


Algorithmics	Student information	Date	Number of session
	UO: UO294067	03/04/24	6
	Surname: Díaz Álvarez	 Escuela de Ingeniería Informática <small>Universidad de Oviedo</small>	
	Name: Paula		



Activity 1. Backtracking

```
private void backtracking(int i, int j) {
    if (j >= size - 2) { // we reached the = number
        i = i + 2;
        j = 0;
    }
    if (counterSol == emptySlots) { // Solution found
        solutionFound = true;
        printSolution(); // O(n^2)
    } else {
        if (canBeModified[i][j]) { //It's an ?
            for (int k = 0; k <= 9; k++) { // Possible numbers O(n)
                board[i][j] = "" + k;
                if (!solutionFound && validRow(i) && validColumn(j)) {
                    counterSol++; //valid number found
                    backtracking(i, j + 2); // Recursive call to next number
                    if (solutionFound)
                        break;
                    board[i][j] = "?"; //reverse changes
                    counterSol--; //not valid number found
                } else {
                    board[i][j] = "?"; //reverse changes
                }
            }
        } else if (!solutionFound) { // already given number or operation
            backtracking(i, j + 2);
        }
    }
}
```

Algorithmics	Student information	Date	Number of session
	UO: UO294067	03/04/24	6
	Surname: Díaz Álvarez		
	Name: Paula		

Maybe it could be better with branch and bound or dynamic programming by storing the invalid or valid combinations for each row, so we don't have to repeat the development of that branch or combination again. For example:

If we try 1 1 1, 3 3 3, 4 4 4, and at the end 4 4 4 is not valid because it doesn't achieve the result of the row, this isn't a solution. We can store it, so if we try with 2 2 2, 6 6 6, 4 4 4, we also know that isn't a solution already.

Test case	Time for first solution (ms)	Time for all the solutions (ms)	Number of solutions found
Test00	LoR (0.051 ms)	LoR (0.076 ms)	1
Test01	LoR (0.102 ms)	LoR (5.9ms)	12
Test02	LoR (0.106 ms)	LoR (0.4 ms)	1
Test03	LoR (0.524 ms)	LoR (22.67 ms)	3
Test04	101 ms	109 ms	2
Test05	LoR (0.33 ms)	LoR (25.5 ms)	5
Test06	LoR (0.285 ms)	329 ms	83
Test07	204 ms	OoT	more than 85