1. Let \( n = \text{rows} \). When \( i = 0 \) the inner loop is executed \( n - 1 \) times, when \( i = 1 \) the inner loop is executed \( n - 2 \) times, when \( i = 2 \) the inner loop is executed \( n - 3 \) times, etc. The work done in the inner loop is constant so the total work is proportional to \( \sum_{i=0}^{n-1} (n - i - 1) = \sum_{i=0}^{n-1} i = n(n - 1)/2 \). In big Oh notation, the answer is \( O(n^{2}) \).

2. (a) \( n^{2} \) is \( O(n^{3}) \) since \( n^{2} \leq 1 \cdot n^{3} \) for all \( n \geq 0 \).
   (b) \( n^{3} \) is not \( O(n^{2}) \) since that would imply there exists a \( c \) such that \( n^{3} \leq cn^{2} \) for all large enough \( n \) but clearly this is impossible if \( n > c \).
   (c) \( 2^{2n} \) is not \( O(2^{n}) \) since this would imply there exists a \( c \) such that \( 2^{2n} \leq c2^{n} \) for all large enough \( n \) but clearly this is impossible if \( 2^{n} > c \) or \( n > \log c \).
   (d) \( \log(2n) \) is \( O(\log(n)) \) since \( \log(2n) = \log(n) + 1 \leq 2\log n \) for all \( n \geq 2 \).

3. A number of solutions are possible. Most came up with something like this:

   ```java
   public List reverseList (List list)
   {
       List reversed = new List();
       while (!list.isEmpty())
       {
           reversed.addToList(list.removeFromTail());
       }
       return reversed;
   }
   ```

4. The first element is moved when the list size becomes 2, 5, 17, 65, ..., etc. The last time it is moved is when the list size becomes \( 4^{\lfloor \log_4 (m-1) \rfloor} + 1 \), i.e., it gets moved about \( \log_4 m \) times. The total number of moves is \( 1 + 4 + 16 + \ldots + 4^{\lfloor \log_4 (m-1) \rfloor} \). This is a geometric sum and is approximately equal to \( 4m/3 \). If \( m \) insertions are performed, the average number of copies per insertion is \( \frac{4m/3}{m} = 4/3 = O(1) \).
5. The simplest way to sort is to push cars onto the siding until car number 1 is pushed. At this point, pop car 1 on its way and pop the rest of the cars back to the left. Now push cars onto the siding until car 2 is pushed. Pop car 2 on its way and pop the rest back to the left. Continue this procedure for cars 3 to n. In the worst case, if the cars are in reverse order this will require n pushes and pops to get car 1, n – 1 pushes and pops to get car 2, etc., for a total of \( \sum_{i=1}^{n} i = n(n + 1)/2 \) pushes and pops. A slightly smarter routine will pop car 2 on its way if it happens to be among the cars pushed when searching for car 1, and similarly when searching for car i, pop i + 1 if possible. In the worst case this still requires \( O(n^2) \) pushes and pops but the constant is a bit smaller (1/4 instead of 1/2). It is possible to do it with \( O(n \log n) \) pushes and pops. Can you see how?