CS170 Second Midterm 28 Oct 1999

NAME: ______________________

TA: ______________________

Be clear and concise. You may use the number of points assigned to each problem as a rough estimate for the number of minutes you want to allocate to the problem. The total number of points is 80.

This exam is closed book/closed computer, but you are allowed one sheet of paper (2 sides) for notes to use during the exam. You are allowed to use, without proof, any results proven in class, lecture notes, homework, or the book, but you must state clearly what result you are using.

We are no longer allowed to post grades using SIDs, because of privacy rulings. We will use the same key that you selected last time in order to post grades.

1	
2	
3	
4	
Total	

1. **(15 points)** The following is a forest formed after some number of UNIONs and FINDs, starting with the disjoint sets A, B, C, D, E, F, G, H and I. Both union-by-rank and path-compression were used.

$$\begin{tikzcd}
 & \mathbf{A} & & \mathbf{F} & \mathbf{H} & \mathbf{I} \\
\mathbf{B} \arrow[ur] & \mathbf{C} \arrow[u] & \mathbf{D} \arrow[ul] & \mathbf{G} \arrow[u] & & \\
\mathbf{E} \arrow[u] & & & & &
\end{tikzcd}$$

(a) Starting with the forest above, we now call the following routines in order:

FIND(B), UNION(G, H), UNION(A, G), UNION(E, I).

Draw the resulting forest, using both union-by-rank and path compression. In case of tie during UNION, assume that UNION will put the lexicographically first letter as root.

Answer:

$$\begin{tikzcd}
 & & & \mathbf{A} & & & \\
\mathbf{E} \arrow[urrr] & \mathbf{I} \arrow[urr] & \mathbf{B} \arrow[ur] & \mathbf{C} \arrow[u] & \mathbf{D} \arrow[ul] & \mathbf{F} \arrow[ull] & \\
 & & & & & \mathbf{G} \arrow[u] & \mathbf{H} \arrow[ul]
\end{tikzcd}$$

(b) Starting with the disjoint sets A, B, C, D, E, F, G, H and I, give a sequence of UNIONs and FINDs that results in the forest shown at the top of the page. In case of a tie during union, assume that UNION will put the lexicographically first letter as root.

Answer: One solution is

UNION(F, G), UNION(A, C), UNION(B, E), UNION(B, D), UNION(D, A).

2. **(25 points)** Let $p(x) = \sum_{i=0}^{n} p_i x^i$ and $q(x) = \sum_{i=0}^{m} q_i x^i$ be polynomials of degrees n and m, respectively, where n and m can be any integers such that $n \geq m$.

   (a) Give an algorithm using the FFT that computes the coefficients of $r(x) = p(x) \cdot q(x)$. How many arithmetic operations does it perform, as a function of m and n? Your answer can use $O()$ notation.

   Answer: (1) Round up $n+m+1$ to the nearest power of 2, i.e. find the smallest k such that $2^k \geq n+m+1$: $k = 2^{\lceil \log_2(m+n+1) \rceil}$. (2) Pad the vectors $[p_0, ..., p_n]$ and $[q_0, ..., q_m]$ with enough zeros to make vectors p' and q' of length 2^k. (3) Compute $\hat{p} = \mathrm{FFT}(p')$ and $\hat{q} = \mathrm{FFT}(q')$. The cost is $3k2^k$ complex operations, or $10k2^k$ real operations. (4) Multiply $\hat{r}_i = \hat{p}_i \cdot \hat{q}_i$ for $i = 0, ..., 2^k - 1$. The cost is 2^k complex operations, or $6 \cdot 2^k$ real operations. (5) Compute $r' = \mathrm{invFFT}(\hat{r})$ and extract the leading $n+m+1$ entries. The cost is $1.5k2^k$ complex operations or $5k2^k$ real operations.

   The total cost is $(4.5k+1)2^k$ complex arithmetic operations, or $(15k+6)2^k$ real arithmetic operations, or more simply $O(n \log n)$ operations.

   (b) Give an algorithm *not* using the FFT that computes the coefficients of $r(x) = p(x) \cdot q(x)$. How many arithmetic operations does it perform, as a function of m and n?

   Answer: For $j = 0$ to $m+n$ compute $r_j = \sum_{i=\max(0,j-m)}^{\min(j,n)} p_i q_{j-i}$. The cost is about $2mn$ complex operations, or $8mn$ real operations, or more simply $O(mn)$ operations.

   (c) Combine the above algorithms to give the fastest possible algorithm depending on m and n. How many arithmetic operations does it perform? Roughly how small (in a O() sense) does m have to be for the non-FFT algorithm to be at least as fast as the FFT algorithm?

   Answer: If $(15k+6)2^k \leq 8mn$ use the FFT based algorithm, else the non-FFT algorithm. Or more roughly, if $\log_2 n < m$, then use the FFT-based algorithm.

3. **(25 points)** Given a set $S = \{s_1, s_2, ..., s_n\}$ of n nonnegative integers, and a positive integer T, find a subset of S that adds up to T. Use dynamic programming; your solution should not have a cost growing like 2^n.

   You should (1) Formulate your algorithm recursively, (2) describe how it would be implemented in a bottom-up iterative manner, (3) give a bound on its running time in terms of n and T, and (4) give a short justification of both the correctness of the algorithm and its running time.

   Answer: Define AddUp(T', i) to be True if a subset of $\{s_1, ..., s_i\}$ adds up to $T' \leq T$, and False otherwise. Clearly AddUp($T', 1$) = True if $s_1 = T'$ and False otherwise, and for larger i AddUp(T', i) = AddUp($T', i-1$) $\vee$ Addup($T' - s_i, i-1$). AddUp can be computed by filling in a T-by-n table of all possible values of AddUp(T', i) for $1 \leq T' \leq T$ and $1 \leq i \leq n$, first filling in all values of AddUp($T', 1$), and then AddUp(T', i) for $i = 2$ to n, at a cost of O(1) per table entry, and $O(Tn)$ overall. Finally, one inspects AddUp(T, n), which is true if and only if the problem can be solved. Another T-by-n table Set where Set(T', i) records which of AddUp($T', i-1$) or Addup($T' - s_i, i-1$) is true (pick arbitrarly if both are true) will let the actual set adding up to T be reconstructed.

4. **(15 points) True or False?** No explanation required, except for partial credit. Each correct answer is worth 1 point, but 1 point will be *subtracted* for each wrong answer, so answer only if you are reasonably certain.

   (a) If we can square a general n-by-n matrix in $O(n^d)$ time, where $d \geq 2$, then we can multiply any two n-by-n matrices in $O(n^d)$ time.
   Answer: True.

   (b) If the frequencies of the individual characters in a file are unique, the file's Huffman code is unique.
   Answer: False.

   (c) Huffman coding can compress any file.
   Answer: False.

   (d) The solution to the recurrence $T(n) = 2T(n/2) + O(n \log n)$ is $T(n) = \Theta(n(\log n)^2)$.
   Answer: True.

   (e) $\log^* \log n = O(\log \log^* n)$
   Answer: False.

   (f) In Union-Find (with union-by-rank and path compression), any union only takes $O(\log^* n)$ time, where n is the number of nodes.
   Answer: False.

   (g) In Union-Find data structure with union-by-rank but no path compression, m union and finds takes $O(m \log m)$ time.
   Answer: True.

   (h) If path compression is not used, but union-by-rank is used, it is possible to arrange m LINK and FIND operations so that is takes $\Omega(m \log m)$ time.
   Answer: True.

   (i) If w is a complex n-th root of unity, then $|w| = 1$, where $|w|$ is the absolute value of w.
   Answer: True.

   (j) If we want to use FFT to multiply two polynomials of degree $n = 2^m$, we need to run the FFT on vectors of length $2n$.
   Answer: False. We need at least $2n + 1$ points, so $4n$ if we stick with powers of two.

   (k) The values of a degree n polynomial at n+2 distinct points determines its coefficients uniquely.
   Answer: True. $n + 1$ points would be enough too.

   (l) To find a optimal way to multiply 6 matrices A1*A2*...*A6, we can find an optimal way to multiply A1*A2*A3, and to multiply A4*A5*A6, and combine the results.
   Answer: False.

   (m) Floyd-Warshall algorithm works with negative edge weights when there are no negative cycles.
   Answer: True.

   (n) Floyd-Warshall algorithm is always asymptotically faster than running Dijkstra n times, where n is the number of vertices.
   Answer: False. Floyd-Warshall is always $O(n^3)$, whereas running Dijkstra (with a binary heap) n times on a graph with positive edge weights is $O(ne \log n)$, which is smaller if the graph is sparse.

   (o) You wrote your name and your TA's name on the first page.