1. Consider the following recurrence relation:

\[ T(n) = \begin{cases} 
25n, & n \leq 50 \\
3n + T(\frac{7n}{10}) + T(\frac{n}{5}), & n > 50 
\end{cases} \]

Prove by induction that the solution is \( T(n) \leq An \), and find the constant \( A \).
2. Given an array \( A[0..n-1] \) and a small constant \( k \). The array is defined to be “\( k \)-near-sorted” if for every element \( A[i] \), at most \( k \) elements to the left of it are greater than it. For example, the following sequence is 2-near-sorted:

\[
\begin{array}{cccccccc}
8 & 1 & 2 & 7 & 3 & 4 & 5 & 6 \\
\end{array}
\]

Obviously, it should be easier to sort an array which is known to be \( k \)-near-sorted, than to sort an arbitrary array.

(a) Write an efficient algorithm (write the code) \( \text{SORT}(A, n, k) \) for completely sorting an array \( A \) which is \( k \)-near-sorted.

(b) What is the time complexity of this algorithm in terms of \( n \) and \( k \)?

(c) Illustrate the algorithm for the sequence below, where \( k = 2 \).

\[
\begin{array}{cccccccc}
8 & 1 & 2 & 7 & 3 & 4 & 5 & 6 \\
\end{array}
\]
3. Given an undirected weighted graph, the Traveling-Salesperson-Problem (TSP) is to find a least-cost tour of the graph. (A tour is a simple cycle which goes through all vertices of the graph exactly once.)

(a) Find a least-cost tour, and its cost, for the above graph by inspection. You can start with vertex 1, though the starting vertex does not matter. (The TSP problem is NP-complete, but for this small graph the solution is easily found.)

(b) Now consider the following GREEDY algorithm for finding a good tour:

   Start with vertex 1 and go to the “NEAREST” vertex. Continue to build a tour by going from the last visited vertex to its nearest unvisited vertex, until you get back to vertex 1.

   Find a tour found by this greedy algorithm, and its cost. Is this greedy solution optimal?
4. (a) The last programming assignment used three different methods to implement \( \text{SELECT}(A, n, k) \), which finds the \( k \)th smallest element in an array of \( n \) elements. List the three algorithms, along with their worst-case time and average-case (expected) time complexities.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case Time</th>
<th>Expected Time</th>
</tr>
</thead>
</table>

(b) If \( k \) is a small constant, say \( k = 10 \), will it change the above worst-case and average-case running times? Explain.

(c) For the special case when \( k \) is a small constant, consider the following algorithm for finding the \( k \)th smallest element:

i. Make a heap of \( n \) elements;

ii. Perform \( \text{DELETE-MIN} \) operations \( k \) times.

Analyze the worst-case time complexity of this algorithm. How does it compare with the above three algorithms?
5. Consider the following undirected weighted graph. (Assume the graph is represented by its adjacency matrix.)

(a) Find a minimum-cost-spanning tree (MST) using PRIM’s algorithm. Show the result after each iteration, and the final MST and its cost.

(b) Use Dijkstra’s algorithm, with vertex 1 as the source, to find the single-source-shortest-paths spanning tree. Show the result after each iteration, and the final tree.