Prove that $4n^2 + 3n$ is $O(n^2)$

Find constants $c > 0$, $n_0 \geq 1$ such that

$$4n^2 + 3n \leq cn^2 \quad \forall n \geq n_0$$

Move $4n^2$ to the right hand side

$$3n \leq (c-4)n^2 \quad \forall n \geq n_0$$

Divide both sides by $n$ (ok as $n > 0$)

$$3 \leq (c-4)n \quad \forall n \geq n_0$$

$c = 1$: $3 \leq 3n \quad \forall n \geq n_0$ This value for $c$ does not work!
$c = 7$: $3 \leq 3n \quad \forall n \geq n_0$ This inequality holds for all $n \geq 1$.

$c = 7$, $n_0 = 1$ make inequality true, so we have proven that $4n^2 + 3n$ is $O(n^2)$

These are not the only values for $c$ and $n_0$ that work, we could also choose

$c = 3$, $3 \leq n \quad \forall n \geq n_0$ $[n_0 = 3]$
Prove that $T_1 n + T_2$ is $O(n)$, where $T_1 > 0$, $T_2 > 0$ are constants.

Find constants $c > 0$, $n_0 \geq 1$ such that

$T_1 n + T_2 \leq c n$, $\forall n \geq n_0$

Move $T_1 n$ to the right

$T_2 \leq (c - T_1) n$, $\forall n \geq n_0$

$c = T_1 + 1$

$T_2 \leq n$ $\forall n \geq n_0 = T_2$
Prove that $4n$ is $O(n^2)$

Find constants $c > 0$, $n_0 \geq 1$ such that

$4n \leq cn^2 \quad \forall n \geq n_0$

Divide both sides by $n$ (can be done as $n > 0$)

$4 \leq cn \quad \forall n \geq n_0$

Choose, for example, $c = 4$ to get

$4 \leq 4n \quad \forall n \geq n_0$

Since the inequality holds for all $n \geq 1$, we choose $n_0 = 1$. 
Prove that \( n^2 \) is not \( O(n) \)

Proof by contradiction:

Assume that \( n^2 \) is \( O(n) \) and derive a contradiction from this assumption.

If \( n^2 \) is \( O(n) \) then there are constants \( c > 0, n_0 \geq 1 \) such that

\[
    n^2 \leq cn \quad \forall n \geq n_0
\]

Divide both sides by \( n \):

\[
    n \leq c \quad \forall n \geq n_0
\]

Since \( n \) can be arbitrarily large, it is not possible that \( n \leq c \) for all values \( n \geq n_0 \), a contradiction!
Algorithm LinearSearch (L,n,x)

\[ i \leftarrow 0 \]

\[ \text{while } (i < n) \text{ and } (L[i] \neq x) \text{ do} \]
\[ i \leftarrow i+1 \]

\[ \text{if } i=n \text{ then return } -1 \]
\[ \text{else return } i \]

\[ t \leftarrow t_< t_\neq t_\land t_+ t_\Rightarrow t_{\text{return}} \]

\[ f(n) = (n+1) t_< + (n+1) t_\neq + n t_\neq +(n+1) t_\land + n t_+ t_\Rightarrow t_{\text{return}} \]
\[ = (t_< + t_\neq + t_\land + t_+)n + \frac{t_< + t_\neq + t_\land + t_+ t_{\text{return}}}{T_1 n + T_2}, \]
\[ T_1 \text{ and } T_2 \text{ are constants whose values are implementation dependent.} \]
Algorithm LinearSearch (L,n,x)
  i ← 0
  while (i < n) and (L[i] ≠ x) do
    i ← i+1
  if i=n then return -1
  else return i

To compute the time complexity we can just count the total # operations

<table>
<thead>
<tr>
<th># operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>i=0</td>
</tr>
<tr>
<td>i=1</td>
</tr>
<tr>
<td>i=2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>i=n-1</td>
</tr>
<tr>
<td>i=n</td>
</tr>
</tbody>
</table>

f(n) = 5n + 5 is O(n)
**Algorithm LinearSearch** \((L, n, x)\)

\[
\begin{align*}
i &\leftarrow 0 \\
\text{while } (i < n) \text{ and } (L[i] \neq x) \text{ do } &\quad \begin{cases} i \leftarrow i+1 \\ \text{if } i=n \text{ then return } -1 \end{cases} \\ \text{else return } &\quad i
\end{align*}
\]

\[
f(n) = \frac{c_1 + c_2 + c_3 n}{c} = c + c_3 n \text{ is } O(n)
\]
Algorithm \text{foo}(n)

\begin{align*}
\text{C}_1 & \left\{ \begin{array}{l}
i \leftarrow 1 \\
K \leftarrow 1 \\
\text{while } i < n \text{ do } \{ \end{array} \right. \\
& \quad \{ \\
& \quad \quad \text{if } i = K \text{ then } \{ \\
& \quad \quad \quad A[i] \leftarrow K \\
& \quad \quad \quad K \leftarrow K + 1 \\
& \quad \quad \quad i \leftarrow 1 \\
& \quad \quad \} \\
& \quad \} \\
\text{C}_2 & \left\{ \begin{array}{l}
i \leftarrow i + 1 \\
\} \\
\end{array} \right. \\
\end{align*}

Let us count the number of iterations performed by the while loop:

\begin{align*}
\text{# iterations} & \\
K = 1, \ i = 1 \quad & 1 \\
K = 2, \ i = 1 \rightarrow 2 \quad & 2 \\
K = 3, \ i = 1 \rightarrow 2 \rightarrow 3 \quad & 3 \\
K = 4, \ i = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \quad & 4 \\
K = 5, \ i = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \quad & 5 \\
\vdots \\
K = n-1, \ i = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \cdots \rightarrow n-1 \rightarrow n \quad & n-1 \\
K = n, \ i = 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \cdots \rightarrow n-1 \rightarrow n \quad & n \\
\text{total} = 1+2+3+\cdots+n-1+n \\
& = \frac{1+2+3+\cdots+n-2+n-1+n}{2} \\
& = \frac{n(n+1)}{2} \\
& = \frac{n^2 + n}{2} \\
& \text{Arithmetic sum} \\
\end{align*}

The total number of iterations is \( 1+2+3+\cdots+n-1+n \) \( = \sum_{j=1}^{n} j = \frac{n(n+1)}{2} = \frac{n^2 + n}{2} \)

The time complexity of the algorithm is then \( C_1 + C_2 \times \text{# iterations} = C_1 + \frac{C_2}{2} n + \frac{C_2}{2} n^2 \)

is \( O(n^2) \)
$Kf(n)$ is $O(f(n))$ for any constant $K > 0$.

Find constants $c > 0$, $n_0 \geq 1$ such that

$$Kf(n) \leq cf(n) \quad \forall n \geq n_0 \quad (1)$$

Move $Kf(n)$ to the right:

$$0 \leq cf(n) - Kf(n) = (c - K)f(n), \quad \forall n \geq n_0$$

Hence, we can choose $c = K$ to get

$$0 \leq 0 \cdot f(n) = 0, \quad \forall n \geq n_0$$

Since this inequality holds for all $n \geq 1$, we can choose $n_0 = 1$.

As we found constant values $c = K, n_0 = 1$ for which inequality (1) holds, then $Kf(n)$ is $O(f(n))$. 

\( f(n) + g(n) \) is \( O(\max \{ f(n), g(n) \}) \):

Find constants \( c > 0, n_0 \geq 1 \) such that

\[
f(n) + g(n) \leq c \max \{ f(n), g(n) \}, \quad \forall n \geq n_0
\]  

(2)

Note that

\[
f(n) \leq \max \{ f(n), g(n) \}, \quad \forall n \geq 1
\]

\[
g(n) \leq \max \{ f(n), g(n) \}, \quad \forall n \geq 1
\]

Adding these inequalities we get

\[
f(n) + g(n) \leq 2 \max \{ f(n), g(n) \}, \quad \forall n \geq 1
\]

Hence choosing \( c = 2 \) and \( n_0 = 1 \) we prove that inequality (2) is true, so \( f(n) + g(n) \) is \( O(\max \{ f(n), g(n) \}) \)