Problem 1. Let $f(n)$ and $g(n)$ be any two functions over the set of nonnegative integers.

1a [5]. True or false: either $f(n) = O(g(n))$ or $g(n) = O(f(n))$.

ANSWER: false. One counterexample is the case where $f(n)=\sin(\pi n)$ and $g(n)=\cos(\pi n)$.

1b [5]. True or false: if $f(n) = \Theta(g(n))$, then there are positive numbers c, d, and n_0 such that for every $n \geq n_0$, $cg(n) \leq f(n) \leq dg(n)$ and $cf(n) \leq g(n) \leq df(n)$.

ANSWER: true. Explanation: we know there are positive numbers c_1, d_1, and n_1 such that for every $n \geq n_1$, $c_1 g(n) \leq f(n) \leq d_1 g(n)$; and positive numbers c_2, d_2, and n_2 such that for every $n \geq n_2$ $c_2 f(n) \leq g(n) \leq d_2 f(n)$. Just let $c = \min(c_1,c_2)$; $d = \max(d_1,d_2)$; and $n_0 = \max(n_1,n_2)$.

Problem 2 [10]. Suppose you are using linked lists to represent an $n \times n$ sparse matrix, as illustrated in Figure 5.5 of the book. In the worst case, how much time will it take to compute the sum of the elements along the main diagonal? A big-O answer is OK as long as it is a tight bound.

ANSWER: $O(n^2)$. We can't do it in time $O(n)$ because to get from $X(i,i)$ to $X(i+1,i+1)$, in the worst case we might need to search row $i+1$ or column $i+1$.

Problem 3 [10]. What is the Knuth-Morris-Pratt matrix for the string "abcabcabc"?

ANSWER:

	a	b	c	a	b	c	a	b	c
a	0	1	2	0	4	5	0	7	8
b	1	0	3	4	0	6	7	0	9
c	1	2	0	4	5	0	7	8	0
Other	1	2	3	4	5	6	7	8	9

Problem 4 [10]. Let S and T be 2-3 trees of the same height h. Suppose every key of S is smaller than every key of T. Explain how to write a function CONCATENATE*(S,T)* that runs in time $O(h)$ and returns an AVL tree that contains all of the nodes of S and T. You don't need to write the actual code, as long as you make it very clear how the program works.

ANSWER: As I pointed out during the exam, there's a typographical error above: I intended to ask for a 2-3 tree rather than an AVL tree.

Obviously it's impossible to construct an entire AVL tree in time $O(h)$, but here's an algorithm that returns a 2-3 tree in time $O(h)$:

```
procedure CONCATENATE(S,T)
    remove the largest key s from S
    if the resulting tree S' has height h then
        return the tree whose root is s and whose left and right subtrees are S' and T
    (if we get to this point then S' has height h-1)
    remove the smallest key t from T
    if the resulting tree T' has height h then
        undo the operations you did to remove s, thus restoring S to its original state
        return the tree whose root is s and whose left and right subtrees are S and T'
    (if we get to this point then both S' and T' have height h-1)
    let U be the tree whose root is s and whose left and right subtrees are S' and T'
    return the result of inserting t into U
```

Problem 5. Suppose P is a set of two-dimensional points.

5a [10]. Is it possible for there to be more than one adaptive k-d tree for P? Explain.

ANSWER: Yes. When choosing what discriminant to use at a node, if the x coordinates and the y coordinates are equally spread out, then one could use either x or y as the discriminant.

5b [10]. Let S and L be the smallest and largest possible heights of any k-d trees for P; and let S' and L' be the smallest and largest possible heights of any *adaptive* k-d trees for P. What is the relation between S, L, S', and L'?

ANSWER: $S \le S' \le L' \le L$. We can't use "<" rather than "≤" in this inequality, because there are cases in which equality occurs.

Problem 6. Suppose I have a set of two-dimensional points, and I want to find the leftmost one (i.e., the one with the smallest x coordinate) whose x coordinate is greater than some value *t.*

6a [10]. How I would do this if *P* were represented as a range tree?

ANSWER: Recall that a 2-dimensional tree consists of a one-dimensional range tree *T* for x, with one-dimensional range trees for y attached to the nodes of *T*. Just ignore the range trees for y and do a binary search of *T* itself. Here are the details if you need to see them:

```
procedure search(node n, integer t)
    while n is not a leaf
        if n's x coordinate is greater than t then n := n's left child
        else n := n's right child
    repeat
    if n's x coordinate is greater than t then return n; else return the next leaf to n's right
```

6b [10]. How I would do this if *P* were represented as a point quadtree?

ANSWER: The basic idea is to use the algorithm discussed in class for finding all points in the region $t<x<\infty$, $-\infty<y<\infty$, but modify it so that it keeps track of the leftmost point it finds:

```
procedure search(node n, integer t)
    declare local variables p and q  that represent two-dimensional points
    p = (∞,∞)
    q  = (∞,∞)
    if n's x coordinate is greater than t then
        (n is to t's right, so the answer is either n or its leftmost descendant)
        if n has a northwest child then p = search(n's northwest child,t)
        if n has a southwest child then q = search(n's southwest child,t)
        return whichever of n, p, and q has the smallest x coordinate
    else
        (n is to t's left, so check whether n has a descendant to t's right)
        if n has a northeast child then p = search(n's northwest child,t)
        if n has a southeast child then q = search(n's southwest child,t)
        return whichever of p and q (but not n this time!) has the smallest x coordinate
        (note that if n has no children to the west, then this will return (∞,∞))
```

Problem 7 [10]. One way to implement the reference-count method described on page 345 is to precede the assignment $P \leftarrow Q$ with the following steps: (i) decrement the count of the cell that *P* points to, (ii) release that cell if the count is zero, (iii) increment the count of the cell that *Q* points to. Modify the above steps so that they will work correctly even if *P* and *Q* point to the same memory location.

ANSWER: Do step iii before step ii.

Problem 8. For each of the following memory management strategies, find a sequence of allocation and deallocation requests that defeats it (i.e., makes it unable to satisfy an allocation request even though the allocation request is less than the total amount of available memory).

8a [10]. The roving pointer strategy

ANSWER: Suppose the total amount of memory is 8 words long.
a = allocate(2); b = allocate(4); free(a); c = allocate(3)

8a [10]. The buddy system

ANSWER: Suppose the total amount of memory is 8 words long.
a = allocate(2); b = allocate(2); free(a); c = allocate(5)

Problem 9 [10]. Let p be the number of bits needed for a pointer, r be the number of bits needed for a record, and α be the load factor. Under what circumstances, in terms of these three parameters, is the hash table organization of Figure 8.7 more economical in its use of storage than that of Figure 8.6?

Clarification posted on the board during the exam:
r is the total number of bits needed in a record, including the pointer.

ANSWER: Recall that $\alpha = n/m$, where m is the table size and n is number of items stored in it. Figure 8.6 uses $mp+nr$ bits. Figure 8.7 uses $mr+(n-m+b)r$ bits, where b is the number of empty buckets. The average value for b is m times the probability that any one bucket is empty. A bucket is empty when all n keys hash to the other $m-1$ locations, which happens with probability $(m-1)^n/m^n$. Thus, the expected number of empty buckets is $(m-1)^n/m^{n-1}$. this means that on the average, Figure 8.7 is superior if $mp+nr > mr+(n-m+(m-1)^n/m^{n-1})r$. This happens if $mp > (m-1)^n/m^{n-1}r$; i.e., if

$$p/r > (1-1/m)^n = (1-1/m)^{m\cdot(n/m)} = e^{-\alpha}.$$

Nobody actually got the above answer—but I gave you substantial partial credit if you said that Figure 8.7 is more economical than Fig. 8.6 when p is large and r is small (so that $p \approx r$) and α is large.

Problem 10 [10]. You are given a sequence of n elements to sort. The input is a sequence of n/m subsequences, each of length m, with all the elements in each subsequence less than all the elements in the next. Describe a way to sort the input in time $O(n \log m)$.

ANSWER: For each subsequence, use merge-sort to sort it in time $O(m \log m)$, and then concatenate the subsequences in time $O(n)$. The total time will be
$(n/m)\, O(m \log m) + O(n) = O(n \log m)$.