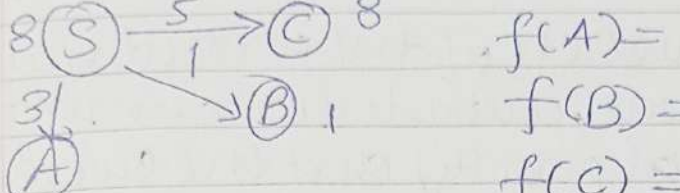
Q-1)
(a) A^*

It works on a formula of calculating actual cost plus heuristic value (h(n))
$g(n)$

Now, $[f(n) = g(n) + h(n)]$.

Aim: to reach from S to G1 hence,



$f(A) = 3+2 = 5$

$f(B) = 1+1 = 2$

$f(C) = 8+5 = 13$ → Not check bcuz found '8' as one of the potential ans

Now, f(B) looks least cost hence we will go that way,

S — B — F $f(F) = 1+2+3 = 6$

B — D $f(D) = 1+4+4 = 9$ (Never check again)

hence,

S — B — F — D $f(D) = 4+4 = 8$

Now,

S — B — F — D — E $f(E) = 1+2+1+2+1 = 7$

Now,

S — B — F — D — E — G1 $= 1+2+1+2+2 = 8$

But $f(A) = 5$ & we might can have a better option, we have check over there too!!).

S → A → D $f(D) = 3+4+4 = 11$ ✗ rejected

A → G1 $f(G1) = 3+10 = 13$ ✗ rejected

hence, according to A^*

S → B — F — D — E — G1 Total cost = 8

↓

This is 'B'

(b) Uniform cost Search

- Insert the root node into the priority queue
- Remove the element with highest priority
- If the removed node is the goal node
  - print total cost and stop the algo
- Else
  - Enqueue all the children of the current node to the priority queue, with their cumulative cost from the root as priority and the current node to the visited list

Let the priority q (queue) be PQ

PQ: S(0)

PQ: B(1) A(3) C(5)

PQ: A(3) F(3) C(5) D(5)

PQ: F(3) C(5) D(5) D(7) G1(13)

PQ: D(4) C(5) D(5) D(7) G1(13)

PQ: C(5) D(5) E(6) D(7) G1(13)

PQ: E(6) D(7) D(7) G1(13)

PQ: D(7) E(7) G1(8) G1(13)

PQ: E(7) G1(8) E(9) G1(13)

PQ: G1(8) G1(9) E(9) G1(13)

found the goal hence cost is 8 and path is

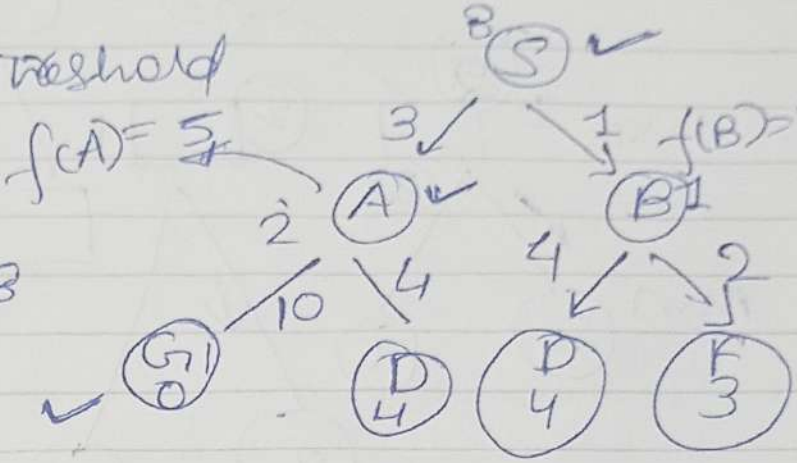
S → B → F → D → B → G1

(c) Iterative deepening A*

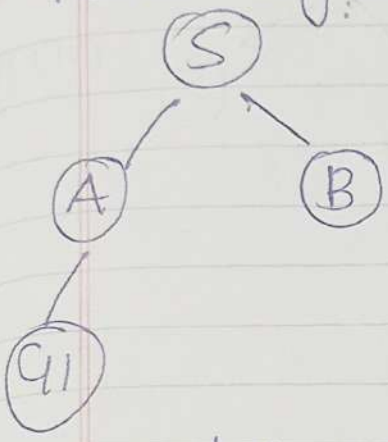
Threshold value initially be the heuristic value of starting node here, Now,

T.V = 8 (Threshold

Now, f(n) =

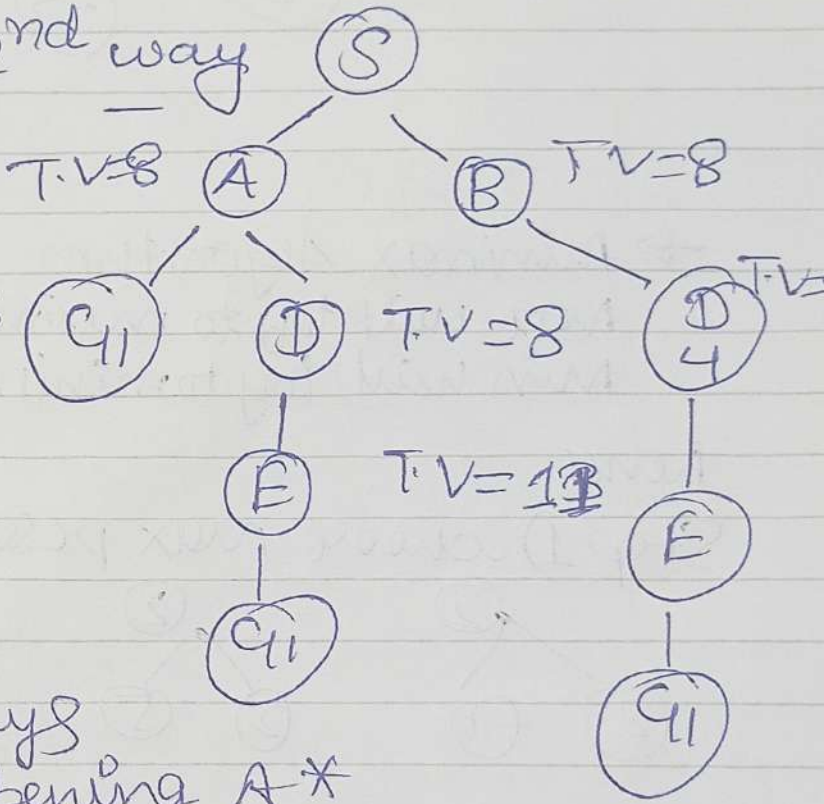


Hence by threshold = 8



reached the goel.

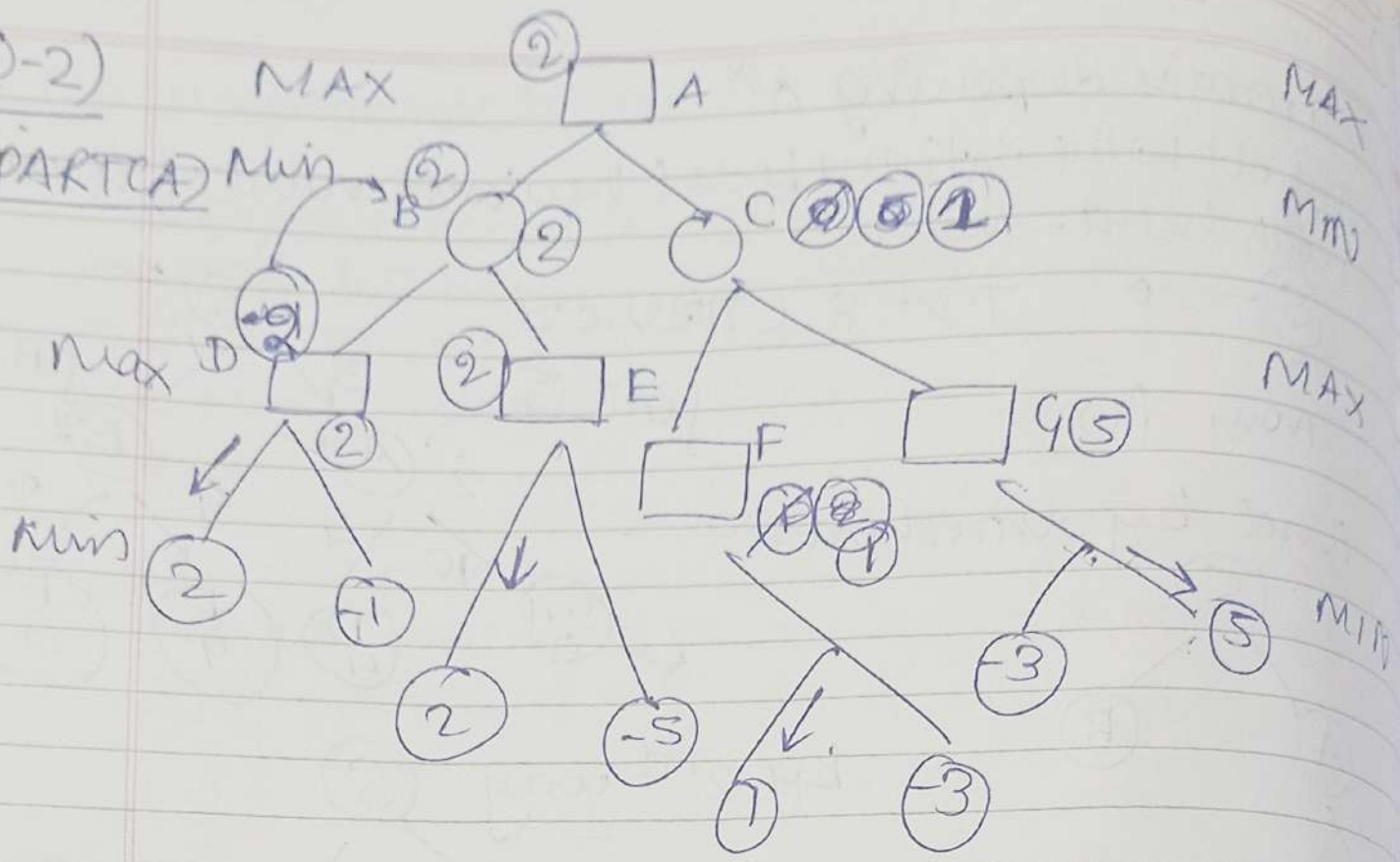
by 2nd way



Hence there are three ways in which iterative deepening A* can reach up to goel with Threshold values: 8, 9, 11

Q-2)

PART(A)



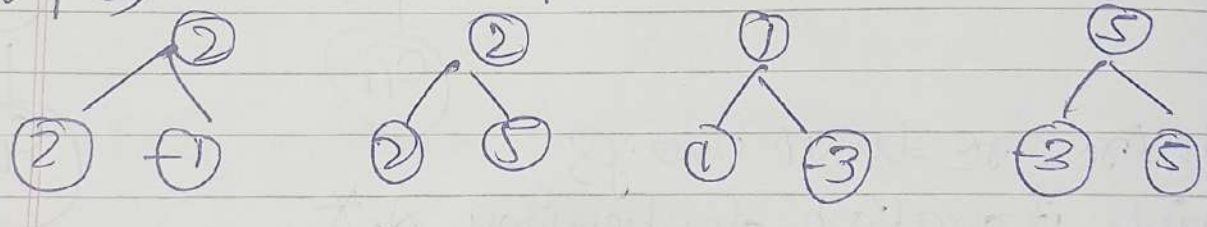
→ Minimax Algorithm

Max will try to maximize its possibility of winning

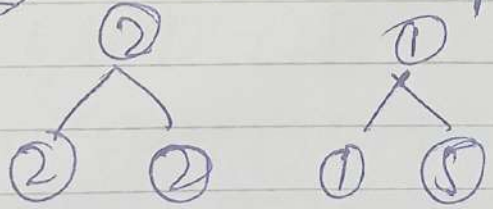
Min will try to minimize ~~it~~ others possibility of winning

hence

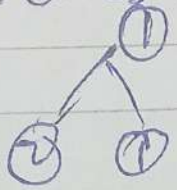
Step 1) choose max possible from root level nodes



step 2) chose min poseibly tabs

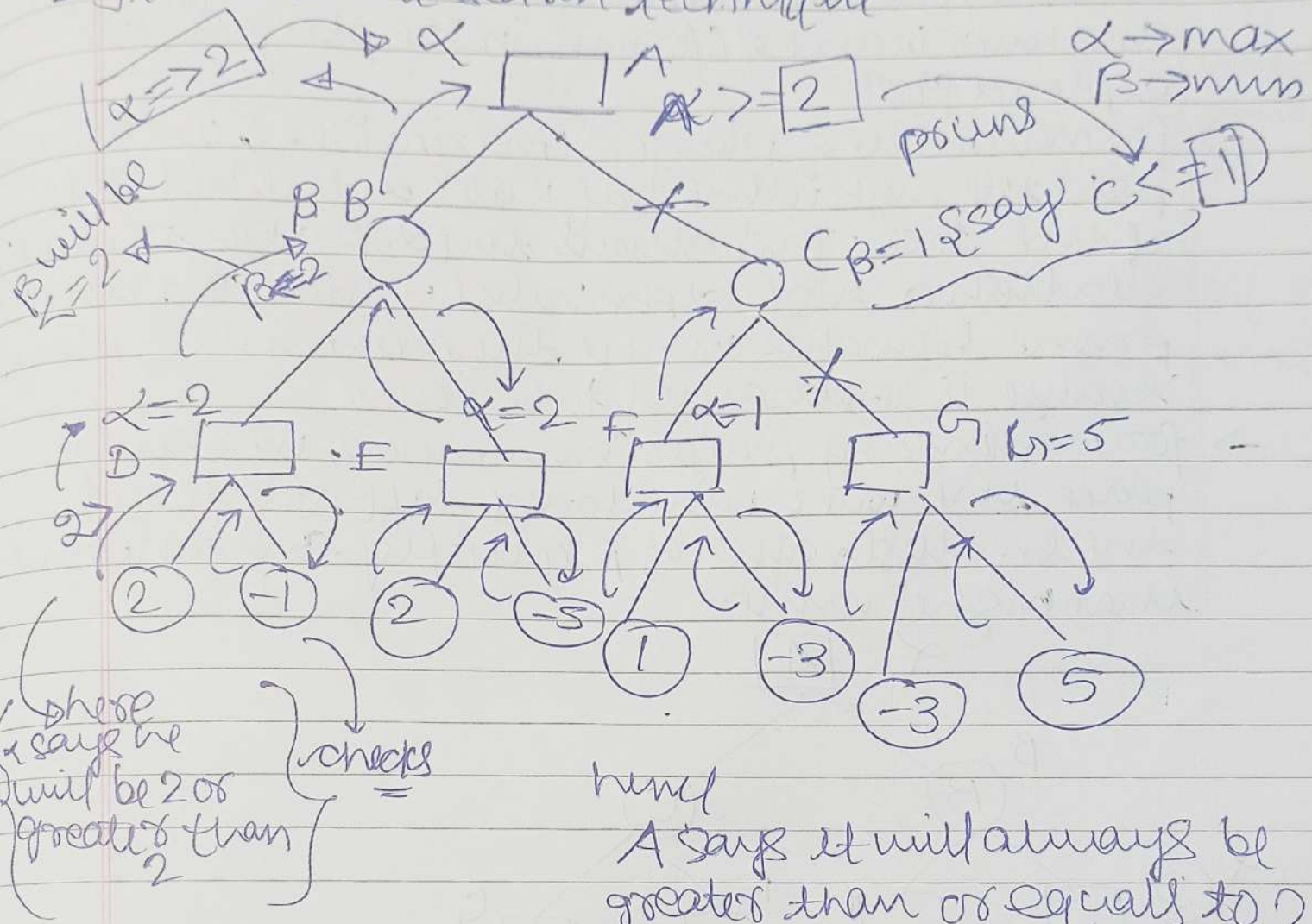


step 3) chose max hence,



Solving it via alpha-beta pruning (α-β)

- Search it as α be always max and β be always min
- It is faster than max-min algo.
- It is faster because it doesn't go through unnecessary paths.
- It's a smart search technique.



Hence

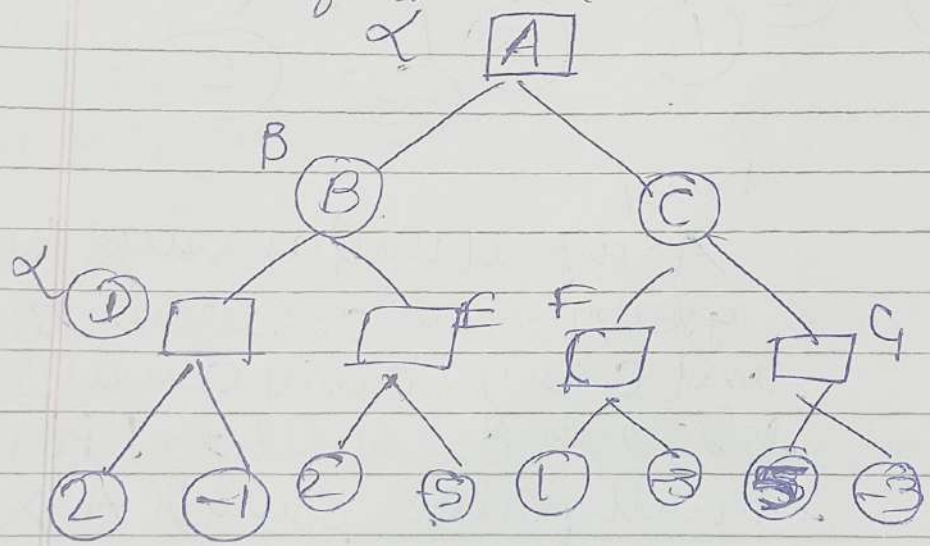
A says it will always be greater than or equals to 2 and C says C will always be less than or equals to 1 hence it will prunes branch A * C because it won't get anything greater than 2.

Part (B)

(a) In the best case scenario for alpha-beta pruning you want to maximize the number of pruned nodes. to do this rearrange the nodes such that pruning happens as early as possible i.e higher value branches (for maximizing players) and lower branches (for minimize players) are explored first.

→ for maximizing players (maximizer), we should place the high value leaf nodes on the left side of the tree. This allows the α-β algo to quickly establish a high alpha value which can then prune branches where the maxmizing player's value is guaranted to be lower

→ for minimizing players (minimizer), we should place the lower value to the left side to get the smallest value of β quickly. and compare that higher value.



higher value on the left side

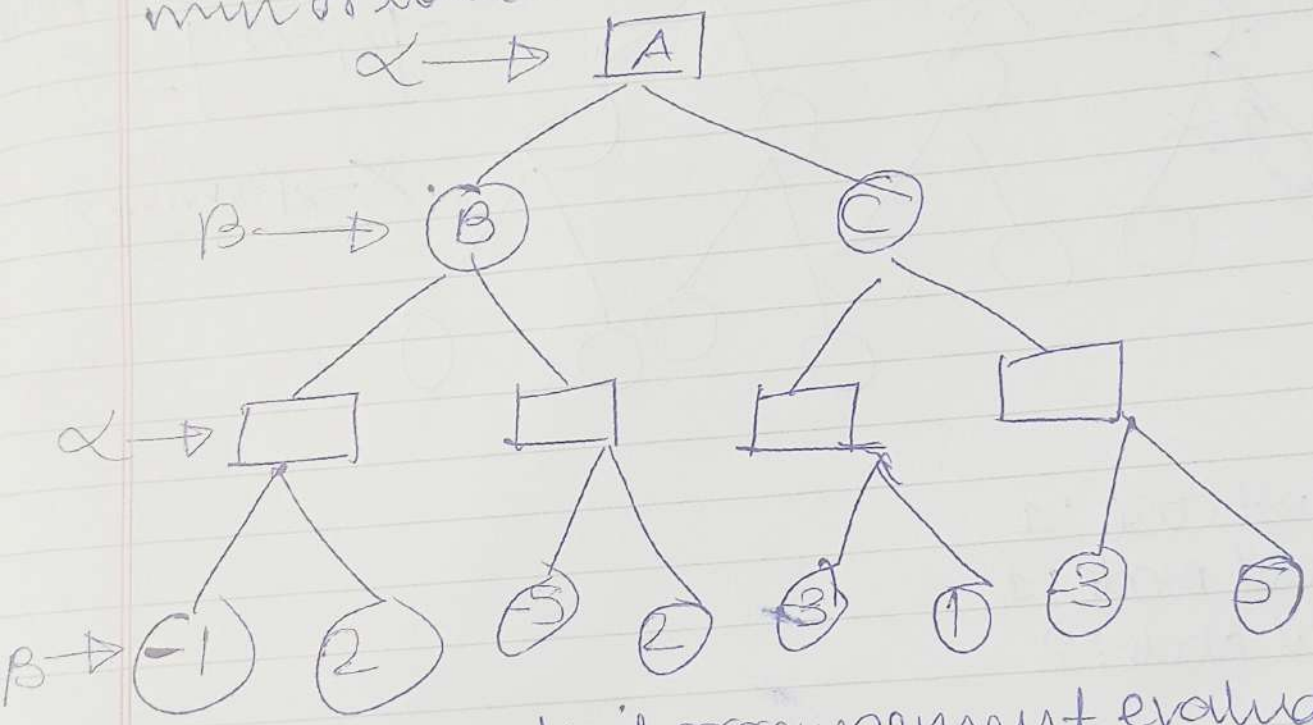
Justification:- By prioritizing branches that will make the α or β bound tightly early you force the algo to prune more branches before they are fully explored reducing no. of nodes evaluations

(6) worst case

In the worst case, the pruning will be minimized hence if we do are not pruning hence its brute-force only (max-min) where we are exploring almost every node so for getting worst case to the following

maximizer's player: will place the value of leafs as such that max values are on right and min values are left side to that he has to explore other branch too

minimizer's player: place the value of leaf nodes as such that max values are at the left and min or lowers at the right.

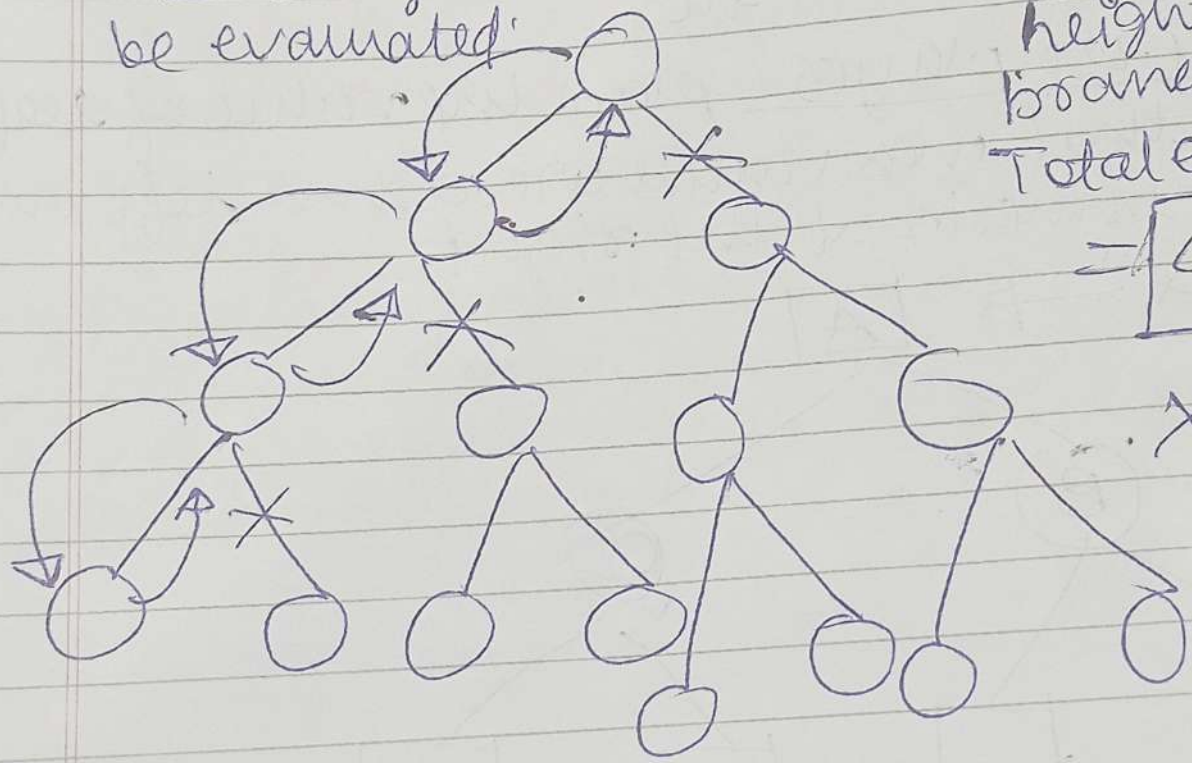


Justification:- In this arrangement evaluation the algo will not be able to prune as many branches because it will take longer to tigh α-β bound leading to exploreation of more unnecessary branch.

Part (c)

The best case time complexity of α-β pruning is $O(b^{d/2})$ where b is branch no. & d is depth of the tree.

a) In the best case α-β pruning allows us to prune roughly half of the nodes at each level of a game tree. This effectively reduces the depth of the tree that needs to be explored by half as you and only need to search one branch at each level, significantly reducing the total no. of nodes that must be evaluated.



height = 4 (depth)
branchor node = 2
Total explored = 4 nodes
$= 4^{2/2} = 4$

X → pruning