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

\begin{verbatim}
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
\end{verbatim}

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

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

\begin{verbatim}
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

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

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

(b) (3 points) Give an asymptotically-tight bound for the worst-case running time of this version of \textsc{SortedInsert}. You do not have to solve the recurrence.
3. (4 points) Solve the following recurrence using the master method. Show your work.

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

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