Problem 4 is worth 40% of this problem set, and problems 1-3 constitute the remaining 60%.

For each problem where you are asked to give an algorithm, more points are given for asymptotically faster algorithms. In judging the number of points to award a correct solution, we only consider the running time in asymptotic notation, i.e. a writeup of an algorithm taking $1000n^2$ steps versus one taking $.01n^2$ steps would receive the same number of points — both would be simply treated as $\Theta(n^2)$-time solutions.

Problem 1

Given $n$ points in the plane and a parameter $0 < p < 1$, we call a line “$p$-heavy” if it contains at least $pn$ of the input points.

(a) (3 points) Show if $n \geq 2/p^2$, then the number of distinct $p$-heavy lines is at most $2/p$.

(b) (7 points) Based on (a), devise a divide-and-conquer algorithm to report all $p$-heavy lines. Full credit is given for solutions with running time $O(C_p n \log n)$ for some constant $C_p$ that only depends on $p$ (optimizing $C_p$ is not necessary to receive full credit). **Hint:** consider treating inputs with fewer than $n_p$ points as a base case, where $n_p$ depends only on $p$.

Problem 2

We construct an infinite sequence of arrays $A_1, A_2, A_3, \ldots$ in the following recursive fashion. First, we specify that $A_1 = [1]$. For $k > 1$, we recursively define $A_k$ to be two copies of $A_{k-1}$ put together, with the number $k$ inserted between the two lists.

To illustrate the above algorithm:

$$A_1 = [1]$$
$$A_2 = [1, 2, 1]$$
$$A_3 = [1, 2, 1, 3, 1, 2, 1]$$
$$A_4 = [1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1]$$
and so on.

Devise an efficient algorithm that accepts some \( k \) and two intervals \([a, b]\) and \([c, d]\), and determines the length of the longest subarray contained in both the arrays \( A_k[a:b] \) and \( A_k[c:d] \). An array \( C \) is a subarray of an array \( B \) if there exist \( i \) and \( j \) such that \( C = B[i:j] \). As always, you must prove the algorithm’s correctness and analyze its runtime.

**Problem 3**

Recall the following text search problem from class. There is an alphabet \( \Sigma \) and two strings \( P \in \Sigma^m, T \in \Sigma^n, n \geq m \) (i.e. strings of length \( m, n \), respectively, made up of characters in \( \Sigma \)). We would like to output a list of all indices \( i \) such that \( T[i, i+1, \ldots, i+m-1] = P \), i.e. \( t_{i+j-1} = p_j \) for \( j = 1, \ldots, m \). In class, when \( \Sigma = \{0, 1\} \) we showed how to use the FFT to solve this problem in \( O(n \log n) \) time (assuming a computer supporting infinite precision complex arithmetic). Making the same precision assumptions:

(a) (2 points) Improve this time to \( O(n \log m) \) for \( \Sigma = \{0, 1\} \).

(b) (2 points) Deal with the case when \( P \) (but not \( T \)) has “don’t care” symbols \(*\). A * symbol can match either a 0 or a 1. Again you should assume \( \Sigma = \{0, 1\} \).

(c) (2 points) Show how to deal with the case \( |\Sigma| > 2 \), without don’t care symbols. You can assume \( \Sigma = \{1, 2, \ldots, |\Sigma|\} \). For this part, full points will be given to solutions with runtime \( O(|\Sigma|n \log(|\Sigma|m)) \).

(d) (2 points) Give a solution for \( |\Sigma| > 2 \) with don’t care symbols (in both \( P \) and \( T \)), which can match any character in \( \Sigma = \{1, 2, \ldots, |\Sigma|\} \). Again, full points will be given to solutions with runtime \( O(|\Sigma|n \log(|\Sigma|m)) \).

(e) (2 points) Devise a new algorithm improving the runtime from (d) to \( O(n \log m) \), i.e. with no dependence on \( |\Sigma| \). Note that an \( O(n \log m) \) solution here naturally implies you don’t have to separately do parts (a) through (d), since the problem solved here is most general. Hint: Characters \( c, c' \) are unequal iff \((c-c')^2 > 0\).

**Programming Problem**

Solve DIVIDENDS on the programming server (https://cs124.seas.harvard.edu).

**Extra Credit Programming Problem**

Solve DIVIDENDSCHARD on the programming server (https://cs124.seas.harvard.edu). No collaboration of any form is allowed on this extra credit problem; you also will not receive any help from TFs (though you may ask TFs for clarifications on the problem statement).