2.1-4. Consider the problem of adding two \( n \)-bit binary integers, stored in two \( n \)-element arrays \( A \) and \( B \). The sum of the two integers should be stored in binary form in an \( (n+1) \)-element array \( C \). State the problem formally and write pseudocode for adding the two integers.

**Solution:**

Declaration of \( A \), \( B \) and \( C \):

\( A[0] \ldots A[n-1] \) (length = \( n \))

\( B[0] \ldots B[n-1] \) (length = \( n \))

\( C[0] \ldots C[n] \) (length = \( n+1 \))

\( A[0] \) and \( B[0] \) are the most significant bits.

Pseudocode:

\[
\text{Carry} = 0 \\
\text{For } i = n - 1 \text{ to } 0 \\
\quad C[i+1] = (A[i] + B[i] + \text{Carry}) \mod 2 \\
\quad \text{Carry} = (A[i] + B[i] + \text{Carry}) / 2 \\
\text{C}[0] = \text{Carry}
\]

2.3.6. Observe that the \textbf{while} loop of lines 5–7 of the INSERTION-SORT procedure in Section 2.1 uses a linear search to scan (backward) through the sorted subarray \( A[1…j-1] \). Can we use a binary search (see Exercise 2.3-5) instead to improve the overall worst-case running time of insertion sort to \( \Theta(n \lg n) \)?

**Solution:**

No. Although you can find the position of \( A[j] \) in \( O(\lg j) \), you need to move \( j-1 \) elements in the worst case. So, binary search cannot improve the overall worst-case running time of insertion sort to \( \Theta(n \lg n) \).