1. (a) \( n^n = 2^{n \log n} \cdot 2^{2(\log n)^{1.5}} = 2^{n \log \sqrt{n}} \). Since \( n \log n = o(n \sqrt{\log n}) \), it follows that \( n^n = O(2^{n \log \sqrt{n}}) \).

(b) This statement need not always be true. As an example, if \( f(n) = n^4 + n^2 \) and \( g(n) = n^4 + n \), \( f(n) - g(n) = n^2 - n \neq \Theta(n^4) \).

2. Consider the following Monte Carlo algorithm:

0. \( i = 0 \);
1. Pick an element \( x \in A \) at random. \( i = i + 1 \);
2. Use binary search to look for \( x \) in \( B \);
3. If \( x \in B \) then output Type II and stop;
4. If \( x \notin B \) and \( i < \alpha n^{1/3} \log n \) then goto 1; else output Type I and stop.

If \( A \) and \( B \) are of Type I then the algorithm above always outputs the correct answer.
If \( A \) and \( B \) are of Type II then the probability of failure during steps 1-2-3 is \( 1 - \frac{n^{2/3}}{n} \).
Probability of failure after executing steps 1-2-3 \( n^{1/3} \alpha \log n \) times is \( (1 - \frac{n^{2/3}}{n})^{\alpha n^{1/3} \log n} \leq n^{-\alpha} \).

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

4. (a) We can use the master theorem here. \( a = 64, b = 4, f(n) = n^3 \). Also, \( n^{\log_b a} = n^3 \).
Case 2 applies. Thus, \( T(n) = \Theta(n^3 \log n) \).

(b) \[ T(n) = T(n^{1/3}) + \log n = T(n^{1/9}) + \log n^{1/3} + \log n = \ldots = \sum_{k=0}^{i-1} \frac{1}{3^k} \log n \]
where \( i = \frac{\log \log n}{\log \frac{3}{2}} \). Therefore \( T(n) = \log n (\sum_{k=0}^{i-1} \frac{1}{3^k}) = \log n^{2/3} \left( \frac{1}{3} - \frac{1}{\log n} \right) = \Theta(\log n) \).

5. Let the array be \( a[1 : n] \). Invoke \( \text{FindTransition}(a, 1, n) \).

\( \text{FindTransition}(a, l, r) \)
(1) $m = \lceil (l + r)/2 \rceil$.
(2) if $a[m - 1] < a[m]$ and $a[m] > a[m + 1]$ return $m$;
(3) if $a[m - 1] < a[m] < a[m + 1]$ return $\text{FindTransition}(a, m + 1, r)$;
(4) if $a[m - 1] > a[m] > a[m + 1]$ return $\text{FindTransition}(a, l, m - 1)$;

If $T(n)$ is the run time of the algorithm on any input of size $n$, then we have: $T(n) \leq T(n/2) + O(1)$ which solves to: $T(n) = O(\log n)$.

6. Here is an algorithm:

**Step 1.** Sort $R$ to get $R'$;
**Step 2.** Merge $S_1, S_2, \ldots, S_n$ together to get $S'$;
**Step 3.** Sort all the elements of $T_1, T_2, \ldots, T_n$ together to get $T'$;
**Step 4.** Finally, merge $R'$, $S'$, and $T'$ to get one sorted sequence.

We will show that each of the steps 1 to 3 can be done in $O(n\sqrt{\log n})$ time. Since each of the sequences $R'$, $S'$, and $T'$ has < $n$ elements, they can be merged in $O(n)$ time.

**Step 1.** Since $R$ has < $n$ elements and there are only $2\sqrt{\log n}$ distinct elements in it, using a result from Homework 2, $R$ can be sorted in $O(n\sqrt{\log n})$ time.

**Step 2.** A result from Homework 2 states that we can merge $\ell$ sorted sequences in $O(n \log \ell)$ time if there are a total of $n$ elements in the $\ell$ sequences together. If we apply this fact for $S_1, S_2, \ldots, S_n$, it follows that they can be merged in $O(n\sqrt{\log n})$ time.

**Step 3.** Note that the total number of elements in the sequences $T_1, T_2, \ldots, T_n$, together, is $O(n/ \log n)$. Thus we can put all of these elements together and sort them in $O(n)$ time (e.g., using merge sort).