**Name: Sundas Shoukat**

**Roll no. 057**

**Section: BSDS-3A**

**Explanation of Min-Max Algorithm**

**How the Code Works**

* The program **imports the math library** to perform mathematical calculations such as logarithms.  
  This helps in finding the number of levels in the game tree.
* A **list named “values”** is created that contains numbers like *3, 1, 5, 2, 7, 8, 9, 9*.  
  These values represent the **leaf nodes** or final results of all possible game outcomes.
* The program then **calculates the total depth (levels)** of the game tree using the formula **log base 2** of the number of values.  
  Since there are 8 values, the result is **3 levels**.  
  This means the game tree will have three layers from top to bottom.
* A **function named “minmaxalgo”** is defined to perform the main logic of the program.  
  This function is **recursive**, meaning it calls itself to explore every possible path in the game tree.
* The function takes several **important parameters**:
  + **currDepth:** shows the current level in the tree.
  + **nodeIndex:** shows which node is being checked.
  + **minTurn:** decides whose turn it is — Min player or Max player.
  + **values:** stores the list of final leaf node values.
  + **totalDepth:** represents how deep the game tree goes.
* The function first checks if the **current depth equals the total depth.**  
  If yes, it means we have reached a leaf node and there are no more moves left.  
  In that case, the function **returns that leaf node’s value**.
* If the current depth is not the total depth, the program then decides **whose turn it is**:
  + If it’s **Min’s turn**, it selects the **smaller value** between the two possible moves because Min tries to reduce the score.
  + If it’s **Max’s turn**, it selects the **larger value** because Max tries to increase the score.
* The function continues this process step by step, moving deeper into the tree and then coming back up with the best possible values.
* Finally, the **main part of the program** calls the function starting from the top of the tree — depth 0, node 0, and Max’s turn first.  
  After completing all recursive calls, the program prints the **best possible score**, which is **2** in this case.

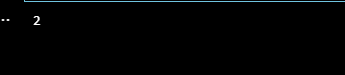
**Why This Code Is Used**

* The Min-Max algorithm is widely used in **two-player games** such as Tic-Tac-Toe, Chess, and Checkers.
* It helps the computer make smart decisions by checking **all possible outcomes** of the game.
* The algorithm assumes that both players play **perfectly and logically**, and it picks the move that gives the **best advantage** to the current player.
* The **Max player** tries to get the highest possible score, while the **Min player** tries to lower it.
* This back-and-forth process teaches how computers can **think strategically**, just like humans do in real games.

**Conclusion**

* This program shows how **artificial intelligence** can make intelligent choices by looking ahead at all possibilities.
* The Min-Max algorithm works by alternating between **maximizing** and **minimizing** moves.
* It helps in finding the **optimal decision** in games where two players compete.
* The result (2 in this case) represents the **best guaranteed score** for the Max player if both players play their best.
* Overall, this program is a clear and simple demonstration of **decision-making in AI** using logic and recursion.

**Output**

****