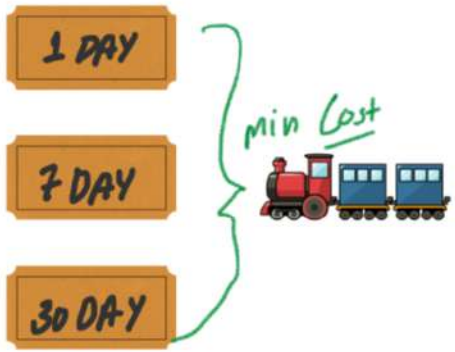
2. Minimum Cost for Tickets (Leetcode-983)

Problem Statement:
You have planned some train traveling one year in advance.
The days of the year in which you will travel are given as an integer array days.
Each day is an integer from 1 to 365.



Train tickets are sold in three different ways:
Way1: a 1-day pass is sold for costs[0] dollars,
Way2: a 7-day pass is sold for costs[1] dollars, and
Way3: a 30-day pass is sold for costs[2] dollars.

Note: The passes allow that many days of consecutive travel.
For example, if we get a 7-day pass on day 2, then we can travel for 7 days: 2, 3, 4, 5, 6, 7, and 8.

ANS: Return the minimum number of dollars you need to travel every day in the given list of days.

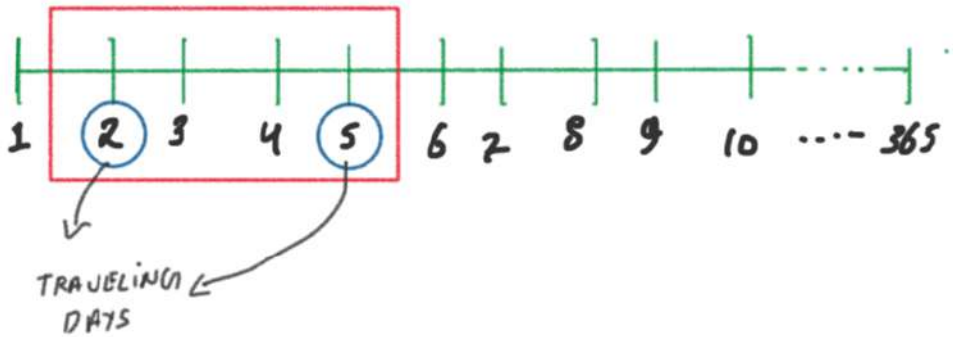
Example 1:
Input: days = [2,5], costs = [1,4,25]
Output: 2

Example 2:
Input: days = [1,2,3,4,5,6,7,8,9,10,30,31], costs = [2,7,15]
Output: 17

Example 3:
Input: days = [1,4,6,7,8,20], costs = [2,7,15]
Output: 11

Explanation

Example 1:
Input: days = [2,5], costs = [1,4,25]
Output: 2



CASE 1 we get 1 DAY pass ON 2 DAY = 1 RS.
" " 7 " " " " " = 4 RS.
" " 30 " " " " " = 25 RS.
} min = 1 RS. cost

{ 1 din me 3 trains se travel kar skta hun. }

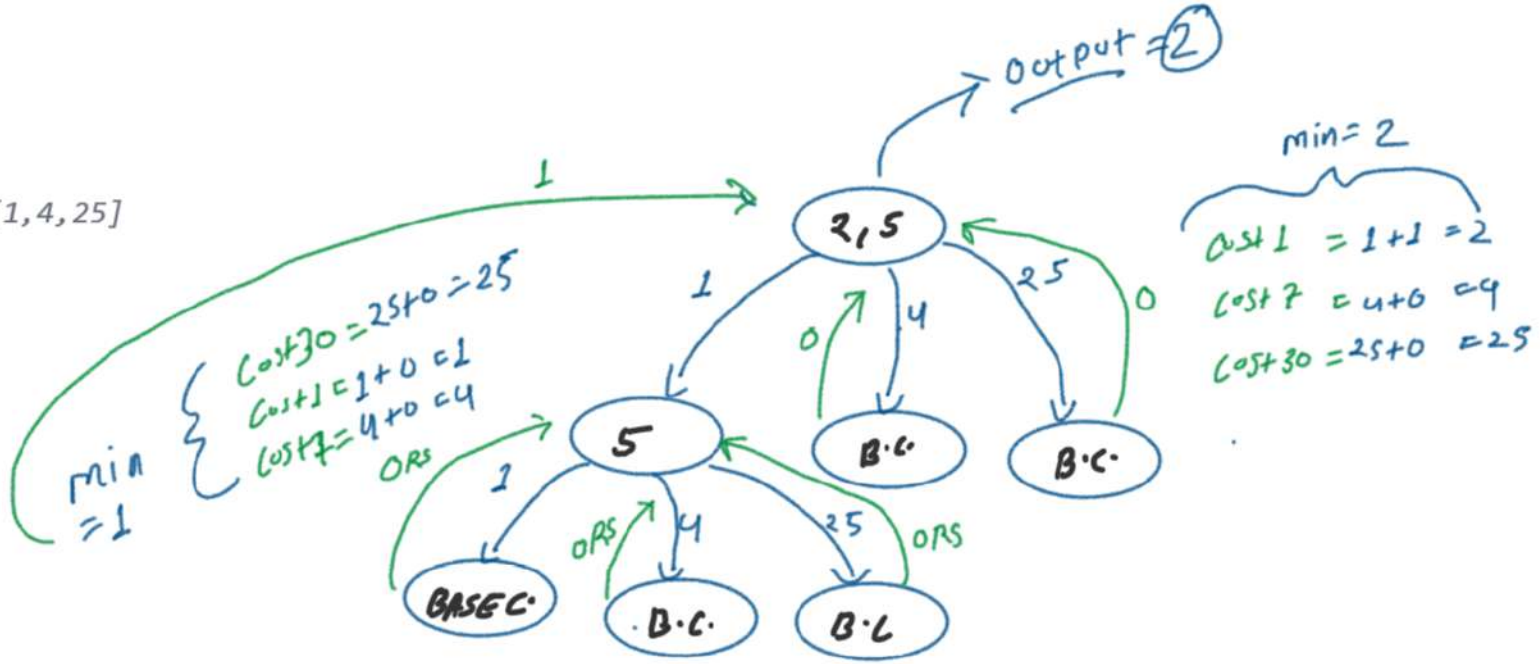
+

CASE 2 we get 1 DAY pass ON 5 DAY = 1 RS.
" " 7 " " " " " = 4 RS.
" " 30 " " " " " = 25 RS.
} min = 1 RS. cost

Total ⇒ 2 RS

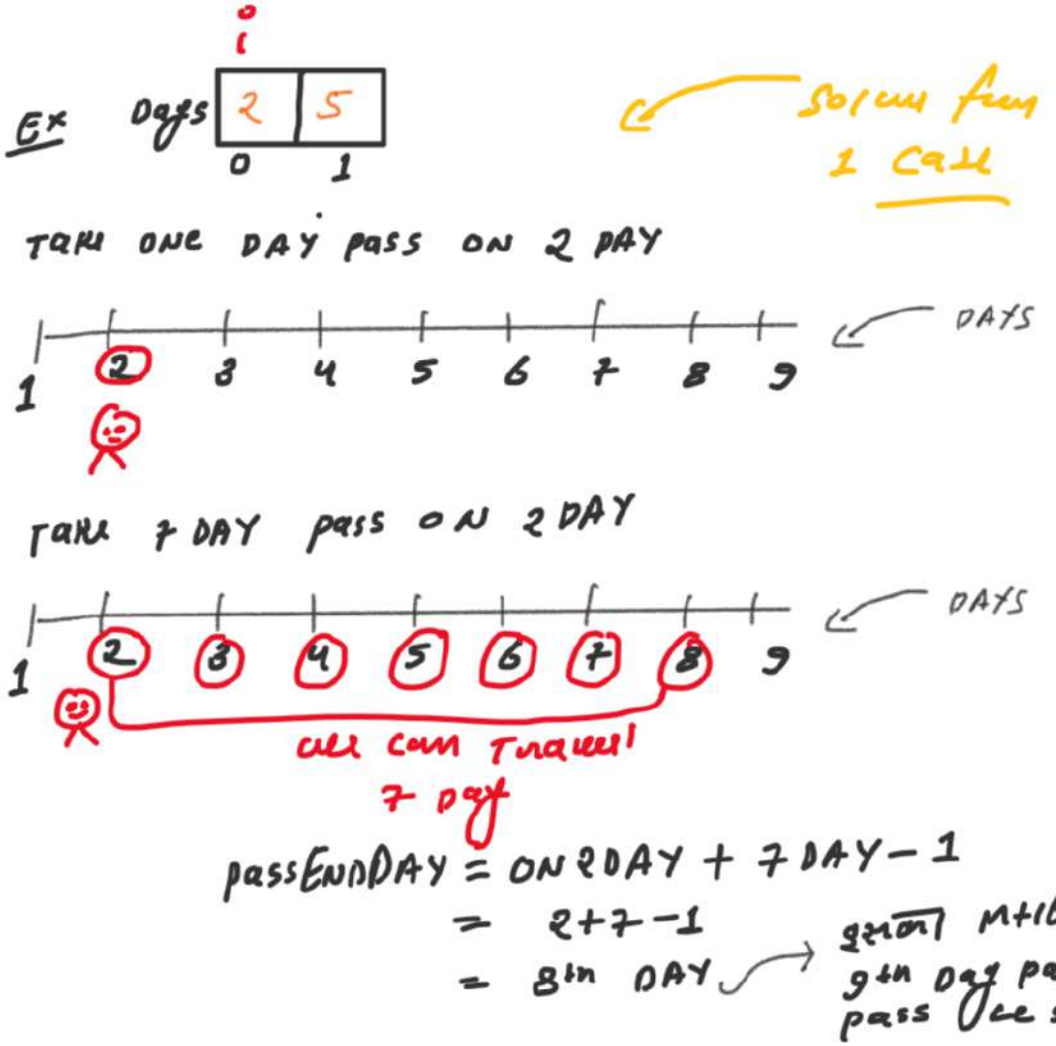
Solve using Rec

Example 1:
Input: days = [2,5], costs = [1,4,25]
Output: 2



```cpp
// 2. Minimum Cost for Tickets (Leetcode-983)
// Approach 1: Normal Recursion
// Time Complexity: O(3)^N
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingRec(vector<int>& days, vector<int>& costs, int i){
        // Base case
        if(i >= days.size()){
            return 0;
        }

        // Solve for one case in three different ways
        // Way 1: 1 Day Pass Taken
        int cost1 = costs[0] + solveUsingRec(days, costs, i + 1);

        // Way 2: 7 Day Pass Taken
        int passEndDay = days[i] + 7 - 1;
        int j = i;
        while(j < days.size() && days[j] <= passEndDay){
            // consecutive travel: 7daysPass se me kis kis din travel kar skta hu
            // and mujhe next pass kab kharidna hai uske j update kar doonga
            j++;
        }
        int cost7 = costs[1] + solveUsingRec(days, costs, j);

        // Way 3: 30 Day Pass Taken
        passEndDay = days[i] + 30 - 1;
        j = i;
        while(j < days.size() && days[j] <= passEndDay){
            j++;
        }
        int cost30 = costs[2] + solveUsingRec(days, costs, j);

        // Return ans
        int ans = min(cost1, min(cost7, cost30));
        return ans;

    }
    int mincostTickets(vector<int>& days, vector<int>& costs) {
        int ans = solveUsingRec(days, costs, 0);
        return ans;
    }
};
```

TLE

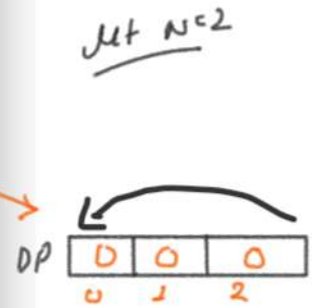


1 DP

```cpp
// 2. Minimum Cost for Tickets (Leetcode-983)
// Approach 2: Top Down
// Time Complexity: O(3N)
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingMemo(vector<int>& days, vector<int>& costs, int i, vector<int> &dp){
        // Base case
        if(i >= days.size()){
            return 0;
        }

        // Step 3: if ans already exist then return ans
        if(dp[i] != -1){
            return dp[i];
        }

        // Step 2: store ans and return ans using DP array
        // Recursive call:
        // Solve for one case in three different ways
        // Way 1: 1 Day Pass Taken
        int cost1 = costs[0] + solveUsingMemo(days, costs, i + 1, dp);

        // Way 2: 7 Day Pass Taken
        int passEndDay = days[i] + 7 - 1;
        int j = i;
        while(j < days.size() && days[j] <= passEndDay){
            j++;
        }
        int cost7 = costs[1] + solveUsingMemo(days, costs, j, dp);

        // Way 3: 30 Day Pass Taken
        passEndDay = days[i] + 30 - 1;
        j = i;
        while(j < days.size() && days[j] <= passEndDay){
            j++;
        }
        int cost30 = costs[2] + solveUsingMemo(days, costs, j, dp);

        // Store ans in dp
        dp[i] = min(cost1, min(cost7, cost30));
        // return ans
        return dp[i];

    }
    int mincostTickets(vector<int>& days, vector<int>& costs) {
        int n = days.size();
        // Step 1: create DP array
        vector<int> dp(n+1, -1);
        int ans = solveUsingMemo(days, costs, 0, dp);
        return ans;
    }
};
```

1 DP

```cpp
// 2. Minimum Cost for Tickets (Leetcode-983)
// Approach 3: Bottom-up
// Time Complexity: O(N)^2
// Space Complexity: O(N)

class Solution {
public:
    int solveUsingTabu(vector<int>& days, vector<int>& costs){
        // Step 1: create DP array
        // Step 2: fill initial data in DP array according to recursion base case
        int n = days.size();
        vector<int> dp(n+1, 0);

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int i = n-1; i >= 0; i--){
            // Solve for one case in three different ways
            // Way 1: 1 Day Pass Taken
            int cost1 = costs[0] + dp[i + 1];

            // Way 2: 7 Day Pass Taken
            int passEndDay = days[i] + 7 - 1;
            int j = i;
            while(j < days.size() && days[j] <= passEndDay){
                j++;
            }
            int cost7 = costs[1] + dp[j];

            // Way 3: 30 Day Pass Taken
            passEndDay = days[i] + 30 - 1;
            j = i;
            while(j < days.size() && days[j] <= passEndDay){
                j++;
            }
            int cost30 = costs[2] + dp[j];
            dp[i] = min(cost1, min(cost7, cost30));
        }

        // Return ans
        return dp[0];
    }
    int mincostTickets(vector<int>& days, vector<int>& costs) {
        int ans = solveUsingTabu(days, costs);
        return ans;
    }
};
```



Ex1 10-02-2024

(Rec + DP) (983)

[Minimum cost for Ticket]

Sold tickets in 3-diff ways

cost-1 @ 1-day pass at costs[0] dollars

cost-2 @ 7-day " " cost[1] dollars

cost-3 @ 30-day " " cost[2] dollars

days: | 1 | 4 | 6 | 7 | 8 | 20 |
idx: 0 1 2 3 4 5

costs: | 2 | 7 | 15 |
0 1 2

Note: Consecutive travel Allowed

Ex: we get 7-day pass on day (2)

↳ we can travel for 7 days from [2, 3, 4, 5, 6, 7, 8]

Return minimum No of dollars we need to travel every day in the given list of days)

Must do DRY RUN TO RE-TRACE

{T.C = O(3^n), S.C = O(N)}

Ex: days[2,5], costs[1,4,25] } output = 2 Rs

T.C = O(N^3)
S.C = O(N)

→ Tabulation