CS 3100 Quiz Day 3

This packet contains the quizzes for this quiz day. This **cover sheet** is here to provide instructions, and to cover the questions until the quiz begins. **do not remove this cover sheet** until your proctor instructs you to do so.

You will have the entire class period to complete these quizzes. Each quiz is two pages (front and back of one sheet of paper) worth of questions. Make sure to **write your name and computing id at the top of each individual quiz**.

When you are done, you will come to the front of the room and cut off the staple to this quiz booklet. Afterward, you will discard this cover sheet and submit each quiz separately in a different pile. The proctors will be available at the front of the room to clarify this if you have any questions.

This quiz is CLOSED text book, closed-notes, closed-calculator, closed-cell phone, closed-computer, closed-neighbor, etc. Questions are worth different amounts, so be sure to look over all the questions and plan your time accordingly. Please sign the honor pledge below.

In theory, there is no difference between theory and practice.
But, in practice, there is.

THIS COVER SHEET WILL NOT BE SUBMITTED. DO NOT PUT WORK YOU WANT GRADED ON THIS PAGE

Quiz - Module 5: Dynamic Programming

Name _____

1. [8 points] Answer the following True/False questions.

Problems with *optimal substructure* always have dynamic programming solutions	**True**	**False**
Dynamic programming solutions are always implemented recursively.	**True**	**False**
The *backtracking* algorithm for *Discrete Knapsack* runs in $\Theta(n)$ time, where n is the number of items you can steal.	**True**	**False**
Most *dynamic programming* algorithms compute the value of the solution only, and another process is needed to construct the solution elements themselves.	**True**	**False**
The dynamic programming solution to *Fibonacci* that we saw in class can be improved to use $\Theta(1)$ space.	**True**	**False**
Our *dynamic programming* solution to the *discrete knapsack problem* ran in $\Theta(n * W)$ time, which (surprisingly) is actually exponential.	**True**	**False**
Our $MakingChange$ solution used $\Theta(A)$ space.	**True**	**False**
When solving the *weighted interval scheduling* problem, we needed to precompute the $p()$ function. Computing these values took $\Theta(n)$ time.	**True**	**False**

2. [6 points] For each of the problems we went over in class, look at the recurrence and fill in the missing blanks.

Problem	**Vars/Funcs**	**Recurrence**
Rod-Cutting	$P[]$	$C(n) = Max_{i=1...n}\{\_\_\_\_\_ + P[i]\}$
Making Change	A, $Denom[]$	$C(i,a) = Min(C(i+1,a), 1 + C(i, \_\_\_\_\_))$
Weighted Interval Scheduling	$V[]$, $P()$	$C(i) = Max(C(i-1), \_\_\_\_\_ + C(\_\_\_\_\_))$
Discrete Knapsack	C, $W[]$, $V[]$	$C(i,w) = Max(C(i-1,w), V[i] + C(\_\_\_\_\_, \_\_\_\_\_))$

Consider the following problem: You want to prank your friend by nesting the gifts inside of multiple boxes (i.e., they open the present and find another smaller present! When they open that one, they find another smaller present, etc.!). You are given a list of boxes sorted by volume $B = \{b_1, b_2, ..., b_n\}$ and a function $f(b_i, b_j)$ that given two boxes, returns true if and only if the first box first inside of the second box fully. These questions will ask you to design a *dynamic programming* solution that computes the **most boxes that can be nested inside of one another**, given the input boxes.

Also consider the following sub-problem definition: Let $C(i)$ be the most nested boxes possible given that you only use the first i boxes AND box i is the most outer box in the solution. Given this, answer the following questions:

3. [2 points] What are the two base cases? Fill in the blanks below

$$C(0) = \_\_\_\_\_ \qquad C(1) = \_\_\_\_\_$$

4. [3 points] Now solve the general form of the recurrence in terms of smaller sub-problems. Fill in the blanks.

$$C(i) = Max_{j=1...i}(1 + \_\_\_\_\_) \quad \text{IF } f(\_\_\_\_\_, \_\_\_\_\_)$$

5. [1 points] Which of the following subproblems stores the *overall solution to the problem*. Choose one.

- $C(n)$
- $Max_{i=0...n}C(i)$

NOTHING BELOW THIS POINT WILL BE GRADED

Quiz - Module 6: Network Flow

Name _____

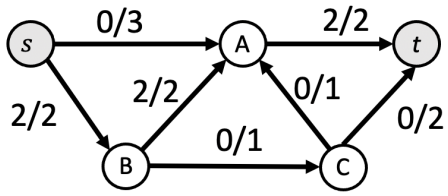
1. [8 points] Answer the following True/False questions.

Ford-Fulkerson runs in time $\Theta(E * f)$, where f is the capacity of the largest edge in the network	**True**	**False**
Ford-Fulkerson cannot be used to solve a network flow problem that contains multiple sources and multiple sinks	**True**	**False**
When reducing *Bi-Partite Matching* to *Network Flow*, we needed to add 2 new nodes and $\Theta(\|E\|)$ new edges	**True**	**False**
After reducing *Bi-Partite Matching* to *Network Flow*, a perfect matching exists if the max-flow is equal to $\frac{V}{2}$	**True**	**False**
If given an arbitrary *(S-T) Cut* in a network flow graph, the *net-flow* across that cut is always equal to the *capacity* of the cut	**True**	**False**
As *Ford-Fulkerson* executes, it is possible that the current flow through the graph goes up and down, but eventually it becomes maximum	**True**	**False**
If a *flow-network* contains a current flow that is maximum, then *DFS* can be used to efficiently find the *cut that is at full capacity*	**True**	**False**
For *flow-networks*, only one cut can be at *full capacity* at a time	**True**	**False**

2. [6 points] Consider the implementation of *Ford-Fulkerson* below. Fill in the blanks to complete the implementation.

```
function FordFulkerson(int[] Gf):

        int flow = __________   //starting flow
        int[] path              //path that DFS finds

        while (path = DFS(Gf)) != NULL:

                int minF = _________    //min flow along path

                for i from 0 to path.length-1:
                        minF = min(Gf[i][i+1], minF)

                for i from 0 to path.length-1:


                        Gf[i][i+1] ___________________


                        Gf[i+1][i] ___________________


                flow += _________
        return flow
```

The graph G below shows a flow graph after Ford-Fulkerson has found one augmenting path, $s - B - A - t$.



3. [1 points] Are there any augmenting paths in the residual flow graph G_f corresponding to the drawing of G shown above? If so, list the vertices in a valid augmenting path in the box below. If there is not, write "none."
   Vertices in path: [ ]

4. [2 points] Look for or find a min-cut in the graph G shown above and list the vertices on the source side of this cut in the box. (Include vertex s in your answer.) [ ]

5. [3 points] Which of the following are true statments about the Θ time-complexity of *Ford-Fulkerson*? Circle zero or more statements that are true.

   - Because each augmenting path may push only 1 unit of flow, there is a factor of $|f|$ in the Θ time-complexity.
   - Because each search using DFS costs $\Theta(V + E)$ and we may have to do one search per vertex, there is a factor of $V^2 + EV$ in the Θ time-complexity.
   - The cost of using DFS to find one augmenting path can be considered as $\Theta(E)$ instead of $\Theta(E + V)$ because we assume each flow graph must be connected in a way that $E \geq V - 1$.

NOTHING BELOW THIS POINT WILL BE GRADED