1. (a) TRUE. Since \( f(n) = O(g(n)) \) there exist constants \( c_1 \) and \( n_1 \) such that \( f(n) \leq c_1g(n) \) for all \( n \geq n_1 \). Similarly, there exist constants \( c_2 \) and \( n_2 \) such that \( g(n) \leq c_2h(n) \) for all \( n \geq n_2 \). As a result, it follows that \( f(n) \leq c_1c_2h(n) \) for all \( n \geq n_0 \) (where \( n_0 = \max\{n_1, n_2\} \)). Thus we conclude that \( f(n) = O(h(n)) \).

(b) FALSE. If \( n^3 = O\left(n^{2\sqrt{\log n}}\right) \), then it will imply that \( n = O\left(2^{\sqrt{\log n}}\right) \), i.e., \( 2^{\log n} = O\left(2^{\sqrt{\log n}}\right) \), which is not true since \( \sqrt{\log n} = o(\log n) \).

2. Consider the following algorithm:

\[
\text{repeat}
\]
\[
\quad \text{Pick a random } j \in [1, n];
\]
\[
\quad \text{if } A[j] < 2 \text{ then output "Type I" and quit;}
\]
\[
\quad \text{if } A[j] > 4 \text{ then output: "Type II" and quit;}
\]
\[
\text{forever}
\]

\textbf{Analysis:} Consider the case of \( A \) being of type I. The probability that \( A[j] = 1 \) on a randomly picked \( j \) is \( \frac{1}{3} \). Thus the probability of quitting in any execution of the repeat loop is \( \frac{1}{3} \). Therefore, the probability of failure in any execution of the repeat loop is \( \frac{2}{3} \). As a result, the probability of failure in the first \( k \) iterations of the repeat loop is \( \left(\frac{2}{3}\right)^k \). We want this probability to be no more than \( n^{-\alpha} \). This happens when \( k \geq \log_{3/2} n \). This implies that the run time of this algorithm is \( \tilde{O}(\log n) \), if the array is of type I. A similar analysis holds when the array is of type II.

3. Keep two 2-3 trees \( N \) and \( H \). In \( N \) store all the records with the name as the key for each record and in \( H \) store all the records with the HuskyOne ID as the key for each record. To process \textbf{Find Name}(HID), we search for a record whose key is \( HID \) in the tree \( H \). The name in this record will be output. The run time is \( O(\log n) \). We process \textbf{Find HID}(Name) in a similar manner.

4. We keep four variables: \textit{average}, \textit{min}, \textit{max}, and \( n \). We use \( n \) to store the number of elements, \textit{min} to store the minimum, \textit{max} to store the maximum, and \textit{average} to store the average of all the elements that have been inserted into the data structure thus far. We process the four operations as follows.
1) **Insert**($x$): $n := n + 1$; if $\min > x$ then $\min := x$; if $\max < x$ then $\max := x$; 
average := $\frac{\text{average} (n-1) + x}{n}$;

2) **FindMin()**: return $\min$;

3) **FindMax()**: return $\max$; and

4) **FindMean()**: return average.

Clearly, each operation takes $O(1)$ time.

5. (a) Here $a = 27$, $b = 3$, and $f(n) = n^4$. $n^{\log_{27}3} = n^3$ and hence the regularity condition holds. As a result, case 3 of Master theorem holds. Therefore, $T(n) = \Theta(n^4)$.

(b) We can use repeated substitutions here. $T(n) = T(n-1) + \log_e n = T(n-2) + \log_e(n-1) + \log_e n = T(n-3) + \log_e(n-2) + \log_e(n-1) + \log_e n$. Continuing this, it follows that $T(n) = 1 + \sum_{i=3}^{n} i \log_e i$. We can approximate this summation with the integral $\int_{3}^{n} i \log_e i \, di = (i \log_e i - i)\big|_{3}^{n} = \Theta(n \log n)$. Therefore, $T(n) = \Theta(n \log n)$.

6. Let $X_i = \{k_{i1}^i, k_{i2}^i, \ldots, k_{n/\log n}^i\}$, for $1 \leq i \leq \log n$. We replace each $k_j^i$ with a pair $(k_j^i, i)$, for $1 \leq i \leq \log n$ and $1 \leq j \leq \frac{n}{\log n}$ and form a sequence $Y$ of these new keys. Specifically,

$$Y = (k_1^1, 1), (k_2^1, 1), \ldots, (k_{n/\log n}^1, 1),$$

$$ (k_1^2, 2), (k_2^2, 2), \ldots, (k_{n/\log n}^2, 2),$$

$$ \ldots,$$

$$ (k_1^{\log n}, \log n), (k_2^{\log n}, \log n), \ldots, (k_{n/\log n}^{\log n}, \log n).$$

We now sort $Y$. In fact we only have to sort this sequence with respect to the first entries of the pairs. Let $Y'$ be the sorted sequence. We scan through this sorted sequence comparing one element with the next in the sequence. If we find two successive pairs $(k, i)$ and $(k, j)$, for some $k$, then we output $i$ and $j$. The total run time is $O(n \log n)$. 

2