Instructions.

1. You could submit a pdf of the homework, either printed or hand-written and scanned, as long as it is easily readable.

2. If your solution is not clearly written, it might not be graded.

3. Prove the correctness of your answer. A correct answer without a proof might not be accepted.

4. If you have discussed the solution with other students, mention their names clearly on the homework. These discussions are not forbidden and are actually encouraged. However, you must write your whole solution yourself.

5. All questions have same weight.

6. You could refer to a data structure studied in class, and just mention briefly their guaranties. For example “It is known that a Red-Black tree could support the insert/delete/find operations on a set of $n$ elements in time $O(\log n)$.”

7. If your answer uses one of the data structures or algorithms that were studied in class, you could refer to it without having to repeat details studied in class. If your answer requires only modifications of one of the algorithm, it is enough to mention the required modifications, and what’s the effect (if any) on the running time and on other operations that the algorithm performs.

8. In general, a complete answer should contain the following parts:

   (a) High level description of the data structures (if needed). E.g. We use a binary balanced search tree. Each node contains, a key and pointers to its children. We augment the tree so each node also contains a field...

   (b) High level description of the algorithms

   (c) Proof of correctness (why your algorithm provides what is required).

   (d) A claim about the running time, and a proof showing this claim.
1. (a) Which algorithm would you use to solve the diet problem, if only two types of food (bananas and oranges) are involved, and you are seeking an efficient algorithm. What is the running time, as the function of the number of vitamins? Express both the worst case running time and the expected running time.

(b) Assume now that you need to answer the diet problem, but now there are \( m \geq 10 \) types of food (oranges, bananas, mangos etc etc). Which algorithm would you use? Why? As you know, linear programming could solve both versions. So please indicate which LP algorithm exactly would you use.

2. (a) You are given a set \( S = \{p_1, \ldots, p_n\} \) of \( n \) points in the plane. Each point \( p_i \) is given by its coordinates \( p_i = (x_i, y_i) \). Suggest an algorithm that in expected time \( O(n) \), finds a line \( \ell \) whose vertical distance from each point is \( \leq 1 \), and each \( p_i \) is on, or above \( \ell \) (but not below \( \ell \)). If no such line exists, the algorithm should report so. Refer to the example in Figure 4.

Hint: assume that the line could be represented in the form \( y = ax + b \), where the parameters you need to compute are \( a \) and \( b \). Then the vertical distance from a point \( (x_i, y_i) \) is \( |y_i - (ax_i + b)| \). For example, if \( (x_i, y_i) \) is exactly on \( \ell \), then \( y_i = (ax_i + b) \).

Note that this definition is different than the distance to the projected line, which we discussed earlier.

(b) Repeat the question, but now \( \ell \) does not have to lay below all the points. See the example in Figure 2.

3. The pseudo-code of the 1DLP, as appears in the slides, does not handle case that \( \ell_i \) does not intersect one or more of the lines in \( \{\ell_1, \ldots, \ell_{i-1}\} \). Suggest a modification of the code that corrects this issue. (Note that there are a few cases to consider).

4. You are given \( n \) squares \( s_1, \ldots, s_n \) in the plane. Each square is given by the coordinates of its corners. Suggest an expected \( O(n) \) time algorithm that determines whether all squares have a point in common. That is, is it true that \( s_1 \cap s_2 \cap \cdots \cap s_n \neq \emptyset \)

5. As in the previous question, but now assume that all squares are axis-parallel, meaning that each edge is parallel to either the \( x \)-axis or the \( y \)-axis. (The square \( s_1 \) in Figure 3 is an example of such a square.) Suggest an algorithm that runs in \( O(n) \) time in the worst case (that is, not expected case). The algorithm again determines whether all squares have a point in common.
Figure 2: Same as previous figure, but now \( p_2 \) and \( p_4 \) are below \( \ell \).

Figure 3: The set of 4 squares. For clarity of visualization, only the boundaries of the squares are shown. Also shown (in orange) the region \( s_1 \cap s_2 \cap s_3 \cap s_4 \).

6. (Slightly tricky.) Suggest an expected \( O(n) \) time algorithm to find if the polygon \( C_n \) contains more than 3 edges on its boundary. The notation is as in the slides.

You could assume that when 2DLP returns the optimal value \( v_n \), it also specifies the two lines \( \ell_i, \ell_j \) that \( v_n \) is their intersection point. That is, \( v_n = \ell_i \cap \ell_j \).

Figure 4: A set of 6 halfplanes, whose intersections \( C_6 = h_1 \cap \cdots \cap h_6 \) (denoted in orange) has 4 edges on its boundary.