CS381 Examples of Exam Questions

The below samples illustrate the types of questions that I could ask in exams (both midterms and finals). An actual exam would have roughly twice as many questions as the below (it would have 100 points whereas the examples below are worth 52 points only). [*The italicized notes between square brackets explain my rationale for asking each sample question – no such notes would be included in an actual exam.*]

Question X. (16 points)
[*This is meant to test proficiency in quickly estimating an algorithm's time complexity – a student who understands the recursion tree method of analysis can do these quickly, without too much complicated math.*]
In all of the four recurrences shown below, it is assumed that $T(1) = d$ for some constant d. State, using the "big oh" notation, the solution to each of the recurrences shown below. Just state the answers – you do not need to justify them.

1. $T(n) = 4T(n/2) + cn^2$
   Answer: $O(n^2 \log n)$
2. $T(n) = 3T(n/3) + cn^2$
   Answer: $O(n^2)$
3. $T(n) = T(n/2) + T(n/3) + cn$
   Answer: $O(n)$
4. $T(n) = T(n/4) + T(3n/4) + cn$
   Answer: $O(n \log n)$

Question Y. (9 points)
[*This is an example of a question based on a homework, to test who actively worked on that homework and understands its solution.*]
Given n distinct numbers $y_1, \ldots, y_n$ (not in sorted order), let y^* be a number y that minimizes $f(y) = \sum_{i=1}^{n} |y - y_i|$ where $|y - y_i|$ denotes the absolute value of $y - y_i$. Mark by T (= True) or F (= False) each of the following statements about this problem:

1. True: We must have $f(y_i) = f(y^*)$ for some $i \in \{1, 2, \ldots, n\}$, i.e., one of the n numbers $y_1, \ldots, y_n$ is guaranteed to minimize the function f.
2. False: We must have $y^* \in \{y_1, y_2, \ldots, y_n\}$, i.e., any value of y that minimizes $f(y)$ *has* to come from the set $\{y_1, \ldots, y_n\}$.
3. False: There are always infinitely many choices for y^*, i.e., infinitely many values of y that minimize $f(y)$.
4. True: If the y_i's were given to us in sorted order, then we could find a y^* in constant time. (I gave points for "False" because it should have explicitly stated "sorted in an array".)

Question Z. (9 points)
[*A question to test understanding of lower bounds and of inorder tree traversal.*]
Let S be a set of n distinct elements (S is not given sorted). A *binary search tree* T for set S is defined as follows: (i) The root of T contains the median (call it $\hat{x}$) of S, (ii) the left subtree of the root is a binary search tree for the elements of S that are smaller than $\hat{x}$, and (iii) the right subtree of the root is a binary search tree for the elements of S that are larger than $\hat{x}$. Mark by T (= True) or F (= False) each of the following statements:

1. False: Given S, T can be built in $O(n)$ time
2. True: Given T, a sorted version of S can be obtained in $O(n)$ time
3. True: The problem of consructing T from the set S has an $\Omega(n \log n)$ time lower bound.

Question V. (12 points)
[*Tests algorithmic thinking without asking for the design of a full-fledged algorithm in a short amount of time.*]
Let $S = (p_1, \ldots, p_n)$ be a set of n two-dimensional points, no two of which have same x coordinate or same y coordinate. Let x_i and y_i denote the x and (respectively) y coordinate of a point $p_i \in S$. Point p_i is said to be *maximal* if no other point p_j of S has both $x_j > x_i$ and $y_j > y_i$. The *maximal elements* problem is: Given such a set S of two-dimensional points, compute the set $Max(S)$ of maximal elements in S. The following are two different algorithms for computing the maximal elements for such a set S. Each description contains pairs of choices, each of which is surrounded by boldface curly brackets (as in **{**Choice1,Choice2**}**): For each such pair of the form **{**Choice1,Choice2**}** you must choose (by circling it) one of the alternatives (either Choice1 or Choice2) in such a way that in the end t he description sketches an $O(n \log n)$ time algorithm for solving the maximal elements problem.

Algorithm A

- First sort the points according to their x-coordinates, then go through the sorted points by **{**increasing,decreasing**}** x-coordinates and maintain as you go along the **{**smallest,largest**}** y-coordinate (call it y') encountered so far: The point at which you are is maximal if and only if its own y-coordinate is **{**smaller,larger**}** than y'.

Algorithm B

- Find the median of the x-coordinates of the points (call it x'). Let S' be the subset of points in S that have x-coordinates $\leq x'$. Let S'' be the points of S not in S'. Recursively compute $Max(S')$, then recursively compute $Max(S'')$. Let y' be the largest y-coordinate in **{**$Max(S'), Max(S'')$**}**: $Max(S)$ consists of all of **{**$Max(S'), Max(S'')$**}**, followed by those points of **{**$Max(S'), Max(S'')$**}** whose y-coordinate is larger than y'.

Answer: decreasing, largest, larger ; $Max(S'')$, $Max(S'')$, $Max(S')$.

Question W. (6 points)
[*A question based on a particular lecture.*]
In the $O(n \log n)$ time algorithm for finding a closest pair among a set S of $n = 2^q$ points in two dimensions, we first partitioned the set of points in two equal sets S_1 and S_2 according to their x coordinates. Then we recursively solved the problem for S_1, and also for S_2. Call δ_1 the closest distance for S_1, δ_2 the one for S_2, and let $\delta = \min\{\delta_1, \delta_2\}$.
Mark by T (= True) or F (= False) each of the following (where c denotes a constant).

1. True: There could be cn pairs of points in S that are (for each pair) closer to each other than δ.

2. True: Let p be any point in S. Then there are at most $O(1)$ points of S whose distance to p is $< \delta$.

3. False: Let p be any point in S_1. Then there are at most $O(1)$ points of S_1 whose distance to p is $< \delta$, but there could be cn points of S_2 whose distance to p is $< \delta$.