1. Using the notation for an array implementation of a stack used in class, give pseudo code to reverse a stack $S$ of $n$ items in linear time and space. Note that you may use additional data structures, but it must be that $S$ becomes reversed and not that some other stack is returned.

Using the notation for stack and queue operations given in class, we can pop everything from the stack onto a queue and then push it all back onto the stack:

- Initialize an empty queue $Q$.
- \textbf{while} $S$ not empty \textbf{do} ENQUEUE(POP($S$),$Q$)
- \textbf{while} $Q$ not empty \textbf{do} PUSH(DEQUEUE($Q$),$S$)

**Time:** $O(1)$ time for each iteration of the two while loops, for a total of $O(n)$ time.

**Space:** At any given time, there are only $n$ elements used between $S$ and $Q$, but both are arrays of $n$ elements, for a total of $2n = O(n)$ space. Note that we could have eliminated the space used by $Q$ by reversing $S$ in place in a way similar to the next problem.
2. Using the notation for an array implementation of a queue used in class (including accessing the elements of the queue array directly), give pseudo-code to reverse a circular queue \( Q \) of \( n \) elements using \( O(n) \) time and only \( O(1) \) space in addition to the space used for \( Q \). Explain why your algorithm works correctly and analyze the asymptotic time and space used.

Using the notation from class for queue parameters and operations:

\[
x = \text{front} \\
y = \text{rear} \\
z = \text{size}
\]

while \( z > 1 \) do begin
    Exchange the elements in positions \( x \) and \( y \).
    \[
    x = (x + 1) \mod n \\
y = (y - 1) \mod n \\
z = z - 2
    \]
end

**Correctness:** If the items of \( Q \) do not wrap around the array, then each iteration reverses two additional items, and the algorithm terminates either with \( z=1 \) for an odd length list or \( z=0 \) for an even length list. Wrap around is taken care of by using modular arithmetic.

**Time:** \( O(1) \) time for each iteration of the while loop.

**Space:** In addition to the space to store \( Q \), \( O(1) \) space for the variables \( x \), \( y \), and \( z \).
3. Using the basic list operations presented in class (CREATE, FIRST, LAST, SIZE, NEXT, PREV, INSERT, DELETE, DATA, etc.), give pseudo-code to determine a maximum value in the a list $L$ of $n$ integers (i.e., return an integer equal to the maximum of any value stored in the list and leave the list unchanged). Analyze the asymptotic time and space used.

\[
p := \text{FIRST}(L)
\]
\[
m := 0
\]
\[\text{while } p \neq \text{nil do begin}
\]
\[
\quad m := \text{MAX}\{m, \text{DATA}(p)\}
\]
\[
\quad p := \text{NEXT}(p)
\]
\[\text{end}
\]

*Time:* $O(1)$ time for each element visited, for a total of $O(n)$ time.

*Space:* $O(1)$ space in addition to the space to store $L$. 
4. Using the basic list operations presented in class (CREATE, FIRST, LAST, SIZE, NEXT, PREV, INSERT, DELETE, DATA, etc.), give pseudo code to remove every other element of a list $L$ of $n$ integers and place these elements in a new list $M$ (that is, $M$ gets the 1st, 3rd, 5th, etc. elements of $L$). The elements that remain in $L$ should stay in the same relative order and the elements of $M$ should be in the same relative order as they were in $L$. For example, if the input is $L = 1 \ 2 \ 3 \ 4 \ 5$, then after completion of the algorithm, $M = 1 \ 3 \ 5$ and $L = 2 \ 4$. Analyze the asymptotic time and space used.

One method is to proceed thorough $L$ two vertices at a time, deleting one and putting the other in a new list $M$. Another way, shown here, is to keep a variable $parity$ to keep track of whether the current node should be moved to $M$ or not:

\[
\begin{align*}
M & = \text{CREATE} \\
p & = \text{FIRST}(L) \\
parity & = 0 \\
\textbf{while} & \ p \neq \text{nil} \ \textbf{do} \begin{align*}
q & = p \\
p & = \text{NEXT}(p,L) \\
\textbf{if} & \ (parity = 0) \ \textbf{then} \ \text{INSERT(DELETE}(q,L),\text{LAST}(M),M) \\
parity & = 1 - parity \\
\textbf{end}
\end{align*}
\]

**Time:** $O(1)$ time for each element visited, for a total of $O(n)$ time.

**Space:** $O(1)$ space in addition to the space to store $L$. 