1. (a) Insert the following sequence of random elements into an initially empty AVL tree. (The first element of the sequence is inserted first, second element is inserted next, and so on.) Show the result immediately after each insertion and then after any rebalancing in each step.

50, 25, 80, 90, 75, 95, 60, 78, 10, 55

<table>
<thead>
<tr>
<th>GRADE</th>
<th>/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>/100</td>
</tr>
</tbody>
</table>

(b) What is the total worst-case time complexity of \( n \) insertions into an initially empty AVL tree? Explain.

(c) What is the total worst-case time complexity of \( n \) insertions into an initially empty Binary-Search-Tree (BST)? Explain.
2. (a) We have a sorted array of \( n \) elements. We want to construct an AVL tree to contain these \( n \) elements. Describe an efficient algorithm for constructing the AVL tree. (Hint: Think about which element of the array should be placed at the root of the AVL tree.) Analyze the worst-case time complexity. Illustrate the algorithm for the following array.

\[5, 8, 9, 10, 12, 14, 16, 20, 32, 35, 36, 40, 42, 45, 48\]

(b) Give the pseudo-code for a recursive algorithm to generate the AVL tree from a sorted array. The recursive algorithm takes as input an array \( A[i..j] \) (with starting index \( i \) and ending index \( j \)) and returns a pointer to the root of the tree that is generated. You may assume a new tree node is generated by NewNode and that each node has three fields: Key, LeftChild (pointer to left child), and RightChild (pointer to right child). The initial call to this recursive algorithm is Generate-AVL (\( A, 0, n - 1 \)).

\[
\text{ptr Generate-AVL (real } A[], \text{ int } i, \text{ int } j)\]

3. Consider the problem of selection: Given an array of \( n \) real-valued random elements and an integer \( k \), we want to find the \( k^{th} \) smallest element. What is the worst-case and average-case time complexity for each of the following algorithms? Provide a brief explanation as needed.

(a) Randomized Selection (quick-select):

(b) Selection algorithm that uses median-of-row-medians.

(c) Quicksort

(d) An adaptation of heapsort, with only \( k \) delete-min operations.

(e) An adaptation of bubble-sort, with only \( k \) iterations.
4. (a) Consider Floyd’s all-pairs-shortest-paths algorithm for a directed weighted graph. What is the time complexity of this algorithm?

Show the working of the algorithm for a graph with the following adjacency matrix $A$, where $A[i, j]$ equals the cost of the edge $(i, j)$. Show the cost matrix after each iteration $k = 1, 2, 3, 4$. (Don’t worry about recording the path.)

$$
\begin{bmatrix}
0 & 17 & 10 & 25 \\
20 & 0 & \infty & 3 \\
\infty & 2 & 0 & 10 \\
10 & \infty & \infty & 0 \\
\end{bmatrix}
$$

(b) Now consider the problem of all-pairs-shortest-paths for a directed but unweighted graph, where the length of a path is simply the number of edges. Suppose the graph is sparse, with at most $3n$ edges, and is represented by its adjacency lists.

Briefly outline a more efficient algorithm for all-pairs-shortest-paths for this special case. Analyze the worst-case time complexity.
5. Consider Prim’s minimum-cost-spanning-tree (MST) algorithm for an undirected weighted graph. Assume the graph is connected, is sparse and is represented by its adjacency-lists.

(a) What is the time complexity of Prim’s algorithm for this graph representation? Briefly explain what data structure is used and how each iteration is implemented.

(b) Now consider a special case when all edges have integer weights in the range 1 to 5. Again, assume the graph is sparse and represented by it adjacency-lists. Describe a more efficient implementation of Prim’s MST algorithm. What data structure do you use to maintain the list of candidate edges? Analyze the time complexity of this implementation.