# Minimax Algorithm

In Minimax, the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.

Example: Consider the game which has 4 final states and the path to reach the final state is from root to state. The maximizer makes the first move and is at the root and the opponent minimizer is at the next level. The final stage is denoted as leaves in the following tree. Which move should a maximizer make, considering that the minimizer also plays optimally?
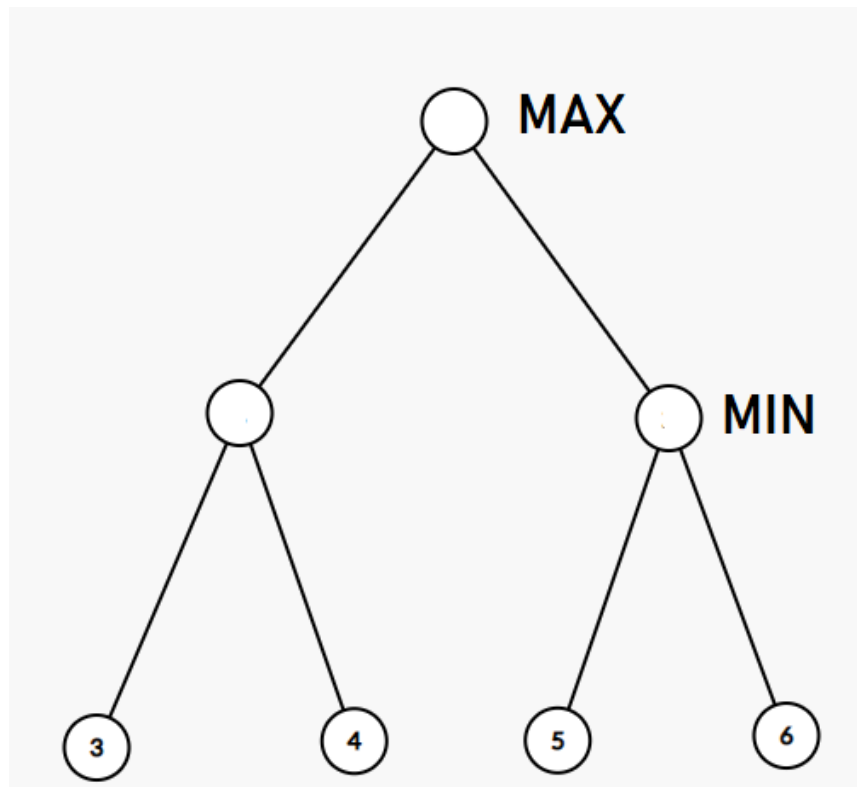
It is maximizers turn first, maximizer has two options:
- MAXIMIZER GOES LEFT: If the maximizer goes left, then after that it is the minimizer's turn and the minimizer will have two options, either to choose 3 or 4. So the minimizer will choose 3 to minimize the score.
- MAXIMIZER GOES RIGHT: If the maximizer goes right, then after that it is the minimizer's turn and the minimizer will have two options, either to choose 5 or 6. So the minimizer will choose 5 to minimize the score.

Being a maximizer, it will choose a larger value i.e, 5, so the maximizer will go right.
The final score will be
Maximizer: 5
Minimizer: 5

// depth: current level initially will be passed 0

```
//   nodeIndex: index of the position, assuming heap like structure
//   isMax: true: if maximizer's turn, false if minimizer's turn
//   scores(type: array): scores at the leaf nodes
//   h: height of the tree

function minimax(depth, nodeIndex, isMax, scores[], h)
{
        if (depth == h)
                return scores[nodeIndex];

        //   If current move is maximizer, find the maximum value
        if (isMax)
                return max(minimax(depth+1, nodeIndex*2, false, scores, h),
                        minimax(depth+1, nodeIndex*2 + 1, false, scores, h));

        //   Else (If the current move is Minimizer), find the minimum value
        else
                return min(minimax(depth+1, nodeIndex*2, true, scores, h),
                        minimax(depth+1, nodeIndex*2 + 1, true, scores, h));
}
```