1. Let $A$ be an array of size $n \geq 2$ containing integers from 1 to $n - 1$ inclusive, one of which is repeated. Describe an algorithm for finding the integer in $A$ that is repeated.

   **Algorithm.**
   The solution is shown in Algorithm 1. The key idea is to subtract the sum of the array from the sum of numbers from 1 to $n$. This informs the number of elements $m$ in the array that are “out of place.” Meaning the number repeated is $n - m$.

   **Algorithm 1 Find repeat**
   **Input:** Array $A$ of length $n$
   **Output:** Repeated integer $x$
   1: $s \leftarrow \sum_{i=1}^{n-1} i$
   2: $m \leftarrow \sum_{a \in A}$
   3: return $n - (s - m)$

   **Note:** It is also acceptable to use extra memory. Fully sorting the data or doing a nested for loop is unacceptable for this problem. They would be slow and inefficient.

2. Describe an algorithm for concatenating two singly linked lists $L$ and $M$, into a single list $L'$ that contains all the nodes of $L$ followed by all the nodes of $M$.

   **Algorithm.**
   Algorithm 2 describes the correct process. The idea, is simply to link the tail of $L$ to the head of $M$. After, we construct $L'$ by denoting the correct head and tail of the concatenated lists. Following this step, we clear $L$ and $M$ by resetting their head and tail values to null.

   **Algorithm 2 Concatenate**
   **Input:** Singly linked lists $L$ and $M$
   **Output:** Singly linked list $L'$
   1: Singly linked list $L' \leftarrow \emptyset$
   2: $L$.tail.next $\leftarrow M$.head
   3: $L'$.head $\leftarrow L$.head
   4: $L'$.tail $\leftarrow M$.tail
   5: Set $L$'s and $M$'s heads and tails to null
   6: return $L'$
3. Describe in detail an algorithm for reversing a singly linked list \( L \) using only a constant amount of additional space.

**Algorithm.**

The algorithm shown in Algorithm 3 will reverse a singly linked list. It iterates over the list and reverses the pointers appropriately. We keep three markers, the “head” of the new list \( n \), the current element \( t \), and the “head” of the old list \( c \). In each step, we make the current element’s next field point to the new head. Then, we increment the pointers appropriately.

**Algorithm 3 Reverse**

**Input:** Singly linked list \( L \)

1: Singly linked list \( L' \leftarrow \emptyset \)
2: Node \( n \leftarrow \text{null} \); Node \( c \leftarrow L.\text{head} \)
3: \( L.\text{head} \leftarrow L.\text{head} \)
4: \( L.\text{tail} \leftarrow c \)
5: while \( c \neq \text{null} \) do
6: Node \( t \leftarrow c \)
7: \( c \leftarrow c.\text{next} \)
8: \( t.\text{next} \leftarrow n \)
9: \( n \leftarrow t \)

**Note:** Another possible solution is to construct \( L' \) by removing from \( L \) and inserting in the reversed order in \( L' \).

4. **Bonus.** Write a method, \( \text{shuffle}(A) \), that rearranges the elements of array \( A \) so that every possible ordering is equally likely. You may rely on a function \( \text{rand}(n) \), which returns a random number between 0 and \( n - 1 \) inclusive.

**Algorithm.**

Algorithm 4 proceeds as follows: in reverse, for each element of the array, pick a random element in the un-shuffled portion to swap with. The key intuition of why this algorithm yields an unbiased permutation is akin to picking elements at random from a hat. The “right” portion of the array represents the already picked elements, while the “left” is the “hat”. By using this process, each element is equally likely to be in any specific location of the array.

**Algorithm 4 Shuffle**

**Input:** Array \( A \)

1: for \( i \leftarrow n - 1 \ldots 1 \) do
2: \( j \leftarrow \text{rand}(i) \)
3: Swap \( A[j] \) and \( A[i] \)