Question I

Part (a): Comparing the given recurrence with the template for the Master Theorem, we note that $a = 1$, $b = 4/3$ and $f(n) = 7n$.

Now, $\log_b a = \log_{4/3} 1 = 0$. Thus, $f(n) = \Omega(n^{0+\epsilon})$, where $\epsilon = 1 > 0$. Therefore, Part III of Master Theorem can be used, provided the regularity condition holds.

Now, $a f(n/b) = f(3n/4) = (3/4) \times f(n)$ for all $n \geq 0$. Thus, the regularity condition holds with $c = 3/4$ and the solution to the recurrence is $T(n) = \Theta(f(n)) = \Theta(n)$.  

Part (b): Comparing the given recurrence with the template for the Master Theorem, we note that $b = 3$ and $f(n) = 2n^2$. Note that $n \log_b a = n \log_3 a$.

Now, if $a > 9$, then $\log_3 a > 2$ and so $n \log_3 a$ would be polynomially larger than $n^2$. Hence, the solution to the recurrence won’t be $O(n^2)$. (It will be $\Theta(n^{\log_3 a})$.) If $a = 9$, then $\log_3 a = 2$ and so $n \log_3 a = \Theta(n^2)$. In that case, the solution to the recurrence would be $O(n^2 \log n)$. Therefore, the largest integer value of $a$ for which the solution remains $O(n^2)$ is 8.

Question II

Part (a): Let $x_i$ be the indicator random variable associated with element $i$, $1 \leq i \leq n$. Thus, $x_i = 1$ if element $i$ is selected and $x_i = 0$ otherwise. Thus, the random variable $X$, which represents the number of elements in $S'$, is given by $X = \sum_{i=1}^{n} x_i$. We need to compute $E[X]$. By linearity of expectation $E[X] = \sum_{i=1}^{n} E[x_i]$.

Now, for any $i$, $E[x_i] = \Pr\{x_i = 1\} = 1/2$, because we use a fair coin to select $i$. Therefore, $E[X] = n/2$.

Part (b): Consider any two values $x$ and $y$ in the array. For any integer $i$, $1 \leq i \leq n^4$, the probability that each of these two values is equal to $i$ is $1/n^4 \times 1/n^4 = 1/n^8$. Since $i$ can take on $n^4$ different values, the probability that $x$ and $y$ are equal is $n^4 \times 1/n^8 = 1/n^4$.

There are $n(n-1)/2$ pairs of values. Thus, by the Union Bound, the probability that any pair of values in $A$ is equal is at most $n(n-1)/2 \times 1/n^4$ which is less than $1/(2n^2)$.

Therefore, the probability that the values in $A$ are all distinct is at least $1 - \frac{1}{2n^2}$.  

Question III

Part (a): The idea is that the second largest value must be stored in one of the two children of the root. Since $n \geq 3$, both of these children exist and their indices are 2 and 3.

Pseudocode for $\text{Second-Max}(A)$:


Since the function involves just one comparison followed by a return statement, the running time is \( O(1) \).

**Part (b):** This problem can be solved in two ways.

**Solution I:** The idea here is that we can do two \textsc{Extract-Max} operations and then insert the first extracted value back into the heap.

**Pseudocode** for \textsc{Extract-Second-Max}(A):

1. Let \( x = \text{Heap-Extract-Max}(A) \).
2. Let \( y = \text{Heap-Extract-Max}(A) \).
3. \textsc{Max-Heap-Insert}(A, x).
4. return \( y \).

Each of the steps 1, 2 and 3 above take \( O(\log n) \) time while Step 4 takes \( O(1) \) time. Therefore, the above algorithm runs in \( O(\log n) \) time.

**Solution II:** As noted in the answer to Part (a), the second largest value must be stored in one of the two children of the root. Using this fact, we can employ an approach similar to that used for \textsc{Heap-Extract-Max} as shown below.

**Pseudocode** for \textsc{Extract-Second-Max}(A):

2. Exchange \( A[j] \) and \( A[\text{heap-size}[A]] \).
4. \textsc{Max-Heapify}(A, j).

Each of the steps 1, 2 and 3 above takes \( O(1) \) time while Step 4 takes \( O(\log n) \) time. Therefore, the above algorithm runs in \( O(\log n) \) time.

---

**Question IV**

**Part (a):** The value returned by the call to \textsc{Partition} is 1. After the call, \( A[1] \) contains \( x_n \), \( A[n] \) contains \( x_1 \) and for \( 2 \leq i \leq n - 1 \), \( A[i] \) contains \( x_i \).

**Reason:** In the given call to \textsc{Partition}, the pivot value is \( x_n \). Since \( x_n \) is the smallest value in the array, no exchanges are done in the loop shown in Step 2 of \textsc{Partition}. So, \( i \) remains 0 at the end of that loop. When the loop ends, values \( A[1] \) and \( A[n] \) are exchanged and the value returned is \( i + 1 = 1 \).

**Part (b):** The basic idea is that in each call to \textsc{Partition}, we would use the middle element of the subarray as the pivot. This would ensure that after the call to \textsc{Partition}, each of the two sides will have at most \( \lceil n/2 \rceil \) elements.

Using \textsc{Partition}, we construct a function \textsc{New-Partition} as follows. The \textsc{QuickSort} algorithm would use \textsc{New-Partition} instead of \textsc{Partition}.
New-Partition \((A, p, r)\)

1. Use Algorithm \(A\) to find the index \(j\) of a middle element of \(A[p .. r]\).
2. Exchange \(A[j]\) and \(A[r]\).
3. return \(\text{Partition}(A, p, r)\).

New-Partition runs in \(O(n)\) time (where \(n\) is the size of the subarray \(A[p .. r]\)). This is because the \(A\) runs in \(O(n)\) time (given), Step 2 takes \(O(1)\) time and Step 3 (call to \(\text{Partition}\)) uses \(O(n)\) time.

Since the middle element is used as the pivot, each of the two sides after a call to New-Partition would contain at most \([n/2]\) elements. Therefore, if \(T(n)\) denotes the running time of the modified version of Quicksort, the recurrence for \(T(n)\) is given by

\[
T(n) \leq 2T([n/2]) + cn, \quad n \geq 2
\]

with \(T(1) = c_1\), where \(c\) and \(c_1\) are constants.

Comparing the above recurrence with the template of the Master Theorem, we note that \(a = b = 2\) and \(f(n) = cn\). Thus, \(n^{\log_b a} = n^1 = \Theta(f(n))\). Hence, Part II of Master Theorem applies and \(T(n) = O(n \log n)\).

**Question V:**

Suppose each internal node of the decision tree has 3 children. Let \(h\) be the height of the decision tree. We know that in the worst-case, the number of comparisons needed is at least \(h\).

Note that the decision tree must have \(n!\) leaves (one corresponding to each possible permutation of the \(n\) input values). When each internal node has 3 children, the maximum number of leaves in the decision tree of height \(h\) is \(3^h\). Therefore, \(3^h \geq n!\) or \(h \geq \log_3(n!)\). Since \(n! > (n/e)^n\), we have \(\log_3(n!) > n(\log_3 n - \log_3 e)\). Since \(\log_3 e < 1\) is a constant, we notice that \(\log_3(n!) = \Omega(n \log_3 n)\). Thus, \(h \geq \log_3(n!) = \Omega(n \log_3 n)\). In other words, the number of comparisons in the worst-case is \(\Omega(n \log n)\).