FINAL EXAM

Please answer all questions. Begin each question on a new page. Reserve some space for scratch work, which you should hand in. All the best!

1. (10 + 15 points) **[Asymptotics and Recurrences]**
Order these functions in order of increasing Θ-order. When two functions have the same Θ-order, please indicate this explicitly.

   (a) Order the following 5 functions:
   (1) $\log^2 n$
   (2) $n/\log^4 n$
   (3) $\sqrt{n}$
   (4) $n2^{-n}$
   (5) $\log\log n$

   (b) Order the following 4 functions:
   (1) $T_1(n) = n + T_1(n/7) + T_1(6n/7)$.
   (2) $T_2(n) = n + T_2(3n/7) + T_2(4n/7)$.
   (3) $T_3(n) = n + 5T_3(n/6))$.
   (4) $T_4(n) = 1 + 3T_4(n/2))$.

   ANSWER:

   (a) (4) $<$ (5) $<$ (1) $<$ (3) $<$ (2)

   (b) We know from homework that $T_1(n) = \Theta(T_2(n)) = \Theta(n \log n)$. Also, by Master Theorem, we know $T_3(n) = \Theta(n)$ and $T_4(n) = \Theta(n^\alpha)$ where $\alpha = \lg 3 = 1.58....$ Thence the answer is (3) $<$ (1) $=$ (2) $<$ (4).

2. (15 points) **[Basic Graph Algorithms]**
How do you compute the transpose of a graph in $O(m + n)$ time? The transposed graph simply reverses the direction of each edge.

   ANSWER: We go through the adjacency list of each vertex i ($i = 1, \ldots, n$). For each j that occurs in the adjacency list of i, we put i into the adjacency list of j. This clearly takes $O(m + n)$.

3. (15+15 points) **[Shortest Path]**
(a) Hand simulate Dijkstra's algorithm on the graph in figure 1(a), using node A as source.

   (b) In part(a) you only need to compute the lengths of the minimum paths from source to the other nodes. Now, extend your simulation in (a) to compute these minumum paths. HINT: introduce another array $\pi[A, B, \ldots, H]$ to keep track of the paths. Do not give us the algorithm; just show the simulation like part (a).

   ANSWER: (a) The array $d[A, B, \ldots, H]$:

	A	B	C	D	E	F	G	H
1	$\underline{0}$	4	1	10	9	∞	∞	∞
2			$\underline{1}$	9	7		17	13
3		$\underline{4}$		6		13		
4				$\underline{6}$		9	13	
5					$\underline{7}$	8	12	10
6						$\underline{8}$	11	9
7							10	$\underline{9}$
8							$\underline{10}$	

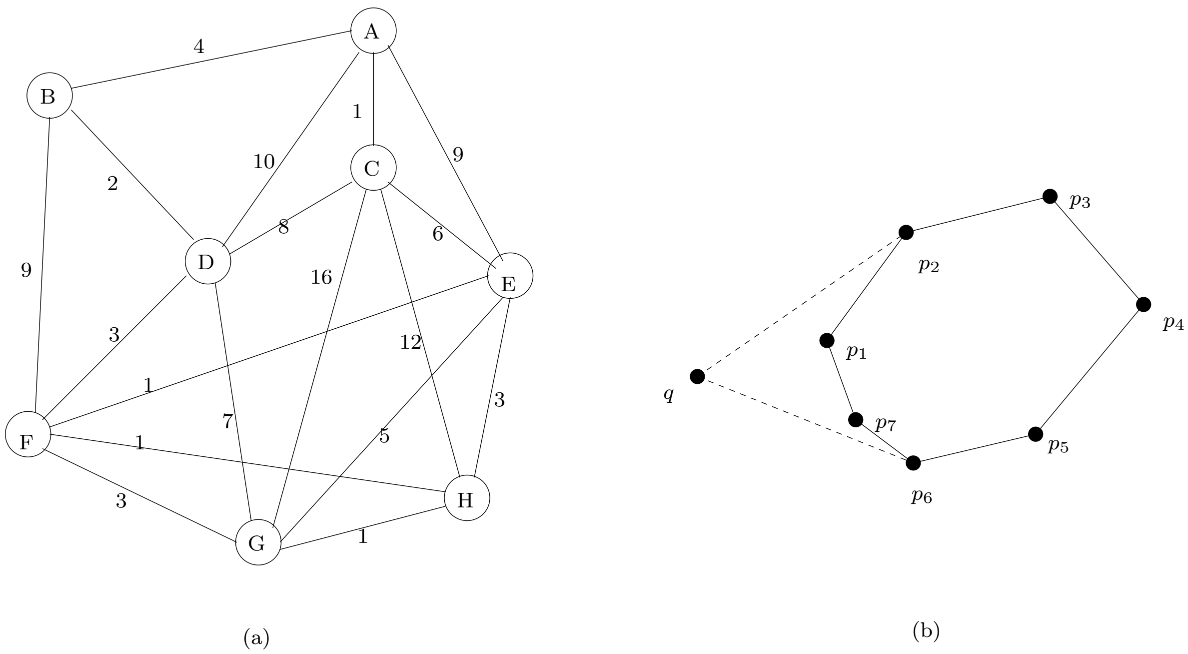

Figure 1: (a) Graph for Dijkstra Algorithm. (b) Tangents from q.

ANSWER: (b) The array $\pi[A, B, \ldots, H]$ indicates the previous node in a shortest path from the source A:

π	A	B	C	D	E	F	G	H
1	$\underline{0}$	A	A	A	A	$--$	$--$	$--$
2			$\underline{A}$	C	C		C	C
3		$\underline{A}$		B		B		
4				$\underline{B}$		D	D	
5					$\underline{C}$	E	E	E
6						$\underline{E}$	F	F
7							H	$\underline{F}$
8							$\underline{H}$	

4. (30 Points) **[Greedy Method and Dynamic Programming]** Recall the grouping problem where, given (M, w) where $w = (w_1, \ldots, w_n)$ are numbers and $w_i \le M$, and you need to compute a partition of w into groups of size at most M. Let $Cost(M, w)$ denote the minimum number of groups.

Useful notations: a **group** is a subsequence of w that has the form $w(i, j) = (w_{i+1}, w_{i+2}, \ldots, w_j)$ where $0 \le i < j \le n$. The **size** of the group $w(i, j)$ is denoted $|w(i, j)|$ and is just the sum of the numbers in the group. E.g., $|w(i-1, i)| = w_i$. A **solution** with k groups can be denoted $[n_1, \ldots, n_k]$ where $1 \le n_1 < n_2 < \cdots < n_k = n$, and $|w(n_{i-1}, n_i)| \le M$, assuming $n_0 = 0$.

We now allow the w_i's to be negative. The original greedy algorithm, even when generalized, no longer works. We want you to (a) design a algorithm, (b) briefly argue why it is correct, and (c) analyze its running time. HINT: Assume that when you solve the problem for (M, w), you also solve it for (M, w') where w' is a suffix of w. Aim for an $O(n^2)$ running time.

ANSWER: Let (M, w) be input. Assume that we have computed the optimum cost for all (M, w^i), $i = 2, \ldots, n$, where $w^i = (w_i, \ldots, w_n)$ is a suffix of w. To compute $Cost(M, w)$

we simply choose it to be $1 + \min_j\{Cost(M, w^j)\}$ where j range over all those "candidate values" where $|w(0, j-1)| \le M$.

Correctness: Clearly, if the first group of the optimum solution is $w(0, j-1)$ then j is among the candidates. The number of groups in the optimum solution then equal to $Cost(M, w^j)$. Thus $Cost(M, w) = 1 + Cost(M, w^j)$.

Complexity: These candidate values can be determined in linear time. Hence $T(n) = n + T(n-1) = O(n^2)$.

5. 15+25 points **[Computational Geometry]**
Recall that if p, q are points in the plane, the text defines a cross product $p \times q$.
(a) Let q, a, b, c denote points. Define the predicate $Inside(q, a, b, c)$ which returns true iff q is strictly inside the triangle formed by a, b, c. Implement this predicate using the cross product operator. HINT: Consider the predicate $Opposite(a, b, c, d)$ which returns true iff c and d are strictly on different side of the line through a, b. For simplicity, you may ignore degeneracies (e.g., what to do if $a = b$, or c lies on the line $\overline{ab}$, etc).

ANSWER (a): It is easy to see that $Inside(q, a, b, c)$ is equal to:

$$Opposite(a, b, c, q); Opposite(b, c, a, q); Opposite(c, a, b, q).$$

(b) Suppose you are given a convex polygon $P = (p_1, \ldots, p_n)$, and also a point q. You are told that q lies outside the polygon. Give an efficient algorithm to compute the two points p_i, p_j $(1 \le i < j \le n)$ such that the lines $\overline{qp_i}$ and $\overline{qp_j}$ are the two tangents to the polygon. E.g., in figure 1(b), we have $i = 2, j = 6$. What is the complexity of your algorithm? Can you do better if the sequence of P is given in an array?

HINT: You can assume an answer to part (a) even if you did not solve part (a). By reordering the points of P, you may assume p_1 is the closest point to q, and $p_1, p_2, \ldots, p_n$ appears in a clockwise order.

ANSWER (b): Note that the polygon is divided into 2 chains, one that is convex towards q, and the other is concave towards q. In the figure, $(p_3, \ldots, p_6)$ is the concave chain. Clearly, p_i is in the convex chain iff $Inside(q, p_{i-1}, p_i, p_{i+1})$. Moreover, the two tangent points that we seek are precisely where the endpoints of the concave chain.

Now, there is a trivial $O(n)$ algorithm: for each p_i, check whether p_i belongs to the convex chain.

OPTIONAL: For a more efficient algorithm, you can do a binary search. But this depends on assuming that the vertices of P are given in an array. We aim for a running time of $O(\log n)$, so our assumption that we can re-order the points of P is no longer valid. But we can overcome this as well. We leave this for an exercise.

6. 15+15 points **[NP-Completeness]**
(a) Suppose someone proves that $NP = P$ tomorrow. What do you think will happen to our practice in Computer Science? This is somewhat open ended, but try to limit yourself to at most two paragraphs.

(b) Let $Comp \subseteq \{0, 1\}^*$ be the set of all binary strings that represent composite numbers (i.e., numbers that are not prime). E.g., the binary string $0101 \notin Comp$, since it represents the prime number 5, but $10101 \in Comp$ since it represents a composite number 21 ($= 3 \times 7$). Prove that $Comp \in NP$.

ANSWER:

(a) It means that we suddenly know that polynomial time algorithms exists for a host of NP-complete problems in every computational fields. Even though these polynomial time algorithm exists, it does not mean they are practical – so there will be a flurry of activity to find the practical versions of these polynomial time solutions.

Since much prior research have been explored to solve NP-complete problem until this new (unlikely) result, it is likely that a very interesting new computational technique must have been used to show $NP = P$. This will probably have interesting consequences in practice.

(b) We need to construct a polynomial time verifier $V(x, y)$ for *Comp*. Let x, y be binary strings. The verifier is easy to construct: define $V(x, y) = 1$ iff y divides x. Clearly, $V(x, y)$ can be computed in polynomial time (in fact in $O(n^2)$). Furthermore $x \in Comp$ iff there exists y such that $V(x, y) = 1$.