Problem 1. (20 points [greedy])

Find a maximum-size subset of non-overlapping intervals (or non-conflicting tasks) from the set of 26 intervals shown below, according to the greedy strategy we discussed in class for Activity Selection. Each letter labels a single interval, e.g., --A-- is an interval five units long; intervals --A--, ---H--, ---O--- and ---V--- are overlapping.

Circle the intervals you take. Put an X through each interval you don’t take.

Answer: We select tasks by early finish time, as explained in class. Solution is marked with ====.

==A== ---B---- --C-- ---D---- --E-- --F--- ----G----
---H-- ==I== ----J--- ---K--- --L-- ---M-- --N--
---O---- ---P---- --Q-- ---R---- ==S== ==T=== ====U====
----V-- --W-- ==X== ===Y==== =Z=

Problem 2. (20 points [greedy])

The Activity Selection problem we discussed in class requires one to find the maximum-size subset of non-overlapping intervals (or non-conflicting tasks). Consider the following greedy algorithm.

1. For each interval $i$ compute the number of other intervals overlapping $i$
2. Sort the task by the number of overlaps, in increasing order (break ties arbitrarily)
3. Pick the task $i$ with the smallest number of overlaps, schedule it, and remove from further consideration tasks that are overlapping with $i$
4. Repeat Step 3 until all tasks are scheduled/discarded (no task is left)

This greedy algorithm is not optimal. Show a counter-example in which this strategy gives a suboptimal solution.

Answer: As said, the strategy is not optimal. Consider the following seven tasks:

The optimal number of tasks is four (boxed), but the algorithm can pick the dashed tasks (the two on the sides have one conflict so they are picked first).

Problem 3. (20 points [greedy (design)])

Assume that you are given two unsorted arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of $n$ distinct positive integers. Give a $O(n \log n)$-time greedy algorithm that determines an ordering
of the elements of $A$ and $B$ such that $W = \sum_{i=1}^n a_i b_i$ is maximized. Explain why your algorithm runs in $O(n \log n)$-time, and prove the greedy choice property for your algorithm.

**Answer:** Here is the greedy algorithm

**Algorithm** Greedy $(A, B)$; sort $A$ and $B$ in decreasing order; return $(A, B)$

Complexity analysis: it runs in $O(n \log n)$-time due to sorting.

Proof of the greedy choice: The usual exchange argument. Consider any indices $i$ and $j$ such that $i < j$, and consider the terms $a_i b_i$ and $a_j b_j$ in the sum $W$ (we leave all the other items alone). We want to show that the objective function $W$ will be lower by taking $a_i b_j$ and $a_j b_i$ instead. In other words, we need to show that

$$a_i b_i + a_j b_j \geq a_i b_j + a_j b_i$$

which means

$$a_i (b_i - b_j) \geq a_j (b_i - b_j)$$

which is true since $A$ and $B$ are sorted in decreasing order, and the elements are distinct, so when $i < j$ we have $a_i > a_j$ and $b_i > b_j$. That implies that $b_i - b_j > 0$, which means that we can cancel it out without changing the direction of the inequality. That results in $a_i \geq a_j$ which is true due to the ordering.

**Problem 4.** (20 points [dynamic programming])

In this problem you are asked to traceback the solution(s) of an instance of LCS for the strings $x = \text{abcd}$, $y = \text{cdabac}$. Recall that we define the table $C[i, j]$ as the length of LCS of between the prefix of $x$ of length $i$ and the prefix of $y$ of length $j$.

$$C[i, j] = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
C[i - 1, j - 1] + 1 & \text{if } i > 0, j > 0 \text{ and } x[i] = y[j] \\
\max\{C[i - 1, j], C[i, j - 1]\} & \text{if } i > 0, j > 0 \text{ and } x[i] \neq y[j]
\end{cases}$$

1. Draw the traceback pointers for the optimal solution(s)
2. Write the corresponding optimal solution(s)

**Note:** guessing the optimal solutions without showing the traceback won’t give any credit.

**Answer:** There are three LCSs. LCS abc corresponds to the path $(5, 6) \rightarrow (4, 6) \rightarrow (3, 6) \rightarrow (2, 5) \rightarrow (2, 4) \rightarrow (1, 3) \rightarrow (0, 2)$; LCS aba corresponds to $(5, 6) \rightarrow (5, 5) \rightarrow (4, 4) \rightarrow (3, 4) \rightarrow (2, 4) \rightarrow (1, 3) \rightarrow (0, 2)$; LCS cda corresponds to $(5, 6) \rightarrow (5, 5) \rightarrow (4, 4) \rightarrow (4, 3) \rightarrow (4, 2) \rightarrow (3, 1) \rightarrow (2, 0)$.

**Problem 5.** (20 points [dynamic programming])

**Answer:** This is another version of 01-knapsack called SUBSETSUM, where there are no benefits and we have to fill the knapsack exactly.

Let $t(i, j) = \text{TRUE}$ if there is a subset of the first $i$ items that has a total of exactly $j$, false otherwise. We have
\[ t(i, j) = \begin{cases} 
(j = 0) & \text{if } i = 0 \\
t(i - 1, j) & i > 0 \text{ AND } a_i > j \\
t(i - 1, j) \text{ OR } t(i - 1, j - a_i) & i > 0 \text{ AND } a_i \leq j
\end{cases} \]

We want to know \( t(n, T) \). Time complexity is \( O(nT) \). Space complexity is \( O(T) \) because we can re-use the rows (we just need the previous row \( i - 1 \) to compute the current row \( i \)).