ArrayLists
&
List Computational Complexity

Generic Operations on a List

- create an empty list
- add(x) – insert x at the end of the list
- add(x, idx) – inserts x into the list at the specified position
- clear( ) – removes all elements from the list
- get(idx) – returns the object at position idx
- indexOf(x) – returns the position of the first occurrence of x
- isEmpty( ) – returns true if the list has no elements
- printList() – prints all elements in the list
- remove(idx) – removes the element at idx
- set(idx, x) – replace the specified position with x
- size() – returns the number of elements in the list
- ...

Simple Array Implementation of a List

- Use an array to store the elements of the list
  - printList is *O*(n)
  - add(x), get(idx) and set(idx, x) are constant time
  - What about add(idx, x) and remove(idx)?

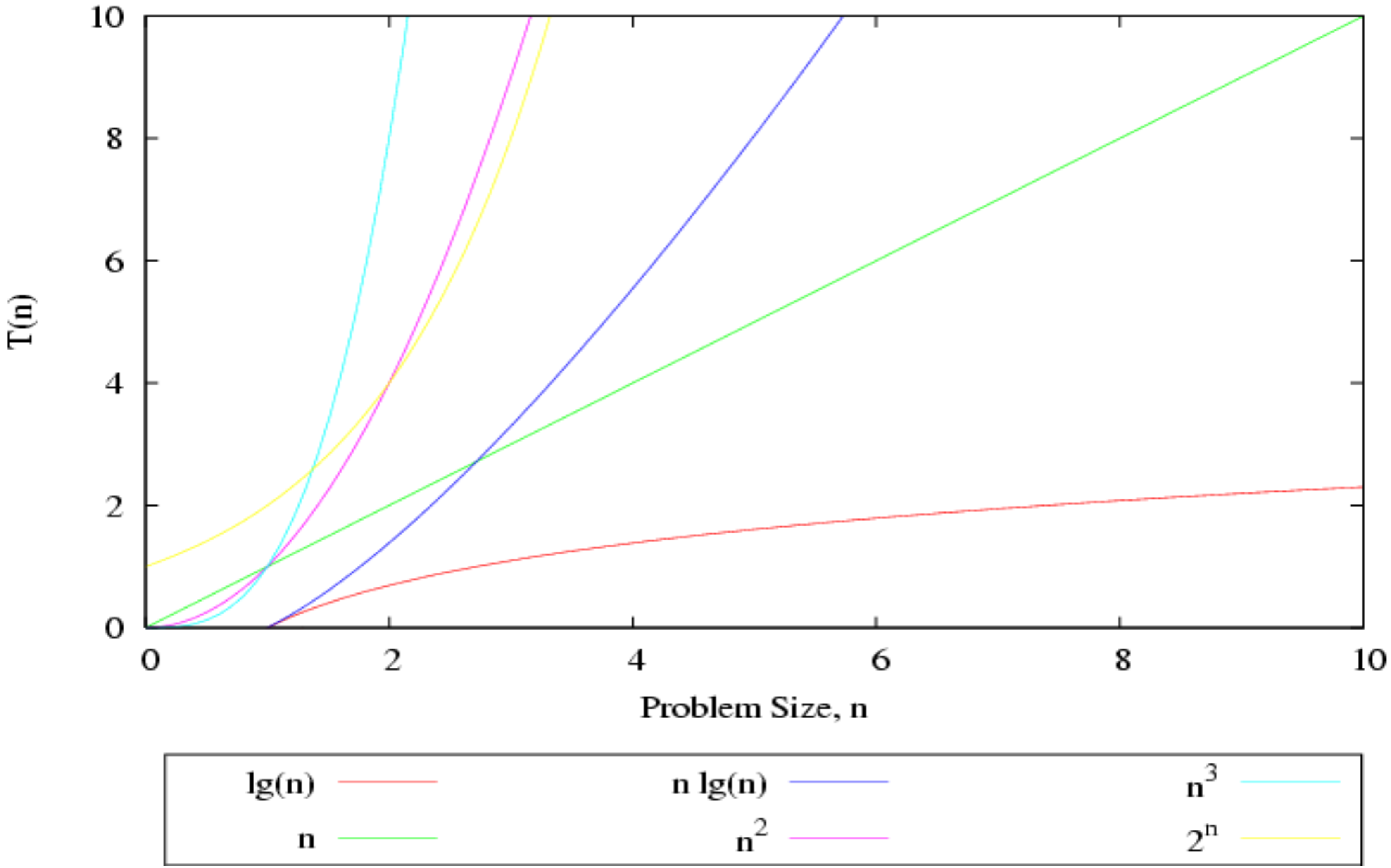
- Also, arrays have a fixed capacity, but we can “resize” them by copying elements to a new larger array

```
int arr[] = new int arr[10];
int newArr[] = new int[arr.length * 2];
for(int i = 0; i < arr.length; i++)
  newArr[i] = arr[i];
arr = newArr;  // arr is now twice as large
```

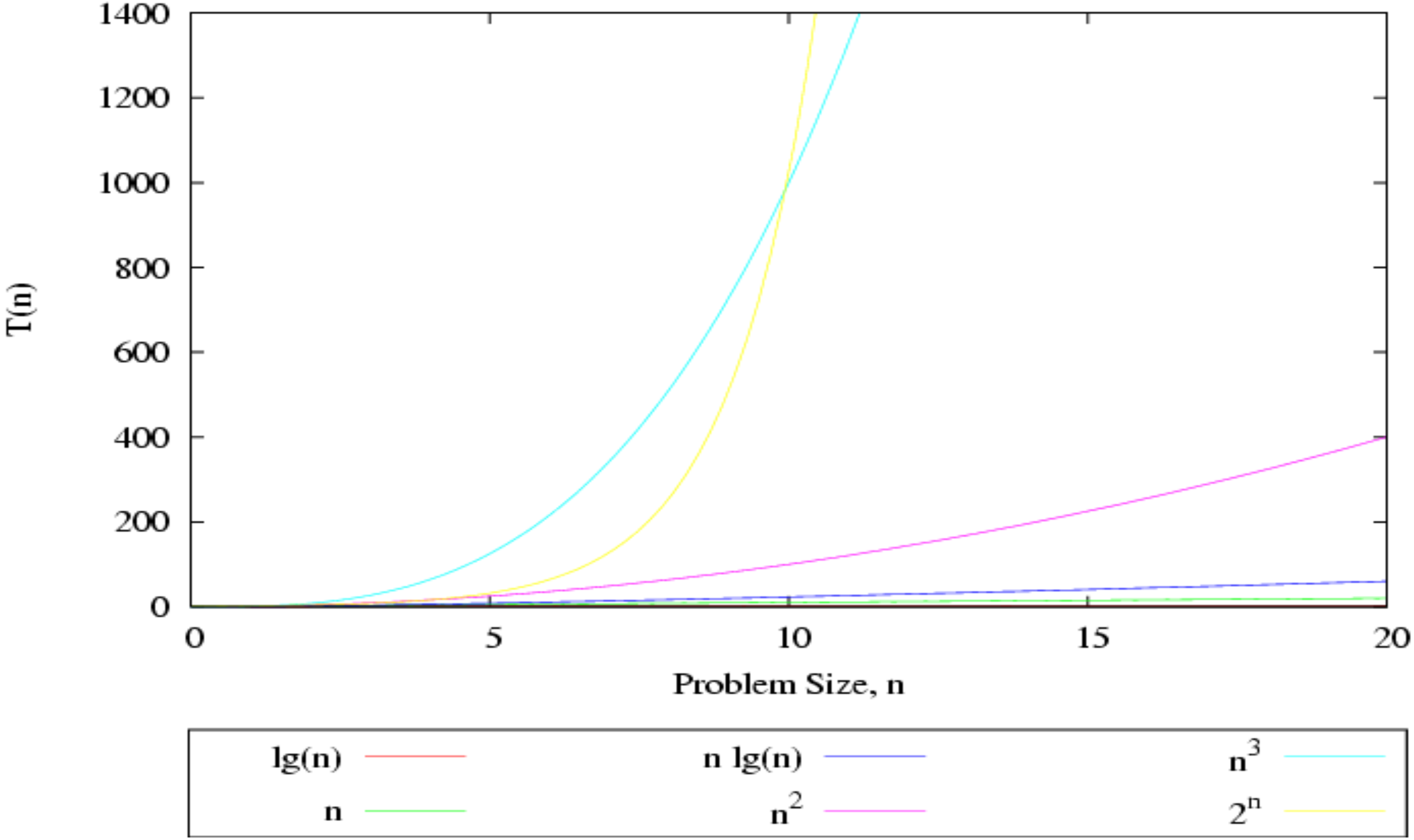
Concrete Implementations of the List ADT in the Java Collections API

- Two concrete implementations of the List API in the Java Collections API with which you are already familiar are:
  - java.util.ArrayList
  - java.util.LinkedList
- Let's examine the methods of these concrete classes that were developed at Sun.

A Graph of Growth Functions



Expanded Scale



List Operations on an ArrayList<E>

- Supports constant time for
  - insertion at the “end” of the list using

    `void add(E element)`
  - deletion from the “end” of the list using

    `E remove(int index)`
  - access to any element of the list using

    `E get(int index)`
  - changing value of any element of the list using

    `E set(int index, E element)`

List Operations on an ArrayList<E> (cont.)

What is the computational complexity for the following?

- insertion at the "beginning" of the list using

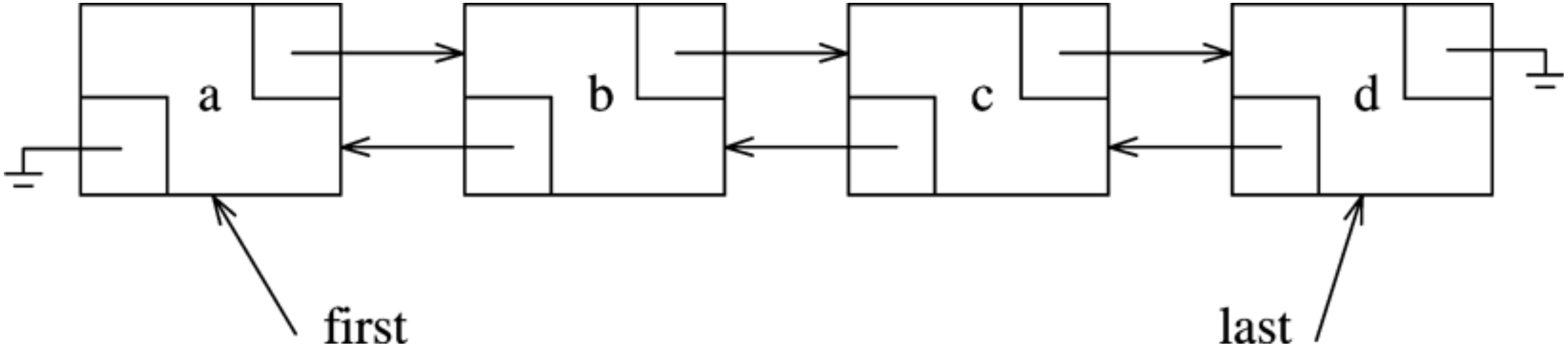
  `void add(int index, E element)`

- deletion from the "beginning" of the list using

  `E remove(int index)`

List Operations on a LinkedList<E>

- Provides doubly-linked list implementation



List Operations on a LinkedList<E>

- Supports constant time for:
  - insertion at the "beginning" of the list using
    `void addFirst(E o)`
  - insertion at the "end" of the list using
    `void addLast(E o)`
  - deletion from the "beginning" of the list using
    `E removeFirst()`
  - deletion from the "end" of the list using
    `E removeLast()`
  - Accessing first element of the list using

    `E getFirst()`
  - Accessing first element of the list using

    `E getLast()`

List Operations on a LinkedList<E>

- What is the complexity for the following?
  - access to the “middle” element of the list using
    `E get(int index)`

Example 1 –ArrayList vs. LinkedList

- What is the running time for an ArrayList versus a LinkedList?

```
public static void
makeList1(List<Integer> list, int N)
{
  list.clear();
  for(int i = 0; i < N; i++)
     list.add(i);
}
```

Example 2 –ArrayList vs. LinkedList

- What is the running time for an ArrayList versus a LinkedList?

```
public static void
makeList2(List<Integer> list, int N)
{
     list.clear();
     for(int i = 0; i < N; i++)
          list.add(0,i);
}
```

Example 3 –ArrayList vs. LinkedList

- What is the running time for an ArrayList versus a LinkedList?

```
public static
int sum(List<Integer> list, int N)
{
    int total = 0;
    for(int i = 0; i < N ; i++)
            total += list.get(i);
    return total;
}
```

- How can we change this code so the running time for both is the same?

Extra Material

Methods from the Collections List ADT

```
//from Collection interface
int size( );
boolean isEmpty( );
void clear( );
boolean contains( AnyType x );
boolean add( AnyType x );
boolean remove( AnyType x );
java.util.Iterator<AnyType> iterator( );
//from List interface
AnyType get( int idx );
AnyType set( int idx, AnyType newVal );
void add( int idx, AnyType x );
void remove( int idx );
ListIterator<AnyType> listIterator(int pos);
```

The Iterator<E> Interface

- The Collections framework provides two very useful interfaces for traversing a *Collection*. The first is the Iterator<E> interface.
- When the *iterator* method is called on a *Collection,* it returns an *Iterator* object which has the following methods for traversing the Collection.

```
boolean hasNext( );
AnyType next( );
void remove( );
```

Using an Iterator to Traverse a Collection

```
public static <AnyType>
void print(Collection<AnyType> coll)
{
    Iterator<AnyType> itr = coll.iterator();
    while(itr.hasNext()){
          AnyType item = itr.next();
          System.out.println(item);
    }
}
```

The Enhanced for Loop

- The enhanced for loop in Java actually calls the *iterator* method when traversing a *Collection* and uses the *Iterator* to traverse the *Collection* when translated into byte code.

```
public static <AnyType> void
print(Collection<AnyType> coll) {
      for(AnyType item : coll)
          System.out.println(item);
}
```

The ListIterator<E> Interface

- The second interface for traversing a *Collection* is the ListIterator<E> interface. It allows for the bidirectional traversal of a *List*.

```
boolean hasPrevious();
AnyType previous();
void add(AnyType x);
void set(AnyType newVal);
```

- A *ListIterator* object is returned by invoking the *listIterator* method on a *List*.

Implementing Your Own ArrayList

- What do you need?
  1. Store elements in a parameterized array
  2. Track number of elements in array (size) and capacity of array

```
public class MyArrayList<AnyType>
implements Iterable<AnyType>
{
  private static final int DEFAULT_CAPACITY=10;

    private int theSize;
    private AnyType[] theItems;
```

3. Ability to change capacity of the array

```
public void ensureCapacity(int newCapacity)
{
   if (newCapacity < theSize)
       return;

   AnyType[] old = theItems;
   theItems = (AnyType[]) new Object[newCapacity];
   for(int i = 0; i < size(); i++)
       theItems[i] = old[i];
}
```

4. get and set Methods

```
public AnyType get(int idx)
{
    if(idx < 0 || idx >= size())
       throw new ArrayIndexOutOfBoundsException();
    return theItems[idx];
}

public AnyType set(int idx, AnyType newVal)
{
    if(idx < 0 || idx >= size())
      throw new ArrayIndexOutOfBoundsException();
    AnyType old = theItems[idx];
    theItems[idx] = newVal;
    return old;
}
```

5. size, isEmpty, and clear Methods

```
public void clear(){
    theSize = 0;
    ensureCapacity(DEFAULT_CAPACITY);
}

public int size(){
    return theSize;
}

public boolean isEmpty(){
    return size() == 0;
}

// constructor invokes the clear method
public MyArrayList(){
    clear();
}
```

6. add Methods

```
public boolean add(AnyType x) {
        add(size(), x);
        return true;
}

public void add(int idx, AnyType x) {
        if(theItems.length == size())
            ensureCapacity(size() * 2 + 1);
        for(int i = theSize; i > idx; i--)
            theItems[i] = theItems[i - 1];
        theItems[idx] = x;
        theSize++;
}
```

7. remove and iterator Method

```
public AnyType remove(int idx) {
   AnyType removedItem = theItems[idx];
   for(int i = idx; i < size() - 1; i++)
            theItems[i] = theItems[i + 1];
   theSize--;
   return removedItem;
}

//required by Iterable<E> interface
public java.util.Iterator<AnyType> iterator() {
    return new ArrayListIterator();
}
```

8. Iterator class

```
 // private inner class for iterator
 private class ArrayListIterator implements
   java.util.Iterator<AnyType>
      {
            private int current = 0;

            public boolean hasNext()
              { return current < size(); }
            public AnyType next()
              { return theItems[current++]; }
            public void remove()
              { MyArrayList.this.remove(--current); }
      }
} // end MyArrayList class
```

Implicit reference to outer class method

Implicit ref. to outer class data

Explicit reference to outer class method

Example 4 –ArrayList vs. LinkedList

- What is the running time for an ArrayList versus a LinkedList?

```
public static void
removeEvensVer3(List<Integer> lst)
{
      Iterator<Integer> itr = lst.iterator();

      while(itr.hasNext())
          if(itr.next() % 2 == 0)
              itr.remove();
}
```