# Task 1 (MinMax Implementation):

import math  
  
  
def minimax(curDepth, nodeIndex,  
 maxTurn, scores\_arr,  
 targetDepth):  
 # base case : targetDepth reached  
 if curDepth == targetDepth:  
 return scores\_arr[nodeIndex]  
  
 if maxTurn:  
 return max(minimax(curDepth + 1, nodeIndex \* 3,  
 False, scores\_arr, targetDepth),  
 minimax(curDepth + 1, nodeIndex \* 3 + 1,  
 False, scores\_arr, targetDepth))  
  
 else:  
 return min(minimax(curDepth + 1, nodeIndex \* 3,  
 True, scores\_arr, targetDepth),  
 minimax(curDepth + 1, nodeIndex \* 3 + 1,  
 True, scores\_arr, targetDepth))  
  
  
scores = [3, 12, 8, 2, 4, 6, 14, 5, 2]  
  
treeDepth = math.log(len(scores), 3)  
  
print("The optimal value is : ", minimax(0, 0, True, scores, treeDepth))

# Task 1 (AlphaBeta Implementation):

# BCS181081 - Shaban  
  
MAX, MIN = 1000, -1000  
  
  
def minimax(depth, nodeIndex, maximizingPlayer, arr, alpha, beta):  
 if depth == 2:  
 return arr[nodeIndex]  
  
 if maximizingPlayer:  
 best = MIN  
  
 for i in range(0, 2):  
  
 val = minimax(depth + 1, nodeIndex \* 3 + i, False, arr, alpha, beta)  
 best = max(best, val)  
 alpha = max(alpha, best)  
  
 # Alpha Beta Pruning  
 if beta <= alpha:  
 break  
  
 return best  
  
 else:  
 best = MAX  
  
 for i in range(0, 2):  
  
 val = minimax(depth + 1, nodeIndex \* 3 + i, True, arr, alpha, beta)  
 best = min(best, val)  
 beta = min(beta, best)  
  
 # Alpha Beta Pruning  
 if beta <= alpha:  
 break  
  
 return best  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 values = [3, 12, 8, 2, 4, 6, 14, 5, 2]  
 print("\n\*\*\* The optimal value is :", minimax(0, 0, True, values, MIN, MAX), "\*\*\*")