An Array-Based Implementation of the ADT List

```
public class ListArrayBased
implements ListInterface {

  private static final int MAX_LIST = 50;
  private Object items[];
    // an array of list items
  private int numItems;
    // number of items in list
```

An Array-Based Implementation of the ADT List

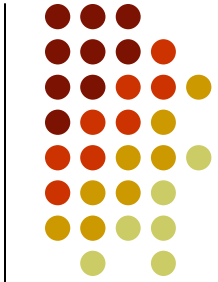
```
public ListArrayBased() {
   items = new Object[MAX_LIST];
   numItems = 0;
 }  // end default constructor
```

An Array-Based Implementation of the ADT List

```
public boolean isEmpty() {
   return (numItems == 0);
} // end isEmpty

public int size() {
   return numItems;
}  // end size
```

Insertion into Array



What happens if you want to insert an item at a specified position in an existing array?

- Write over the current contents at the given index (which might not be appropriate), or
- The item originally at the given index must be moved up one position, and all the items after that index must ***shuffled up***

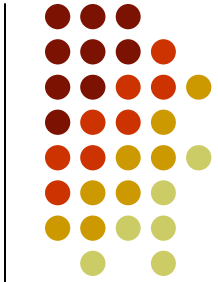
An Array-Based Implementation of the ADT List

```
public void add(int index, Object item)
throws  ListException,
        ListIndexOutOfBoundsException {
   if (numItems >= MAX_LIST) {
     throw new ListException("ListException on add:"+
       " out of memory");
   }  // end if
   if (index < 1 || index > numItems+1) {
     // index out of range
     throw new ListIndexOutOfBoundsException(
      "ListIndexOutOfBoundsException on add");
  }  // end if
```

An Array-Based Implementation of the ADT List

```
    // make room for new element by shifting all items at
    // positions >= index toward the end of the
    // list (no shift if index == numItems+1)
       for (int pos = numItems; pos >= index; pos--) {
           items[pos] = items[pos-1];
       } // end for
       // insert new item
       items[index-1] = item;
       numItems++;
} //end add
```

Removal from Arrays



What happens if you want to remove an item from a specified position in an existing array?

- Leave ***gaps*** in the array, i.e. indices that contain no elements, which in practice, means that the array element has to be given a special value to indicate that it is “empty”, or
- All the items after the (removed item’s) index must be ***shuffled down***

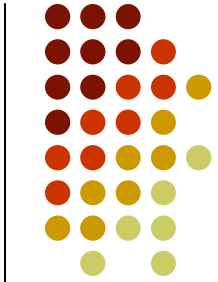
An Array-Based Implementation of the ADT List

```
public void remove(int index)
throws ListIndexOutOfBoundsException {
   if (index >= 1 && index <= numItems) {
     // delete item by shifting all items at
     // positions > index toward the beginning of the list
     // (no shift if index == size)
     for (int pos = index+1; pos <= size(); pos++) {
       items[pos-2] = items[pos-1];
     }  // end for
     numItems--;
   }
   else {  // index out of range
     throw new ListIndexOutOfBoundsException(
               "ListIndexOutOfBoundsException on remove");
   }  // end if
 } // end remove
```

An Array-Based Implementation of the ADT List

```
public Object get(int index)
throws ListIndexOutOfBoundsException {
   if (index >= 1 && index <= numItems) {
     return items[index-1];
   }
   else  {  // index out of range
     throw new ListIndexOutOfBoundsException(
       "ListIndexOutOfBoundsException on get");
   }  // end if
 } // end get
```

An Array-Based Implementation - Summary

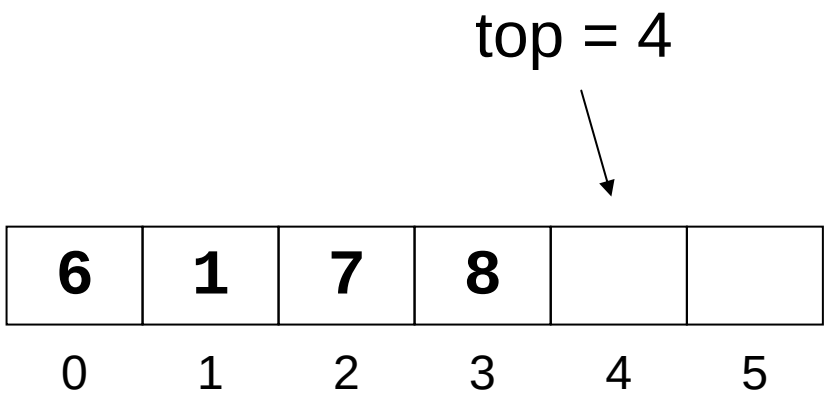


- **Good** things:
  - Fast, random access of elements
  - Very memory efficient, very little memory is required other than that needed to store the contents (but see bellow)
- **Bad** things:
  - Slow deletion and insertion of elements
  - Size must be known when the array is created and is fixed (static)

ADT List using Dynamic arrays

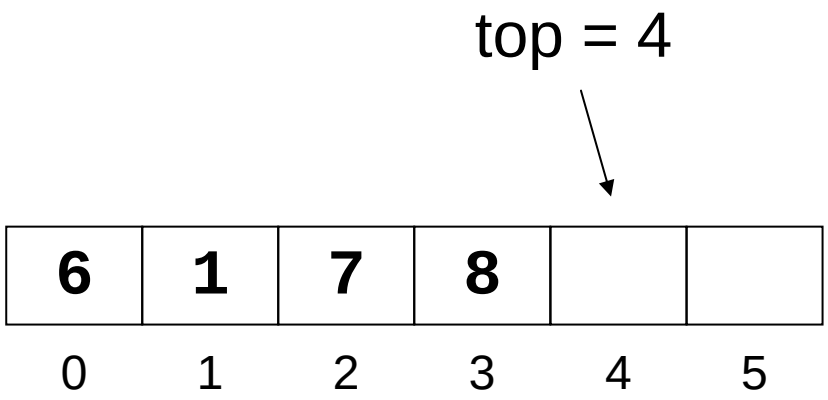
- A dynamic data structure is one that changes size, as needed, as items are inserted or removed
  - The Java **`ArrayList`** class is implemented using a dynamic array
- There is usually no limit on the size of such structures, other than the size of main memory
- Dynamic arrays are arrays that grow (or shrink) as required
  - In fact a new array is created when the old array becomes full by creating a new array object, copying over the values from the old array and then assigning the new array to the existing array reference variable

Dynamic Array

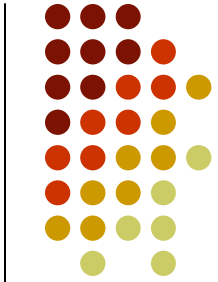


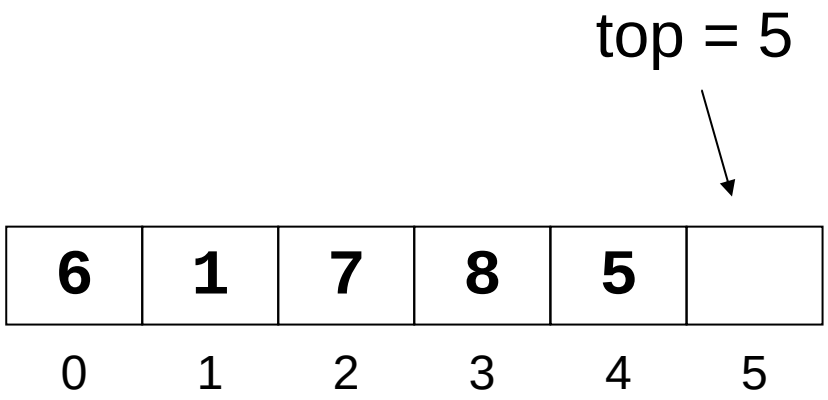
Dynamic Array

insert 5

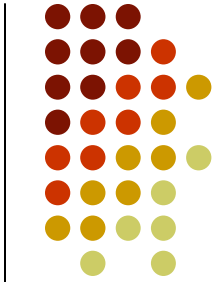


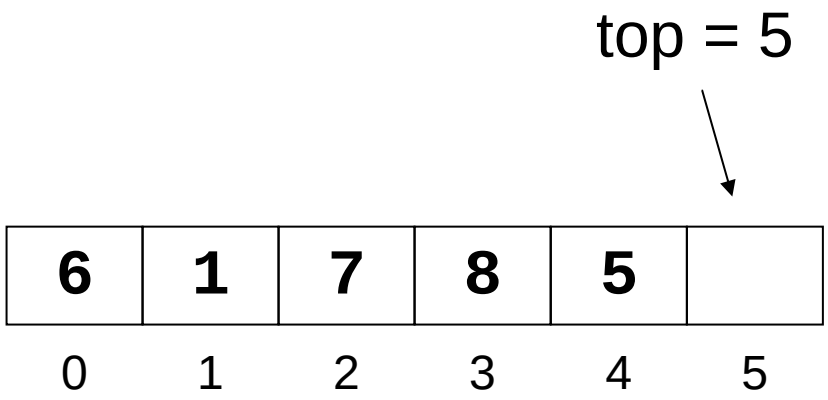
Dynamic Array





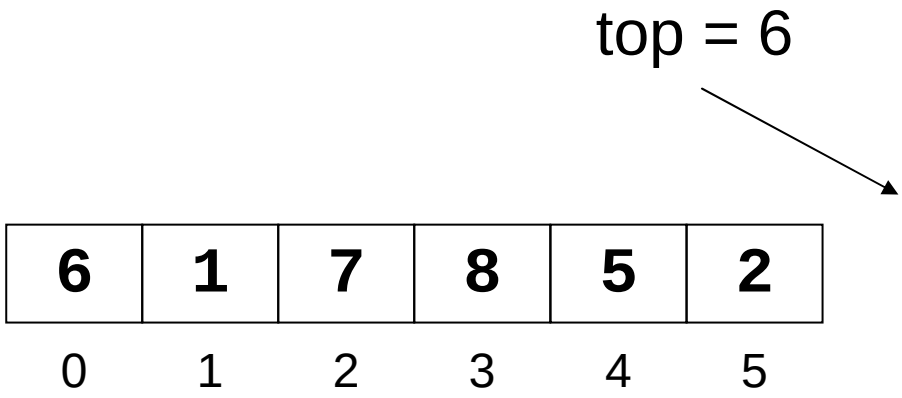
Dynamic Array





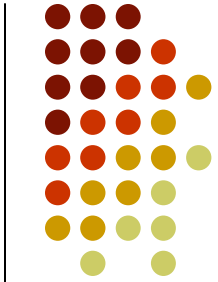
insert 2

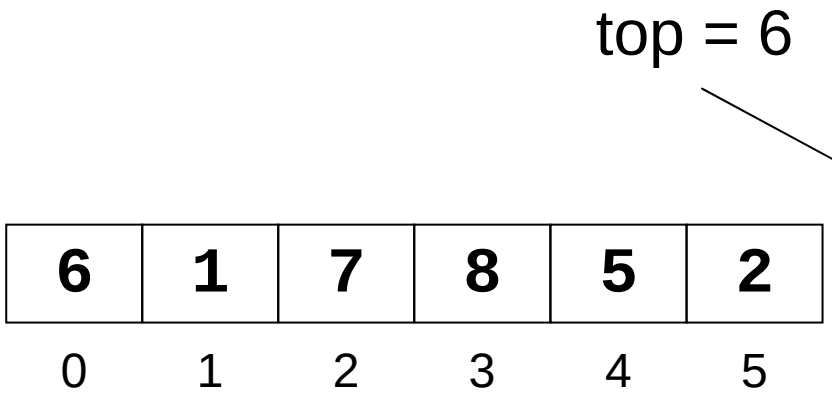
Dynamic Array



insert 3

Dynamic Array

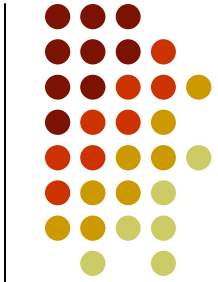


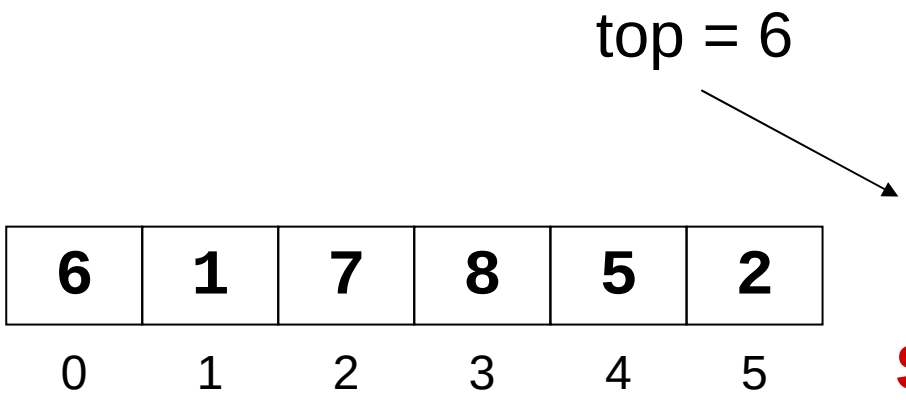


insert 3

!The array is full and there is no room for a new item!

Dynamic Array





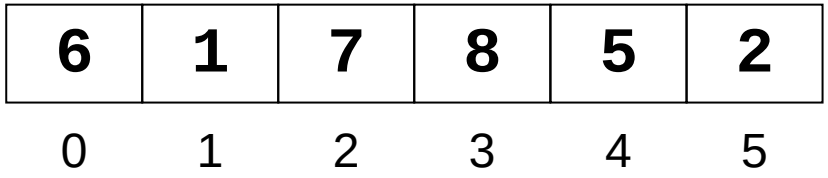
insert 3

So we will create a new, bigger array …

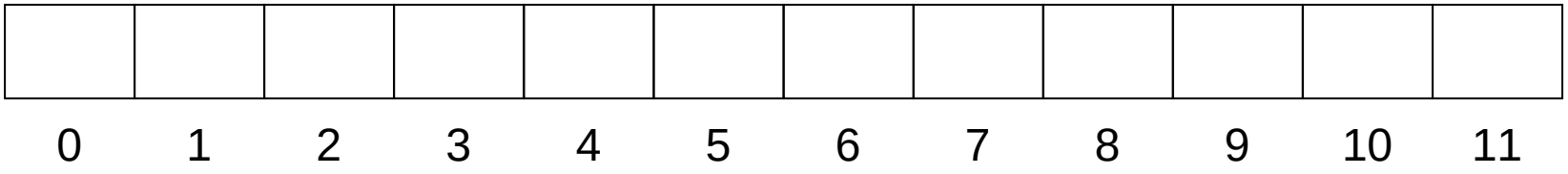
Dynamic Array

top = 6

insert 3



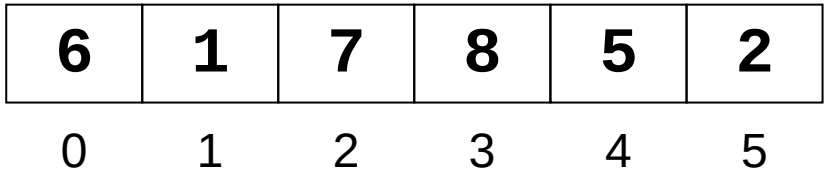
So we will create a new, bigger array …



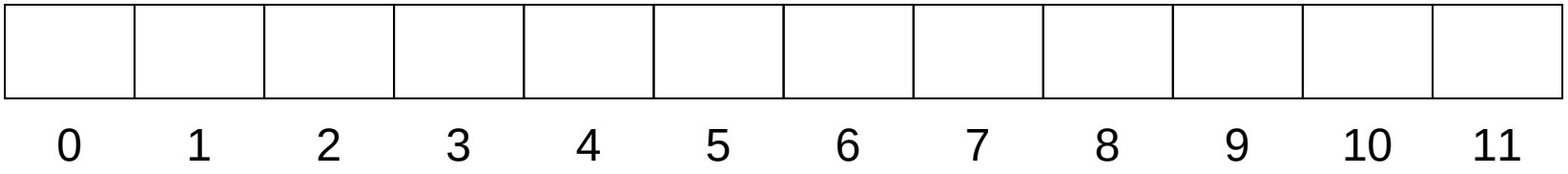
Dynamic Array

top = 6

insert 3

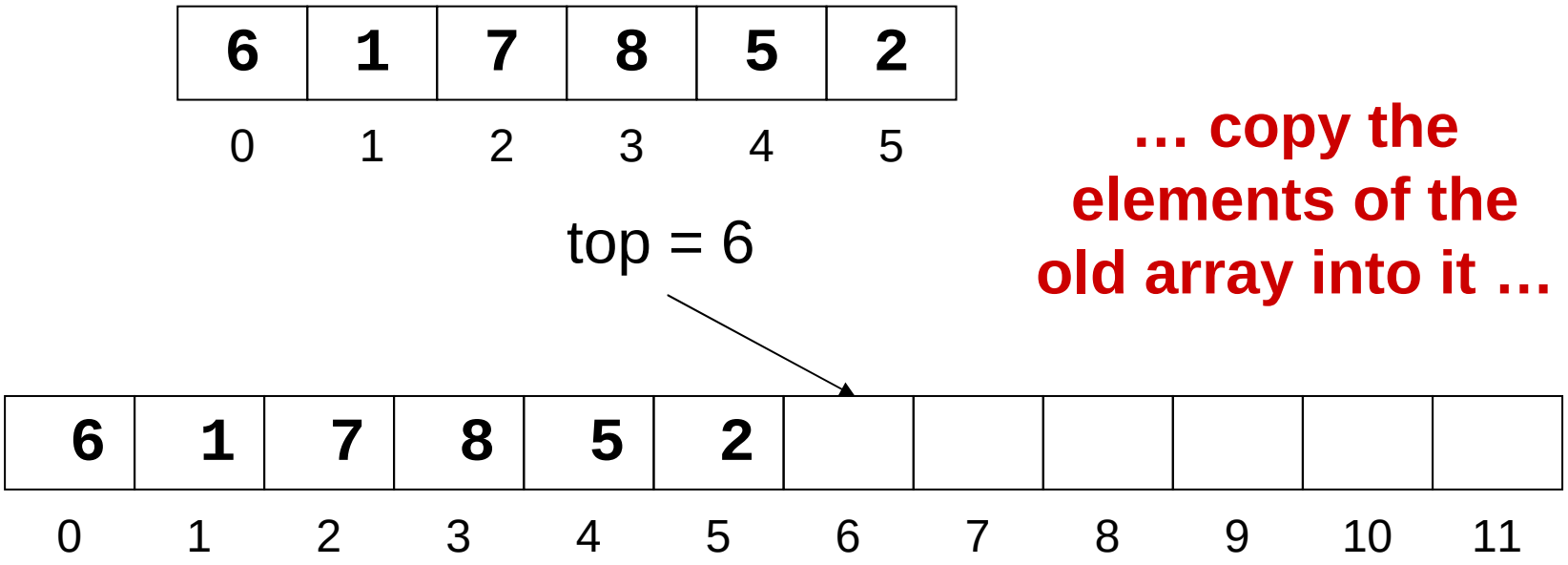


… copy the elements of the old array into it …



Dynamic Array

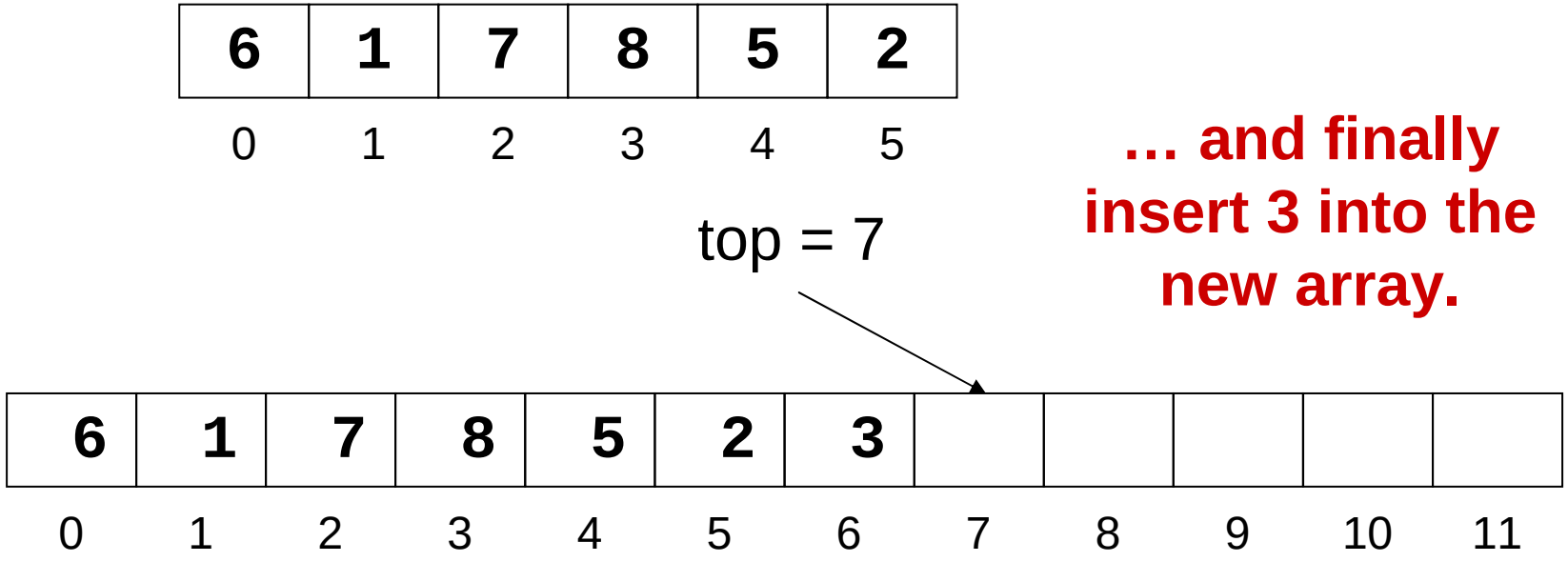
insert 3



… copy the elements of the old array into it …

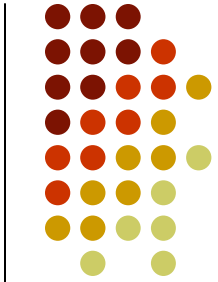
Dynamic Array

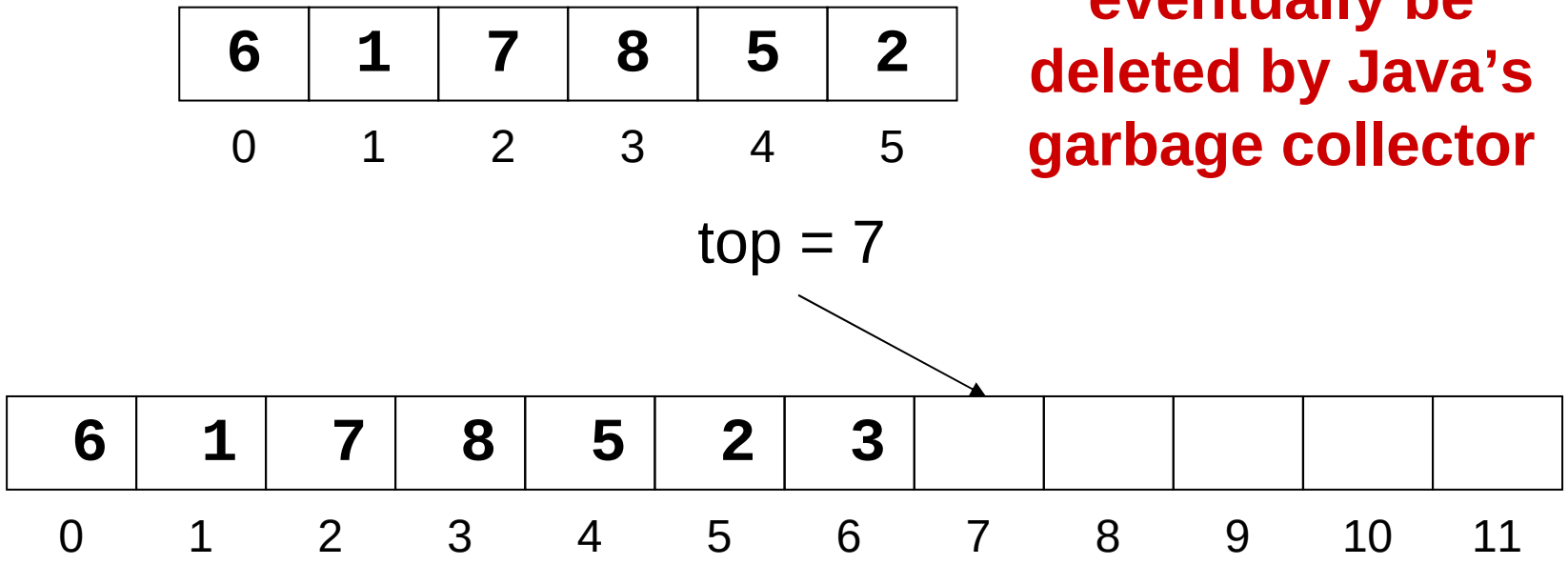
insert 3



… and finally insert 3 into the new array.

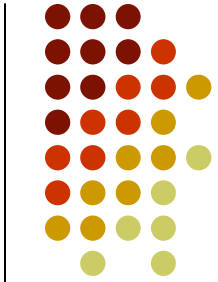
Dynamic Array





The old array will eventually be deleted by Java's garbage collector

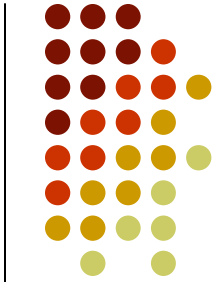
Dynamic Array Summary



- Before every insertion, check to see if the array needs to grow

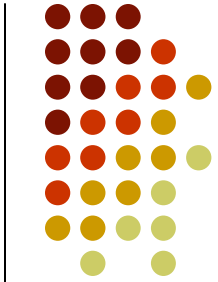
- Question: When growing array, how much to grow it?
  - Memory efficient? (by 1)
  - Time efficient?

Dynamic Array Summary



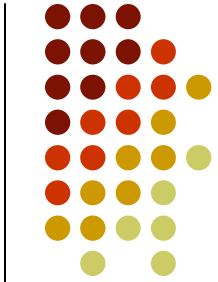
- Before every insertion, check to see if the array needs to grow
- Growing by doubling works well in practice, because it grows very large very quickly
  - 10, 20, 40, 80, 160, 320, 640, 1280, …
  - Very few array re-sizings must be done
  - To insert n items you need to do ≈ log(n) re-sizings
- While the copying operation is expensive it does not have to be done often

Dynamic Array Problems



- When the doubling does happen it may be time-consuming
- And, right after a doubling half the array is empty
- Re-sizing after each insertion would be prohibitively slow
- Deleting and inserting in the middle (on average) is still O(n)

(Optional) Homework



- Modify implementation of ListArrayBased class so that it works as dynamic List (it will never throw an exception ListException).

  (Basically it is enough to modify add() method.)