Final – Answer Key

Problem 1 (*40 points)*

No explanation needed:

- What is the solution of the recurrence $T(n) = 5T(n/4) + n^3$ in Θ-notation?
  Using the Master Theorem, we obtain $T(n) = \Theta(n^3)$.
- An undirected graph with n nodes is connected, and then an edge is deleted. What is the largest number of connected components the new graph can have?
  By deleting an edge, we can at most split the graph into 2 components.
- An undirected graph with n nodes is connected, and then a node is deleted. What is the largest number of connected components the new graph can have?
  By deleting a node, the largest number of connected components left is $n-1$ (e.g. all the other nodes connected only to the node removed).
- A directed graph is strongly connected, and then an edge is deleted. What is the largest number of strongly connected components the new graph can have?
  By deleting an edge, we end up with at most n strongly connected components (e.g. the initial graph is a SCC cycle).
- What is $7^{5^{3,000}} \bmod 47$?
  47 is prime, so $7^{5^{3000}} \pmod{47} = 7^{5^{3000} \pmod{46}} \pmod{47}$.
  Moreover, $46 = 2 \cdot 23$ and it is relatively prime to 5, so $5^{3000} \pmod{46} = 5^{3000 \pmod{22}} \pmod{46}$. We get:
  $3000 \pmod{22} = 8$;
  $5^{3000} \pmod{46} = 5^8 \pmod{46} = 39$;
  $7^{5^{3000}} \pmod{47} = 7^{39} \pmod{47} = 7^{32+4+2+1} \pmod{47} = 64 \cdot 4 \cdot 2 \cdot 7 \pmod{47} = 21$
- In RSA, if $p = 17$, $q = 11$, and $e = 3$, what is d?
  $d = e^{-1} \mod (p-1)(q-1) = 3^{-1} \mod 160 = 107$.
- In the same RSA, if the message is 7, what is the encoding?
  Encoding is $7^3 \mod 187 = 156$.
- You want to multiply two polynomials of degree 6 using the FFT. What is your ω going to be?
  The degree f the product polynomial is 12, so we need at least 13 points. The smallest N we can use is 16, so $\omega = e^{2\pi i/16} = e^{\pi i/8}$.
- Assume one of the polynomials happens to be $x^6 + 1$, and the result of its FFT is $(A_0, A_1, \ldots)$. What is A_0? What is A_1?
  $A_0 = p(\omega^0) = p(1) = 1$; $A_1 = p(\omega^1) = p(\omega) = 1 + e^{3\pi i/4}$

- **Extra credit:** What is $(x-a)\cdot(x-b)\cdot(x-c)\cdots(x-z)$?

  It is 0; the term $(x-x)$ appears in the product.

- What is the dual of this linear program?

  maximize $x_1 + x_2 + x_3$

  subject to

  $$x_1 + x_3 = 2$$
  $$x_1 + 2x_3 \leq 4$$
  $$x_1 - x_2 \leq 2$$

  and $x_1, x_2, x_3 \geq 0$

  The dual is:

  minimize $2y_1 + 4y_2 + 2y_3$

  subject to

  $$y_1 + y_2 + y_3 \geq 1$$
  $$-y_3 \geq 1$$
  $$y_1 + 2y_2 \geq 1$$

  and $y_2, y_3 \geq 0$

- Does the original LP have an optimum? If so write down the optimal value and the corresponding x_1, x_2, x_3. If not, what does this imply for the dual?

  No, the primal is unbounded, since x_2 can get arbitrarily large. Hence, the dual is infeasible.

Problem 2 (*30 points)*

You are given a graph (V, E) with positive edge weights, and its minimum spanning tree. Now you are told that the length of edge $[u, v]$ (it may or may not be on the tree) has changed (increased or decreased) to ℓ_{new}. Describe an $O(|E|)$ algorithm for finding the new minimum spanning tree.

For the case where the weight decreases we do the following:

- If the edge $e = \{u, v\}$ with decreased weight is in the tree T , then we return T .
- Otherwise, we add the edge $e = \{u, v\}$ to the tree T, creating a single simple cycle. We can determine all edges in the cycle in $O(|V|)$ time by running a DFS from node u (using only edges in T) until we reach node v. Once we have found all edges in the simple cycle, we then delete the highest cost edge e' from the cycle (with $e' = e$ possible) to obtain a new spanning tree T' .

For the case where the weight increases we do the following:

- If the edge e with increased weight is not in T, then return T.
- Otherwise, we remove $e = \{u, v\}$ from the tree to obtain two connected components, one containing u and one containing v. We can then run two BFSs in $O(|V| + |E|)$ time (only using tree edges) to label the nodes in us component L1, and label the nodes in vs component L2. Using these labels, we can then iterate over all edges to nd the least cost edge e' crossing from us component to vs component. We then return the tree $T' = T - e + e'$.

What if you are told that the lengths of m edges have changed? How fast could you find the new minimum spanning tree? Explain very briefly.

We can run the above algorithm m times using as an input MST, the MST computed at the previous iteration. The crucial observation is that each run of the algorithm must be done after the previous one is completed. This gives a total running time of $O(m|E|)$.

For which values of m is the algorithm of the previous question faster than finding the MST from scratch?

For values $m \leq \log |V|$ (or $\log^* |V|$ if we use path compression) it is better to use the algorithm of the previous question. For larger values of m it's better to build the MST from scratch.

Problem 3 (*15 points)*

Recall that the RUDRATA PATH problem (Given a graph (V, E), find a path (no repeated nodes) of length (number of edges) $|V| - 1$) is NP-complete. The HALF-RUDRATA PATH problem is the following: Given a graph (V, E), find a path (no repeated nodes) of length $\left\lfloor \frac{|V|}{2} \right\rfloor$ (number of edges). Prove that it is NP-complete.

The problem is in NP: given a set of nodes it is easy (i.e. can be done in polynomial time) to check that they form a path of length $\left\lfloor \frac{|V|}{2} \right\rfloor$.

There are two obvious reductions.

The first one takes an instance of RUDRATA PATH and copies it; the two copies are independent in the sense that we add no edges between them. This is a valid instance for HALF-RUDRATA PATH and it is easy to see that it has a a path of length $\left\lfloor \frac{|V|}{2} \right\rfloor$ iff the initial RUDRATA PATH instance had a path of length $|V| - 1$.

An alternative reduction is to take an instance of RUDRATA PATH and add $|V|$ singleton nodes (that are not connected anywhere). Again, this is an instance of HALF-RUDRATA PATH and it has a a path of length $\left\lfloor \frac{|V|}{2} \right\rfloor$ iff the initial RUDRATA PATH instance had a path of length $|V| - 1$.

Problem 4 (*45 points)*

You are the CEO of Falafel International. Your company is planning to deploy falafel carts at street corners in San Francisco. You are considering n corners $1, 2, \ldots, n$, aligned on a path, and you want to decide for each one of them whether to deploy a cart there or not. For each street corner i your marketing people have given you three numbers, a_i, b_i, and c_i, possibly negative, which are the estimated daily profits from street corner i if a cart is deployed there and, respectively, none, or one, or two of the adjacent corners $i-1$ and $i+1$ also have a cart (obviously, c_1 and c_n have no meaning, and can be anything). You want to use the latest corporate planning trend called Dynamic Programming to find the best possible allocation of carts to street corners.

(a) Define a subproblem, write the recurrence equation, show how to recover the optimum solution, and tell us how long your algorithm takes to run.

We will define a series of subproblems encoding the optimum so far, under the condition that at current position n we have: no cart (Z[i]), one cart with no neighbors (O[i]), one cart with only a right neighbor (R[i]), one cart with only a left neighbor (L[i]), one cart with both neighbors (B[i]). The update rules are as follows:

$Z[i] = \max(Z[i-1], O[i-1], R[i-1])$

$O[i] = Z[i-1] + a_i$

$R[i] = \max(L[i-1], B[i-1]) + b_i$

$L[i] = Z[i-1] + b_i$

$B[i] = \max(L[i-1], B[i-1]) + c_i$

Of course, we need to satisfy the constraints at $i = 0$ (all arrays there are undefined except $Z[0] = 0, O[0] = a_0, R[0] = b_0$. The value of the optimal solution is the maximum of Z[n], O[n] and R[n]. In order to recover the solution, we can keep an extra array for each Z, R and B subproblem, that indicates where we came from. The total running time is linear, because for every i, we only look at a constant number of elements.

(b) Suppose now that the possible corners lie on a graph G, instead of a path, and again you are given profit estimates a_i, b_i, and c_i, in the case in which none, or any one, or any two or more of the adjacent corners are occupied by carts. Show that now finding the optimum allocation is NP-complete.

First we'll show that the search problem is in NP. Given a cart assignment, how to we check it has a certain value? Easy, just look at each vertex and add the amount generated by it - polynomial time. Now we show completeness, by reducing INDEPENDENT SET to our CART problem.

For a given INDEPENDENT SET problem with graph G=(V, E) and value k, we create a CART problem with the same graph G and objective k. For each vertex, we set $a_i = 1, b_i = -|V|, c_i = -|V|$. We need to show that an independent of size k generates a CART of value k, and viceversa. First direction is easy, all vertices in the independent set S have no neighbors in S, so if we choose those in CART we make a profit of k. For the other direction, we notice, the following fact: in our CART solution, we never choose two adjacent vertices (because then the total value would be negative), so any CART solution of value k defines k vertices with no edges between them (i.e. an independent set)

Problem 5 (*40 points)*

True or False? Circle the right answer. No explanation needed. No points will de deducted for wrong answers, so *give us your best guess.*

F The following is a legitimate choice for RSA: $N = pq = 35$, $e = 3$.

T Computing $a^b \bmod c$ for n-bit numbers a, b, c can be done in $O(n^{2.99})$ time.

T If from a particular vertex v in an undirected graph there is a unique path to every other vertex, then the graph is a tree.

T The shortest paths between all pairs of vertices in a graph with both positive and negative weights can be found in $O(|V|^3)$ time.

F If a graph has no negative cycles reachable from s, then Dijkstra's algorithm works in finding shortest paths from s.

T The Bellman-Ford algorithm takes time $O(|V||E|)$.

F If all weights in a graph are different, then there is a unique shortest path from s to t.

T A set of n Horn clauses can be tested for satisfiability in polynomial time.

T $(\bar{x} \vee \bar{y})$ is a Horn clause.

F $(x \vee y)$ is a Horn clause.

F Because of recursion overhead, recursion with memoization is always slower than direct dynamic programming.

F The dynamic programming algorithm for the traveling salesman problem may take time proportional to $n!$ for some instances.

T There is a polynomial algorithm for linear programming.

F Simplex is a polynomial-time algorithm.

F If in a network we increase the capacity of an edge in the minimum cut, the maximum flow gets increased.

T If in a network we decrease the capacity of an edge in the minimum cut, the maximum flow gets decreased.

T If in a network we increase the capacity of an edge which is not in the minimum cut, the maximum flow does not get increased.

T In a zero-sum game it is always possible for both players to play best response to what the other is playing.

For the remaining questions there are *four* possible answers: 1. true (T); 2. false (F) 3. true iff P = NP (=); and 4. true iff P $\neq$ NP ($\neq$). In each case, choose the right one.

T 3SAT is NP-complete.

= HORNSAT (finding a satisfying truth assignment for a set of Horn clauses) is NP-complete.

T There is a polynomial reduction from HORNSAT to 3SAT.

=	There is a polynomial reduction from 3SAT to HORNSAT.
=	3SAT is in P.
T	HORNSAT is in P.
T	There is no known polynomial algorithm for 3SAT.
F	There is no known polynomial algorithm for HORNSAT.
T	FACTORING is in NP.
$\neq$	There is no polynomial time algorithm for computing the optimum tour of an instance of the traveling salesman problem.
T	There is a polynomial time algorithm for telling whether in an unweighted graph there is a path of length 10 from s to t.
T	There is a polynomial reduction from maximum flow to linear programming.
T	There is a polynomial reduction from minimum spanning tree to linear programming.
=	There is a polynomial reduction from integer linear programming to linear programming.