1. Problem 4-5 (VLSI chip testing) - This is a really good divide and conquer problem that I left out of the last hw

2. Show via induction that a full parenthesization of an $n$ element expression has exactly $n - 1$ pairs of parenthesis.

3. Find the optimal parenthesization for a matrix-chain product whose sequence of dimensions is: $(3, 2, 4, 1, 2)$. (Don't forget to include the table used to compute your result)

4. A bakery sells donuts in boxes of three different quantities, $x_1$, $x_2$, and $x_3$. In the Donut Buying problem, you are given the numbers $x_1$, $x_2$ and $x_3$, and an integer $n$ and you should return either 1) the minimum number of boxes needed to obtain exactly $n$ donuts if this is possible, along with a set of boxes that obtains this minimum; or 2) “DOH!” if it is not possible to obtain exactly $n$ donuts.

For example if $x_1 = 4$, $x_2 = 6$, $x_3 = 9$ and $n = 17$, then you should return that 3 boxes suffices, with 2 boxes of size 4, and 1 box of size 9.

However, if $n = 11$, you should return “DOH!” since it is not possible to buy exactly 11 donuts with these box sizes.

(a) For any positive $x$, let $m(x)$ be the minimum number of boxes needed to buy $x$ donuts if this is possible, or INFINITY otherwise. Write a recurrence relation for the value of $m(x)$. Don't forget the base case(s)!

(b) Give an efficient algorithm for solving Donut Buying. How does its running time depend on $x_1$, $x_2$, $x_3$, and $n$? Is it an algorithm that runs in polynomial time in the input sizes?

5. Gus wants to open franchises of his restaurant, *Los Pollos Hermanos*, along Central Avenue. There are $n$ possible locations for franchises,
where location \( i \) is at mile \( i \) on Central. Each location \( i > 1 \), is thus a distance of 1 mile from the previous one. There are two rules.

- At each location, there can be at most one restaurant, and the profit of a restaurant at location \( i \) is \( p_i \).
- Any two restaurants must be at least 2 miles apart.

(a) Jesse proposes the following algorithm: Sort the locations by decreasing \( p_i \) values, then greedily choose the next possible location, provided that it doesn’t conflict with previously chosen locations. Show that Jesse’s algorithm doesn’t always give maximum profit.

(b) Now consider a dynamic programming approach to this problem. For \( i \geq 0 \), let \( m(i) \) be the maximum profit obtainable by using locations 1 through \( i \). Write a recurrence relation for \( m(i) \). Don’t forget the base case(s).

(c) Describe how you would create a dynamic program using the previous recurrence. What is the run time of your algorithm?

(d) Now Gus wants to solve a generalization of the problem. There are two changes. First, for \( 1 < i \leq n \), location \( i \) is now distance \( d_i \) from location \( i - 1 \). Second, any two restaurants must now be distance \( k \) apart for some parameter \( k \). Write a new recurrence relation for this problem. Don’t forget the base case(s).

6. A h-tree is a rooted binary tree defined as follows. For \( \ell \) a power of 2, a tree with \( \ell \) leaf nodes is a h-tree iff the tree is perfect (recall that a perfect binary tree is one where all non-leaf nodes have two children, and all leaf nodes have the same depth). For \( \ell \) not a power of 2, a tree with \( \ell \) leaf nodes is a h-tree if and only if (1) the root node, \( r \), has two children; (2) the left subtree of \( r \) is the root of a perfect binary tree with \( 2^{\lfloor \log \ell \rfloor} \) leaf nodes; and (3) the right subtree of \( r \) is a h-tree.

Show the following by induction:

- For all positive \( \ell \), there is a unique h-tree with \( \ell \) leaf nodes.
- Call the h-tree with \( \ell \) leaf nodes h-tree(\( \ell \)). Then, the height of h-tree(\( \ell \)) is \( \lceil \log \ell \rceil \)

7. We can use h-trees to design “self-healing” overlay networks, since h-trees can be merged quickly. In the self-healing application of h-trees, the leaf nodes are associated with actual machines in a network,
and the internal nodes represent additional “router nodes” (a scarce resource). To merge a list of h-trees, \( h_1, h_2, \ldots, h_x \) we want to create a single new h-tree, \( h \), which contains as leaf nodes all the leaf nodes in \( h_1, h_2, \ldots, h_x \), and adds the smallest number of new internal nodes as possible.

- Show how you can quickly merge a collection of \( x \) h-trees, each of size no more than \( n \), into a single big h-tree by adding no more than \( O(x \log n) \) additional internal nodes. What is the runtime of your algorithm?

Hint: Think about how to set up a correspondence between binary numbers and h-trees, and binary addition and h-tree merging.