1. The run time of the algorithm is
\[ \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} = \sum_{i=1}^{n} \sum_{j=1}^{i} \frac{i(j+1)}{2} = \frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} = \Theta(n^3). \]

2. (a) Let \( f(n) = 14n^3 \log n + 5n^2 \) and \( g(n) = n^3 \log n \). We have to show that \( f(n) = O(g(n)) \) and that \( f(n) = \Omega(g(n)) \). To show that \( f(n) = O(g(n)) \) we have to find constants \( c \) and \( n_0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \). For example, choose \( c = 19 \) and \( n_0 = 2 \). Clearly, \( 14n^3 \log n + 5n^2 \leq 19n^3 \log n \) for all \( n \geq 2 \). Thus, \( f(n) = O(g(n)) \).

(b) \( \log n! = \sum_{i=1}^{n} \log i \). We can both upper bound and lower bound this summation with the integral \( \int \log x \ dx = x \ln x \) to get the desired result. Alternatively, one could also use Stirling’s approximation for \( n! \).

(c) Let \( f(n) = (\sqrt{\pi})^{\sqrt{n}} \) and \( g(n) = 2^{6.5} \). Note that \( f(n) = 2^{(\sqrt{\pi} \log n)/2} \). Thus,
\[
\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{1}{2^{n^{0.5}-(\sqrt{\pi} \log n)/2}} = 0.
\]
As a result, \( f(n) = o(g(n)) \).

3. (a) This statement is not a theorem. As an example, let \( f(n) = 5n \) and \( g(n) = n \). In this case, \( f(n) = O(g(n)) \) but \( 2f(n) \neq O(2g(n)) \).

(b) This statement is always true (for non-negative functions \( f() \) and \( g() \)). Let \( F(n) = \max\{f(n), g(n)\} \) and \( G(n) = f(n) + g(n) \). Clearly, \( F(n) \leq G(n) \) for all \( n \geq 1 \). Thus, \( F(n) = O(G(n)) \). Also, \( F(n) \geq \frac{1}{2} G(n) \) for all \( n \geq 1 \). Therefore, \( F(n) = \Omega(G(n)) \). In summary, \( F(n) = \Theta(G(n)) \).

(c) Assume that \( b \) is an integer. Using the binomial theorem, \((n+a)^b = \sum_{i=0}^{b} \binom{b}{i} n^i a^{b-i} \).
The leading term in this polynomial is \( n^b \). Thus \((n+a)^b = \Theta(n^b) \).

4. A simple algorithm for this problem could work as follows. For every element \( z \) in the array look at the other elements in the array and check if one of them is \( u - z \). This algorithm takes \( \Theta(n^2) \) time.

An \( O(n \log n) \) algorithm can be devised as follows. To begin with, sort the array in \( O(n \log n) \) time (using heap sort, for example). Now, for every element \( z \) in the array check if the array has \( u - z \) as another element. Use binary search here. For every element of the array, searching takes \( O(\log n) \) time. Thus the run time of the entire algorithm is \( O(n \log n) \).
5. An algorithm similar to the one given in class can be conceived of for this problem:

**Algorithm** FindTwoRepeatedElements(a, n):

**Repeat**
  - Pick a random $i$ in the range $[1, n]$;
  - Pick a random $j$ in the range $[1, n]$;
  - If $i \neq j$ and $a[i] = a[j]$ then
    - FirstElement := $a[i]$;  Output $a[i]$;  Quit;
  
**Forever**

**Repeat**
  - Pick a random $i$ in the range $[1, n]$;
  - Pick a random $j$ in the range $[1, n]$;
  - If $i \neq j$ and $a[i] = a[j]$ and $a[i] \neq$ FirstElement then
    - Output $a[i]$;  Quit;
  
**Forever**

**Analysis:** We can compute the run times of the two Repeat loops separately. Consider the first loop. Probability of success in one basic step is $\frac{(n/2)(n/4-1)}{n^2}$ which is $\geq \frac{1}{10}$ for all $n \geq 20$. Therefore, the probability of failure in one basic step is no more than $\frac{9}{10}$. As a result, the probability of failure in the first $k$ basic steps is $\leq (9/10)^k$. We want this probability to be $\leq n^{-\alpha}$. This happens for $k \geq c \frac{\alpha \log n}{\log(10/9)}$. Thus, the run time of the first Repeat loop is $\tilde{O}(\log n)$.

In the second Repeat loop, the probability of success in one basic step is $\frac{(n/4)(n/4-1)}{n^2}$ which is $\geq \frac{1}{20}$ for all $n \geq 20$. Proceeding along the same lines, we see that the run time of the second loop is also $\tilde{O}(\log n)$.

In summary, the run time of the algorithm is $\tilde{O}(\log n)$. 