B1 Algorithms (25 points)

Select any 5 of the following 6 items (do only 5 -- if you do all 6, only the first 5 will be graded). For each item that you select, give a short explanation of the item (data structure, algorithm, or technique). You should capture the main idea/ideas of the data structure, algorithm, or technique in your explanation, and, for algorithms, state the running time.

1) Kruskal's algorithm

2) Fibonacci trees

3) Decision trees (and give a brief example)

4) Depth first search

5) Quicksort

6) FFT

B2 Algorithms (25 points)

Do part a or part b or part c. *Do not do more than one part.* If more than one part is attempted, then only the first part will be graded.

a. *PARALLEL ALGORITHMS*

i. [7 points] Carefully describe the CREW PRAM model. Your description should include both what constitutes the PRAM model and what the CREW variant is all about.

ii. [9 points] There exist problems that require time O(log n) on a CREW PRAM, but can be solved in time O(1) on a CRCW PRAM.

- State one such problem.
- For the problem that you stated, outline an O(log n) CREW algorithm for that problem and state the work required by that algorithm.
- For the problem that you stated, outline an O(1) CRCW algorithm for that problem and state the work required by that algorithm.

iii. [9 points] Sketch an O(log n) time CREW algorithm for merging two ordered lists. How much work does your algorithm use?

b. *PATTERN MATCHING*

Assume that T is a text string and that P is a pattern, both drawn from a finite alphabet Σ. Pattern matching is the problem of locating the leftmost occurrence (if any) of P in T.

Give the major ideas behind each of the following pattern matching algorithms. Your descriptions should capture the essential ideas of the algorithms and should be carefully explained.

i. [12 points] The Boyer-Moore pattern matching algorithm.

ii. [13 points] The Knuth-Morris-Pratt pattern matching algorithm.

c. *NETWORK FLOW*

i. [10 points] Define the network flow problem.

ii. [15 points] Consider the problem of ***bipartite k-matching***: You are given a positive integer k, sets $A = \{A_1, A_2, ..., A_n\}$ and $B = \{B_1, B_2, ..., B_m\}$ and a set of compatibility constraints of the form $<A_i, B_j>$. Here, we say that A_i and B_j are *compatible*. The problem is to find k disjoint pairs (A_i, B_j) such A_i and B_j are compatible. Here, disjoint means that any given A_i or B_j can appear in at most one pair.

Bipartite k-matching can be solved by changing the problem into a network flow problem. Carefully explain how to do that.

B3 Algorithms (25 points)

Do all three parts.

a. [6 points] Define P. NP, and NP-complete. You may assume that polynomially reducible has already been defined.

b. [8 points] Name and briefly describe four different NP-complete problems (do not use any type of CLIQUE problem).

c. [11 points] Recall that CLIQUE is the problem, given a positive integer k and an undirected graph G=(V,E), does G contain a clique (i.e. a complete subgraph) of size k? It is well-known that CLIQUE is NP-complete.

A related problem is WEIGHTED COMPLETE CLIQUE. Here, you are given a positive integer k, a real number B, and a COMPLETE weighted undirected graph G'=(V',E', w), where w is a function that associates a non-negative weight with each edge. The question is: does there exist a k-clique in G' of total weight not exceeding B (i.e. if we add the weights of all of the edges in the k-clique, their total weight is less than or equal to B)?

Give a reduction from CLIQUE to WEIGHTED COMPLETE CLIQUE. You do not have to prove that the construction that you give is correct. Just give the construction itself.

B4 Algorithms (25 points)

Do all 6 parts.

Let A be an array of n distinct numbers. Selection of the k-th smallest element may be done in O(n) expected time. CLRS describes the algorithm as follows.

Randomized-Select(A, p, r, i) // return i-th smallest in A[p..r].

1. If p = r, then return A[p]
2. q = Randomized-Partition(A, p, r) // now A[p..q-1] < A[q] < A[q+1..r].
3. k = q+1 - p // rank of A[q]
4. if k = i, then return A[q]
5. elseif i < k, return Randomized-Select(A, p, q-1, i)
6. else return Randomized-Select(A, q+1, r, i-k)

Thus, for example, Randomized-Select(A, 1, n, n/2) returns the median of the array A of length n.

a) [5 pts] Consider the problem of finding the deciles of the data. That is, the nine elements Randomized-Select(A, 1, n, d*n/10), for d = 1, 2, ..., 9. Clearly this can be done faster by modifying Randomized-Select than by calling it 9 times independently. Write an algorithm to do this.

b) [5 pts] What is the worst case asymptotic run time of your deciles algorithm? Explain. Assume Randomized-Partition(A, p, r) runs in O(r-p) worst case time.

c) [4 pts] Suppose Randomized-Partition is replaced by q = Perfect-Partition(A, p, r), which partitions the range A[p..r] exactly in half, i.e., q = floor((p+r)/2)), and also runs in O(r-p) time. In this case, say more precisely how much faster your algorithm is than 9 calls to Perfect-Select (which is like Randomized-Select except that it uses Perfect-Partition).

Consider this sorting algorithm.

Stooge-Sort(A, i, j) // sort A[i..j]

1. n = j+i - i
2. if A[i] > A[j], swap (A[i],A[j])
3. if n <= 2, return
4. k = floor(n/3)
5. Stooge-Sort(A, i, j - k) // first two thirds
6. Stooge-Sort(A, i + k, j) // second two thirds
7. Stooge-Sort(A, i, j - k) // first two thirds again

d) [5 pts] Explain why Stooge-Sort(A, 1, n) correctly sorts an array A of length n.

e) [4 pts] Give a recurrence relation for the worst case running time of Stooge-Sort.

f) [2 pts] Solve the recurrence.