

CS 580 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 94 points only).

Question A. (12 points) State, using the “ $O(\cdot)$ ” notation, the solution to each of the three recurrences shown below. State the answers as a function of n , not of q . Just state the answers – you do not need to justify them.

1. $T(n) = T(n^{1/2}) + 1$
 $T(2) = 1$
assuming $n = 2^{(2^q)}$ for some integer q .
2. $T(n) = T(n/2) + \log n$
 $T(1) = 1$
assuming $n = 2^q$ for some integer q .

Question B. (8 points) Order the following functions by increasing growth rate (you may want to simplify some of these functions before attempting to take limits of ratios etc):

$$\sqrt{n}, 2^{\log \log n}, n^{1/\log n}, 2^{3 \log n}, 4^{\log n}, \log(n!)/\log n, (1.1)^n, n^{10}, (\log n)^2, 2^{3 \log n}, n^{10}, (1.1)^n.$$

Question C. (6 points) 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. There could be cn pairs of points in S that are (for each pair) closer to each other than δ .
2. Let p be any point in S . Then there are at most $O(1)$ points of S whose distance to p is $< \delta$.
3. 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 $< \delta$.

Question D. (6 points) We covered in class an $O(n)$ time algorithm for doing $Select(S, k)$, which returns in $O(n)$ time the k th smallest item in a set S of n items. The $O(n)$ time algorithm we described partitioned the input into $n/5$ pieces of size 5 each, ... etc. Write the recurrence relation that would result if we had used 3 instead of 5, that is, if we had started by partitioning the set into $n/3$ chunks of size 3 each, ... etc.

(Just write down the resulting recurrence – you do not have to justify it.)

Question E. (6 points) Suppose we have available a linear-time selection procedure $\text{BadSelect}(S, k)$ that is defective, in that it works properly only if $k = |S|/2$, but if $k \neq |S|/2$ then it can return the wrong answer or even crash; here $|S|$ denotes the number of elements in set S . We want to use this defective selection procedure to produce a correct selection procedure $\text{CorrectSelect}(S, k)$. Fill the blank spaces in the following description of $\text{CorrectSelect}(S, k)$ so it works in linear time even when k is not equal to $|S|/2$.

$\text{CorrectSelect}(S, k)$:

1. If $|S| \leq 20$ then solve the problem by sorting S and returning the k th entry in the sorted version of S . If $|S| > 20$ then proceed to Step 2.
2. Let $x = \text{BadSelect}(S, |S|/2)$, i.e., x is the median of S . If $k = |S|/2$ then return x as the answer, otherwise proceed to Step 3.
3. Create the sets A and B where A contains the elements of S that are $< x$, and B contains the elements of S that are $> x$.
4. If $k < |S|/2$ then return _____,
otherwise (i.e., if $k > |S|/2$) return _____.

Question F. (6 points) This question is about the definition of the Discrete Fourier Transform, and of its inverse. Let $B = (b_0, \dots, b_{n-1})$ be the Discrete Fourier Transform of $A = (a_0, \dots, a_{n-1})$. Let $w_n = e^{2\pi\sqrt{-1}/n}$.

1. Write, in the space below, b_i as a function of (a_0, \dots, a_{n-1}) and of the powers of w_n .
2. Write, in the space below, a_i as a function of (b_0, \dots, b_{n-1}) and of the powers of w_n .

Question G. (8 points) Let A and B be two problems. Suppose that we were able to establish the following fact: “Any algorithm that solves A in time $O(T(n))$ can be used to solve B in time $O(n^2 + T(n))$.” Mark by T (= True) or F (= False) each of the following statements:

1. _____ If A has an $\Omega(n^2)$ time lower bound then B does too
2. _____ If B has an $\Omega(n^2)$ time lower bound then A does too
3. _____ If A has an $\Omega(n^3)$ time lower bound then B does too
4. _____ If B has an $\Omega(n^3)$ time lower bound then A does too

Question H. (10 points) For any sequence of numbers X , we use $|X|$ to denote the length of X , $\text{Sorted}(X)$ to denote a version of X sorted by *increasing* order, $\text{Reverse}(X)$ to denote the reverse of X (obtained by reading X backwards), $\text{LIS}(X)$ to denote a longest *increasing* subsequence of X , and $\text{LDS}(X)$ to denote a longest *decreasing* subsequence of X . For any pair of sequences X and Y , we use $\text{LCS}(X, Y)$ to denote a longest subsequence that is common to X and Y . Mark by T (= True) or F (= False) each of the following statements:

1. ----- $|LIS(X)| = |LDS(Reverse(X))|$
2. ----- $|LDS(X)| = |LCS(Reverse(X), Sorted(Reverse(X)))|$
3. ----- $|LDS(X)| = |LCS(Reverse(X), Reverse(Sorted(X)))|$
4. ----- $\max\{|LIS(X)|, |LDS(X)|\} \geq \sqrt{|X|}$
5. ----- $\min\{|LIS(X)|, |LDS(X)|\} \leq \sqrt{|X|}$

Question I. (6 points) In the pattern matching problem, if we let $P = p_1 \dots p_m$ be the pattern and $T = t_1 \dots t_n$ be the text, then the first occurrence of P in T is said to *start at position i* if i is the first (i.e., smallest) index such that $t_i = p_1, t_{i+1} = p_2, \dots, t_{i+m-1} = p_m$ (the second occurrence of P in T starts at the second smallest index i such that $t_i = p_1, t_{i+1} = p_2, \dots, t_{i+m-1} = p_m$, the third occurrence starts at the third such index, etc – recall that these occurrences may overlap, and that there may be none of them if P does not occur at all in T).

This question is about the special case when m is a power of two and $T = PP$, i.e., T is the concatenation of P with P . Note that this implies that $n = 2m$, that the first occurrence of P in T starts at position 1 in T , and that there is another occurrence of P starting at position $m + 1$ in T . Mark by T (= True) or F (= False) each of the following statements about this special case:

1. ----- Suppose the second occurrence of P in T starts at a position j in T such that $j < m/2$ and $j - 1$ is a power of 2. Then there is a description of P that uses $O(j)$ space rather than $O(m)$ space.
2. ----- It is possible for the second occurrence of P in T to start at a position j in T such that $m/2 < j < m$ and $m - j + 1$ is a power of 2.

Question J. (10 points) This question is about how long a string can be, as a function of alphabet size, when it does not contain certain “forbidden” substrings and subsequences. Let X be a string, whose length we denote by $|X|$, over an alphabet of size 100. In the rest of this question we assume that X does not contain any *substring* of the form yy where y is an alphabet symbol, that is, there are no two adjacent occurrences of the same symbol (note that this does not rule out non-adjacent multiple occurrences of a symbol, that is, $yy \dots y$ can occur as a *subsequence* of X because subsequence symbols need not be adjacent).

1. Suppose that, in addition to not containing any adjacent repetitions, X does not contain any *subsequence* of the form zyz where y and z are distinct alphabet symbols. Then the length of X cannot exceed (choose the best alternative from the below):
 - (a) 99
 - (b) 100
 - (c) 101
 - (d) 199
 - (e) 201

- (f) 4950
- (g) 5050
- (h) 10000

2. Assume X does not contain any *subsequence* of the form $yzyz$ where $y \neq z$. Then the length of X cannot exceed (choose the best alternative from the below):

- (a) 99
- (b) 100
- (c) 101
- (d) 199
- (e) 201
- (f) 4950
- (g) 5050
- (h) 10000

Question K. Let $A = a_1 a_2 \dots a_n$ be a sequence of numbers (some of which are negative), where n is a power of 2.

For $1 \leq i \leq j \leq n$ where $j - i + 1$ is a power of 2, we define the recursive algorithm $SolveIt(i, j)$ that returns the four quantities:

- $S_{i,j} = a_i + a_{i+1} + \dots + a_j$
- $\Delta_{i,j} = \max_{i \leq i' \leq j' \leq j} S_{i',j'}$
- $l_{i,j} = \max_{i \leq k \leq j} S_{k,j}$
- $r_{i,j} = \max_{i \leq k \leq j} S_{i,k}$

1. (9 points) Fill the blanks in the below description of the various steps of $SolveIt(i, j)$ (note that the algorithm maintains the invariant that $j - i + 1$ is a power of 2).

Algorithm $SolveIt(i, j)$

- (a) If $j = i$ then return $S_{i,j} = a_i$, $\Delta_{i,i} = a_i$, $l_{i,j} = a_i$, $r_{i,j} = a_i$. Otherwise continue with the next step.
- (b) Recursively solve the problem for the left half of the index range $[i, j]$. In other words, letting $\tau = (i + j - 1)/2$, we recursively call $SolveIt(i, \tau)$. The call returns $S_{i,\tau}$, $\Delta_{i,\tau}$, $l_{i,\tau}$, and $r_{i,\tau}$.
- (c) Recursively solve the problem for the right half of the index range $[i, j]$. In other words, recursively call $SolveIt(\tau + 1, j)$. The call returns $S_{\tau+1,j}$, $\Delta_{\tau+1,j}$, $l_{\tau+1,j}$, and $r_{\tau+1,j}$.

- (d) Using the 8 quantities returned by the above two recursive calls, compute in $O(1)$ time the desired quantities

$$S_{i,j} = S_{i,\tau} + S_{\tau+1,j}$$

$$\Delta_{i,j} = \text{-----}$$

$$l_{i,j} = \text{-----}$$

$$r_{i,j} = \text{-----}$$

and return them.

2. (3 points) Complete writing, in the space below, the recurrence relation for *SolveIt*(1, n):

$$T(1) = c_1 \text{ where } c_1 \text{ is a constant}$$

If $n > 1$ then

$$T(n) = \text{-----}$$

3. (4 points) State in the space below the solution to the above recurrence (using the $O(\cdot)$ notation). No detailed derivation is necessary, just writing down the answer is enough.
