1. To show that \( n^3 + 2n^2 \) is \( O(n^3) \) we must find constants \( c > 0 \) and \( n_0 \geq 1 \) integer such that
\[
 n^3 + 2n^2 \leq cn^3, \quad \forall n \geq n_0
\]
Move \( n^3 \) to the right hand side of the above inequality to get
\[
 2n^2 \leq (c-1)n^3, \quad \forall n \geq n_0
\]
Divide both sides of the first inequality by \( n^2 \) to get
\[
 2 \leq (c-1)n, \quad \forall n \geq n_0
\]
Now we can choose \( c = 2 \) and the above inequality becomes
\[
 2 \leq n, \quad \forall n \geq n_0
\]
Finally, we can choose \( n_0 = 2 \), as
\[
 2 \leq n, \quad \forall n \geq n_0.
\]

2. To show that \( n(n + 1) \) is not \( O(n) \) we use a proof by contradiction. We will assume that \( n(n + 1) \) is \( O(n) \) and derive a contradiction from this assumption. If \( n(n + 1) \) is \( O(n) \), this means that we can find constants \( c > 0 \) and \( n_0 \geq 1 \) integer such that
\[
 n(n + 1) \leq c \times n, \quad \forall n \geq n_0.
\]
Dividing both sides by \( n \) (this can be done since \( n \geq n_0 \geq 1 \), so \( n \) is positive) we get
\[
 1 + \frac{1}{n} \leq c, \quad \forall n \geq n_0,
\]
which clearly cannot hold, since \( n \) grows without bound and so \( 1 + \frac{1}{n} \) cannot be upper-bounded by any constant \( c \). Therefore, \( n(n + 1) \) is not \( O(n) \).

3. Note that since \( f(n) \) is \( O(g(n)) \) then we know that constant values \( c' > 0 \) and \( n'_0 \geq 1 \) integer exist such that
\[
 f(n) \leq c'g(n), \quad \forall n \geq n'_0. \tag{1}
\]
Also, since \( g(n) \) is \( O(h(n)) \) we know that constant values \( c'' > 0 \) and \( n''_0 \geq 1 \) integer exist such that
\[
 g(n) \leq c''h(n), \quad \forall n \geq n''_0. \tag{2}
\]
To show that \( f(n) \) is \( O(h(n)) \), we need to find constants \( c > 0 \) and \( n_0 \geq 1 \) integer such that
\[
 f(n) \leq c \times h(n), \quad \forall n \geq n_0. \tag{3}
\]
To find these constants let us combine (1) and (2) to get
\[ f(n) \leq c'g(n) \leq (c' \times c'')h(n), \quad \forall n \geq \max\{n'_0, n''_0\}. \quad (4) \]

Note that the inequality in (1) holds for all \( n \geq n'_0 \) and the inequality in (2) holds for all \( n \geq n''_0 \), hence the inequality in (4) holds for all \( n \geq \max\{n'_0, n''_0\} \).

Comparing (4) and (3) we see that if we choose \( c = c' \times c'' \) and \( n_0 = \max\{n'_0, n''_0\} \) inequality (3) holds, and since \( c \) and \( n_0 \) are constants we have shown that \( f(n) \) is \( O(h(n)) \).

4. Algorithm AtLeastTwice\((A, n)\)

\[ \text{In: Array } A \text{ storing } n \text{ integer values} \]
\[ \text{Out: True if every value in } A \text{ appears at least twice; false otherwise} \]
\[
\begin{align*}
\text{for } i & \leftarrow 0 \text{ to } n - 1 \text{ do } \\
& \quad c \leftarrow 0 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{// counter for the number of copies of } A[i] \\
& \quad \text{for } j \leftarrow 0 \text{ to } n - 1 \text{ do } \\
& \quad \quad \text{if } A[j] = A[i] \text{ then} \\
& \quad \quad \quad c \leftarrow c + 1 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{// count the number of copies of } A[i] \\
& \quad \text{if } c < 2 \text{ then return false} \quad \quad \text{// } A[i] \text{ appears only once in } A \\
& \quad \text{return true} \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{// Every value } A[i] \text{ appears at least twice}
\end{align*}
\]

- **Termination.** The outside loop is repeated once for each value of \( i \) between 0 and \( n - 1 \), so the outside loop is repeated \( n \) times. The inside loop is repeated \( n \) times for each iteration of the outside loop. Therefore, the total number of iterations of the inside loop is at most \( n^2 \). Since each loop iterates a finite number of times, the algorithm terminates after a finite amount of time.

- **Correct output.** The first loop considers all values \( i \in \{0, 1, \ldots, n - 1\} \). For each one of these values, the value of \( c \) is set first to zero and then, as the second loop considers all values \( j \in \{0, 1, \ldots, n - 1\} \) the value of \( c \) is increased whenever a copy of \( A[i] \) is found. Hence, at the end of the second loop the value of \( c \) indicates the number of times that the value \( A[i] \) appears in the array. If \( c < 2 \), the value \( A[i] \) appears only once in \( A \) and the algorithm correctly returns the value \text{false}. On the other hand if \( c \geq 2 \) then \( A[i] \) appears at least twice and so a new iteration of the first loop is performed to test whether the next value in \( A \) also appears at least twice.

Observe that if every value appears at least twice in \( A \) then the condition of the second if statement is always false causing the algorithm to return the value \text{true} as required.

- **Worst case.** The worst case for the above algorithm is when every value appears at least twice in the array as in this case the condition of the second if statement is always false causing the first for loop to perform the maximum number of iterations.

- **Time complexity.** We analyze the inner-most for loop first. In every iteration of this loop a constant number \( k \) of operations is performed and the loop is repeated \( n \) times. Thus, the total number of operations performed by this loop is \( kn \).
As for the first for loop, in every iteration it performs a constant number $k'$ of operations to set $c$ to zero and to check the condition of the second if statement plus the $kn$ operations of the second for loop. Since the first loop is repeated $n$ times in the worst case, the total number of operations performed by the algorithm is $n(k' + kn) = k'n + kn^2$. Outside the first loop a constant number $k''$ of operations is performed, so the total number of operations performed by the algorithm is $k'' + k'n + kn^2$ is $O(n^2)$. 