(A)

Ideas

Notice that among three options, $C(i-1, j-1)$ and $C(i+2, j-2)$ uses j smaller than current indicies. Thus we need to evalulate j from 1 to n. The remaining one option $C(i+j, j)$ uses i bigger than current indicies, that means we need to evaluate i from n to i, in **decreasing** order. We notice that all three options only uses results in position $C(*, j)$, $C(*, j-1)$, and $C(*, j-2)$, so we only need to store at most $3n$ entries.

Algorithm

Declare 2D array $A[3, n]$, let $A[0, *] = 0$ and $A[1, *] = 0$. **Takes O(n)**

For $j : 0 \to n$;

- For $i : n \to 0$:
  - Declare three options O_1, O_2, O_3
  - If $j-1 \geq 1$ and $i-1 \geq 1$, let $O_1 = A[0, i-1] + f(i, j)$, otherwise $O_1 = f(i, j)$ **Takes O(1)**
  - If $j-2 \leq n$ and $i+2 \geq 1$, let $O_2 = A[1, i+2] + g(i, j)$, otherwise $O_2 = g(i, j)$. **Takes O(1)**
  - If $i+j \leq n$, let $O_3 = A[2, i+j] + h(i+j)$, otherwise $O_3 = h(i+j)$. **Takes O(1)**
  - Let $A[2, i] = \max(O_1, O_2, O_3)$. **Takes O(1)**
- Let $A[0, *] = A[1, *]$; Let $A[1, *] = A[2, *]$. **Takes O(n)**

Return $A[2, n]$

Note: Time complexity are noted on in bold. The notation $A[0, *]$ is a shorthand for the array slice, where $*$ is the wildcard.

Complexity

Since f, g, h are constant function, and we loop each i, j once, the total time complexity is $O(n^2)$. Since we only store $A[3, n]$ which requires $3n$ space, the space complexity is $O(n)$.

(B)

Idea

Notice that $C(i, j)$ depends on only subproblems with $k > j$, therefore we need to evaluate j from n to 0, in **decreasing** order. It does not depend on subproblem with same j, therefore we could evaluate i in either increasing or decreasing order. Notice that it requires all us to keep all rows of previous data (if we think j represents rows), and we could not really do optimization as f, g depends on i, j and k. Given that we only need to calculate $O(\lfloor n/2 \rfloor, 1)$, technically all the configuration need to be calculated is like a "waterfall", but that only saves constant factor time and space, so it is not in the discussion.

Algorithm

Declare 2D array $A[n, n]$, where

$$A[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ and } i = n \\ 0 & \text{if } j = n \\ \infty & \text{otherwise} \end{cases}$$

For $j : n-1 \to 0$

- For $i : 1 \to n-1$
  - Let minimum value $m = \infty$
  - For $k \in [j+1, n]$ such that $g(i, j, k) > 0$:
    - Update $m = \min(m, A[i-1, k] + A[i+1, k] + f(i, j, k))$ **Takes O(1)**
  - Update $A[i, j] = m$ **Takes O(1)**

Return $A[\lfloor n/2 \rfloor, 1]$

Complexity

Since we compute all $A[i, j]$, we have $O(n^2)$ tasks, in each tasks, we iterate k which in worst case will have $O(n)$ iterations, in each iteration, we update m with $O(1)$ tasks as f, g are both constant function. So, the time complexity is $O(n^3)$. Since we store $A[i, j]$, the space complexity is $O(n^2)$.