B1 Algorithms (25 points)

Do all three parts.

a) [6 points] This question deals with heaps and Heapsort.

i) Describe a procedure to build a binary heap of n elements in linear time. Briefly argue that the procedure is done in time O(n).

ii) Describe the Heapsort algorithm and analyze its time complexity.

b) [9 points] Suppose E1 and E2 are arrays, each with n keys sorted in ascending order. Describe an O(log n) algorithm to find the nth smallest of the 2n keys (this is the median of the combined set.) For simplicity, you may assume the keys are distinct.

c) [10 points] Suppose that we want to augment the binomial and Fibonacci heaps to support an operation Increase-Key(h,i,d), which, in the heap h of n elements, increases the value of element i by $d > 0$.

i) For binomial heaps, give and analyze a way to add this operation that runs in $O(\log^2 n)$ or better.

ii) For Fibonacci heaps, give and analyze a way to add this operation that runs in amortized time O(log n), and does not change the amortized running time for the other operations.

B2 Algorithms (25 points)

Do all three parts.

a) [4 points] Define ***NP-complete***. You may assume that the term ***polynomially reducible*** has already been defined. If you use any other term related to NP-completeness, you need to briefly define it.

b) [10 points] For each of the following problems, state whether or not it is NP-complete. If it is not NP-complete, then sketch (one or two lines) a linear time algorithm for solving the problem. If it is NP-complete, do nothing more.

i) Given an undirected graph G, does there exist a circuit that contains every vertex exactly once?

ii) Given an undirected graph G, does there exist a circuit that contains every edge exactly once?

iii) Given an undirected graph G, does there exist a 2-coloring of G?

iv) Given an undirected graph G, does there exist a 3-coloring of G?

c) [11 points] Give a complete proof that the following problem is NP-complete. You may assume that 3SAT, CLIQUE, PARTITION and HAMILTONIAN PATH are known NP-complete problems.

IS: INDEPENDENT SET

INSTANCE: An integer B, and an undirected graph G=(V,E).

QUESTION: Does G contain an independent set of size B?

Note: An independent set S of graph G=(V,E) is a subset of V such that there are no edges in E with both endpoints in S. The size of the independent set is the number of vertices in S.

B3 Algorithms (25 points)

Do all three parts.

a) [5 points] Suppose an algorithm A exists for matrix multiplication that works in $O(n^k)$ arithmetic operations for some constant k in the case when n is a power of 2. Show that A can be adapted so that it works for all n, still using $O(n^k)$ arithmetic operations.

b) [10 points] The standard method of multiplying two numbers, $x = x_1 B + x_0$ and $y = y_1 B + y_0$, in base B involves 4 B-digit multiplications (each digit of x times each digit of y). But x and y can be multiplied using only 3 multiplications (of "digits" less than 2B) as follows:

$z_0 := x_0 * y_0$
$z_2 := x_1 * y_1$
$w := (x_1 + x_0) * (y_1 + y_0)$
$z_1 := w - z_0 - z_2$
$ans := z_2 B^2 + z_1 B + z_0$

Note that the final step only involves shifting and handling carries. It is not required to multiply by B or B^2. Show how this may be used to multiply two numbers having n B-digits in $O(n^{\lg(3)})$ B-digit operations.

c) [10 points] Polynomial time reductions of one problem to another are a useful tool in the study of the classes P and NP and the theory of NP-Completeness. Reductions can also be useful in other situations where the ultimate classification of the cost of problems remains undetermined. For example the arithmetic cost of multiplying n by n matrices is in $O(n^3)$ using the classical algorithm, in $O(n^{2.81})$ using Strassen's algorithm, and in $O(n^{2.37})$ using the Coppersmith/Winograd algorithm. The minimal exponent ω such that multiplication is in $O(n^{\omega})$ is unknown and may be as low as 2.

A number of other linear algebra problems are equivalent to matrix multiplication in the sense that the arithmetic cost is in $O(n^{\omega})$. This is shown by reductions between problems. Give a reduction for ***any one*** of these problem pairs with the reduction cost being in $O(n^2)$. You may assume the matrix is in general position.

- MM to SQ
- MM to TMI
- TMI to MM
- MM to LUD
- LUD to MM
- LSS to LUD
- BMM to TC

Definitions:

- MM Multiply 2 n by n matrices.
- BMM Multiply 2 n by n Boolean matrices.
- SQ Square an n by n matrix.
- TMI Invert an n by n lower triangular matrix.
- LUD Decompose an n by n matrix as a product of a lower triangular matrix and an upper triangular matrix. Assume the leading principal minors are nonsingular (thus no pivoting is required).

- LSS Linear System Solution. Given n by n matrix A and n-vector b, Solve Ax = b for n-vector x.
- TC Compute transitive closure. The transitive closure of graph G = (V, E) is G* = (V, E*), where E* = {(i,j) : there is a path from vertex i to vertex j in G}.

B4 Algorithms (25 points)

Select any 4 of the following 5 items (do only 4 -- if you do all 5, only the first 4 will be graded).

1) Suppose that p(x) and q(x) are polynomials of degree n-1. Give an O(n log n) algorithm to compute the polynomial h(x) = p(x)*q(x).

2) Sketch the main ideas in an algorithm that takes as input a string S and a pattern P, and returns the index of the first occurrence of P in the string SP. The algorithm that you sketch should have a linear worst case running time.

3) Give the fastest known algorithm for finding the connected components of an undirected graph. Be sure to: a) name the algorithm that you give; b) carefully explain the algorithm; c) state the running time of the algorithm.

4) Give the fastest known algorithm for finding the weight of a minimum spanning tree of an edge weighted undirected graph. Be sure to: a) name the algorithm that you give; b) carefully explain the algorithm; c) state the running time of the algorithm.

5) Give a data structure for efficiently performing each of the following operations in O(log n) time: MIN, INSERT, DELETE_MIN, UNION. Be sure to explain the data structure and why each operation can be done in O(log n) time.