• This exam is closed book, closed notes, 80 minutes long
• Read the questions carefully
• No electronic equipment allowed (cell phones, tablets, computers, . . . )
• Write legibly. What can’t be read will not be graded
• Use pseudocode, Python, or English to describe your algorithms (no C/C++/Java)
• When designing an algorithm, you are allowed to use any algorithm or data structure we explained in CS 141 or CS 14, without giving its details, unless the question specifically requires that you give such details
• Always remember to analyze the time complexity of your solution
• If you have a question about the meaning of a question, come to the front of the class
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.

```
--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-
->->->->->->->->->->->->->time->->->->->->->->->->->->->->->
```
**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.
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.
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 
\(x = \text{abcda}, y = \text{cdabac}\). 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.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 5. (20 points [dynamic programming])

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of $n$ positive integer and let $T$ be another integer. Design a dynamic programming algorithm that determines whether there exists a subset of the integers in $A$ whose total sum is exactly $T$. Analyze the time- and space-complexity of your solution.

For instance, if $A = \{4, 5, 17, 23, 11, 2\}$ and $T = 35$ the algorithm should return True because the subset $\{5, 17, 11, 2\}$ sums to 35. For the same set of numbers if we choose $T = 31$ the problem has no solution, and the algorithm will return False.