# Time Complexity Analysis.

1. Naive Recursive Approach.

→ Here we check all the possible combinations and if even one such permutation satisfies our condition we return it.

∴ if no. of colors = m
no. of nodes = n

So, No. of all possible combinations

= (m) (m) (m) ..... n times
= $m^n$.

From the pseudocode we can observe that at every call stack we call the function 'm' times using a for loop and their are m such stack levels.

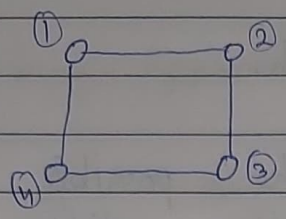
∴ Time complexity = $O(m^n)$.

And the Space Complexity. :-

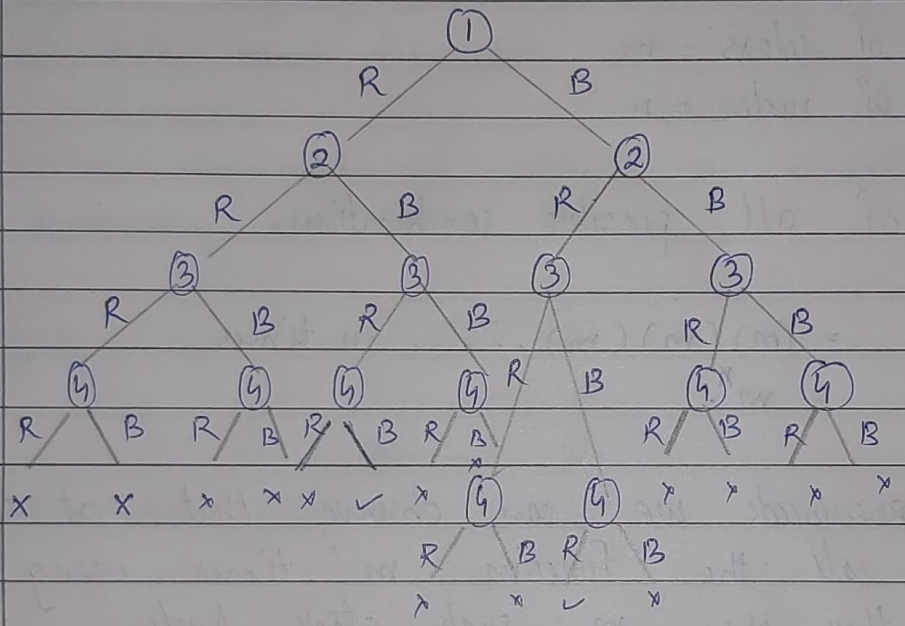
⇒ Recursive stack of graphColoring ( ) function will require O(n) space.

∴ Space Complexity = O(n) [Stack space].

Eg.




Available Colours = 2 (R/B)




Height is equal to n+1

∴ Space Complexity: O(n)

∴ Possible ans: RBRB / BRBR

∴ It checks all $m^n = 2^4 = 16$ combinations

∴ Time Complexity = $O(m^n)$, where m = no. of colors
Space Complexity = $O(n)$ and n = no. of nodes.

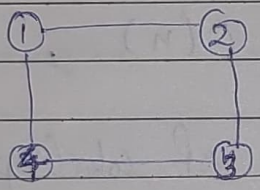
## 2. Backtracking Approach

→ Here we will assign colors one by one to different vertices, starting from the vertex 0.

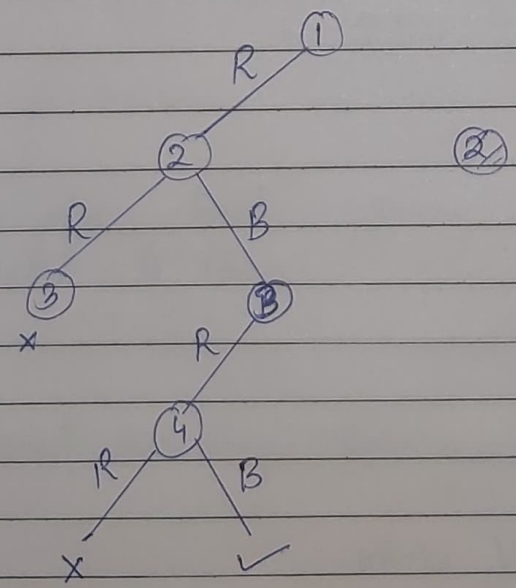
Before assigning color, we will check if its valid or not. If it is valid, then we will assign it and move forward.

If no, assignment of colors is possible then we will backtracking and return false.

Eg :-




Available colors = 2 (R/B)




∴ It will not check all $2^4$ i.e. $m^n$ combinations

Thus, reducing the time taken.

But here, we can see that

in the worst case, this algorithm to takes (checks) all $m^n$ combinations.

∴ Worst Case Time complexity : $O(m^n)$

But, the average time taken will be less.

∴ Better than the naive recursive approach.

=> Space Complexity : $O(n)$ ,

- Recursive stack of colorGraphRecursive(...) function will require $O(n)$ space.

3. BFS Approach.

→ The approach here is to color each node from 1 to n initially by color 1. And start travelling BFS from an unvisited starting node. to cover all connected components in one go.

The number of times each vertex is visited is 1. i.e. O(V). V is the no. of times the while loop runs.

⇒ Total running time = sum of all the vertices size of adjacency list

$$= \sum_{v \in V} |Adj[v]|$$

$$= 2|E| \quad \text{[Undirected graph]}$$

{Using Handshaking lemma}.

⇒ We also touch every vertex to visit the edges connected to it.

∴ Total running time = $O(V+E)$

where V = no. of vertices.
E = no. of edges.

Space complexity = $O(V)$

where V is no. of vertices.

↳ For storing a visited array.