CS 162C+
Unordered Arrays

Arrays

As we learned earlier, arrays are sequential locations in memory that contain similar items. Very often we want to search an array to see if a given item is contained in it.

Unordered Arrays

First unordered arrays; these are arrays that contain values in no particular order. For most of the work in this term, and for most of the use of arrays as storage devices in general, you want to maintain a compact array. That is, you want all of the values stored in the array to be in the locations array [0] to array[count-1] where count is how many items are being stored. Any other locations are assumed to be undefined and will not be referenced when storing new values into them.

Since all of the locations of interest contain meaningful values and none of the others matter, there is no reason to initialize such an array or to set locations to some special value when removing values.

Adding new items to an Unordered Array

Let's walk through what we will be doing with an illustration before we code it:

Consider the following array of integers and an index called count, that is initialized to 0, and a size of the array that is set to six.

array[0]	
array[1]	
array[2]	
array[3]	
array[4]	
array[5]	
count	0

If we add the number six to the array, then it should look like this:

array[0]	6
array[1]	
array[2]	
array[3]	
array[4]	
array[5]	
count	1

If we then add the number 32, it should look like this:

array[0]	6
array[1]	32
array[2]	
array[3]	
array[4]	
array[5]	
count	2

Remember that we do not care what happens to be in memory at the locations in the array that are blank.

Coding this functionality using a simple pseudo code results in:

```
addItem(int value)
   array[count] = value
   count += 1
```

This code also does not check for whether there is room for a new item before adding it as the following expanded code does:

Removing an item from an Unordered Array

Let's walk through what we will be doing with an illustration before we code it. We will use the same array as before, but now it has had three more items added and appears as below:

array[0]	6
array[1]	32
array[2]	4
array[3]	0
array[4]	10
array[5]	
count	5

If we wanted to remove the last added item, we just decrement count to show there are now only four items and return the value.

```
removeItem()
   count -= 1
   return array[count]
```

array[0]	6
array[1]	32
array[2]	4
array[3]	0
array[4]	10
array[5]	
count	4

Note that we do not need to do anything to the location array[4] since we have already said that any locations greater than array[count-1] are not defined.

Also, note that the decrement must occur before the reference to the array and this algorithm does not do any error checking to make sure the array has items before removing one.

Finding an item in an Unordered Array

Using the same array as above, lets search for an item:

array[0]	6
array[1]	32
array[2]	4
array[3]	0
array[4]	10
array[5]	
count	4

In this case, let's check if the number **7** is present. We start by looking at **array[0],** if that is not correct, we look at the next location until we have inspected all locations with valid numbers in them. If we find it, we return true, if it is not in the array, we return false.

Here is the algorithm for this:

```
findValue(int value)
   for (int i = 0; i < count; i++)
      if (array[i] == value)
         return true
   return false
```

This is a sequential search since we search each location in sequence. We do not need to worry about an empty array, it will simply return false for not found. No error checking is required.

Finding and removing an item in an Unordered Array (maintaining order)

The final algorithm to consider is how to find and remove an item from an unsorted array. Consider the array that we have below. If we want to find and remove the number 4 from it, we have two choices.

array[0]	6
array[1]	32
array[2]	4
array[3]	0
array[4]	7
array[5]	
count	5

The first choice is to maintain the same order, but with the number 4 removed:

array[0]	6
array[1]	32
array[2]	0
array[3]	7
array[4]	7
array[5]	
count	4

See how the zero and seven that were in positions array[3] and array[4] are now moved to the next lower slots. Again, we do not care about array[4] when we are done, since it is outside the range of locations we consider.

The algorithm that did this compression is as follows. Note the only change from the findValue method above is to add a for loop to compress the array:

```
removeValue(int value)
  for (int i = 0; i < count; i++)
    if (array[i] == value)
      for (int j = i, j < count; j++)
        array[j] = array[j+1]
      count -= 1
      return true
  return false
```

Finding and removing an item in an Unordered Array (without maintaining order)

The final algorithm to consider is how to find and remove an item from an unsorted array when we do not need to preserve the initial order:

array[0]	6
array[1]	32
array[2]	4
array[3]	0
array[4]	7
array[5]	
count	5

We simply copy the value in the last occupied position into that where the item was, in this case, **seven** replaces the **four**.

array[0]	6
array[1]	32
array[2]	7
array[3]	0
array[4]	7
array[5]	
count	4

Again, we do not care about **array[4]** since its content will change when we add a new value.

The algorithm for this is:

```
bool removeValue(int value)
    for(int i = 0; i < count; i++)
        if ( array[i] == value )
            count -= 1
            array[i] = array[count]
            return true;
    return false;
```

This function is more efficient than the array compression method above since it does not need to copy **count/2** elements on average.