Assignment 3

1. Swap two adjacent elements by adjusting only the links (and not the data) using
   a. Singly linked list
   b. Doubly linked list

For singly linked lists:

```
  // beforeP is the cell before the two adjacent cells that are to be
swapped.
  // Error checks are omitted for clarity.

  public static void swapWithNext( Node beforep )
  {
    Node p, afterp;

    p = beforep.next;
    afterp = p.next;   // Both p and afterp assumed not null.

    p.next = afterp.next;
    beforep.next = afterp;
    afterp.next = p;
  }
```

(b) For doubly linked lists:

```
  // p and afterp are cells to be switched.  Error checks as before.

  public static void swapWithNext( Node p )
  {
    Node beforep, afterp;

    beforep = p.prev;
    afterp = p.next;

    p.next = afterp.next;
    beforep.next = afterp;
    afterp.next = p;
    p.next.prev = p;
    p.prev = afterp;
    afterp.prev = beforep;
  }
```

2. Implement the contains routine for MyLinkedList

```
public boolean contains( AnyType x )
{
   Node<AnyType> p = beginMarker.next;
   while( p != endMarker && !(p.data.equals(x) ))
   {
         p = p.next;
   }

   return (p != endMarker);
}
```

3. What is the running time of the following code?

```
public static List<Integer> makeList( int N)
{
      ArrayList<Integer> lst = new ArrayList<>();
      for( inti  =0; i < N; i++)
      {
            lst.add(i);
            lst.trimToSize();
      }
}
```

$O(N^2)$. The trim method reduces the size of the array, requiring each add to resize it. The resize takes $O(N)$ time, and there are $O(N)$ calls.

4. The following routine removes the first half of the list passed as a parameter:

```
public static void removeFirstHalf(List<?> lst)
{
      int theSize = lst.size() /2
      for( inti =0; i < theSize; i++ )
            lst.remove(0);
}
```

a. Why is theSize saved prior to entering the for loop?
b. What is the running time of removeFirstHalf if lst is an ArrayList?
c. What is the running time of removeFirstHalf if lst is a LinkedLIst?
d. Does using an iterator make removeFirstHalf faster for either type of List

(a) Because the remove call changes the size, which would affect the loop.

(b) $O(N^2)$. Each remove from the beginning requires moving all elements forward, which takes $O(N)$ time.

(c) $O(N)$. Each remove can be done in $O(1)$ time.

(d) No. Since it is always the first element being removed, finding the position does not take time, thus an iterator would not help.