One possible Monte Carlo algorithm will be to randomly select $k$ elements (for some relevant $k$) of $A$, find the median of this random sample and output this median. Clearly, the run time of this algorithm will be $O(k)$.

Let $S$ be the random sample. The above algorithm will give an incorrect answer if $\geq \frac{k}{2}$ elements of $S$ have ranks either greater than $\left(\frac{1}{2} + \delta\right)n$ or less than $\left(\frac{1}{2} - \delta\right)n$. We can use Chernoff bounds to calculate the probability of each of these possibilities.

Let $X$ stand for the number of sample elements whose ranks are $\geq \left(\frac{1}{2} + \delta\right)n$. Note that $X$ is $B(k, \frac{1}{2} - \delta)$. As a result, $\text{Prob.}[X \geq \frac{k}{2}] \leq \exp\left(-\frac{4\delta^2 k (\frac{1}{2} - \delta)}{3(1 - 2\delta)^2}\right) = \exp\left(-\frac{2\delta^2 k}{3(1-2\delta)}\right) \leq \exp\left(-\frac{2\delta^2 k}{3}\right)$.

Along the same lines, the probability of the second possibility is $\leq \exp\left(-\frac{2\delta^2 k}{3}\right)$. The probability of either possibility is $\leq 2 \exp\left(-\frac{2\delta^2 k}{3}\right)$. We want this probability to be $\leq n^{-\alpha}$. This will happen when $k \geq \frac{3(1+\alpha \log n)}{2\delta^2}$.

In summary, the run time of the above Monte Carlo algorithm is $O\left(\frac{\log n}{\delta^2}\right)$.

2. The data-structures used here are 3 separate balanced trees (such as 2-3 trees). The keys used in the first, second, and third trees are $CID$, $A$, and $t$, respectively. A customer is represented in each tree with a node. Each node in the first tree contains a triple $(CID, A, t)$. This node also contains pointers to $CID$’s nodes in the second and third trees. Each node in the second tree has a pair $(CID, A)$ and each node in the third tree has a pair $(CID, t)$.

ProcessTransaction$(CID, A, t)$ would first check if $CID$ is present in the first tree. If present the current values of $A$ and $t$ are updated. $CID$’s nodes in the second and third trees are updated as well. This will involve deleting and reinserting them with new key values. After these updates, new pointers are stored in the first tree. Since we perform only a constant number of operations, the run time is $O(\log n)$.

BestCustomer() This operation can be performed with a search for the maximum 100 keys in the second tree. In a 2-3 tree, for example, these are the 100 rightmost leaves. Run time is $O(\log n)$.

DeleteInactive$(t)$ Search for the smallest key $t'$ in the third tree. If $t'$ is less than $t$, then we delete this key. If $t'$ is deleted, then we also delete the corresponding nodes in the first and second trees.
We repeat the above process until the smallest key in the third tree is \( \geq t \). For each node deleted we spend \( O(\log n) \) time. Thus the total run time is \( O(q \log n) \), where \( q \) is the number of nodes deleted.

3. Create a singleton set for every person.

   for each \((i, j)\) in \( R \) do
   
   if \( \text{Find}(i) \neq \text{Find}(j) \) then
   
   \( \text{Union}(i, j) \)
   
   Then run the following:
   
   if \( \text{Find}(p_1) = \text{Find}(p_2) \) then
   
   Output “\( p_1 \) and \( p_2 \) are related”
   
   else

   Output “\( p_1 \) and \( p_2 \) are not related”

   With \( \text{CollapsingFind} \) and \( \text{WightedUnion} \), the run time will be \( O((m+n)\alpha(m+n)) \), where \( \alpha(.) \) is the inverse Ackermann’s function.

4. \( A_1 \cap A_2 \cap \ldots \cap A_m = ((A_1 \cap A_2) \cap A_3) \ldots A_n). \)

   \( A_1 \cap A_2 \) can be computed by merging the two lists. (Note that the lists are in sorted order.)

   The computation time for \( (A_1 \cap A_2) \cap A_3 \) is \( |A_1| + |A_2| + (\text{Min}(|A_1|, |A_2|) + |A_3|) \).

   \( (A_1 \cap A_2) \cap A_3 \ldots A_m \) can be computed in time

   \[
   \frac{|A_1| + |A_2| + \text{Min}(|A_1|, |A_2|) + |A_3| + \ldots + \text{Min}(|A_1|, |A_2|, \ldots, |A_{m-1}|) + |A_m|}{2} \leq 2 \times (|A_1| + |A_2| + |A_3| + \ldots + |A_m|) = O(n)
   \]

5. Let \( A(n) \) be the expected run time of quickselect. The pivot \( k \) can be any element of \( X \) all
   with an equal probability of \( 1/n \). Let \( \pi_1, \pi_2, \ldots, \pi_n \) be the sorted order of \( X \). If \( k = \pi_1 \), the
   expected run time is \( n + A(n - 1) \). If \( k = \pi_2 \), the expected run time is \( n + A(n - 2) \); \ldots; If
   \( k = \pi_{i-1} \), the expected run time is \( n + A(n - i + 1) \). If \( k = \pi_i \), the expected run time is \( n \).
   Also, if \( k = \pi_{i+1} \), the expected run time is \( n + A(i) \); if \( k = \pi_{i+2} \), the expected run time is
   \( n + A(i + 1) \); \ldots; if \( k = \pi_n \), the expected run time is \( n + A(n - 1) \). Putting these together,
   we realize that

   \[
   A(n) = \frac{1}{n} [A(n - 1) + A(n - 2) + \cdots + A(n - i + 1) + A(i) + A(i + 1) + \cdots + A(n - 1)] + n.
   \]
We can now use induction to show that $A(n) \leq cn$ for some constant $c$. Assume the hypothesis for all inputs of size up to $n - 1$. We’ll prove it for an input of size $n$. Using the induction hypothesis we see that:

$$A(n) \leq \frac{c}{n} \left[ \frac{n(n-1)}{2} - \frac{(n-i)(n-i+1)}{2} + \frac{n(n-1)}{2} - \frac{i(i-1)}{2} \right] + n.$$

Upon simplification,

$$A(n) \leq \frac{c}{n} \left[ \frac{n^2}{2} - \frac{3}{2}n^2 - i^2 + ni + i \right] + n.$$

The value of $ni + i - i^2$ is maximum when $i = \frac{n+1}{2}$. As a result, it follows that

$$A(n) \leq \frac{c}{n} \left[ \frac{3}{4}n^2 - n + \frac{1}{4} \right] + n = \frac{3}{4}cn - c + \frac{c}{4n} + n.$$

$\frac{3}{4}cn - c + \frac{c}{4n} + n$ will be $\leq cn$ if $c \geq 4$.

In summary, $T(n) \leq 4n$.

6. Matrices $A$ and $B$ could be divided into $k$ submatrices each of size $n \times n$, $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$ respectively. Then $AB =$

$$\begin{pmatrix}
  A_1 \\
  A_2 \\
  \vdots \\
  A_k \\
\end{pmatrix}
\begin{pmatrix}
  B_1 & B_2 & \ldots & B_k \\
\end{pmatrix}
= 
\begin{pmatrix}
  A_1B_1 & A_1B_2 & \ldots & A_1B_k \\
  A_2B_1 & A_2B_2 & \ldots & A_2B_k \\
  \vdots & \vdots & \ldots & \vdots \\
  A_kB_1 & A_kB_2 & \ldots & A_kB_k \\
\end{pmatrix}$$

Thus $AB$ could be found by computing $A_iB_j, 1 \leq i, j \leq k$. Each of $A_iB_j$ can be found by using Strassen’s algorithm in $O(n^{\log_7 7})$ time, thus total time is $O(k^2n^{\log_7 7})$. 