NAME: _____

CSE 331
Introduction to Algorithm Analysis and Design
Sample Final Exam Solutions

1. ($5 \times 2 = 10$ points) Answer True or False to the following questions. **No justification** is required. (Recall that a statement is true only if it is logically true in all cases while it is is false if it is not true in some case).

   Note: *I'm providing justifications for the questions below for your understanding. In the actual exam for Q1, you of course only have to say True or false.*

   (a) Depth First Search (DFS) is a linear time algorithm.

   True. The input graph can be represented as an adjacency list of total size $O(m+n)$. We saw in the lecture that DFS runs in time $O(m+n)$ and thus, has a linear running time.

   (b) n is $O\left((\log n)^{\log n}\right)$.

   True. Note that $n = 2^{\log n}$ and $(\log n)^{\log n} = 2^{\log\log n \cdot \log n}$. Now the statement follows as $\log n \leq \log\log n \cdot \log n$ for large enough n.

   (c) There is no algorithm that can compute the Minimum Spanning Tree (MST) of a graph on n vertices and m edges in time asymptotically faster than $O(m \log n)$.

   False. As I mentioned in the class, there exists algorithms to compute the MST in time $O(m\alpha(m,n))$, where $\alpha(m,n)$ is the inverse Ackerman function and is asymptotically (much) smaller than $\log n$. (This fact was also mentioned with Q2 in HW 8.)

   (d) Let G be a graph with a negative cycle. Then there is no pair of vertices that has a finite cost shortest path.

   False. Consider a graph that has an edge (s,t) with cost 1 and a disjoint negative cycle. In this graph the shortest $s-t$ path has cost 1.

   (e) Every computational problem on input size n can be solved by an algorithm with running time polynomial in n.

   False. There are many ways to show this is false, here is one. Consider the problem, where given n numbers as input, the algorithm has to output all the permutations of the n numbers. Since there are $n!$ permutations that need to be output, every algorithm for this problem runs in exponential time.

2. ($5 \times 6 = 30$ points) Answer True or False to the following questions and **briefly JUSTIFY** each answer. A correct answer with no or totally incorrect justification will get you 2 out of the total 6 points. (Recall that a statement is true only if it is logically true in all cases while it is is false if it is not true in some case).

   (a) The following algorithm to check if the input number n is a prime number runs in polynomial time.

For every integer $2 \leq i \leq \sqrt{n}$, check if i divides n. If so declare n to be *not* a prime. If no such i exists, declare n to be a prime.

False. The run time of the algorithm is $O(\sqrt{n})$ but the input size is $\log n$ and thus, the run time is exponential and not polynomial (in the input size).

(b) Let $G = (V, E)$ be a directed graph with positive costs, i.e. $c_e \geq 0$ for every $e \in E$. The following is true for every real number $\delta > 0$. Consider the instance where the input graph is still G but the costs are now $c'_e = c_e + \delta$ (for every $e \in E$). Then for some distinct $s, t \in V$, the shortest $s - t$ path in the two instances are different.

False. Consider the graph with just one edge (s, t). Now no matter how the weight of (s, t) is changed, the shortest $s - t$ is always the edge (s, t).

(c) Every weighted graph has a unique Minimum Spanning Tree (MST).

False. Consider the "triangle" graph with all edge weights being 1. This graph has three spanning trees all of which have weight 2.

(d) The weighted interval scheduling problem on n jobs can be solved in $O(n)$ time.
True. We saw in the class a dynamic programming algorithm that runs in $o(n)$ time.

(*Note:* The above assumes that the jobs are sorted by their finish time (and the $p(\cdot)$ values have been pre-computed). So if you say false and justify it by assuming that the jobs are not sorted, then you will get full credit too.)

(e) Given n integers $a_1, \ldots, a_n$, the third smallest number among $a_1, \ldots, a_n$ can be computed in $O(n)$ time.

True. Do a linear scan and remember the three smallest numbers seen so far. Whenever you encounter a new number, one can figure out in constant time, if it should displace any of the current three minimum guys. At the end of the linear scan, output the third smallest number. (The running time is $O(n)$ as the algorithm spends only $O(1)$ time per element.)

3. ($5 + 15 = 20$ points) In this problem you will show that the naive recursive algorithm (that we saw in class) to compute the value of the optimal schedule for the weighted interval scheduling problem takes exponential time.

(a) As a warmup, we will begin with a recurrence relation, that does not have anything to do with the weighted interval scheduling per se. Consider the following recurrence relation: $F(0) = 0$, $F(1) = 1$ and for every $n \geq 2$, $F(n) = F(n-1) + F(n-2)$. Prove that $F(n) \geq (3/2)^{n-2}$. (Note that you have to prove a *lower bound*.)

Hint: Use the guess and verify using induction technique of solving recurrences.

Solution: We will show by induction on n that $F(n) \geq (3/2)^{n-2}$. The inequality is definitely true when $n = 0, 1$ or 2. For the inductive hypothesis, assume that for every $0 \leq i \leq n-1$, we have $F(i) \geq (3/2)^{i-2}$. Now by the definition of the recurrence, $F(n) = F(n-1) + F(n-2)$. By the inductive hypothesis, $F(n-1) \geq$

$(3/2)^{n-3}$ and $F(n-2) \geq (3/2)^{n-4}$. Thus, we have $F(n) \geq (3/2)^{n-3} + (3/2)^{n-4} = (3/2)^{n-2}\left(\frac{2}{3} + \frac{4}{9}\right) > (3/2)^{n-2}$, which completes the inductive step.

(b) Using the part (a) or otherwise, prove that for every $n \geq 2$, the following recursive algorithm for the weighted interval scheduling problem takes $\Omega((1.5)^n)$ time. (Recall that the input to the weighted interval scheduling problem are n jobs where the ith job is the tuple (s_i, f_i, v_i). Further, note that we assume that $f_1 \leq f_2 \leq \cdots \leq f_n$ and the values $p(j)$ are known. Finally, recall that $p(j)$ is the the largest job index $i \leq j$ such that $s_j \geq f_{p(j)}$.)

Compute-Opt(j)
If $j == 0$ **return** 0.
return $\max\{v_j + \textbf{Compute-Opt}(p(j)), \textbf{Compute-Opt}(j-1)\}$.

Solution. Consider the following generalization of the bad example for the algorithm we saw in class (and also appears in the book). Consider the following n jobs where the ith job is $(i, i+1, 1)$ (for $1 \leq i \leq n$). Note that in this case, for every $2 \leq i \leq n$, $p(i) = i-2$ and $p(1) = 0$. Thus, if $T(n)$ was the run time for **Compute-Opt**(n), then the following is true

$$T(n) \geq T(n-1) + T(n-2).$$

It is also easy to see that $T(0) \geq 0$ and $T(1) \geq 1$. Thus, by part (a), we have $T(n) \geq (1.5)^{n-2}$, which is $\Omega((1.5)^n)$, as desired.

4. $(5 + 15 = 20$ points) A *boolean* polynomial $P(X)$ of *degree* d is the formal polynomial $P(X) = \sum_{i=0}^{d} p_i X^i$, where for every $0 \leq i \leq d$, $p_i \in \{0, 1\}$ (also called the ith *coefficient*) and X is a variable. Note that a polynomial is specified once the coefficients $p_0, \ldots, p_d$ are specified. (E.g. $X^4 + X^2 + X + 1$ is a degree 4 polynomial).

Given two polynomials $P(X)$ and $Q(X)$ of degree at most $n-1$, their product $R(X) = P(X) \cdot Q(X)$ is defined as the formal polynomial

$$\left(\sum_{i=0}^{n-1} p_i X^i\right)\left(\sum_{i=0}^{n-1} q_i X^i\right) = \sum_{i=0}^{2n-2}\left(\sum_{j=\max(0,i-n+1)}^{i} p_j q_{i-j}\right) X^i.$$

For example, if $P(X) = X^2 + 1$ and $Q(X) = X^2 + X + 1$, then $P(X) \cdot Q(X) = X^4 + X^3 + 2X^2 + X + 1$. (Note that the resulting polynomial can have coefficients taking values outside of $\{0, 1\}$.)

(a) Let $P(X) = X^3 + X + 1$ and $Q(X) = X^2 + X$. What is $P(X) \cdot Q(X)$?
Solution. $X^5 + X^4 + X^3 + 2X^2 + X$

(b) In this part, you will design an algorithm to solve the polynomial multiplication problem. In particular, the input to the problem for input size $n \geq 1$ are the coefficients of the two polynomials $p_0, \ldots, p_{n-1}$ and $q_0, \ldots, q_{n-1}$. The output should be the coefficient $r_0, \ldots, r_{2n-2}$, where $P(X) = \sum_{i=0}^{n-1} p_i^i X^i$, $Q(X) = \sum_{i=0}^{n-1} q_i X^i$, $R(X) = \sum_{i=0}^{2n-2} r_i X^i$ such that $R(X) = P(X) \cdot Q(X)$.

Design a divide and conquer algorithm that runs in time asymptotically faster than $O(n^2)$. Justify the correctness of your algorithm (formal proof is not required). Also state the running time of your algorithm (and very briefly justify it.)

Hint: Think of a similar problem we solved in class by a divide and conquer algorithm.

Solution. The solution mirrors the divide and conquer algorithm we saw for integer multiplication.

Define $P^0(X) = \sum_{i=0}^{\lceil n/2 \rceil - 1} p_i X^i$, $P^1(X) = \sum_{i=0}^{n-\lceil n/2 \rceil - 1} p_{i+\lceil n/2 \rceil} X^i$, $Q^0(X) = \sum_{i=0}^{\lceil n/2 \rceil - 1} q_i X^i$, and $Q^1(X) = \sum_{i=0}^{n-\lceil n/2 \rceil - 1} q_{i+\lceil n/2 \rceil} X^i$. It can be verified that

$$P(X) = P^0(X) + X^{\lceil n/2 \rceil} \cdot P^1(X) \text{ and } Q(X) = Q^0(X) + X^{\lceil n/2 \rceil} \cdot Q^1(X).$$

With the above understanding, like in the integer multiplication, we can deduce the following:

$$\begin{aligned} P(X) \cdot Q(X) =& (P^1(X) \cdot Q^1(X)) \cdot X^{2\lceil n/2 \rceil} \\ &+ ((P^1(X) + P^0(X))(Q^1(X) + Q^0(X)) - P^1(X) \cdot Q^1(X) - P^0(X) \cdot Q^0(X)) \cdot X^{\lceil n/2 \rceil} \\ &+ P^0(X) \cdot Q^0(X). \end{aligned}$$

With the above recurrence relation, the rest is the same as integer multiplication, see the book for the algorithm (replace "2" by "X" and the digits of the first number by the coefficients of $P(X)$ and the digits of the second number by the coefficients of $Q(X)$) and the analysis to show that it runs in time $O(n^{\log_2 3})$. (As with integer multiplication, note that adding or subtracting two polynomials of $O(n)$ degree takes $O(n)$ time. Finally, multiplying a polynomial by X^i just amounts to shifting the coefficients of the polynomial by i positions to the "left" and thus, multiplying by $X^{2\lceil n/2 \rceil}$ and $X^{\lceil n/2 \rceil}$ above are both $O(n)$ time operations.)

5. You are almost done!

   (a) (20 points) You given as input n real numbers $x_1, \ldots, x_n$. Design an efficient algorithm that uses the minimum number m of unit intervals $[\ell_i, \ell_i+1)$ $(1 \leq i \leq m)$ that cover all the input numbers. A number x_j is covered by an interval $[\ell_i, \ell_i+1)$ if $\ell_i \leq x_j < \ell_i + 1$. Argue why your algorithm is correct (formal proof is not required).

   For example, consider the input for $n = 4$: $0.1, 0.9, 1.1, 1.555$. Then the two intervals $[0.1, 1.1)$ and $[1.1, 2.1)$ cover all the input numbers (i.e. in this case $m = 2$).

   Solution First, if necessary by sorting, we will assume that $x_1 \leq x_2 \leq \cdots x_n$. (This part is important.)

   We will assign unit intervals greedily. Starting from $i = 1$, we will use the unit interval $[x_1, x_1 + 1)$ to cover as many points as possible. Let x_{i_1} be the last point

covered by this interval. Then use a new unit interval $[x_{i_1+1}, x_{i_1+1})$ to cover as many points as possible. Continue this process till all points are covered.

By the "greedy stays ahead" paradigm we can prove the optimality of the algorithm above. In particular, for any $1 \leq i \leq m$ (where m is the number of intervals used by the greedy algorithm above), if O_i is the index of the last point covered by the ith interval in an optimal solution (we assume that the intervals in the optimal solutions are sorted by their "start time") and G_i is the corresponding index for the greedy algorithm then $O_i \leq G_i$. This is true by definition of the greedy algorithm for $i = 1$. Now for the inductive hypothesis assume that for every $1 \leq i < j$, $O_i \leq G_i$. Now for the jth interval note that the j th interval for the optimal solution can be used by the greedy algorithm jobs up to O_j. Thus, by the greedy choice it can only do better.

(b) (**Bonus**) (10 points) I am too lazy to put in a bonus problem here but there will be one in the actual final exam.