CS 390 HW 2 Q1

Aaryan Wadhwani

February 7, 2025

Problem 2

Algorithm 1 Longest Convex Subsequence

Let LCS(i, j) represent the length of the longest convex subsequence ending with A[i] as the second to last element and A[j] as the last element, where i < j.

```
function LCS(i, j)
    max_len ← 2
    for k ← 1 to i − 1 do
        if A[j] − A[i] ≥ A[i] − A[k] then
            max_len ← max(max_len, LCS(k, i) + 1)
    return max_len
End function
```

return $\max\{\text{LCS}(i, j) \mid 1 \leq i < j \leq n\}$

We apply dynamic programming to this problem by caching the solutions to the subcalls. There are n^2 subproblems and we spend $O(n)$ time on each of them excluding recursive calls so the function takes $O(n^3)$ time.

Problem 4

Algorithm 2 Longest odd/even LIS

Let OELIS(i) represent a tuple (e, o), where e is the length of the longest increasing subsequence with an even sum and o is the length of the longest increasing subsequence with an odd sum

```
function OELIS(i)                          ▷ Returns (even, odd) for subsequences ending at i
    if A[i] even then
        e, o ← 1, 0
    else
        e, o ← 0, 1
    for j ← 1 to i − 1 do
        if A[j] < A[i] then
            (e_prev, o_prev) ← OELIS(j)
            if A[i] even then
                e ← max(e, e_prev + 1)
                o ← max(o, o_prev + 1)
            else
                e ← max(e, o_prev + 1)
                o ← max(o, e_prev + 1)
    return (e, o)
End function
```

return max(OELIS(i)) for all i

We apply dynamic programming to this problem by caching the solutions to the subcalls. There are n subproblems and we spend $O(n)$ time on each of them excluding recursive calls so this is an $O(n^2)$ algorithm.

Problem 5

Algorithm 3 AmericanLIS

Let AMERICANLIS(i) represent the length of the longest increasing subsequence ending at index i, where the colors of the elements in the subsequence follow the American flag color order (red, white, blue, red, white, blue, ...).

```
function AmericanLIS(i)
    max_len ← 1
    for j ← 1 to i − 1 do
        if A[j] < A[i] and color[i] is next in sequence from color[j] then
            current ← AmericanLIS(j) + 1
            max_len ← max(max_len, current)
    return max_len
End function
```

return $\max\{\text{AmericanLIS}(i) \mid 1 \le i \le n\}$

We apply dynamic programming to this problem by caching the solutions to the subcalls. There are n subproblems and we spend $O(n)$ time on each of them excluding recursive calls so this is an $O(n^2)$ algorithm.

Problem 6

Algorithm 4 Longest Palindrome Subsequence

Let LPS(i, j) represent the length of the longest palindromic subsequence in the substring of A starting at index i and ending at index j (inclusive).

function $\mathrm{LPS}(i, j)$ ▷ Length in substring $A[i..j]$

  if $i > j$ **then return** 0

  else if $i = j$ **then return** 1

  return $\max \begin{cases} 2 + \mathrm{LPS}(i+1, j-1) \\ \mathrm{LPS}(i+1, j) \\ \mathrm{LPS}(i, j-1) \end{cases}$

End function

return $\mathrm{LPS}(1, n)$

We apply dynamic programming to this problem by caching the solutions to the subcalls. There are n^2 subproblems and we spend $O(1)$ time on each of them excluding recursive calls so this is an $O(n^2)$ algorithm.

Problem 9: Longest Increasing Subsequence with Sum Divisible by Its Length

Given an array $A[1..n]$ of integers, compute the length of the longest increasing subsequence such that the sum of the elements in the subsequence is divisible by the number of elements in the subsequence.

Formally, find the maximum k such that there exists a subsequence $A[i_1], A[i_2], \ldots, A[i_k]$ where:

- $i_1 < i_2 < \cdots < i_k$,
- $A[i_1] < A[i_2] < \cdots < A[i_k]$, and
- $(A[i_1] + A[i_2] + \cdots + A[i_k]) \equiv 0 \pmod{k}$.

Algorithm 5 DivideLIS

Computes the length of the longest increasing subsequence such that the sum of the elements in the subsequence is divisible by the number of elements in the subsequence.

function $\textsc{DivideLIS}(i, k, \texttt{mod}, \texttt{last})$ ▷ prolly a much simpler way to do this which is easier to visualize as well but this guarantees (?) ig
    if $i = 0$ **then**
        if $k = 0$ **then return** 0
        if $\texttt{mod} = 0$ **then return** k
        elsereturn -1
    $\texttt{skip} \leftarrow \text{DivideLIS}(i-1, k, \texttt{mod}, \texttt{last})$
    $\texttt{take} \leftarrow -1$
    if $A[i] < \texttt{last}$ **then** ▷ Ensure increasing order
        $\texttt{new\_k} \leftarrow k + 1$
        $\texttt{new\_mod} \leftarrow (\texttt{mod} \cdot k + A[i]) \bmod \texttt{new\_k}$ ▷ Update modulus
        $\texttt{result} \leftarrow \text{Divide LIS}(i-1, \texttt{new\_k}, \texttt{new\_mod}, A[i])$
        if $\texttt{result} \neq -1$ **then**
            $\texttt{take} \leftarrow \texttt{result} + 1$
    return $\texttt{DP}[i][k][\texttt{mod}][\texttt{last}]$
End function

return $\max_{k \in [1,n]} \text{DivideLIS}(n, 0, 0, \infty)$ ▷ Start with $k = 0$, $\texttt{mod} = 0$

We apply dynamic programming to this problem by caching the solutions to the subcalls. There are n^4 subproblems and we spend $O(1)$ time on each of them excluding recursive calls so this is an $O(n^4)$ algorithm.
