Question 1 (3 points)

Suppose you come up with a Divide & Conquer algorithm which when given a problem of size n, divides it into 3 sub-problems each of size half of the original problem. Moreover, say your algorithm takes $\Theta(n)$ time in total to divide and combine the solutions to the sub-problems.

Let T(n) denote the running time of your algorithm on an input problem of size n. Assuming appropriate base cases, T(n) satisfies the following recurrence: $T(n) = 2T(n/3) + \Theta(n)$.

- ◯ True
- ◯ False

Question 2 (3 points)

A flow network with unique edge capacities could have more than one min cut.

- ◯ True
- ◯ False

Question 3 (3 points)

The Bellman-Ford algorithm may take different number of iterations to converge, depending on the order in which it examines the vertices of the graph.

- ◯ True
- ◯ False

Question 4 (3 points)

Recall the Subset Sum problem we studied in class: Given a set of n requests (in which request i takes time wi to process) and an integer W such that you can schedule requests any time between 0 and W, the objective was to schedule requests to maximize the machine's utilization.

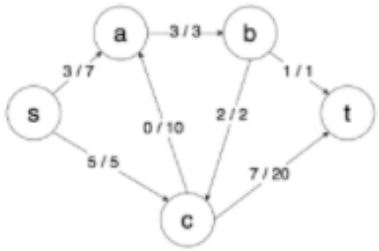
We presented a solution to this problem which had a pseudo-polynomial run time of O(nW).

Now suppose we learn that W is upper bounded by n^2. Let us call this new problem (i.e. the one where W is upper bounded by n^2) as Upset Sum. Then, the same algorithm solves upset sum in polynomial-time.

- ◯ True
- ◯ False

Question 5 (3 points)

Given a flow network below where each edge e is labelled fe/ce where fe is the flow on e and ce is the capacity of e. Then, f (the flow given) is a max-flow for this network.



- ◯ True
- ◯ False

Question 6 (3 points)

The Ford-Fulkerson algorithm is not guaranteed to converge to max flow for some non-integer edge capacities.

- ◯ True
- ◯ False

Question 7 (3 points)

In a flow network G with no two edges having the same capacity, let e be the edge with the smallest capacity. Then, e must cross some min-cut of G.

- ◯ True
- ◯ False

Question 8 (6 points)

Which of the following best describes the meaning of pseudo-polynomial runtime?

- ◯ The runtime includes both polynomial and logarithmic terms.
- ◯ The runtime is polynomial in terms of the numeric value of the input.
- ◯ The runtime is polynomial in terms of the number of integers in the input.
- ◯ The runtime is polynomial in terms of the number of bits in the input.

Question 9 (6 points)

Which two of the following recurrences have the solution with the same complexity?

- [ ] $T(n) = 2T(n/2) + 1$
- [ ] $T(n) = 2T(n/3) + n$
- [ ] $T(n) = 2T(n/2) + n$
- [ ] $T(n) = 2T(n/2) + n\log n$

Question 10 (6 points)

Which of the following best describes how to calculate the runtime of dynamic programming algorithms?

- ◯ The number of unique subproblems times the input size of each subproblem.
- ◯ The number of unique subproblems times the input size of each subproblem plus the runtime of solving each subproblem.
- ◯ The number of unique subproblems times the runtime of solving each subproblem.
- ◯ The number of unique subproblems times the runtime of solving each subproblem times the number of occurrences of each subproblem.

Open Response

Question 11 (12 points)

Recall the weighted interval scheduling problem we studied in class: given n requests where request i has start time si, finish time fi and weight wi -- the goal is to select a subset S of mutually non-overlapping requests such that it maximizes the total weight of S.

Let us denote each request i by the 3-tuple (si, fi, wi). Given the following set of requests: {(5, 7, 20), (1, 3, 10), (4, 9, 40), (2, 6, 20), (8, 10, 30)}, what is the weight of the optimal subset S (i.e. what is the value of the optimal solution)?

You can use the definition of the sub-problems and the recurrence formula given in class {opt(i) = Max(wi + opt(p(i)), opt(i-1))} for this problem and then solve the problem numerically. We are not looking for the optimal subset, only for the weight of the optimal subset.

You need to show all your steps in solving this problem using dynamic programming with the given recurrence formula. In other words, just giving the final answer will not receive any credit.

Question 12 (19 points)

The cows are back in Exam 2! Except this time, they will be jumping hay-bales.

Consider an n x m field of hay-bales. The "#"'s identify the locations of the hay-bales.

```
..#.###.
.###.#..
..#.#.#.
.#..#...
```

The cow starts in the top left corner and is trying to make its way to the bottom right corner. The cow can jump either right or down in the field a distance of up to k cells with a single jump. Your task is to determine the minimum number of jumps needed to reach the bottom right cell without the cow landing headfirst on any hay-bales.

Design a dynamic programming solution to this problem. Analyze the runtime and memory usage. Your runtime should not be worse than O(nmk) and your memory usage should not be worse than O(nm).

a) Define (in plain English) unique subproblems to be solved (4 points).

b) Write the recurrence relation for subproblems. No justification needed (6 points).

c) Using the recurrence formula in part b, write pseudocode (using iteration) to compute the minimum number of jumps needed to reach the bottom right cell without landing headfirst on any hay-bales. Make sure you have initial values properly assigned (6 points).

d) Compute the runtime of the algorithm described in part c and state whether your solution runs in polynomial time or not (3 points).

Question 13 (19 points)

Given n positive integers a_1, a_2, ..., a_n. What is the maximum value of a_i / a_j where i < j. Design an O(n)-time divide-and-conquer algorithm, and justify the time complexity of your algorithm.

a) Describe your divide step (2 points).

b) Describe the information you will send back to the parent subproblem during recursion at each step (6 points).

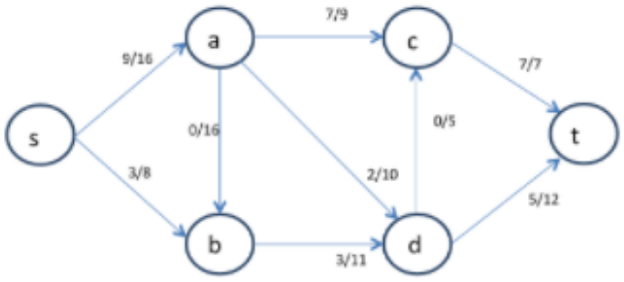
c) Describe your combine step (2 points).

d) Write complete pseudocode to present your divide and conquer solution (6 points).

e) Justify the runtime complexity of your algorithm (3 points).

Question 14 (15 points)

Consider the below flow-network, for which an s-t flow has been computed. The numbers x/y on each edge shows that the capacity of the edge is equal to y, and the flow sent on the edge is equal to x.



a) What is the current value of flow? (2 points)

b) Draw (or list edges with edge capacities of) the residual graph for the corresponding flow-network. (4 points)

c) Calculate the maximum flow in the graph using the Ford-Fulkerson algorithm. You need to show/state the augmenting path at each step and final max flow. (6 points)

d) Show/state the min cut found by the max flow you found in part c. (3 points)