Questions

1. The `add(i, e)` method says to first copy all the elements to bigger array, and then to add the new element at the suitable place. This is slightly inefficient since adding the element could be done *while* copying. Write out the method for doing so. Assume the underlying array is `a[ ]`. (This is a good exercise for practicing indices.)

2. If you just want to add an element to a list and you don't care where the element goes, then the simplest approach is to add the element at the end of the list. Or, you just might *want* to add an element at the end of a list. Either way, the method is `add(e)`. Write our the method. Assume the underlying array is `a[ ]`.

3. Give an algorithm for reversing all the elements of an array list which uses a constant amount of additional space (other than the array itself). That is, you are not allowed to use a new array for your solution.

   The main idea is to use a 'swap' method which you should be familiar with from COMP 202.

   ```
   swap(j, k){
     tmp   =  a[j]
     a[j]  =  a[k]
     a[k]  =  tmp
   }
   ```

4. Give a $O(N)$ algorithm for removing the first instance of a given object `e` in a list, assuming the list is represented as an array list and the size of the list is N. That is, give an algorithm for `remove(e)`. In your answer, you can use methods given in the lecture.

   In the lecture, I presented a `remove(i)` algorithm which removes the element at index i in the list. Here I'm asking for a `remove(e)` algorithm which removes the first instance of object in the list, if at least one instance is present.

5. An important property of arrays is that they have constant time access. This property follows from the fact that array slots all have the same size and the address of any slot is the address of the first slot in the array plus some multiple of the array index, where the multiple is the *constant* amount of memory used by each slot.

   What about an array of strings, in which the strings have possibly different lengths? Does this contradict the property just mentioned? Does one still have constant time access to an element in an array of strings?

Answers

1.
```
if (size == length){
     b = new array with 2 * length slots
     for (int j=0; j < i; j++)
        b[j] = a[j]

     b[i] = e

     for (int j = i; j < size; j++)
        b[j+1] = a[j]          // elements must be shifted
        size = size + 1

        a = b
  }
```

2.
```
  if (size == length){
     //  same as above,  make a bigger array and copy into it}
  }
  a[size] = e                        //  insert into now empty slot
  size = size + 1
```

3. The algorithm then swaps the first and the last, the second and second last, etc.

```
reverseArrayList() {
   for (i = 0; i < size/2; i++){
      swap(i, size-1-i)
}
```

   If the list has a odd number of elements, then it doesn't touch the middle one, which is fine. For example, consider the case $i = 13$. It swaps 0,1,2,3,4,5 with 12,11,10,9,8,7, respectively, and doesn't touch 6.

4. The following algorithm loops through the list and examines each element at most once. Hence it takes time proportional to N in the worst case, and so we say it is $O(N)$.

```
remove( e ) {
  i = 0
  found = false
  while ((i < size) and (found == false)){
     if a[i] == e
        found = true
        return remove(i) //  this method was discussed in the lecture
     else
        i++    //  means  i = i + 1
  }
```

```
  print("Could not find it.")
}
```

5. The slots in the "array of strings" don't contain strings. Rather they contain references to strings, that is, they contain addresses of strings. The references themselves are all the same size. The strings may be different size and each is located somewhere (at some address) in memory.