EECE 2560:
Fundamentals of Engineering Algorithms

Array-Based Implementations

The ADT Approach

- An ADT is
  - A collection of data … and …
  - A set of operations on that data
- Specifications indicate
  - What ADT operations do
  - But not how to implement
- First step for implementation
  - Choose data structure

Core Methods

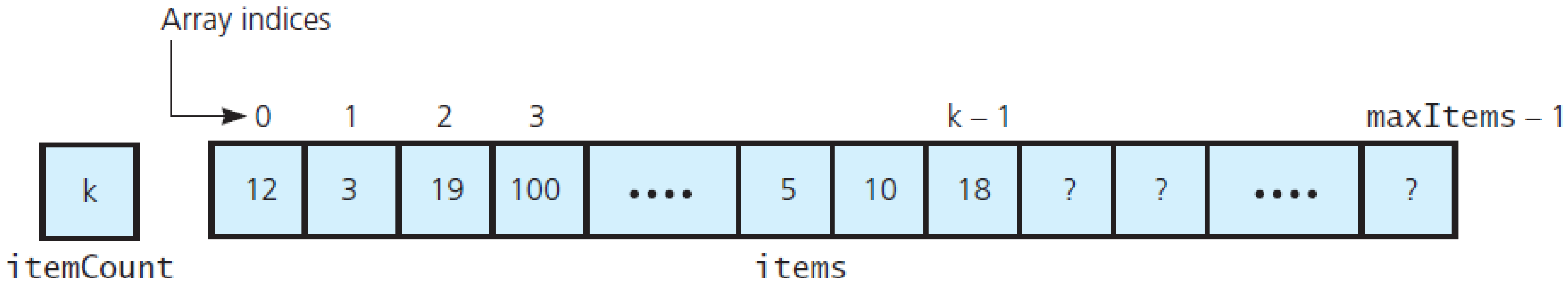
- Poor approach
  - Define entire class and attempt test
- Better plan – Identify, then test basic (core) methods
  - Create the container (constructors)
  - Add items
  - Display/list items
  - Remove items

ADT Bag Header File

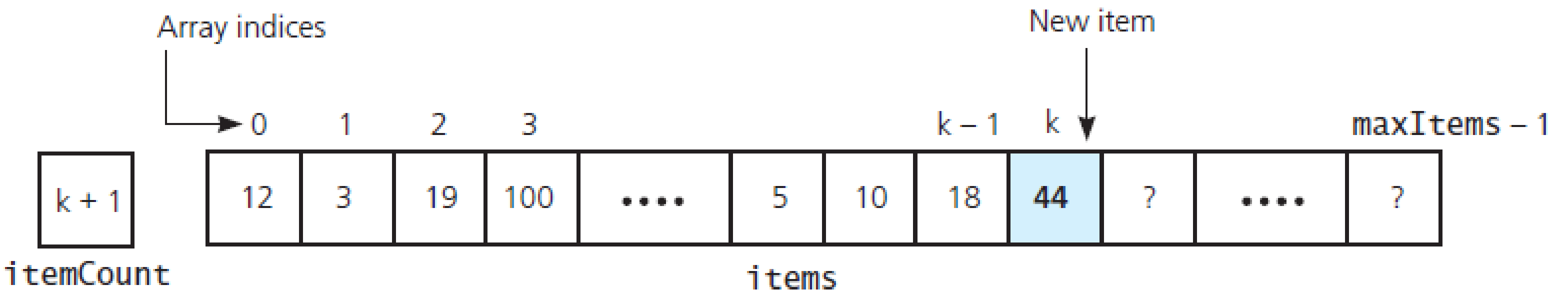
```
template<class ItemType>
class Bag {
 private:
   static const int DEFAULT_BAG_SIZE = 6;
   ItemType items[DEFAULT_BAG_SIZE]; // array of bag items
   int itemCount;                    // current count of bag items
   int maxItems;                     // max capacity of the bag
   // Returns the index of the element in the array items
   int getIndexOf(const ItemType& target) const;
public:
        Bag();
        int getCurrentSize() const;
        bool isEmpty() const;
        bool add(const ItemType& newEntry);
        bool remove(const ItemType& anEntry);
        void clear();
        bool contains(const ItemType& anEntry) const;
        int getFrequencyOf(const ItemType& anEntry) const;
    std::vector<ItemType> toVector() const;
};  // end Bag
```

Using Fixed-Size Arrays

- Must keep track of array elements used, available
- Decide if first object goes in element 0 or 1
- Consider if the **add** method places elements in consecutive elements of array
- What happens when **add** method has used up final available element?



The add Method



```
template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newEntry)
{
        bool hasRoomToAdd = (itemCount < maxItems);
        if (hasRoomToAdd)
        {
                items[itemCount] = newEntry;
                itemCount++;
        }  // end if

        return hasRoomToAdd;
}  // end add
```

The `getFrequencyOf` Method

```cpp
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
   int frequency = 0;
   int curIndex = 0;       // Current array index
   while (curIndex < itemCount)
   {
      if (items[curIndex] == anEntry)
      {
         frequency++;
      }  // end if

      curIndex++;          // Increment to next entry
   }  // end while

   return frequency;
}  // end getFrequencyOf
```

The getIndexOf Method

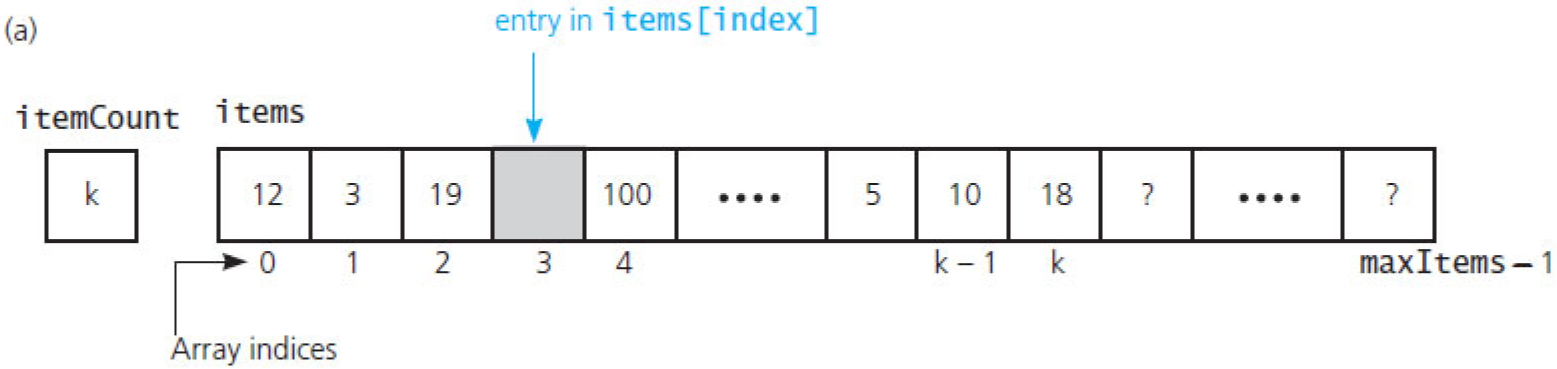
```
// private
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
{
   bool found = false;
   int result = -1;
   int searchIndex = 0;
   // If the bag is empty, itemCount is zero, so loop is skipped
   while (!found && (searchIndex < itemCount)) {
      if (items[searchIndex] == target) {
         found = true;
         result = searchIndex;
      }
      else   { searchIndex++; }
   }  // end while

   return result;
}  // end getIndexOf
```

The contains Method

```
template<class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anEntry) const
{
       return getIndexOf(anEntry) > -1;
}  // end contains
```

The `remove` Method (1 of 4)

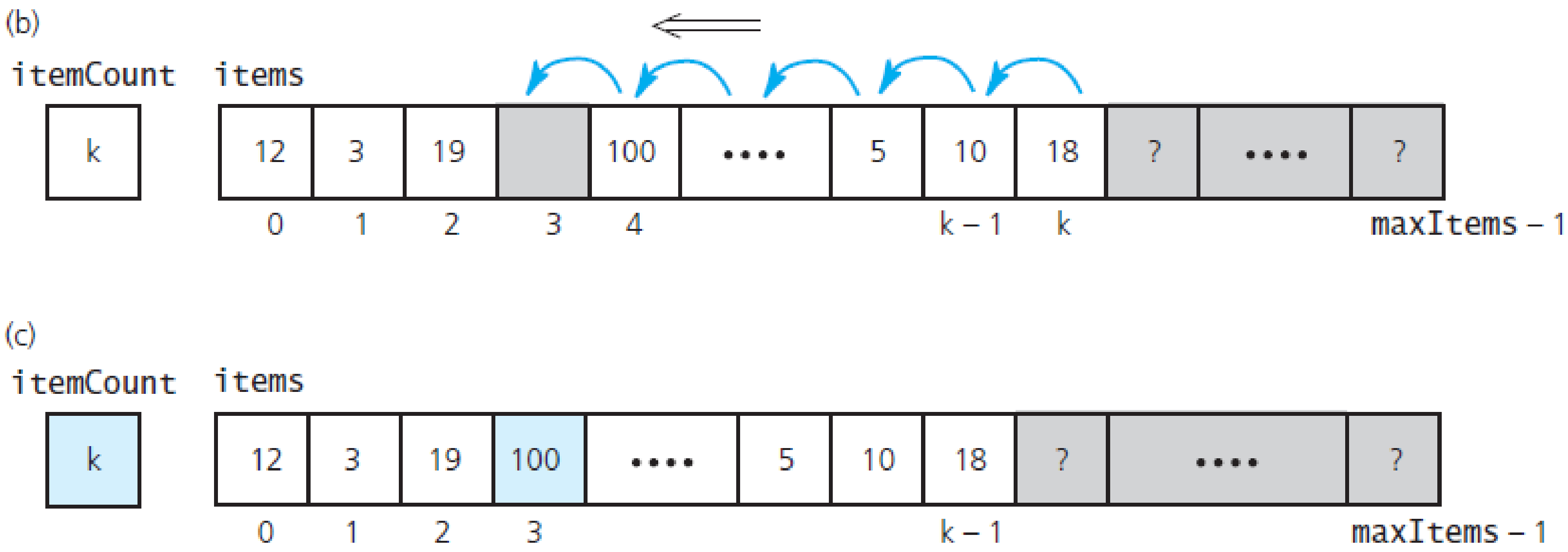
A gap in the array items after the entry in **items[index]** and decrementing **itemCount**:



12	3	19		100	••••	5	10	18	?	••••	?
0	1	2	3	4			k – 1	k			maxItems – 1

The remove Method (2 of 4)

Shifting subsequent entries to avoid a gap:



(b)

itemCount: k

items: 12 | 3 | 19 | | 100 | •••• | 5 | 10 | 18 | ? | •••• | ?

0 1 2 3 4 k – 1 k maxItems – 1

(c)

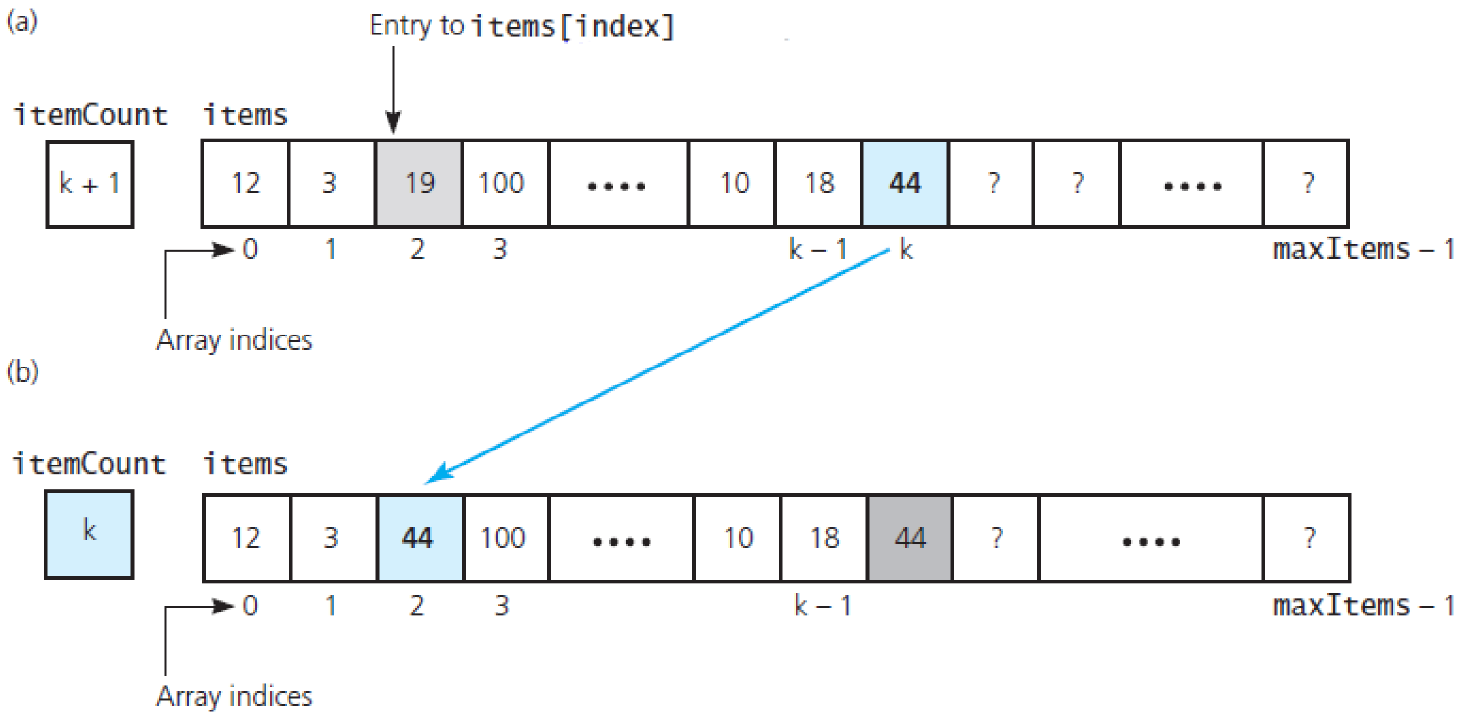
itemCount: k

items: 12 | 3 | 19 | 100 | •••• | 5 | 10 | 18 | ? | •••• | ?

0 1 2 3 k – 1 maxItems – 1

The `remove` Method (3 of 4)

Avoiding a gap in the array while removing an entry:



The remove Method (4 of 4)

```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
   int locatedIndex = getIndexOf(anEntry);
       bool canRemoveItem = !isEmpty() && (locatedIndex > -1);
       if (canRemoveItem)
       {
               itemCount--;
               items[locatedIndex] = items[itemCount];
       }  // end if

       return canRemoveItem;
}  // end remove
```