This is a closed-book, closed-notes, closed-computer, closed-neighbor quiz.

1. Below is pseudocode for the iterative algorithm `SortedInsert`, which inserts the key $k$ into an already sorted array $A$ and maintains the sorted order of the array.

   ```pseudocode
   SortedInsert(A, k)
   1  i = length(A)
   2  while $i > 0$ and $k < A[i]$
   3      A[i + 1] = A[i]
   4      i = i - 1
   5  A[i + 1] = k
   ```

   (a) (4 points) Perform a line-by-line analysis of `SortedInsert` and derive a precise (non-asymptotic) expression of the running time $T(n)$, where $n = length(A)$. You may assume a cost $c_i = 1$ for each line of pseudocode.

   See line-by-line analysis above, where $t$ is the number of elements of the array $A$ greater than key $k$.

   $$T(n) = 3t + 3$$

   (b) (3 points) Describe the best-case scenario for `SortedInsert` and give both a precise and asymptotically-tight bound on the best-case running time.

   The best-case scenario is when the key $k$ is larger than any element in $A$, and the body of the while loop does not execute (i.e., $t = 0$). Thus, $T(n) = 3 = \Theta(1)$.

   (c) (3 points) Describe the worst-case scenario for `SortedInsert` and give both a precise and asymptotically-tight bound on the worst-case running time.

   The worst-case scenario is when the key $k$ is smaller than any element in $A$, and the body of the while loop executes $n$ times (i.e., $t = n$). Thus, $T(n) = 3n + 3 = \Theta(n)$.

2. Consider the following alternative approach to `SortedInsert` that first uses binary search to find the position for key $k$ and then shifts over the elements to the right and inserts $k$.

   ```pseudocode
   SortedInsert(A, k)
   1  n = length(A)
   2  i = Search(A, k, 1, n)
   3  for $j = n$ to $i + 1$
   5  A[i + 1] = k
   ```
\( \text{Search}(A,k,p,r) \)

1. if \( p < r \)
2. \quad \( q = \lceil \frac{p + r}{2} \rceil \)
3. \quad if \( k \leq A[q] \)
4. \quad \quad return Search(A,k,p,q)
5. \quad else return Search(A,k,q + 1,r)
6. else return \( p \)

(a) (4 points) Give a recurrence describing the worst-case running time \( T(n) \) of \text{Search}, where \( n = r - p + 1 \).

\[
T(n) = \begin{cases} 
\Theta(1) & n = 1 \\
T(n/2) + \Theta(1) & n > 1
\end{cases}
\]

(b) (3 points) Give an asymptotically-tight bound for the worst-case running time of this version of \text{SortedInsert}. You do not have to solve the recurrence.

\text{Search} will always take \( \Theta(\lg n) \) time. In the worst case, \text{Search} will return the lowest value for \( i \), causing the \textbf{for} loop to shift the entire array to the right, which takes \( \Theta(n) \). Thus, the total running time of \text{SortedInsert} will be \( T(n) = \Theta(\lg n) + \Theta(n) + \Theta(1) = \Theta(n) \).

3. (4 points) Solve the following recurrence using the master method. Show your work.

\[
T(n) = \begin{cases} 
\Theta(1) & n = 1 \\
T(n/2) + \Theta(1) & n > 1
\end{cases}
\]

Master method: \( a = 1, b = 2, f(n) = \Theta(1) \). Thus, \( n^{\log_b a} = n^{\log_2 1} = n^0 = \Theta(1) \), which equals \( f(n) \). Therefore, we are in case 2 of the master theorem, and \( T(n) = \Theta(\lg n \cdot \Theta(1)) = \Theta(\lg n) \).

4. (4 points) Use the substitution method to show that \( T(n) = O(n) \) for the recurrence in Problem 3.

Show that \( T(n) = O(n) \leq cn \).

Assume \( T(n/2) \leq cn/2 \).

\[
T(n) \leq \frac{cn}{2} + \Theta(1) \\
\leq \frac{cn}{2} + \Theta(1) + (\frac{cn}{2} - \Theta(1)) \\
\leq cn
\]

given that \( \frac{cn}{2} - \Theta(1) \geq 0 \), or \( c \geq 2\Theta(1)/n \). Since \( \Theta(1) \) represents a constant, for large enough \( n \), \( c \) can be a valid constant and still satisfy the inequality. Thus, \( T(n) = O(n) \).