Homework Assignment 04

Name: **Asrar Syed**
Due by: **November 17, 2025**
Course Section: **CSC 4520-006**

Recursive Algorithm Design Paradigm: SRTBOT Analysis

- S — Subproblem definition
- R — Relate subproblem solutions recursively
- T — Topological order on subproblems
- B — Base cases
- O — Original problem
- T — Time analysis

Notes: Dynamic programming requires two core ideas

- Optimal substructure — the solution can be broken into subproblems.
- Overlapping subproblems — the same subproblems repeat.

Dynamic Programming = solving subproblems + reusing results

- Top-down DP: recursion + memoization
- Bottom-up DP: tabulation + iteration

Outside Resources Used:

https://www.youtube.com/watch?v=r4-cftqTcdI

https://martinlwx.github.io/en/solving-dynamic-programming-problems-using-srtbot/

Answers start next page

Problem 01 (30 pts)

- S — Subproblem definition

  dp[i] is the minimum number of jumps needed to reach the last platform (index n-1) starting from platform i. If it is impossible from i, dp[i] = ∞.

- R — Relate subproblem solutions recursively

  For each platform i:
  dp[i] = 1 + min(dp[j]) for all j such that i < j ≤ i + jumps[i]
  If no valid j exists, dp[i] = ∞.

- T — Topological order on subproblems

  Compute dp values from i = n-1 down to 0, because dp[i] depends only on dp[j] where j > i.

- B — Base cases

  1. dp[n-1] = 0 (already at the goal).
  2. If jumps[i] = 0 and i ≠ n-1, then dp[i] = ∞.

- O — Original problem

  Return dp[0]. If dp[0] = ∞, return -1 because the last platform is unreachable.

- T — Time analysis

  Worst-case each dp[i] checks up to n transitions then O(n^2) time and O(n) space.

Problem 02 (30 pts)

- S — Subproblem definition

  dp[i] = maximum number of rounds the player can still win starting from deck position i (meaning the next unused card is cards[i]).

  If fewer than 5 cards remain then dp[i] = 0.

- R — Relate subproblem solutions recursively

  Dealer hand: d = cards[i] + cards[i+1]
  Player 2-card hand: p2 = cards[i+2] + cards[i+3]
  Player 3-card hand: p3 = p2 + cards[i+4]

  win2 = 1 if (p2 ≤ 21 and p2 > d) else 0
  win3 = 1 if (p3 ≤ 21 and p3 > d) else 0

  dp[i] = max(win2 + dp[i+4], win3 + dp[i+5])
  (3-card option valid only if enough cards remain)

- T — Topological order on subproblems

  Compute dp from i = n down to 0 because dp[i] depends only on dp[j] for j > i.

- B — Base cases

  1. If n - i < 5, then dp[i] = 0 (cannot play another round).
  2. Values beyond end of array (i ≥ n) is dp[i] = 0.

- O — Original problem

  The final answer is dp[0].

- T — Time analysis

  We compute dp[i] once and each step does O(1) work -> O(n) time, O(n) space.

Problem 03 (40 pts)

- S — Subproblem definition

  dp[i][c] = minimum cost to paint houses from index i to the end, if house i is painted with color c.

  Where:
  c = 0 is red
  c = 1 is green
  c = 2 is blue

- R — Relate subproblem solutions recursively

  House i painted with color c, so the next house (i+1) must be any color other than c:

  dp[i][c] = costs[i][c] + min(dp[i+1][c'] for c' != c)

- T — Topological order on subproblems

  We compute the DP from:

  i = n-1 down to 0

  (last house to first house)
  Because dp[i] depends on dp[i+1].

- B — Base cases

  For the last house (i = n-1):

  dp[n-1][c] = costs[n-1][c]

  (just paint the last house and pay its cost)

- O — Original problem

  We want the best color choice for house 0:

  answer = min(dp[0][0], dp[0][1], dp[0][2])

- T — Time analysis

  Each dp state checks 2 other colors; there are 3n states -> O(n) time and O(n) space