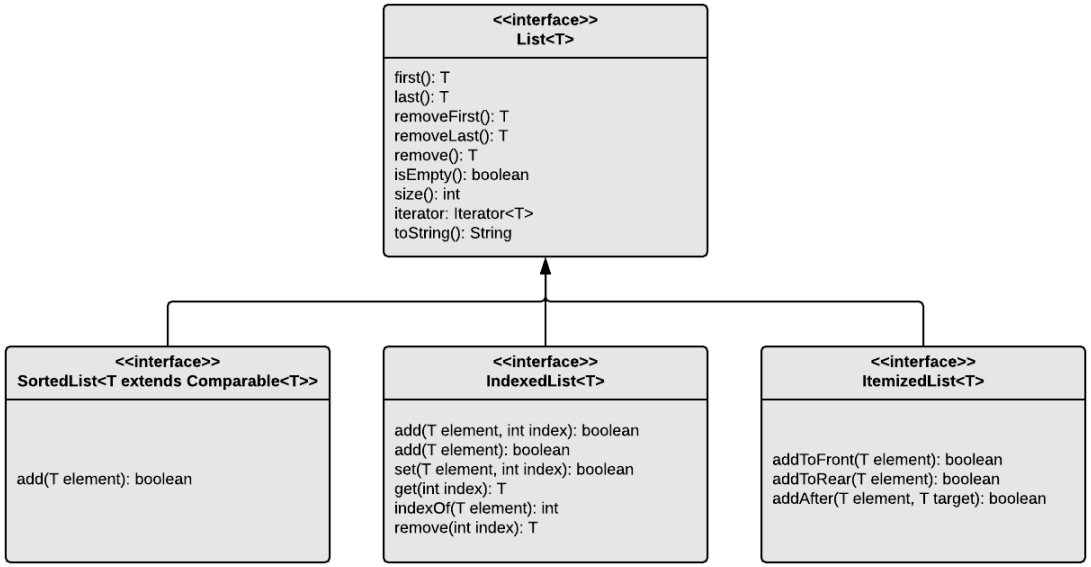
O(1)

O(1)

Performance analysis

Performance analysis

Based on what we know about implementation patterns, we should be able to describe the time complexity of any list method, even without a reference implementation to study.



Performance analysis

	IndexedList		ItemizedList		SortedList	
method	**Array**	**Nodes**	**Array**	**Nodes**	**Array**	**Nodes**
remove(element)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
addAfter(element, target)	•	•	O(N)	O(N)	•	•
add(element)	O(1)	O(1)	•	•	O(N)	O(N)
add(index, element)	O(N)	O(N)	•	•	•	•
get(index)	O(1)	O(N)	•	•	•	•
indexOf(element)	O(N)	O(N)	•	•	•	•

Performance analysis

	IndexedList		ItemizedList		SortedList	
method	**Array**	**Nodes**	**Array**	**Nodes**	**Array**	**Nodes**
remove(element)	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
addAfter(element, target)	•	•	O(N)	O(N)	•	•
add(element)	O(1)	O(1)	•	•	O(N)	O(N)
add(index, element)	O(N)	O(N)	•	•	•	•
get(index)	O(1)	O(N)	•	•	•	•
indexOf(element)	O(N)	O(N)	•	•	•	•

If we could use binary search on a node-based structure, we could add, remove, and search a sorted list in O(log N) time.