Task 8:

Explanation:

1. Minimax

Minimax is a recursive algorithm used to find the best move in a game by assuming that:

One player (Maximizer) tries to get the highest possible score.

The other player (Minimizer) tries to get the lowest possible score.

The algorithm works by simulating all possible moves and choosing the best one.

2. Understanding the Code
    a) Function Definition

```
Def minimax(depth, node_index, is_max, scores, height):
```

Depth: The current depth of the decision tree.

Node_index: The index of the current node in the tree.

Is_max: A boolean (True/False) to check if it's the turn of the Maximize

Height: The total depth of the decision tree.

    b) Base Case (Stopping Condition)

```
If depth == height:
  Return scores[node_index]
```

If we reach the maximum depth, return the score of the current node.

    c) Recursive Calls (Maximizer & Minimizer Turns)

```
If is_max:
  Return max(
    Minimax(depth + 1, node_index * 2, False, scores, height),
    Minimax(depth + 1, node_index * 2 + 1, False, scores, height)
  )
```

If it's the Maximizer's turn (is_max=True), we take the maximum of the two possible moves.

```
Else:
   Return min(
      Minimax(depth + 1, node_index * 2, True, scores, height),
      Minimax(depth + 1, node_index * 2 + 1, True, scores, height)
   )
```

If it's the Minimizer's turn (is_max=False), we take the minimum of the two possible moves.

d) Running the Algorithm

```
Scores = [3, 5, 2, 9, 12, 5, 23, 23]
Tree_height = math.log2(len(scores))
Optimal_value = minimax(0, 0, True, scores, int(tree_height))
```

3.

If scores = [3, 5, 2, 9, 12, 5, 23, 23], the algorithm will evaluate the decision tree and find the best possible outcome.