• This exam is closed book, closed notes, 50 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. (25 points [greedy])

Draw the Huffman tree and find the optimal prefix code for the symbols in the following frequency table

<table>
<thead>
<tr>
<th>symbol</th>
<th>frequency</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
Problem 2. (25 points [greedy])

In the activity selection problem we discussed in class, we are given a set of activities $S$, where each activity $a$ is defined by a start time $s_a$ and a finish time $f_a$; the objective is to produce the largest subset of non-conflicting activities. The greedy algorithm first sorts the activities in $S$ by finish time, then schedules first the activity in $S$ that has the earliest finish time (greedy choice). Prove that there exists an optimal solution that begins with a greedy choice. Complete the proof:

Proof. Suppose $A$ is an optimal solution for $S$.

Let $a$ be the activity in $A$ which has the earliest finish time.

- If $a$ is the activity in $S$ which has the earliest finish time (i.e., $A$ contains the greedy choice), then . . .

- If $a$ is not the activity in $S$ which has the earliest finish time (i.e., $A$ does not contain the greedy choice), then . . .
Problem 3. (25 points [dynamic programming])

In this problem you are asked to traceback the solution of an instance of 0-1 knapsack with 
\( n = 6 \) items, with weights \( \{w_1 = 2, w_2 = 4, w_3 = 6, w_4 = 3, w_5 = 5, w_6 = 3\} \),
benefits \( \{b_1 = 1, b_2 = 3, b_3 = 5, b_4 = 4, b_5 = 4, b_6 = 2\} \), and a knapsack size \( W = 11 \).

Recall that we define the table \( P[i, k] \) as the maximum profit possible using items \( \{i, i+1, \ldots, n\} \) 
and residual (knapsack) capacity \( k \). The table is filled according to the following rules:

\[
P[i, k] = \begin{cases} 
0 & \text{if } i = n \text{ and } w_n > k \\
b_n & \text{if } i = n \text{ and } w_n \leq k \\
P[i + 1, k] & \text{if } i < n \text{ and } w_i > k \\
\max\{P[i + 1, k], P[i + 1, k - w_i] + b_i\} & \text{if } i < n \text{ and } w_i \leq k
\end{cases}
\]

1. Draw the traceback pointers for the optimal solution(s) from the cell in \textbf{bold}.

2. Write the optimal solution(s): which items are selected? Compute that total benefit and the 
total weight of the optimal solution(s).

\textbf{Note:} guessing the optimal solutions without showing the traceback won’t give any credit.

\[
\begin{array}{cccccccccccc}
\text{0} & \text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{6} & \text{7} & \text{8} & \text{9} & \text{10} & \text{11} \\
\hline
6 & 0 & 0 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\
5 & 0 & 0 & 0 & 2 & 2 & 4 & 4 & 4 & 6 & 6 & 6 \\
4 & 0 & 0 & 0 & 4 & 4 & 4 & 6 & 6 & 8 & 8 & 8 \\
3 & 0 & 0 & 0 & 4 & 4 & 4 & 6 & 8 & 9 & 9 & 10 \\
2 & 0 & 0 & 0 & 4 & 4 & 6 & 7 & 8 & 9 & 9 & 10 \\
1 & 0 & 0 & 1 & 4 & 4 & 5 & 6 & 7 & 8 & 9 & 10
\end{array}
\]
Problem 4. (25 points [dynamic programming])

Consider the weighted version of the activity selection problem we discussed in class, where now tasks have benefits. You are given an array $A$ of $n$ tasks described by start time $s_i$, finish time $f_i$, and benefit $b_i$, which are all integers. Tasks are sorted in increasing order of $s_i$ in $A$. Give a dynamic programming algorithm that computes the largest possible total benefit for a subset of non-conflicting tasks in $A$. Your algorithm should run in $O(n^2)$ time (explain why).

For instance, if $A = \{(1,4,5), (1,3,6), (2,3,2), (4,5,1), (4,6,3), (5,6,3)\}$ where each triple is $(s_i, f_i, b_i)$, the highest total benefit is 10 (see figure on the right).

Hints: Define $P[i]$ to be the largest total benefit for a subset of tasks $\{1, 2, \ldots, i\}$ which includes task $i$. The structure of the recurrence relation for $P[i]$ is similar to the one for longest increasing subsequence.