B1 Algorithms (25 pts)

Complete any 6 of the following 7. If you do all 7, only the first 6 will be graded. Each part is worth 4 points. You get one point for free.

a) State the solutions to each of the following recurrences using order of magnitude notation:

$T(n) = T(2n/3) + n$

$T(n) = 2T(n/2) + n$

$T(n) = 2T(n/2) + 1$

$T(n) = 4T(n/2) + 1$

$T(n) = T(n/2) + 1$

$T(n) = T(\text{sqrt}(n)) + 1$

Useful fact: $\log_2 n = 1/(\log_n 2)$

b) Describe quicksort. What is the worst case running time? Explain your answer.

c) Give an algorithm for finding the median of a set of n numbers in linear worst case time.

d) Define binary search tree.

e) Name two methods of hash table collision resolution.

f) What is amortized analysis? (be sure to name and describe at least two methods for performing such analyses).

g) Give an example of dynamic programming. Provide enough details in your explanation that the reader can understand that the example actually works.

B2 Algorithms (25 pts)

Complete any 3 of the following 4. If you do all 4, only the first 3 will be graded. Each part is worth 8 points. You get one point for free.

a) Explain the iterative version of the FFT.

b) Suppose there was a matrix multiplication algorithm that multiplied two 4 by 4 matrices of integers using 23 integer multiplications and 100 integer additions. Answer the following four questions.

1) Explain how to best use this algorithm to multiply two n by n matrices of integers.

2) What would be the resulting recurrence?

3) What is the solution to that recurrence?

4) How many integer multiplications would the method use to multiply two 64 by 64 matrices?

c) Answer the following three questions.

1) Describe the varieties of PRAMs. Be sure to mention to include at least three CRCW models.

2) What is pointer jumping? Give an example of a parallel algorithm that uses pointer jumping. Be complete in describing how the algorithm works.

3) Give an O(1) algorithm for finding the maximum of n integers on a CRCW PRAM. How much work does your algorithm use?

d) Answer the following three questions.

1) Define NP-complete.

2) Minimization problems are often known to be NP-hard, but not NP-complete. What is the apparent problem that makes it tough to prove them NP-complete?

3) The problem of 3SAT can be reduced to the problem of CLIQUE using the construction given below. Prove that the construction is correct.

From an instance of 3SAT consisting of k clauses, we construct the following undirected graph:

For each 3SAT clause $C_i = (l_{i1}, l_{i2}, l_{i3})$, we construct three vertices v_{i1}, v_{i2}, v_{i3} (these are in 1:1 correspondence with the literals in the clause).

We place an edge between v_{ir} and v_{js} if: $i \neq j$ -and- l_{ir} is not the negation of l_{js}

These are the only vertices and edges in the graph.

We claim that the graph contains a clique of size k iff the instance of 3SAT is satisfiable.

B3 Algorithms (25 pts)

Minimum Spanning Trees. Answer all 7 parts. Each part is worth 3.5 points. You get 0.5 points for free.

Let G be a connected graph with positive weights on its edges. Let M be the subgraph whose edges are the minimum weight edges from G. Thus the edges in M all have the same weight and this weight is strictly less than the weight of any other edge in G. The set of vertices of M is the set of endpoints of these minimum weight edges.

a) Define a *spanning forest* in a graph, such as M, which may not be connected. Be precise.

b) Define a *minimum spanning tree* in a graph, such as G, which is connected. Be precise.

c) Show that every minimum spanning tree of G contains at least one edge from M.

d) Show that each edge from M is contained in some minimum spanning tree of G.

e) Must every minimum spanning tree of G contain every edge from M? Prove or give counterexample.

f) Show that every minimum spanning tree of G contains a spanning forest of M.

g) Show that every spanning forest of M is contained in a minimum spanning tree of G.

B4 Algorithms (25 pts)

Mergable Heaps. Answer all 5 parts.

a) (7 pts) Describe the data structure used in Binomial heaps.

b) (7 pts) Describe the data structure used in Fibonacci heaps.

c) (6 pts)

1. State the worst case asymptotic run time of a single *insert* operation on a Binomial heap of size n.
2. State the worst case asymptotic run time of a single *insert* operation on a Fibonacci heap of size n.
3. State the worst case asymptotic run time of a single *extract-minimum* operation on a Binomial heap of size n.
4. State the worst case asymptotic run time of a single *extract-minimum* operation on a Fibonacci heap of size n.
5. State the worst case asymptotic run time of a single *union* operation on a Binomial heap of size n.
6. State the worst case asymptotic run time of a single *union* operation on a Fibonacci heap of size n.

d (3 pts) State the asymptotic run time of the Fibonacci heaps operations (*insert*, *extract-minimum*, and *union*) as amortized over a sequence of n of the operations.

1. *insert*:
2. *extract-minimum*:
3. *union*:

e (2 pts) Considering the total range of mergable heap operations, do Binomial heaps have any advantage asymptotically over Fibonacci heaps?