ECS 122A – Algorithm & Analysis
Homework 04 Solution

Question 1 (30 points total)

Read Section 4.2 *Strassen's algorithm for matrix mltiplications* in the textbook **Introduction to Algorithms**

1. (10 points) Use Strassen's algorithm to compute the matrix product

$$\begin{pmatrix} 3 & 8 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 & 3 \\ 4 & 9 \end{pmatrix}$$

Answer:

$$\begin{aligned} A_{11} &= [3] & A_{12} &= [8] & A_{21} &= [5] & A_{22} &= [2] \\ B_{11} &= [2] & B_{12} &= [3] & B_{21} &= [4] & B_{22} &= [9] \end{aligned}$$

$$\begin{aligned} S_1 &= B_{12} - B_{22} = [-6] \\ S_2 &= A_{11} + A_{12} = [11] \\ S_3 &= A_{21} + A_{22} = [7] \\ S_4 &= B_{21} - B_{11} = [2] \\ S_5 &= A_{11} + A_{22} = [5] \\ S_6 &= B_{11} + B_{22} = [11] \\ S_7 &= A_{12} - A_{22} = [6] \\ S_8 &= B_{21} + B_{22} = [13] \\ S_9 &= A_{11} - A_{21} = [-2] \\ S_{10} &= B_{11} + B_{12} = [5] \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 = [-18] \\ P_2 &= S_2 \cdot B_{22} = [99] \\ P_3 &= S_3 \cdot B_{11} = [14] \\ P_4 &= A_{22} \cdot S_4 = [4] \\ P_5 &= S_5 \cdot S_6 = [55] \\ P_6 &= S_7 \cdot S_8 = [78] \\ P_7 &= S_9 \cdot S_{10} = [-10] \end{aligned}$$

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 = [55 + 4 - 99 + 78] = [38] \\
C_{12} &= P_1 + P_2 = [81] \\
C_{21} &= P_3 + P_4 = [18] \\
C_{22} &= P_5 + P_1 - P_3 - P_7 = [55 - 18 - 14 + 10] = [33]
\end{aligned}
$$

$$
\begin{pmatrix} 3 & 8 \\ 5 & 2 \end{pmatrix} \cdot \begin{pmatrix} 2 & 3 \\ 4 & 9 \end{pmatrix} = \begin{pmatrix} 38 & 81 \\ 18 & 33 \end{pmatrix}
$$

2. (20 points) Write the pseudo-code for Strassen's algorithm.

   Answer:

```
strassen(A, B):
    n = number of rows in A
    if n == 1:
        return A * B

    Let C be a n * n matrix
    Partition A, B, and C into
        A11, A12, A21, A22,
        B11, B12, B21, B22,
        C11, C12, C21, C22 as in equations (4.9)

    S1 = B12 - B22
    S2 = A11 + A12
    S3 = A21 + A22
    S4 = B21 - B11
    S5 = A11 + A22
    S6 = B11 + B22
    S7 = A12 - A22
    S8 = B21 + B22
    S9 = A11 - A21
    S10 = B11 + B12

    P1 = strassen(A11, S1)
    P2 = strassen(S2, B22)
    P3 = strassen(S3, B11)
    P4 = strassen(A22, S4)
    P5 = strassen(S5, S6)
    P6 = strassen(S7, S8)
    P7 = strassen(S9, S10)

    C11 = P5 + P4 - P2 + P6
    C12 = P1 + P2
    C21 = P3 + P4
    C22 = P5 + P1 - P3 - P7

    return C
```

   (You do not have to specify how to partition the matrices using indexing.)

Question 2 (40 points total)

Given an array of strings, we want to find the longest common prefix of the strings.

1. (20 points) Provide a brute-force algorithm for the problem and analyze the time complexity of your algorithm. Write your algorithm in pseudo-code.

   Answer:

```
LCP(words[0...n-1]):
  prefix = words[0]
  for i = 1 to n-1:
    prefix = common-prefix(prefix, words[i])
  return prefix

common-prefix(w1, w2):
  prefix = ""
  m = w1.length
  n = w2.length
  i = 0
  while (w1[i] == w2[i]) and (i < m) and (i < n):
    prefix += w1[i]
    i += 1
  return prefix
```

   Time complexity: $O(n \times m)$, where n is the length of the input array of string, m is the longest string in the array.

   - The for loop goes through n iterations.
   - The runtime of `common-prefix` is $O(k)$, where k is the length of the longer input string.

2. (20 points) Provide a divide-and-conquer algorithm for the problem and analyze the time complexity of your algorithm. Write your algorithm in pseudo-code.

   [Hint: The divide-and-conquer algorithm for this problem does not improve the time efficiency.]

   Answer:

```
LCP(words[0...n-1]):
  if words.length == 0:
    return ""
  if words.length == 1:
    return words[0]
  return common-prefix(LCP(words[0...n/2]), LCP(words[(n/2)+1 ... n-1]))

common-prefix(w1, w2):
  prefix = ""
  m = w1.length
  n = w2.length
  i = 0
  while (w1[i] == w2[i]) and (i < m) and (i < n):
    prefix += w1[i]
    i += 1
  return prefix
```

   Time complexity: $O(n \times m)$, where n is the length of the input array of string, m is the longest string in the array.

   Explanation: $T(n) = 2T(\frac{n}{2}) + O(m)$, where n is the length of the input array of string, m is the longest string in the array. We didn't learn how to solve the recurrence. But every character in every string in the array will be checked so the runtime is still $O(n \times m)$.

Question 3 (10 points each, 30 points total)

The algorithm we described in class for the activity selection problem is not the only greedy algorithm. For each of the following alternative greedy choices, either **prove or disprove** that the resulting algorithm has the greedy choice property.

1. Pick the compatible activity with the earliest start time

   Answer:

   The resulting algorithm does not have the greedy choice property.

Activity	Start time	Finish time
a_1	0	10
a_2	1	2
a_3	4	5

   The greedy solution is $\{a_1\}$. The optimal solution is $\{a_2, a_3\}$.

2. Pick the compatible activity with the shortest duration

   Answer:

   The resulting algorithm does not have the greedy choice property.

Activity	Start time	Finish time
a_1	5	7
a_2	1	6
a_3	6	10

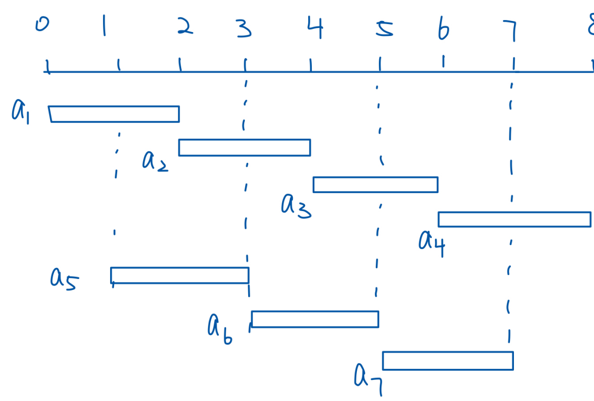
   The greedy solution is $\{a_1\}$. The optimal solution is $\{a_2, a_3\}$.

3. Pick the compatible activity that conflicts with the fewest other activities

   Answer:

   The resulting algorithm does not have the greedy choice property.

Activity	Start time	Finish time
a_1	0	2
a_2	2	4
a_3	4	6
a_4	6	8
a_5	1	3
a_6	3	5
a_7	5	7



The optimal solution is $\{a_1, a_2, a_3, a_4\}$.

Let the greedy solution be $B = \{\}$.

- Step 1: Randomly pick one from a_1 and a_4: $B = \{a_1\}$.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7
number of activities in conflict with	1	2	2	1	2	2	2

- Step 2:

	a_2	a_3	a_4	a_5	a_6	a_7
number of activities in conflict with	2	2	1	1	2	2

  a_5 is not compatible with B. Pick a_4: $B = \{a_1, a_4\}$.

- Step 3:

	a_2	a_3	a_5	a_6	a_7
number of activities in conflict with	2	2	1	2	1

  a_5 and a_7 are not compatible with B. Randomly pick one from a_2, a_3 and a_6. Pick a_6: $B = \{a_1, a_4, a_6\}$.

None of the remaining activicities is compatible with B. So $B = \{a_1, a_4, a_6\}$, smaller than the optimal solution.