Assignment 2

Satyanand
14EC10049

Problem 1

This problem is popularly known as the defective chessboard problem.A defective chessboard is a $2^n * 2^n$ board of squares with exactly one defective square.

- When n=0, there is only one possible defective chessboard.
- When n = 1, there are 4 possible defective chessboard.

Now, a defective chessboard with n = 0 can be easily covered as it has no non-defective squares. Here, the number of tiles are zero.

For n = 1, there are exactly three non-defective squares(unshaded in the above picture) and they can easily be covered with the L- shaped tile in one of the orientations.

Algorithm

- We can use divide and conquer methodology to solve this problem. The method suggests reducing the $2^n * 2^n$ board into smaller defective chessboards and then tiling them. A natural way of thinking is to divide the $2^n * 2^n$ board into four $2^(n-1) * 2^(n-1)$ boards.
- Note that only one of the four sub-squares are defective. So, we place the L - shaped tile in such a way that all the remaining three sub-squares are also defective.

- This partitioning technique converts the original problem into 4 sub- problems which can be solved recursively. The recursion terminates when the board has been reduced to 1X1(which has no non-defective square, hence no tiles need to be placed).

Problem 2

The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. We can calculate the smallest distance in O($nLogn$) time using Divide and Conquer strategy. A O($nx(Logn)^2$) approach is implemented using divide and conquer method.

Algorithm

Following are the detailed steps of a O($n(Logn)^2$) algortihm. Input: An array of n points P[] Output: The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to x coordinates.

1. Find the middle point in the sorted array, we can take P[n/2] as middle point.
2. Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].
3. Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of *dl* and *dr*. Let the minimum be *d*.
4. From above 3 steps, we have an upper bound *d* of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip[] of all such points.

5. Sort the array *strip*[] according to y coordinates. This step is O($nLogn$). It can be optimized to O(n) by recursively sorting and merging.

6. Find the smallest distance in *strip*[]. This is tricky. From first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in *strip*, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate).

7. Finally return the minimum of d and distance calculated in above step (step 6)