Recursive α-β pruning: R&N Fig. 5.7

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state) then return EVAL(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state) then return EVAL(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

Simple stub to call recursion functions
Initialize alpha, beta; get best value
Score each action; return best action

If Cutoff reached, return Eval heuristic
Otherwise, find our best child:
If our options become too good, our min ancestor will never let us come this way, so prune now & return best value so far
Finally, return the best value we found

If Cutoff reached, return Eval heuristic
Otherwise, find our worst child:
If our options become too bad, our max ancestor will never let us come this way, so prune now & return worst value so far
Finally, return the worst value we found

Recursive α-β pruning variant: Prune when α ≥ β

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state) then return EVAL(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(state,a), α, β))
    α ← MAX(α, v)
    if α ≥ β then return v
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if CUTOFF-TEST(state) then return EVAL(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(state,a), α, β))
    β ← MIN(β, v)
    if α ≥ β then return v
  return v
```

This variant has a conceptually simpler pruning rule (α ≥ β), but when pruning occurs it makes one extra call to MAX(). Both variants yield the same pruning behavior, and **both are considered correct on tests.**