CS 577 # Assignment 6 – Divide and Conquer 2 Spring 2023

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Carson Miller Wisc id: 9081473028

Divide and Conquer

1. *Kleinberg, Jon. Algorithm Design (p. 248, q. 5)* Hidden surface removal is a problem in computer graphics where you identify objects that are completely hidden behind other objects, so that your renderer can skip over them. This is a common graphical optimization.

   In a clean geometric version of the problem, you are given n non-vertical, infinitely long lines in a plane labeled $L_1 \dots L_n$. You may assume that no three lines ever meet at the same point. (See the figure for an example.) We call L_i "uppermost" at a given x coordinate x_0 if its y coordinate at x_0 is greater than that of all other lines. We call L_i "visible" if it is uppermost for at least one x coordinate.

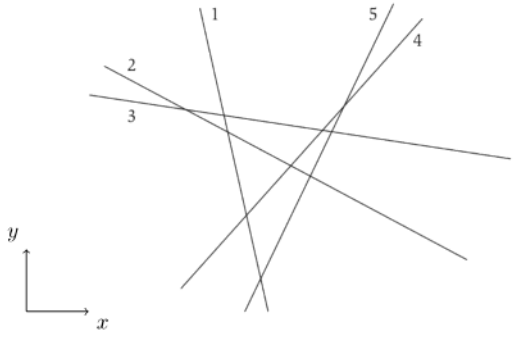


Figure 5.10 An instance of hidden surface removal with five lines (labeled *1–5* in the figure). All the lines except for *2* are visible.

(a) Give an algorithm that takes n lines as input and in $O(n \log n)$ time returns all the ones that are visible.

Solution:

Given an array X of x coordinates 1 to n,
    If(length(X) == 1)
        Add line L with greatest y at X[1] to set V, if not already included
    At X[n/2], find the line L that has the greatest y and add it to set V of all visible lines
    recursiveCall(1:n/2)
    recursiveCall(n/2:n)

(b) Write the recurrence relation for your algorithm.

Solution:

T(n) = 2T(n/2) + 2n

The work done at each step is 2n because we are checking the length (O(n) in worst case) and searching through every n line to find the greatest y value at a given x value. There are 2 recursive calls made, each on 1/2 of the passed array.

(c) Prove the correctness of your algorithm.

Solution:

Base Case:
length of X, array of x coordinates, is 1. We find the line with greatest y at this x coordinate and add it to set V

Inductive Step:
Assume that the recursive calls correctly adds the line with greatest y at a X[n/2] for its given array X. My algorithm will find the line with greatest y at every given x coordinate in the array X of all x coordinates by checking the midpoint of every passed array.

2. In class, we considered a divide and conquer algorithm for finding the closest pair of points in a plane. Recall that this algorithm runs in $O(n \log n)$ time. Let's consider two variations on this problem:

   (a) First consider the problem of searching for the closest pair of points in 3-dimensional space. Show how you could extend the single plane closest pairs algorithm to find closest pairs in 3D space. Your solution should still achieve $O(n \log n)$ run time.

   Solution:

   Similar to the extension from 1D to 2D, we can add a third sorting P_z where the points are sorted by z-coordinate. After splitting in half, we would still have Q_x, Q_y, Q_z, R_x, R_y, and R_z without resorting. For the combination of the solutions, we must still consider some pairs that cross the dividing line, but this will remain the same as the 2D case

   (b) Now consider the problem of searching for the closest pair of points on the surface of a sphere (distances measured by the shortest path across the surface). Explain how your algorithm from part a can be used to find the closest pair of points on the sphere as well.

   Solution:

   The algorithm can mostly remain the same, but the distance formula would need to be modified to calculate the curved distance, due to the shortest path needing to be measure across the surface.

   (c) Finally, consider the problem of searching for the closest pair of points on the surface of a torus (the shape of a donut). A torus can be thought of taking a plane and "wrap" at the edges, so a point with y coordinate MAX is the same as the point with the same x coordinate and y coordinate MIN. Similarly, the left and right edges of the plane wrap around. Show how you could extend the single plane closest pairs algorithm to find closest pairs in this space.

   Solution:

   Once again, the algorithm can mostly remain the same besides the distance formula calculations. If all points lay on this torus plane, we can use the same algorithm while taking distance measurements between two points using some complex geometrical formula that can calculate the distance along a torus surface.

3. *Erickson, Jeff. Algorithms (p. 58, q. 25 d and e)* Prove that the following algorithm computes $\gcd(x, y)$ the greatest common divisor of x and y, and show its worst-case running time.

```
BINARYGCD(x,y):
if x = y:
    return x
else if x and y are both even:
    return 2*BINARYGCD(x/2,y/2)
else if x is even:
    return BINARYGCD(x/2,y)
else if y is even:
    return BINARYGCD(x,y/2)
else if x > y:
    return BINARYGCD( (x-y)/2,y )
else
    return BINARYGCD( x, (y-x)/2 )
```

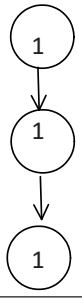
Solution:
Assume the recursive call returns the greatest common divisor of its two parameters. This algorithm halves even inputs and replaces one input with average of both if both inputs are odd. All inputs x and y will be positive integers because half of an even number is an odd number and the average of two odd numbers is even. Eventually after halving and averaging long enough, x and y will be equal.

The worst-case occurs when x != y and x and y are prime numbers (their only divisors are themselves and 1). This algorithm will halve or average one (or both) inputs until they are equal. Therefore, the runtime depends on the larger input as it must be halved/averaged until it becomes 1. This gives us O(log(max(x,y))) runtime.

4. Here we explore the structure of some different recursion trees than the previous homework.
   (a) Asymptotically solve the following recurrence for $A(n)$ for $n \geq 1$.

$$A(n) = A(n/6) + 1 \qquad \text{with base case} \qquad A(1) = 1$$

Solution:

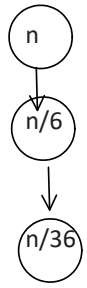


Height of tree * work done in each node
$\log_6 n * 1 = O(\log_6 n)$

(b) Asymptotically solve the following recurrence for $B(n)$ for $n \geq 1$.

$$B(n) = B(n/6) + n \qquad \text{with base case} \qquad B(1) = 1$$

Solution:



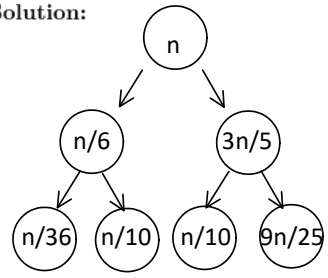
Sum up work done in all nodes

$$n * \sum_{i=0} \left(\frac{1}{6}\right)^i = 1.2n = O(n)$$

(c) Asymptotically solve the following recurrence for $C(n)$ for $n \geq 0$.

$$C(n) = C(n/6) + C(3n/5) + n \qquad \text{with base case} \qquad C(0) = 0$$

Solution:



n

23n/30

529n/900

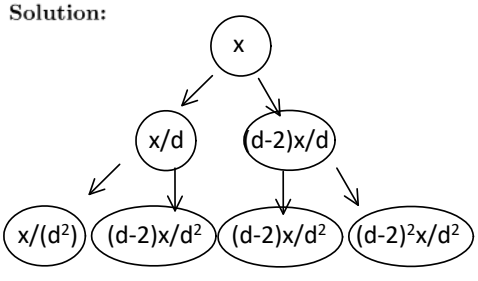
Sum up work done in all nodes

$$n * \sum_{i=0} \left(\frac{23}{30}\right)^i = \left(\frac{30}{7}\right) * n = O(n)$$

(d) Let $d > 3$ be some arbitrary constant. Then solve the following recurrence for $D(x)$ where $x \geq 0$.

$$D(x) = D\left(\tfrac{x}{d}\right) + D\left(\tfrac{(d-2)x}{d}\right) + x \qquad \text{with base case} \qquad D(0) = 0$$

Solution:



X

(d-1)x/d

(d²-2d+1)x/d²

Sum up work done in all nodes

$$x * \sum_{i=0} \left(\frac{d-1}{d}\right)^i = x * d$$

$$= O(x)$$

5. Implement a solution in either C, C++, C#, Java, Python, or Rust to the following problem.

   Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the line $y = 1$. Create a set of n line segments by connecting each point p_i to the corresponding point q_i. Your goal is to develop an algorithm to determine how many pairs of these line segments intersect. Your algorithm should take the $2n$ points as input, and return the number of intersections. Using divide-and-conquer, you should be able to develop an algorithm that runs in $O(n \log n)$ time.
   Hint: What does this problem have in common with the problem of counting inversions in a list?

   Input should be read in from stdin. The first line will be the number of instances. For each instance, the first line will contain the number of pairs of points (n). The next n lines each contain the location x of a point q_i on the top line. Followed by the final n lines of the instance each containing the location x of the corresponding point p_i on the bottom line. For the example shown in Fig 1, the input is properly formatted in the first test case below.

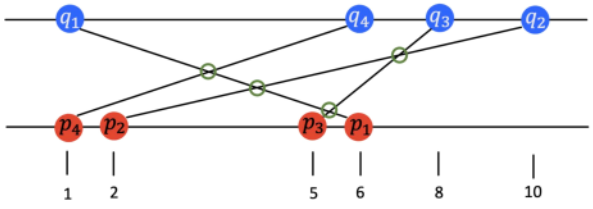


Figure 1: An example for the line intersection problem where the answer is 4

Constraints:

- $1 \leq n \leq 10^6$
- For each point, its location x is a positive integer such that $1 \leq x \leq 10^6$
- No two points are placed at the same location on the top line, and no two points are placed at the same location on the bottom line.
- Note that in C\C++, the results of some of the test cases may not fit in a 32-bit integer. If you are using C\C++, make sure you use a 'long long' to store your final answer.

Sample Test Cases:

input:
2
4
1
10
8
6
6
2
5
1
5
9
21
1
5
18
2
4
6
10
1

expected output:
4
7