Sample Midterm Exam

1 Recurrence Relations [20 points]

Suppose that we implement a variation of mergesort called 3-mergesort where we split our list into three equal size lists, sort each of these recursively with a call to 3-mergesort (if the list is of size 3 or more), and then merge the three sorted lists together to form the final sorted list.

Assume that the procedure we use for merging the three lists uses at most $2n$ comparisons to merge 3 lists of size $n/3$.

(a) Write a recurrence relation which describes the running time of this sorting algorithm.

(b) Solve the recurrence derived in part (a). (A $\Theta(\cdot)$ answer is fine.)

(c) Now suppose that whenever we do a recursive call to 3-mergesort we have to copy the entire original array at a cost of $N$ where $N$ was the original list size. What is the running time for 3-mergesort in this case? (A $\Theta(\cdot)$ answer is fine.)

2 Randomized Algorithms [25 points]

Suppose that you have two lists of integers all between 1 and 100. $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$.

You know that either $Y$ is an exact copy of $X$ (including the order of the items), or $Y$ is a list of random integers drawn uniformly between 1 and 10.

Part A. Give a Monte Carlo algorithm which tests if the two lists are identical and makes an error with probability at most $1/10^9$ for any input list $X$. Design an algorithm which is as fast as possible. (thus your algorithm should output either “copy” or “random” to describe list $Y$).

Note that this is for the worst possible input list $X$, which would have all numbers at most 10.

Part B. Justify the error bound for your algorithm.

Part C. What is the expected running time of your algorithm?

3 Dynamic Programming [35 points]

We consider some variations of the 0/1 knapsack problem. As before there are $n$ items, each with a integer weight $w_i$ and an integer value $v_i$, and a weight limit $W$. We want the subset of items of maximum value such that their weight is at most $W$. 
Part A. Suppose that you can now take 0, 1, or 2 copies of each item. Assume that we still want to compute an $n$ by $W$ matrix $T$ where $T[i,j]$ is the best value solution of weight exactly $j$ we can get using items 1 to $i$.

Give a formula for computing $T[i,j]$ and briefly justify your answer. Remember to specify the first row values also.

Part B. Give the running time to fill in the $T$ matrix to within $\Theta$ accuracy.

Part C. If the values were real numbers rather than integers would this affect the running time of part B? What if the weights were real numbers? Justify your answers.

Part D. Now suppose we have the same situation as in part A, but now if you use two copies of an item the second one will only contribute half its value (thus if an item has weight 10 and value 30 and we take two copies of it, the weight contribution is 20 but the value contribution is $30+15 = 45$).

Give a new formula for computing a $T[i,j]$ value in this setting.

4 Lower Bound [20 points]

Suppose you are given two sorted lists of $n$ real numbers, $X = x_1, < x_2, \ldots, < x_n$ and $Y = y_1, < y_2, \ldots, < y_n$.

We want to check if the two lists are identical. Prove that any comparison based algorithm for checking if the two lists are the same must use $n$ comparisons on its worst input.

5 Hashing. [20 points]

Consider the perfect hashing scheme discussed in class. Given a fixed set of keys $S$ this scheme allowed us to do lookups which determine in $O(1)$ time if an element $x$ was in the set $S$. Now suppose that we have constructed our perfect hashing scheme for $S$ and we then want to modify $S$.

Suppose we want to add an element $x_{subi}$ to $S$. As in a) describe how to modify the perfect hashing scheme to accommodate adding $x_{subi}$ to $S$. Your scheme should keep the worst case lookup time at $O(1)$ even if multiple insertions are made. Describe your insertion scheme which supports insertions and give both its worst case time to insert a new item and the amortized time for an insertion (that is, if you do $k$ inserts, what is the time to do all $k$ inserts).

Your goal is amortized expected $O(1)$ insert time and $O(1)$ worst case lookup time (note: we are now allowing possibly many new inserts).

6 Data Structures [20 points]

Given a set $S$ of numbers, suppose we want to answer queries of the form: I want $x \text{th}$ through $y \text{th}$ smallest numbers in $S$ (e.g. the 7-12th smallest numbers).

How would you handle this:

a) If the set $S$ were static.

b) If there were insertions and deletions to the set $S$.

In each case give the worst case run time of your solution and justify its correctness.