Implement Alpha – Beta Pruning.

```
def alpha_beta_pruning(node, alpha, beta, maximizing_player):
    if type(node) is int:
        return node

    if maximizing_player:
        max_eval = -float('inf')
        for child in node:
            eval = alpha_beta_pruning(child, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return max_eval
    else:
        min_eval = float('inf')
        for child in node:
            eval = alpha_beta_pruning(child, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return min_eval

def build_tree(numbers):
    current_level = [[n] for n in numbers]

    while len(current_level) > 1:
        next_level = []
        for i in range(0, len(current_level), 2):
            if i + 1 < len(current_level):
                next_level.append(current_level[i] + current_level[i + 1])
            else:
                next_level.append(current_level[i])
        current_level = next_level

    return current_level[0]

def main():
    print("Output: 1BM22CS290")
    numbers = list(map(int, input("Enter numbers for the game tree (space-separated): ").split()))
    tree = build_tree(numbers)

    alpha = -float('inf')
    beta = float('inf')
    maximizing_player = True

    result = alpha_beta_pruning(tree, alpha, beta, maximizing_player)
    print("Final Result of Alpha-Beta Pruning:", result)

if __name__ == "__main__":
    main()
```

```
Output: 1BM22CS290
Enter numbers for the game tree (space-separated): 10 9 14 18 5 4 50 3
Final Result of Alpha-Beta Pruning: 50
```