Sample Midterm Exam

Instructions:
Open book and notes. Please write clearly and succinctly; be sure to give an overview before plunging into details

1 Data Structures

Suppose we have $n$ data items which are stored in a ternary search tree, TST, in which each node has three children. Thus an internal node has two keys $k_1 < k_2$ with the property that every key $k$ in the leftmost subtree has a value $k < k_1$, each key in the middle subtree has $k_1 < k \leq k_2$ and all in the right subtree have $k > k_2$. For simplicity you may assume a complete balanced TST (thus it has one node, or 4 nodes, or 13 nodes, ...) with all data stored in the leaves (see example below).

Assume that the data has a key field with values $x_1 < x_2 < \ldots < x_n$.

**Part A.** Suppose you want to answer a rank query in a TST (thus given an integer $r$ you are to return $x_r$, the $r$th smallest element). Assuming each node contains a size field which is the number of leaves in its subtree, describe how to answer a rank query in a TST. Give the worst case run time of your algorithm in terms of comparisons.

**Part B.** Now assume we want to answer a range query: given values $L \leq H$ return all keys with $L \leq x_i \leq H$. Describe how to answer such a query in a TST and give the worst case run time.

2 Amortized Analysis

We consider the situation where we repeatedly insert items into a table, and when the table overflows we allocate a new larger table. As in the text we count the total number of insert operations.

Suppose we have a size $n$ table with $n$ items in it. When we add item $n+1$, we allocate a new table of size $3n$ and insert the old $n$ items plus the new one into that table (so the cost of this insert is $n+1$). We assume initially that we have one item in a table of size 3.

We can analyze this setting using a potential function whose value after the $i$th insert is:

$$num_i - s_i/2$$

where $num_i$ is the current number of items in the table, and $s_i$ is its current size.

**Part A.** Calculate the amortized cost of an insert (i.e. actual cost + change in potential) using this potential function.

**Part B.** Analyze the total cost of inserting $m$ items (one at a time) using this potential function.
3 Dynamic Programming

Note: This problem would rather hard for our exam since we didn’t cover knapsack this quarter. However, it is still indicative of the kind of question I might ask about dynamic programming.

We consider a variation of the 0/1 knapsack problem (so each item can be used at most once). As before there are $n$ items, each with a integer weight $w_i$ and an integer value $v_i$, and a weight limit $W$. In addition, each item has a size $s_i$ and we have a size limit $S$ (imagine there is both a weight limit on what you can carry and a size limit of what you can fit in the knapsack). We want the subset of items of maximum value such that their weight is at most $W$ and their size is at most $S$.

We can solve this problem by computing the values of a 3D table $T$ where $T[i, j, k]$ is the best value solution with weight exactly $j$ and size exactly $k$ using a subset of items $1, 2, \ldots, i$ (value - infinity if no solution has this weight and size). Below you are asked to fill in some details.

Part A. Describe how to fill in the initial table values for $i = 1$.

Part B. Give a formula for computing $T[i, j, k]$ assuming you have already computed the $T[i-1, j, k]$ values. Briefly justify your answer.

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

Part D. Assuming we have filled in the entire table $T$ (and saved it), how do we then find the actual optimal set of items to use?

4 Upper and Lower Bounds

Suppose you are given two sets of $n$ integers, $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$.

We want to check if the two sets are disjoint; no element in $X$ is also in $Y$. We consider several settings below. You may assume that you know the value of $n$ and both sets have $n$ elements

Part A. Assume that the only operation you can perform on the sets is a call to EQUAL($i, j$) which returns:

- invalid if $i$ or $j$ are out of range ($> n$ or $< 1$)
- yes if $x_i = y_j$,
- no if $x_i \neq y_j$.

Prove a lower bound on the number of calls to EQUAL required to determine if $X$ and $Y$ are disjoint.

Your lower bound should be as strong as possible. (hints: use an adversary argument, and the correct answer should match the run time of a simple algorithm to solve this).

Part B. Now assume that you can compare any pair of elements (both in $X$, both in $Y$, or one in $X$ and one in $Y) and get back the usual $<, >, =$ . In this setting describe an efficient algorithm (NOT a lower bound) to determine if $X$ and $Y$ are disjoint. A good answer to parts A,B will have an algorithm for $B$ which is faster than your lower bound for $A$. Analyze the number of compares used by your solution.

Part C. Now assume that you have the sets $X$ and $Y$ stored in two length $n$ arrays. You can now use an arbitrary $C$ program to test if $X$ and $Y$ are disjoint.

Describe a fast Las Vegas algorithm (good expected time, no errors) which determines if $X$ and $Y$ are disjoint. What is the expected run time of your algorithm? (hint, you should be able to beat your part B answer)?