**Program Structure**

The memorization table consists of 3 items: mem[index][lastRoutine][streak]

|  |  |
| --- | --- |
| Dimension | Meaning |
| Index | The current day (or routine index, from 0 to n – 1) |
| lastRoutine | The routine chosen the previous day (0 = A, 1 = B, 2 = C, 3 = none) |
| Streak | Number of consecutive times the same routine has been done (0 to 2) |

Thus, on each recursive calls, the algorithm will:

* Try all 3 routes (A, B, C)
* Only continue if streak limit (<= 2) is not violated
* Chooses the maximum score among the valid transitions

Thus, there are 3 possible states that the cells can be populated in the table in the following way as shown in the image below.

* With each cell consisting of an array of length 3 to store the current streak (how many times the exercise has been completed consecutively)
* Furthermore, at each state, the current max score will also be stored in each cell as shown in Figure 2 as the table populates.

A graph with lines drawn on it

AI-generated content may be incorrect.

Figure 1: mem table storing the current state

A white grid with blue writing

AI-generated content may be incorrect.

Figure 2: score table stored in each state

**Time Complexity Analysis**

int KnapSack(int n, int[] T) - - O(n) time in total

* O(1) - - No more days left
* O(1) - - Determine which routine was done last and how many times
  + O(1) - - for (int r = 0; r < 3; r++) (since the loop will only run for 3 times)
* O(1) - - return already computed value from mem
* O(n) (loop will run for 3 times) - - Try each routine A(0), B(1), C(2)
  + O(1) - - cannot do the same exercise more than 2 times
  + O(1) - - Create a new array to represent T for the next recursion (runs in O(1) times since the array will always be of length 3)
  + O(1) - - If same routine is chosen again, increment streak, else (different routine chosen), reset streak to 1
  + knapsack = max(KnapSack(n + SA, T[A] + 1), KnapSack(n + SB, T[B] + 1), KnapSack(n + SC, T[C] + 1))
    - since there are n days, 3 unique states in each cell (as it is not possible to do more than 2 of the same exercise consecutively) & 4 possible lastRoutine(0=A, 1=B, 2=C, 3=None) 🡪 n \* 3 \* 4 = 12n = O(n) subproblems in the question
    - furthermore, the code will solve each subproblem in O(1) time
    - therefore, the entire code will run for O(n) time
  + O(1) - - Memoize the result for the current state & find max

int maxScore(int[] A, int[] B, int[] C)

* O(n) - - Merge into 2D array for easier indexing
* O(n) - - Fill memo table with uncomputed values
* O(n) - - Initialise T[] with 0s (no consecutive streaks yet)