Answer 1. The proof is by induction on $n$, the limit of the summation. For the basis case we consider the smallest legal value of $n$, namely 1. We have

$$
\sum_{i=1}^{1} i(i-1) = 0 = \frac{1(1-1)(1+1)}{3}.
$$
as desired. For the induction step, we will assume that the formula holds for all the values 1, 2, ..., $n-1$, then show that it holds for $n$. The standard method is to get rid of the last term of the sum, use the induction hypothesis to apply the formula to the the sum consisting of the first $n-1$ terms, and then add the last term back in again and simplify.

$$
\begin{align*}
\sum_{i=1}^{n} i(i-1) &= \left(\sum_{i=1}^{n-1} i(i-1)\right) + n(n-1) \\
&= \frac{(n-1)((n-1)-1)((n-1)+1)}{3} + n(n-1) \quad \text{(by ind. hyp.)} \\
&= \frac{(n-1)(n-2)n}{3} + n(n-1) = \frac{(n-2)n(n-1)+3n(n-1)}{3} \\
&= \frac{(n-2+3)n(n-1)}{3} = \frac{n(n-1)(n+1)}{3}.
\end{align*}
$$
as desired.

Answer 2.

(a) True. Since $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$, it follows from the definition that there exist constants $c_1, c_2 > 0$ and positive integers $n_1, n_2$ such that $T_1(n) \leq c_1 f(n)$ for $n \geq n_1$ and $T_2(n) \leq c_2 f(n)$ for $n \geq n_2$. This implies that, $T_1(n) + T_2(n) \leq (c_1 + c_2)f(n)$ for $n \geq \max(n_1, n_2)$. Thus, $T_1(n) + T_2(n) = O(f(n))$.

(b) False. For a counterexample to the claim, let $T_1(n) = n^2, T_2(n) = n, f(n) = n^2$. Then $T_1(n) = O(f(n))$ and $T_2(n) = O(f(n))$ but $\frac{T_1(n)}{T_2(n)} = n \neq O(1)$

(c) False. We can use the same counterexample as in part (b). Note that $T_1(n) \neq O(T_2(n))$

Answer 3.

<table>
<thead>
<tr>
<th></th>
<th>(A)</th>
<th>Relation:</th>
<th>(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(n^3 + n \log n)</td>
<td>(\Omega, \Theta, O)</td>
<td>(n^3 + n^2 \log n)</td>
</tr>
<tr>
<td>(b)</td>
<td>(\log \sqrt{n})</td>
<td>(\Omega)</td>
<td>(\sqrt{\log n})</td>
</tr>
<tr>
<td>(c)</td>
<td>(n \log_3 n)</td>
<td>(\Omega, \Theta, O)</td>
<td>(n \log_4 n)</td>
</tr>
<tr>
<td>(d)</td>
<td>(2^n)</td>
<td>(\Omega)</td>
<td>(2^{n/2})</td>
</tr>
<tr>
<td>(e)</td>
<td>(\log(2^n))</td>
<td>(\Omega, \Theta, O)</td>
<td>(\log(3^n))</td>
</tr>
</tbody>
</table>
Notes:

(a) Both are $\Theta(n^3)$, the lower order terms can be ignored. Note that if $A(n) = \Theta(B(n))$, then automatically $A(n) = O(B(n))$ and $A(n) = \Omega(B(n))$.
(b) After simplifying, $A$ is $(1/2)\lg n$, and $B$ is $\sqrt{\lg n}$. Substituting $m = \lg n$, we can see ratio $A/B$ grows as $m/2\sqrt{m} = \sqrt{m}/2$ which tends to infinity as $n$ (and hence $m$) tends to infinity.
(c) Log base conversion only introduces a constant factor.
(d) The ratio is $2^n/2^{n/2} = (2)^{n/2}$ which goes to infinity in the limit.
(e) After simplifying these are $n\lg 2$ and $n\lg 3$, both of which are $\Theta(n)$.

Answer 4.

(a) $T(n) = O(n)$.
(b) $T(n) = O(\log n)$
(c) $T(n) = O(n)$.
(d) $T(n) = O(n)$.
(e) $T(n) = O(n\log n)$.
(f) $T(n) = O(n^2)$.

Answer 5.

The recurrence for the number of comparisons is:

\[
T(1) = 0 \\
T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + n - 1.
\]

(Note that if you use the following recurrence for the running time: $T(1) = 1; T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + n$, you will obtain slightly different results.)

(a) Recursion tree for merge sort ($n = 13$):

```
               13
              /  \  \\
             6  7  10
         /  \   / \   /
        3  3  5  5  4
  / \  / \ / \ / \ / \ /  \\
 1  1 1 1 2 2 1 1 2 2 2 2 2
```

```
LEVEL

0
1
2
3
4
```
(b) There are 5 levels in the recursion tree.
(c) Number of comparisons at levels 0, 1, 2 and 3 are 12, 11, 9 and 5, respectively.
(d) The total number of comparisons is 37.
(e) For general n, the number of levels is 1 + log n, the number of comparisons at each level is $O(n)$, and the total number of comparisons is $O(n \log n)$.

Answer 6. For any value of $n$, $\max(f(n), g(n))$ is either equal to $f(n)$ or equal to $g(n)$. Therefore, for all $n$,
$$\max(f(n), g(n)) \leq f(n) + g(n).$$
Using $c = 1$ and $n_0 = 1$ in the big-oh definition, it follows that
$$\max(f(n), g(n)) = O(f(n) + g(n)).$$
Also, for all $n$,
$$\max(f(n), g(n)) \geq f(n)$$
and
$$\max(f(n), g(n)) \geq g(n).$$
Adding we have
$$2 \times \max(f(n), g(n)) \geq f(n) + g(n).$$
Therefore,
$$\max(f(n), g(n)) \geq \frac{1}{2}(f(n) + g(n))$$
Using $c = 1/2$ and $n_0 = 1$ in the Omega definition, it follows that
$$\max(f(n), g(n)) = \Omega(f(n) + g(n))$$
Since $\max(f(n), g(n)) = O(f(n) + g(n))$ and $\max(f(n), g(n)) = \Omega(f(n) + g(n))$, it implies that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$. 
(a) It computes the Fibonacci numbers, which are defined by the following recurrence relation:

\[
\begin{align*}
F(0) &= F(1) = 1 \\
F(n) &= F(n - 1) + F(n - 2) & \text{if } n > 1
\end{align*}
\]

(b) 

The recursion tree is shown in the figure. It is easy to see that \texttt{unknown[i]} is executed once for \(i = 5\), twice for \(i = 4\), three times for \(i = 3\), five times for \(i = 2\), eight times for \(i = 1\), and five times for \(i = 0\).

(c) 12 additions are performed to compute \texttt{unknown(6)}.

(d) Let \(T(n)\) denote the time taken to compute \texttt{unknown(n)}. Then the recurrence relation for \(T(n)\) is:

\[
\begin{align*}
T(0) &= T(1) = 1 \\
T(n) &= T(n - 1) + T(n - 2) + 1 & \text{if } n > 1
\end{align*}
\]

(e) We claim that \(T(n) \geq c(1.5)^n\) for some constant \(c\). Without knowing what \(c\) is, we proceed with the proof by induction. For the basis case, we need to check for both \(n = 0\) and \(n = 1\). Note that \(T(0) = 1 \geq c \cdot (1.5)^0\), for \(c \leq 1\), and \(T(1) = 1 \geq c \cdot (1.5)^1\), for \(c \leq 2/3\). So let us choose \(c = 2/3\). For the induction step, we assume the induction hypothesis that for all \(0 \leq k < n\), \(T(k) \geq c(1.5)^k\), and then we show that the \(T(n) \geq c(1.5)^n\). If we apply the definition of \(T\) and the induction hypothesis and simplify we get:
\[ T(n) = T(n-1) + T(n-2) + 1 \geq \frac{2}{3}(1.5)^{n-1} + \frac{2}{3}(1.5)^{n-2} + 1 \]
\[ \geq \frac{2}{3}(1.5)^{n-2}(1.5 + 1) + 1 \]
\[ \geq \frac{2}{3}(1.5)^{n-2}(2.5) + 1 \]
\[ \geq \frac{2}{3}(1.5)^{n-2}(1.5)^2 + 1 \]
\[ \geq \frac{2}{3}(1.5)^n + 1 \]
\[ \geq \frac{2}{3}(1.5)^n, \]

which completes the induction proof. It follows that \( T(n) = \Omega(1.5^n) \).

(f) Note that the recurrence given for \( T(n) \) also applies to the number of additions. Hence the number of additions performed to compute \texttt{unknown}(100)
\[ \geq \frac{2}{3}(1.5)^{100}. \]
Since the computer can perform a million additions each second, it takes \( \geq \frac{2}{3}(1.5)^{100}/10^6 \) seconds. This simplifies to \( \geq (2.71)10^{11} \)
seconds or more than 86 centuries.

(g) \texttt{float unknown(int n)}
\{
    \texttt{F[0] = F[1] = 0;}
    \texttt{for i = 2 to n \{}
        \texttt{F[n] = F[n-1] + F[n-2];}
    \texttt{\}}
    \texttt{return(F[n]);}
\}

This program takes \( O(n) \) time to compute \texttt{unknown}(n). In the recursive program, the same values are computed \textit{repeatedly} (see part (b)). But in the new program, we do not compute the same values again and again; instead each value \( F[i] \) is computed \textit{exactly once} and \textit{stored} for future reference.