Problem Set 4

Due dates: Electronic submission of the pdf file of this homework is due on **2/17/2023 before 11.59pm** on canvas.

Name: Ayushri Jain

Resources. Below resources were used for reference -

- Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3rd edition, The MIT Press, 2009 (or 4th edition)
- https://www.overleaf.com/learn/latex/Integrals%2C_sums_and_limits
- https://www.cse.iitk.ac.in/users/manindra/CS681/2005/Lecture6.pdf
- https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework. The solutions given in this homework are my own work.

Signature: 

Problem 1. (20 points)

Solution. We know that if we multiply a matrix A with its inverse A^{-1}, we get the identity matrix, i.e. $A \cdot A^{-1} = I$ in which the values at the main diagonal are 1 and all other values are 0. So, to show that inverse of $F = (\omega^{ij})_{i,j\in[0..n-1]}$ is given by $F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k\in[0..n-1]}$, we will show that $F \cdot F^{-1} = I$.
F is $i \times j$ matrix and F^{-1} is $j \times k$ matrix. Let V $i \times k$ denote the multiplication matrix of F and F^{-1}. Consider an entry of V at [i,k] = $V_{i,k}$

$$V_{i,k} = \sum_{j=0}^{n-1}(\omega^{ij})(\omega^{-kj}/n) = \sum_{j=0}^{n-1}(\omega^{ij-kj})/n = \sum_{j=0}^{n-1}(\omega^{j(i-k)})/n$$

Here, if $i = k$, then $V_{i,k} = \sum_{j=0}^{n-1}(\omega^0)/n = (1 + 1 + \cdots + 1)/n = 1$.
This means that the values of main diagonal of matrix V are 1.
We also know that a geometric or exponential series summation is given by

$$\sum_{j=0}^{n} x^j = \frac{x^{n+1} - 1}{x - 1}$$

Keeping $x = \omega^{i-k}$ and summation from j=0 to n-1, we get

$$V_{i,k} = \sum_{j=0}^{n-1}(\omega^{j(i-k)})/n = \frac{1}{n}\frac{\omega^{((n-1)+1)(i-k)} - 1}{\omega^{i-k} - 1} = \frac{1}{n}\frac{\omega^{n(i-k)} - 1}{\omega^{(i-k)} - 1}$$

Since ω is n^{th} root of 1, therefore $\omega^{n(i-k)} = 1^{i-k} = 1$, substituting this in above result, we get

$$V_{i,k} = \frac{1}{n}\frac{1 - 1}{\omega^{(i-k)} - 1} = 0$$

It means that when $i \neq k$, then the value $V_{i,k} = 0$. Thus, matrix V is an identity matrix.
We have proved that product matrix V of F and F^{-1} is an identity matrix. Hence the assumption of inverse that $F = (\omega^{ij})_{i,j\in[0..n-1]}$ is equal to $F^{-1} = \frac{1}{n}(\omega^{-jk})_{j,k\in[0..n-1]}$ is true.

Problem 2. (20 points)

Solution. Assumption : n is a power of 2. If not, then we will pad the polynomial so that their degree becomes next power of 2, implying n is power of 2.
Let $A(x)$ and $B(x)$ be two degree-bound n polynomials, i.e. $0 \leq$ degree $\leq n-1$. If polynomial $C(x)$ represents their product, degree(C) = degree(A) + degree(B), implying that if A is a polynomial of degree-bound n_a and B is a polynomial of degree-bound n_b, then C is a polynomial of degree-bound $n_a + n_b - 1$. Since a polynomial of degree-bound k is also a polynomial of degree-bound k + 1, we will normally say that the product polynomial C is a polynomial of degree-bound n_a + n_b. To multiply A(x) and B(x), we should first represent both of them in coefficient representation.

$$A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$$

$$B(x) = b_0x^0 + b_1x^1 + \cdots + b_{n-1}x^{n-1}$$

The resultant polynomial C(x) will be degree-bound 2n, i.e. $0 \leq$ degree of C(x) $\leq 2n - 1$.

$$C(x) = c_0x^0 + c_1x^1 + \cdots + c_{2n-1}x^{2n-1}$$

Therefore, we should append zeros to both coefficient vectors so that their length is $2n$. This is because the FFT requires that the length of the input be a power of two, and we want to ensure that both vectors have the same length. Then, we perform the FFT on both coefficient vectors, resulting in two point-value representations, $A(\omega)$ and $B(\omega)$, where ω is a primitive $2n$th root of unity. To obtain the point-value representation of the product polynomial $C(\omega) = A(\omega)B(\omega)$, we should then perform pointwise multiplication of $A(\omega)$ and $B(\omega)$. We then perform the IFFT on the resulting point-value representation $C(\omega)$ to obtain the coefficient representation of the product polynomial, $C(x)$. If required, we should trim the result coefficient vector to length $n-1$ (if originally, it was not a power of 2) to obtain the coefficient representation of the product polynomial $C(x)$. The length of the FFT and IFFT needed is $2n$, where n is the degree of the original polynomials. This is because we need to append zeros to the original coefficient vectors to ensure they have the same length, and this new length is $2n$.

Problem 3. (20 points)

Solution. In the polynomial multiplication algorithm based on FFT and IFFT, a polynomial is represented as $A(x) = a_0x^0 + a_1x^1 + \cdots + a_{n-1}x^{n-1}$. We can replace x by 10 for our requirement, for example 234 can be represented as $2 * 10^2 + 3 * 10^1 + 4 * 10^0$. This will give us the point value representation of number and like we represent the polynomials in degree-bounds of 2n, we will pad our numbers so that they have same length also. Then using the FFT algorithm, we will find the value of each integer at $2n^{th}$ roots of unity. These roots of unity are same as the complex numbers which are present on the unit circle. Once we have the point value representations, we will multiply them together point-wise. Then we will perform IFFT algorithm and get output polynomial c. We will handle the carries by adding them to the next pair of coefficients as we multiply them.
Note :

- FFT(A) : cooley-tukey algorithm to compute fast fourier transform of A
- IFFT(A) : cooley-tukey algorithm to compute inverse fast fourier transform of A
- convertToPolynomial(a) converts integer a to polynomial representation
- padZeros(A,n) pads a polynomial A with zeros to make it of length n
- nextPowerOfTwo(a) gets smallest power of two greater $\geq a$
- removeLeadingZeroes(a) removes any leading 0s present in a

```
function multiply_integers(a, b):
    # Convert integers to polynomial representations
    A = convertToPolynomial(a)
    B = convertToPolynomial(b)
    # Pad polynomials with zeros so that their length is equal and next power of 2
    n = nextPowerOfTwo(max(length(A), length(B)))
    A = padZeros(A, n)
    B = padZeros(B, n)
    # Apply the FFT algorithm to the polynomial representations
    Afft = FFT(A)
    Bfft = FFT(B)
    # pointwise multiply both
    Cfft = Afft * Bfft
    # Apply IFFT algorithm
    c = IFFT(Cfft)
    # get back the integer from product polynomial
    ans = 0
    carry = 0
    for i = 0 to 2n-1:
        c[i] += carry
        carry = floor(c[i] / 10)
        c[i] %= 10
        ans += c[i] * 10^i

    # Remove leading zeros if present
    ans = removeLeadingZeroes(ans)

    return ans
```

Problem 4 (20 points)**.**

Solution. Let's assume we have 4 matrices A, B, C, D with dimensions such that $p_0, p_1, p_2, p_3, p_4 = 70, 30, 6, 3, 70$. If we follow Professor Capulet's approach, then $p_0 p_k p_4$ is minimum when $k = 3$. So we need to solve subproblem of multiplying ABC and D. By her algorithm, ABC will split at 2, i.e., overall we should follow this parenthesization - $((AB)C)D$). Cost to multiply the matrices in this way is given by number of scalar multiplications =

$$70 * 30 * 6 + 70 * 6 * 3 + 70 * 3 * 70 = 28560$$

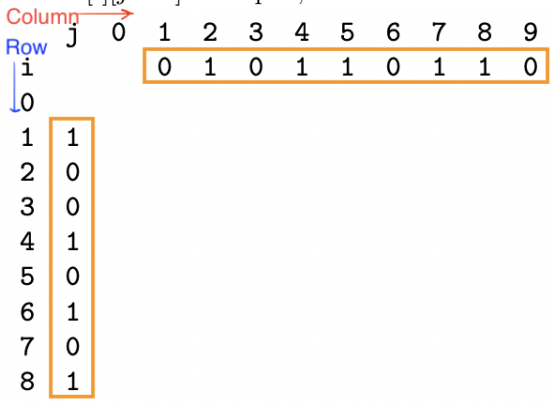
But if we considered another order, i.e. followed paranthesization like $((A(BC))D)$, then cost will be given by these many scalar multiplications =

$$30 * 6 * 3 + 70 * 30 * 3 + 70 * 3 * 70 = 21540$$

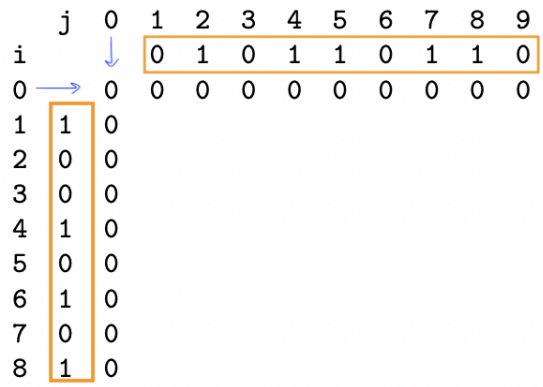
21540 is less than 28650 obtained from Professor Capulet's method. So, in this case her greedy approach yields suboptimal solution.

Problem 5 (20 points)**.**

Solution. We will use dynamic programming to find LCS of given 2 sequences. First, we will create a 2D array lcs with dimensions $(8+1) \times (9+1)$, i.e. 9×10. Please note that we don't need the values below the main diagonal. Also, in the upcoming representations, first row and first column actually represent the row numbers and column numbers; second row and second column represents the sequences given in question. We will assume i to be row iterator and j to be column iterator. If $seq1[i] = seq2[j]$, then $lcs[i][j] = lcs[i-1][j-1] + 1$. Otherwise, $lcs[i][j] = max(lcs[i-1][j], lcs[i][j-1])$, in this case if $lcs[i-1][j]$ and $lcs[i][j-1]$ are equal, we will assume the value comes from $lcs[i-1][j]$.



Next, for all j columns, we will mark $lcs[0][j] = 0$. Then, for all i rows, we will mark $lcs[i][0] = 0$.



We will now fill the next row $i = 1$.
$j = 1 : 1 \neq 0$, so $lcs[1][1] = max(lcs[0][1], lcs[1][0]) = 0$
$j = 2 : 1 = 1$, so $lcs[1][2] = lcs[0][1] + 1 = 0 + 1 = 1$
$j = 3 : 1 \neq 0$, so $lcs[1][3] = max(lcs[0][3], lcs[1][2]) = 1$
$j = 4 : 1 = 1$, so $lcs[1][4] = lcs[0][3] + 1 = 0 + 1 = 1$

$j = 5 : 1 = 1$, so $lcs[1][5] = lcs[0][4] + 1 = 0 + 1 = 1$
$j = 6 : 1 \neq 0$, so $lcs[1][6] = max(lcs[0][6], lcs[1][5]) = 1$
$j = 7 : 1 = 1$, so $lcs[1][7] = lcs[0][6] + 1 = 0 + 1 = 1$
$j = 8 : 1 = 1$, so $lcs[1][8] = lcs[0][7] + 1 = 0 + 1 = 1$
$j = 9 : 1 \neq 0$, so $lcs[1][9] = max(lcs[0][9], lcs[1][8]) = 1$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
→ 1	1	0	0	1	1	1	1	1	1	1	1
2	0	0									
3	0	0									
4	1	0									
5	0	0									
6	1	0									
7	0	0									
8	1	0									

We will now fill the next row $i = 2$.
$j = 1 : 0 = 0$, so $lcs[2][1] = lcs[1][0] + 1 = 1$
$j = 2 : 0 \neq 1$, so $lcs[2][2] = max(lcs[1][2], lcs[2][1]) = 1$
$j = 3 : 0 = 0$, so $lcs[2][3] = lcs[1][2] + 1 = 2$
$j = 4 : 0 \neq 1$, so $lcs[2][4] = max(lcs[1][4], lcs[2][3]) = 2$
$j = 5 : 0 \neq 1$, so $lcs[2][5] = max(lcs[1][5], lcs[2][4]) = 2$
$j = 6 : 0 = 0$, so $lcs[2][6] = lcs[1][5] + 1 = 2$
$j = 7 : 0 \neq 1$, so $lcs[2][7] = max(lcs[1][7], lcs[2][6]) = 2$
$j = 8 : 0 \neq 1$, so $lcs[2][8] = max(lcs[1][8], lcs[2][7]) = 2$
$j = 9 : 0 = 0$, so $lcs[2][9] = lcs[1][8] + 1 = 2$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
→ 2	0	0	1	1	2	2	2	2	2	2	2
3	0	0									
4	1	0									
5	0	0									
6	1	0									
7	0	0									
8	1	0									

We will now fill the next row $i = 3$.
$j = 1 : 0 = 0$, so $lcs[3][1] = lcs[2][0] + 1 = 1$
$j = 2 : 0 \neq 1$, so $lcs[3][2] = max(lcs[2][2], lcs[3][1]) = 1$
$j = 3 : 0 = 0$, so $lcs[3][3] = lcs[2][2] + 1 = 2$
$j = 4 : 0 \neq 1$, so $lcs[3][4] = max(lcs[2][4], lcs[3][3]) = 2$

$j = 5 : 0 \neq 1$, so $lcs[3][5] = max(lcs[2][5], lcs[3][4]) = 2$
$j = 6 : 0 = 0$, so $lcs[3][6] = lcs[2][5] + 1 = 3$
$j = 7 : 0 \neq 1$, so $lcs[3][7] = max(lcs[2][7], lcs[3][6]) = 3$
$j = 8 : 0 \neq 1$, so $lcs[3][8] = max(lcs[2][8], lcs[3][7]) = 3$
$j = 9 : 0 = 0$, so $lcs[3][9] = lcs[2][8] + 1 = 3$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0									
5	0	0									
6	1	0									
7	0	0									
8	1	0									

We will now fill the next row $i = 4$.
$j = 1 : 1 \neq 0$, so $lcs[4][1] = max(lcs[3][1], lcs[4][0]) = 1$
$j = 2 : 1 = 1$, so $lcs[4][2] = lcs[3][1] + 1 = 2$
$j = 3 : 1 \neq 0$, so $lcs[4][3] = max(lcs[3][3], lcs[4][2]) = 2$
$j = 4 : 1 = 1$, so $lcs[4][4] = lcs[3][3] + 1 = 3$
$j = 5 : 1 = 1$, so $lcs[4][5] = lcs[3][4] + 1 = 3$
$j = 6 : 1 \neq 0$, so $lcs[4][6] = max(lcs[3][6], lcs[4][5]) = 3$
$j = 7 : 1 = 1$, so $lcs[4][7] = lcs[3][6] + 1 = 4$
$j = 8 : 1 = 1$, so $lcs[4][8] = lcs[3][7] + 1 = 4$
$j = 9 : 1 \neq 0$, so $lcs[4][9] = max(lcs[3][9], lcs[4][8]) = 4$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
5	0	0									
6	1	0									
7	0	0									
8	1	0									

We will now fill the next row $i = 5$.
$j = 1 : 0 = 0$, so $lcs[5][1] = lcs[4][0] + 1 = 1$
$j = 2 : 0 \neq 1$, so $lcs[5][2] = max(lcs[4][2], lcs[5][1]) = 2$
$j = 3 : 0 = 0$, so $lcs[5][3] = lcs[4][2] + 1 = 3$
$j = 4 : 0 \neq 1$, so $lcs[5][4] = max(lcs[4][4], lcs[5][3]) = 3$

$j = 5 : 0 \neq 1$, so $lcs[5][5] = max(lcs[4][5], lcs[5][4]) = 3$
$j = 6 : 0 = 0$, so $lcs[5][6] = lcs[4][5] + 1 = 4$
$j = 7 : 0 \neq 1$, so $lcs[5][7] = max(lcs[4][7], lcs[5][6]) = 4$
$j = 8 : 0 \neq 1$, so $lcs[5][8] = max(lcs[4][8], lcs[5][7]) = 4$
$j = 9 : 0 = 0$, so $lcs[5][9] = lcs[4][8] + 1 = 5$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
→ 5	0	0	1	2	3	3	3	4	4	4	5
6	1	0									
7	0	0									
8	1	0									

We will now fill the next row $i = 6$.
$j = 1 : 1 \neq 0$, so $lcs[6][1] = max(lcs[5][1], lcs[6][0]) = 1$
$j = 2 : 1 = 1$, so $lcs[6][2] = lcs[5][1] + 1 = 2$
$j = 3 : 1 \neq 0$, so $lcs[6][3] = max(lcs[5][3], lcs[6][2]) = 3$
$j = 4 : 1 = 1$, so $lcs[6][4] = lcs[5][3] + 1 = 4$
$j = 5 : 1 = 1$, so $lcs[6][5] = lcs[5][4] + 1 = 4$
$j = 6 : 1 \neq 0$, so $lcs[6][6] = max(lcs[5][6], lcs[6][5]) = 4$
$j = 7 : 1 = 1$, so $lcs[6][7] = lcs[5][6] + 1 = 5$
$j = 8 : 1 = 1$, so $lcs[6][8] = lcs[5][7] + 1 = 5$
$j = 9 : 1 \neq 0$, so $lcs[6][9] = max(lcs[5][9], lcs[6][8]) = 5$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
5	0	0	1	2	3	3	3	4	4	4	5
→ 6	1	0	1	2	3	4	4	4	5	5	5
7	0	0									
8	1	0									

We will now fill the next row $i = 7$.
$j = 1 : 0 = 0$, so $lcs[7][1] = lcs[6][0] + 1 = 1$
$j = 2 : 0 \neq 1$, so $lcs[7][2] = max(lcs[6][2], lcs[7][1]) = 2$
$j = 3 : 0 = 0$, so $lcs[7][3] = lcs[6][2] + 1 = 3$
$j = 4 : 0 \neq 1$, so $lcs[7][4] = max(lcs[6][4], lcs[7][3]) = 4$

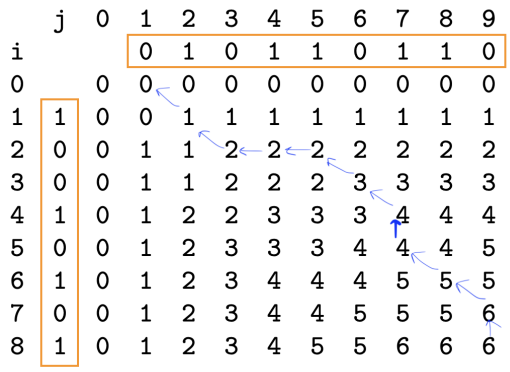
$j = 5 : 0 \neq 1$, so $lcs[7][5] = max(lcs[6]5[], lcs[7][4]) = 4$
$j = 6 : 0 = 0$, so $lcs[7][6] = lcs[6][5] + 1 = 5$
$j = 7 : 0 \neq 1$, so $lcs[7][7] = max(lcs[6][7], lcs[7][6]) = 5$
$j = 8 : 0 \neq 1$, so $lcs[7][8] = max(lcs[6][8], lcs[7][7]) = 5$
$j = 9 : 0 = 0$, so $lcs[7][9] = lcs[6][8] + 1 = 6$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
5	0	0	1	2	3	3	3	4	4	4	5
6	1	0	1	2	3	4	4	4	5	5	5
7	0	0	1	2	3	4	4	5	5	5	6
8	1	0									

We will now fill the next row $i = 8$.
$j = 1 : 1 \neq 0$, so $lcs[8][1] = max(lcs[7][1], lcs[8][0]) = 1$
$j = 2 : 1 = 1$, so $lcs[8][2] = lcs[7][1] + 1 = 2$
$j = 3 : 1 \neq 0$, so $lcs[8][3] = max(lcs[7][3], lcs[8][2]) = 3$
$j = 4 : 1 = 1$, so $lcs[8][4] = lcs[7][3] + 1 = 4$
$j = 5 : 1 = 1$, so $lcs[8][5] = lcs[7][4] + 1 = 5$
$j = 6 : 1 \neq 0$, so $lcs[8][6] = max(lcs[7][6], lcs[8][5]) = 5$
$j = 7 : 1 = 1$, so $lcs[8][7] = lcs[7][6] + 1 = 6$
$j = 8 : 1 = 1$, so $lcs[8][8] = lcs[7][7] + 1 = 6$
$j = 9 : 1 \neq 0$, so $lcs[8][9] = max(lcs[7][9], lcs[8][6]) = 6$

	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
5	0	0	1	2	3	3	3	4	4	4	5
6	1	0	1	2	3	4	4	4	5	5	5
7	0	0	1	2	3	4	4	5	5	5	6
8	1	0	1	2	3	4	5	5	6	6	6

After creating the 2-D array *lcs*, we can trace back the solution paths (arrows in figure below) for longest common subsequence. We will start from bottom left and move diagonal up-left if same characters, else up or left based on how we derived the values.



	j	0	1	2	3	4	5	6	7	8	9
i			0	1	0	1	1	0	1	1	0
0		0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3
4	1	0	1	2	2	3	3	3	4	4	4
5	0	0	1	2	3	3	3	4	4	4	5
6	1	0	1	2	3	4	4	4	5	5	5
7	0	0	1	2	3	4	4	5	5	5	6
8	1	0	1	2	3	4	5	5	6	6	6

Clearly, the length of longest common subsequence is 6 and it is $\langle 1, 0, 0, 1, 1, 0 \rangle$. Please note that there are other subsequences of the same length, like $\langle 1, 0, 1, 0, 1, 1 \rangle$, $\langle 0, 1, 0, 1, 0, 1 \rangle$, etc.