Report for Assignment 2

Deep Kiran Shroti
Roll Number- 12EC35013

March 9, 2016

1 Divide and Conquer

1.1 Theory

Though the divide-and-conquer algorithm is a powerful technology, but it is difficult to be used in practice
The problems which can be solved by the divide-and-conquer algorithm generally have following characters.

1. When the scale of the problem is reduced to certain extent, it can be easily solved.
2. The problem can be divided into many same problems with small scale, i.e. the problem has the character of the optimal sub-structure.
3. The solutions of the sub-problems of the problem can be combined as the solution of the problem.
4. Various sub-problems divided by this problem are independent each other, i.e. the public problem is not contained among sub-problems.

2 Tiling A Defective Chessboard

Assignment Statement Given a nxn board where n is a power of 2 with minimum value as 2, with one missing cell (of size 1x1) of a known location, fill the board using L shaped titles. An L shaped title is a 2x2 square with one cell (of size 1x1) missing

2.1 Implementation Details

1. **First Step:** Let n be the size of a square, and let ***coordinateType*** *filled* be the location of the missing cell
   Base case: n = 2, A 2 x 2 square with one cell missing is nothing but a tile and can be filled with a single tile.

2. **Second Step:** Place a L shaped tile at the center such that it does not cover the n/2 * n/2 subsquare that has a missing square. Now all four subsquares of size n/2 x n/2 have a missing cell (a cell that doesn't need to be filled).

3. **Third Step:** Solve the problem recursively for following four. Let filled1, filled2, filled3 and filled4 be positions of the 4 missing cells in 4 squares.
   - *T*ile(n/2, filled1)
   - *T*ile(n/2, filled2)
   - *T*ile(n/2, filled3)
   - *T*ile(n/2, filled4)

2.2 Conclusion

In the algorithm of the chessboard coverage problem, suppose that T(k) is the time the algorithm takes to cover chessboard of $2^k x 2^k$. so from the divide-and-conquer strategy equation of the algorithm, T(k) satisfies following recursion equation.

$$T(k) = \begin{cases} O(1) & \text{k = 0} \\ 4T(k-1) + O(1) & \text{k} > 0 \end{cases}$$

Solving this recursion equation, we obtain T(k) = $O(4^k)$. Because the amount of L-type domino to cover one chess board of $2^k x 2^k$ is $(4^k-1)/3$, the algorithm of Chessboard is the optimal on the asymptotic meaning.

3 Closest Pair of Points

Assignment Statement Given an array of n points in the plane, and the problem is to find out the closest pair of points in the array.

3.1 Implementation Details

Following are the detailed steps of a $O(n(\log n)^2)$ algorithm.

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

1. **First Step:** ind the middle point in the sorted array, we can take P[n/2] as middle point.

2. **Second Step:** Divide the given array in two halves. The first sub-array contains points from P[0] to P[n/2]. The second sub-array contains points from P[n/2+1] to P[n-1].

3. **Third Step:** Recursively find the smallest distances in both sub-arrays.

4. **Fourth Step:** 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. **Fifth Step:** 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. **Sixth Step:** 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). See this for more analysis.

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

3.2 Conclusion

Let Time complexity of above algorithm be T(n). Let us assume that we use a O(n*log*n) sorting algorithm. The above algorithm divides all points in two sets and recursively calls for two sets.
After dividing, it finds the strip in O(n) time, sorts the strip in O(n*log*n) time and finally finds the closest points in strip in O(n) time. So T(n) can expressed as follows.

$$T(n) = 2T(n/2) + O(n) + O(n\,log\,n) + O(n)$$
$$T(n) = 2T(n/2) + O(n\,log\,n)$$
$$T(n) = T(n \text{ x } log \text{ x } log\,n)$$

Quick sort which can be O(n^2) in worst case. To have the upper bound as O(n $(log\,n)^2$), a O(n*log*n) sorting algorithm like merge sort or heap sort can be used.