Question 2.

(5 points) Among the algorithms and other ideas that you saw in this course, what did you like the most, and why ? And what was the most surprising algorithm/idea that you saw in the course ?

Keep your answers short, succinct and accessible.

Question 3.

(4 points) Arrange the following functions in increasing order of their growth rate, with g(n) following f(n) if and only if f(n) = O(g(n)). No explanation is necessary. You get full points for the correct answer and zero for *any* errors. There are no partial credits for this question.

1. $\sqrt{n}$
2. $2^{\sqrt{\log n}}$
3. $2^{2^{\sqrt{\log \log n}}}$
4. 2^n
5. n
6. $n \log n$
7. $n \cdot (\log n) \cdot (\log \log n)$

Question 4.

(5 points) Let T(n) be a function that satisfies $T(1) = 1$ and $T(n) = T(n/2) + T(2n/5) + 7n$. Then, state and formally prove a bound on the asymptotic behavior of T(n) in Theta (Θ) notation.

Question 5.

An array of integers is said to be *unimodal* if all its entries are distinct and its entries are in increasing order up until its maximum element, after which its elements are in decreasing order.

1. **(4 points)** Design an algorithm to compute the maximum element of a unimodal array that runs in O(log n) time (or the fastest algorithm that you can).
2. **(4 points)** Prove the correctness of your algorithm.

Question 6.

For this problem, our input is a natural number n, given in its binary encoding. Consider the following algorithm.

Algorithm begins

Set F = 1 and i = 2;

While (F > 0){

If i divides n, output i and stop;

Set i = i+1 ;

*Set j = i*i*i;*

If j > n, set F = 0;

}

Output -1 and stop;

Algorithm ends

Algorithm ends

Now, based on the above algorithm, answer the following questions.

1. **(1 point)** What is the number of bits in the input ?
2. **(2 points)** What is the time complexity of the algorithm on input n (no explanation necessary)?
3. **(1 point)** Based on the above two answers, is this an efficient (a polynomial time) algorithm (YES/NO) ?
4. **(4 points)** Prove that if a natural number n is of the form $n = a \times b \times c$, where a, b, c are natural numbers with $a \geq b \geq c \geq 2$, then the algorithm correctly outputs a prime factor of n when invoked with input n.
5. **(2 points)** Give an example of a composite number n such that on input n, this algorithm does not output a prime factor of n correctly.

Question 7.

Recall that an interval is just a contiguous subset of real numbers, given by a pair of real numbers (s, e) with $s < e$. We refer to s as the starting point of the interval and and e as the end point.

For this problem, we are given a set of n intervals $S = \{(start_i, end_i) | 1 \leq i \leq n\}$. We want to find a maximum subset S' of S such that no pair of intervals in S' overlap. To this end, we will try to understand the applicability of variants of the greedy strategy towards designing algorithms for this problem.

1. **(2 points)** Give an example (i.e an instance of this problem given by a collection of intervals) where the following algorithm fails.

while(S is not empty){
a. Select the interval I that overlaps the least number of other intervals ;
b. Add I to final solution S';
c. Remove all intervals from S that overlaps with I and continue;
}

2. **(2 points)** Another variant of the greedy srategy for this problem is to start by selecting the interval that starts earliest, i.e. $start_i$ is smallest, including this interval in the solution set, deleting any intervals that intersect this and continuing. Give an example where this strategy fails.
3. **(2 points)** Yet another variant is to start by selecting the shortest interval, including it in the solution set, deleting any overlapping intervals and continuing. Give an example where this strategy fails.
4. **(4 points)** Finally design an algorithm based on the greedy strategy that outputs the optimal solution for this problem. You are expected to design the fastest algorithm that you can.

5. **(4 points)** Prove the correctness of your algorithm in the previous part and analyse its running time.

Question 8.

Suppose you are given a directed graph G= (V,E), (|V|=n, |E|=m), with a positive integer capacity c_e on each edge e, a source vertex s and a sink vertex t. You are also given a maximum feasible s-t flow f in G, defined by a flow value of f_e on edge e. Now suppose we pick a specific edge e^* and increase its capacity by 1 i.e. increase c_{e^*} by 1 to give a new edge capacitated graph G'.

1. **(3 points)** Let f' be a maximum flow in G'. Show that |value(f)-value(f')| is at most 1.
2. **(3 points)** Give an algorithm to find a maximum flow in G' in O(n+m) time.
3. **(3 points)** Argue the proof of correctness and the running time of the algorithm that you mentioned in the second part.

Question 9.

A graph is called d-regular if every vertex of the graph has degree d. Let G=(V=(L,R),E) be a d-regular bipartite graph with |L|=|R|=n.

1. **(3.5 points)** Consider a directed graph $G' = (V \cup \{s, t\}, E')$ with $E' = \{(x, y) \mid x \in L, y \in R, (x, y) \in E\} \cup \{(s, u)$ and unit capacity on each edge in E'. Show that there is a feasible flow in G' of value n.
2. **(3.5 points)** Use the first part to show that G has a perfect matching.

Question 10.

For this question, the goal is to design two search to decision reductions of the kind that we saw in the class.

1. **(6 points)** Suppose you have access to a subroutine that magically solves SAT in constant time, i.e, given a Boolean formula F as input, this subroutine correctly outputs in constant time whether F is satisfiable or not. Use this subroutine to design a polynomial time algorithm that takes as input a Boolean formula F and outputs an assignment to the variables that satisfies F if F is satisfiable or correctly states that F is unsatisfiable.
2. **(6 points)** For this question suppose that we have access to a subroutine that solves the decision version of minimum vertex cover in constant time, i.e. given an undirected graph G, and a natural number k, this subroutine correctly decides if the graph has a vertex cover of size at most k or not. Using this subroutine, design a polynomial time algorithm that given an undirected graph G, outputs a vertex cover of *minimum size* in G.