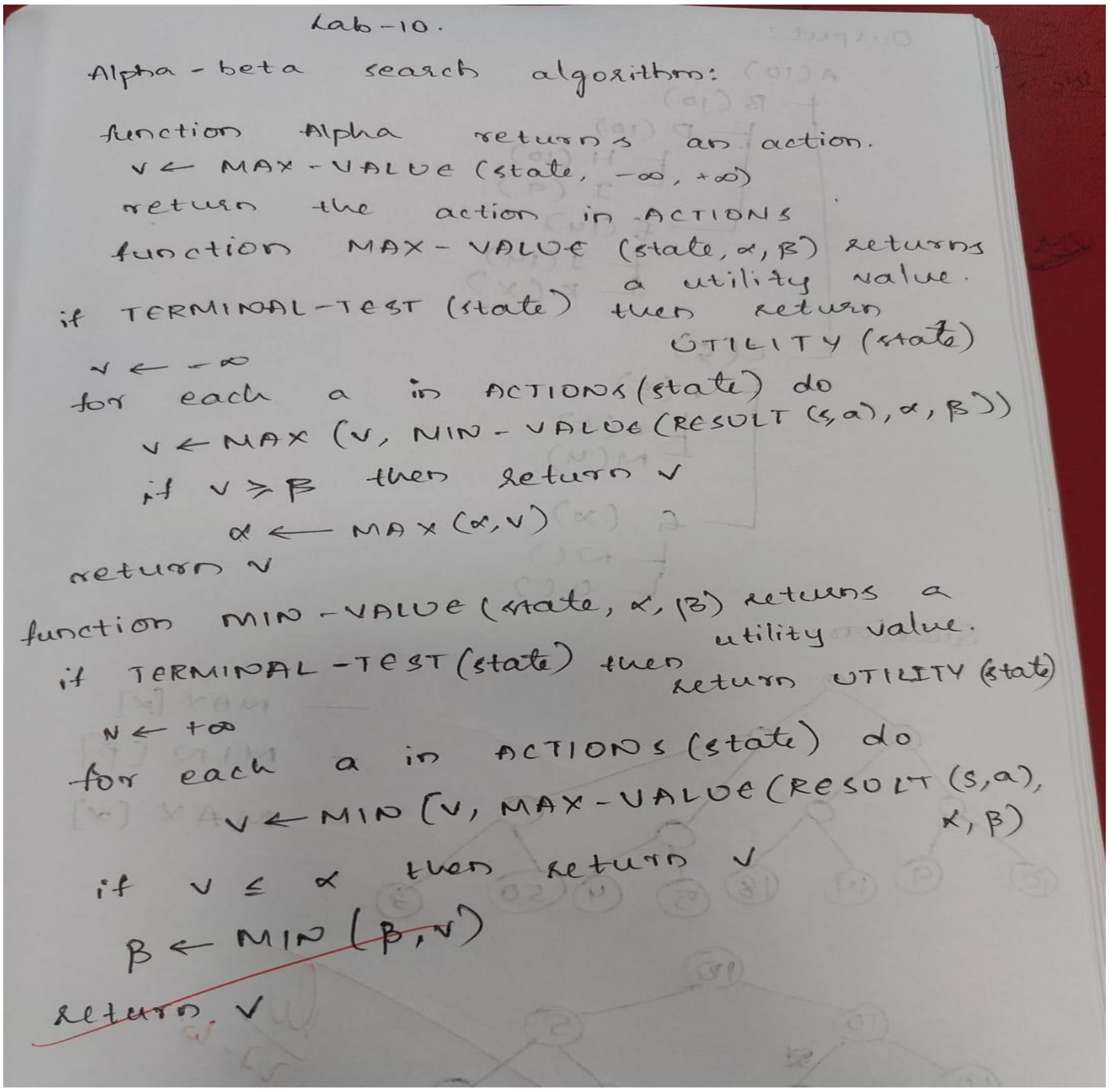
WEEK – 10

ALPHA-BETA PRUNING :

Algorithm:



Lab-10.

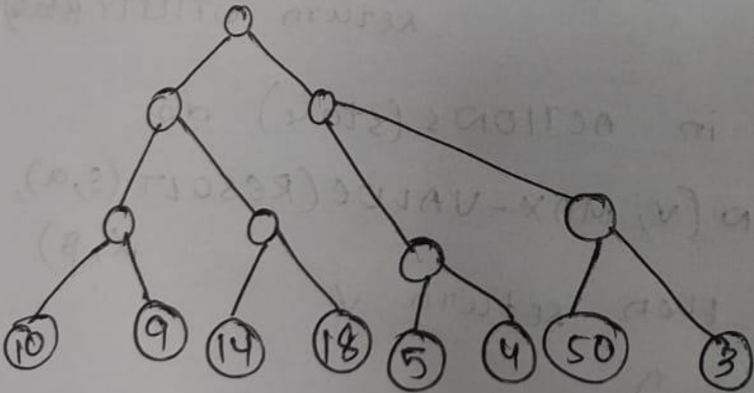
Alpha-beta search algorithm:

```
function Alpha returns an action.
  v <- MAX-VALUE (state, -∞, +∞)
  return the action in ACTIONS
  function MAX-VALUE (state, α, β) returns
                                   a utility value.
if TERMINAL-TEST (state) then return
                              UTILITY (state)
 v <- -∞
for each a in ACTIONS(state) do
    v <- MAX (v, MIN-VALUE (RESULT (s,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 TERMINAL-TEST (state) then return UTILITY (state)
  v <- +∞
 for each a in ACTIONS (state) do
       v <- MIN (v, MAX-VALUE (RESULT (s,a),
                                     α, β)
   if v ≤ α then return v
   β <- MIN (β,v)
return v
```

Output:

```
A(10)
 ├ B(10)
 │  ├ D(10)
 │  │  ├ H(10)
 │  │  └ I(9)
 │  └ E(14)
 │     ├ J(14)
 │     └ K(x)
 └ C(5)
    ├ F(5)
    │  ├ L(5)
    │  └ M(4)
    └ G(x)
       ├ N()
       └ O()
```

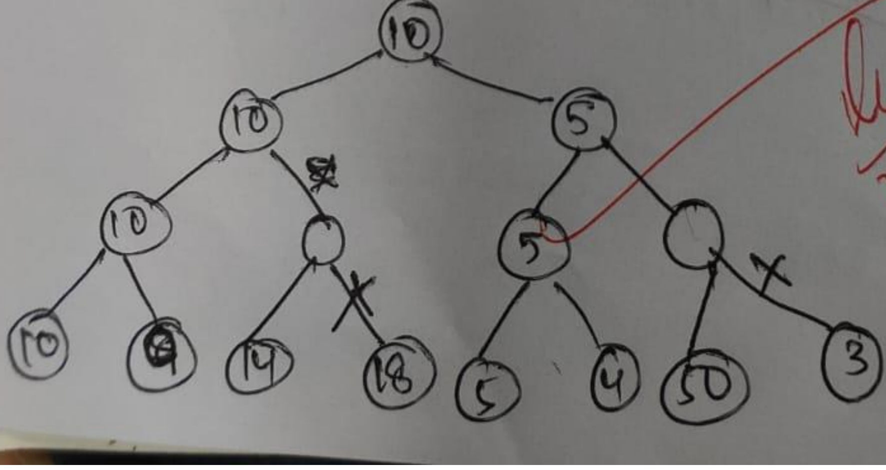
Question.



— MAX [α]

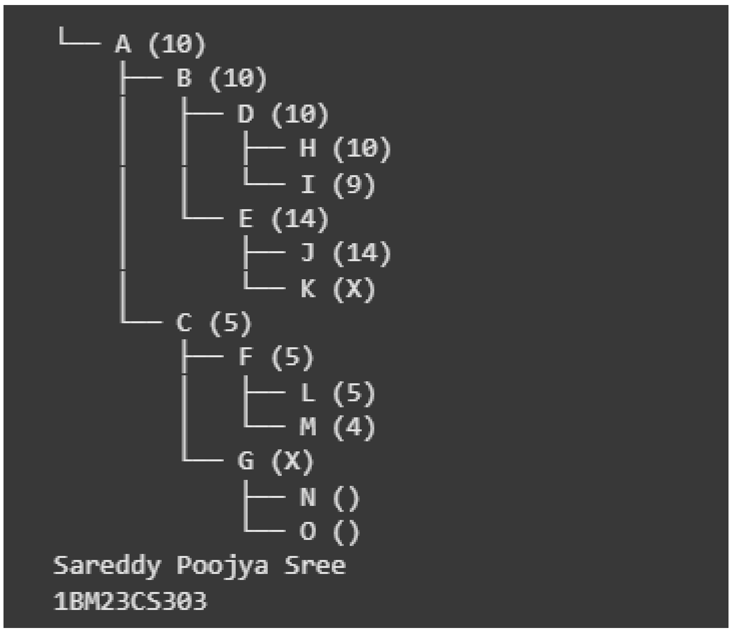
– MIN [β]

– MAX [α]



[illegible]
27-10

Output:


```
└── A (10)
    ├── B (10)
    │   ├── D (10)
    │   │   ├── H (10)
    │   │   └── I (9)
    │   └── E (14)
    │       ├── J (14)
    │       └── K (X)
    └── C (5)
        ├── F (5)
        │   ├── L (5)
        │   └── M (4)
        └── G (X)
            ├── N ()
            └── O ()
Sareddy Poojya Sree
1BM23CS303
```

Code:

```
import math


tree = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H', 'I'],
    'E': ['J', 'K'],
    'F': ['L', 'M'],
    'G': ['N', 'O'],
    'H': [], 'I': [], 'J': [], 'K': [],
    'L': [], 'M': [], 'N': [], 'O': []
}


# Leaf node values
values = {
    'H': 10, 'I': 9,
    'J': 14, 'K': 18,
    'L': 5, 'M': 4,
    'N': 50, 'O': 3
}

# to store final display values
node_values = {}

def get_children(node):
    return tree.get(node, [])

def is_terminal(node):
    return len(get_children(node)) == 0

def evaluate(node):
    return values[node]

def alpha_beta(node, depth, alpha, beta, maximizing):
    if is_terminal(node) or depth == 0:
        val = evaluate(node)
        node_values[node] = val
        return val

    if maximizing:
        value = -math.inf
        for child in get_children(node):
            val = alpha_beta(child, depth - 1, alpha, beta, False)
            value = max(value, val)
            alpha = max(alpha, val)
            if beta <= alpha:
                # mark remaining children as pruned
                for rem in get_children(node)[get_children(node).index(child)+1:]:
                    node_values[rem] = "X"
                break
        node_values[node] = value
        return value
    else:
        value = math.inf
        for child in get_children(node):
            val = alpha_beta(child, depth - 1, alpha, beta, True)
            value = min(value, val)
            beta = min(beta, val)
            if beta <= alpha:
                for rem in get_children(node)[get_children(node).index(child)+1:]:
                    node_values[rem] = "X"
                break
        node_values[node] = value
        return value

# Run pruning
alpha_beta('A', depth=4, alpha=-math.inf, beta=math.inf, maximizing=True)


def print_tree(node, prefix="", is_last=True):
    connector = " └── " if is_last else " ├── "
    value = node_values.get(node, "")
    print(prefix + connector + f"{node} ({value})")
    children = get_children(node)
    for i, child in enumerate(children):
        new_prefix = prefix + ("    " if is_last else "│   ")
        print_tree(child, new_prefix, i == len(children)-1)

# Display the final tree
print("\nFINAL TREE\n" )
print_tree('A')

print("Sareddy Poojya Sree\n1BM23CS303")
```