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: Zinnia Nie Wisc id: 908 319 4044

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.

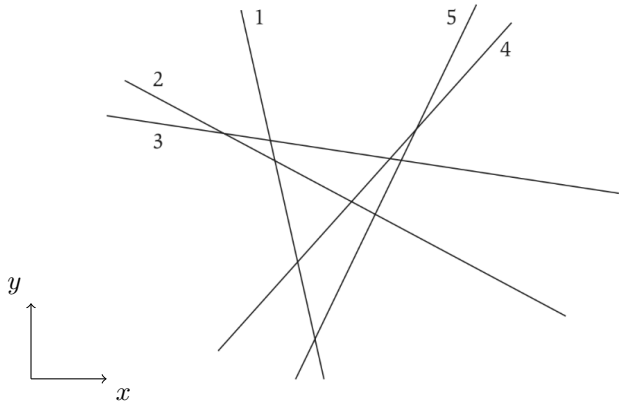
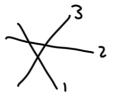


Figure 5.10 An instance of hidden surface removal with five lines (labeled *1–5* in the figure). All the lines except for *2* are visible.

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

recursing on halves
list of lines
N work per call

Solution: preprocessing: sort lines by increasing slope




```
UPPERMOST (L[1..n] ← list of lines):
if |A| ≤ 3: first and third lines are visible (min)
            second is visible if intersect with first is higher than intersect of 1st and 3rd
            return A with second line or not
upper = UPPERMOST(upper half by slope)
lower = UPPERMOST(lower half by slope)
Merge(upper, lower)

MERGE(A[1...n], B[1...m]):
K = merge sorted lists
Visible = K[1,2]
For i=3 to n+m:
   if K[i] is visible, add to Visible
   remove previous lines K[i] covers by comparing highest intercepts
return Visible
```

(b) Write the recurrence relation for your algorithm.

Solution:

2 recursive calls

$\frac{n}{2}$ size of input each recursive call

$O(n)$ = runtime of each call
↳ MERGE is $O(n)$ because it only adds/removes each line once

$T(n) = 2T(n/2) + O(n)$ ← mergesort

$= O(n \log n)$

(c) Prove the correctness of your algorithm.

Solution: Base case: With 3 lines, the largest and smallest slopes will always be visible lines if they are not equal because they will intercept each other. The third line will be visible if it intersects the min slope line to the left of the max slope line. This means it will be higher than both lines until it intercepts the max slope line.

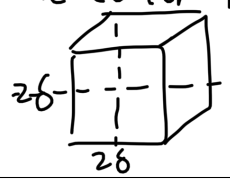
MERGE: once the two halves are merged, they are in order from smallest to largest slope.
Then using a similar method to the base case, we can check if each increasing slope line is visible and what lines it covers.
If the line intersects a lower slope line highest, then that means all lines in between the two lines are covered by the higher slope and thus should be removed.
The correct lines will be removed in each recursive call and merge.

Termination: Each recursive call reduces the size by 2, so eventually we will reach the base case.

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: Still divide the population of points in half and find closest points in each half.
Then find a 3D area that is $+\delta$ and $-\delta$ away from the center line. Then each 3D cube with sides $\frac{\delta}{2} \times \frac{\delta}{2} \times \frac{\delta}{2}$ only contains 1 point. So the algorithm needs to check the 63 points in the cube around each point in the slab.
Finally, choose the min distance between the 3.



   (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:
We can split a circle in half along a circumference, and have an area $+\delta$ and $-\delta$ from the circumference. Then, there is a square $\frac{\delta}{2} \times \frac{\delta}{2}$ that only has one point. We can check the distance between those points around the circumference in $O(n)$ time and pick min distance between, left, right, and center closest pairs.

   (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:
Instead of just applying the midpoint crossing algorithm to the center line area, also do it to the endpoint on one side. That way, we find the shortest distance crossing the midpoint, and the endpoint wrap around.

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: Base case: $x = y$, if the two inputs are equal, then the GCD will be the value itself

Recursive: both even means 2 is a factor of both, so dividing both by 2 doesn't change the GCD calculation, and the GCD will be multiplied by 2

One being even means 2 is a factor and since 2 is the smallest factor besides 1, if GCD is larger than that, dividing by 2 doesn't affect the calculation

Euler's states $\gcd(x,y) = \gcd(x-y,y)$ for $x > y$, since we know x and y are odd, x-y will be even and we can similarly divide by 2. Similar logic for when $y > x$ in the last case.

Eventually, by reducing x and y, we will get them equal to each other, which will then return the GCD at the end.

Runtime: We are reducing x and/or y by $\frac{1}{2}$ each call, so we call BINARY GCD $\log_2(x) + \log_2(y)$ times. Each call does $O(1)$ time. Total runtime = $O(\log_2(x) + \log_2(y))$

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:

$A(n) = A(n/6) + 1$

$= A(n/36) + 1 + 1$

$= A(n/216) + 1 + 1 + 1$

$= A\left(\frac{n}{6^k}\right) + k$

$1 = \frac{n}{6^k} \rightarrow k = \log_6(n)$

$= A(1) + \log_6(n)$

$= 1 + \log_6(n)$

$= O(\log_6 n)$

(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:

$B(n) = B(n/6) + n$

$= B(n/36) + \frac{n}{6} + n$

$= B(n/216) + \frac{n}{36} + \frac{n}{6} + n$

$= B(1) + \sum_{i=0}^{k} \frac{n}{6^i}$

geometric sequence $\sum_{k=0} ar^k = \frac{a}{1-r}$

$= 1 + \frac{n}{1-\frac{1}{6}}$

$= 1 + \frac{n}{\frac{5}{6}}$

$= 1 + \frac{6n}{5}$

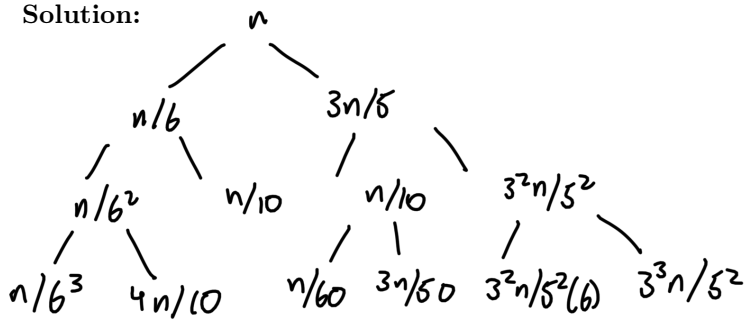
$= O(n)$

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

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

$= C(5n/30) + C(18n/30) + n$

Solution:



$n + \frac{23n}{30} + \frac{529n}{900} + \vdots + \frac{23^k n}{30^k}$

$\sum_{i=0}^{k} \frac{23^i n}{30^i}$

$= \frac{n}{1-\frac{23}{30}}$

$= \frac{n}{\frac{7}{30}}$

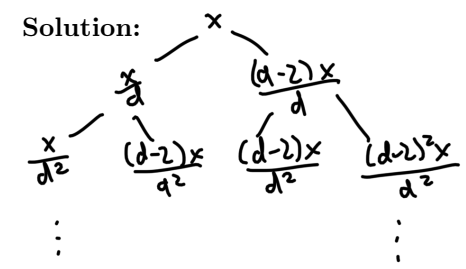
$= \frac{30n}{7}$

$= O(n)$

(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:



$x + \frac{x+(d-2)x}{d^2} + \frac{x+2(d-2)x+(d-2)^2x}{d^2} + \vdots + \frac{x(1+(d-2))^k}{d^k}$

$\sum_{i=0}^{k} \frac{(1+(d-2))^k x}{d^k}$

$= \frac{x}{1-\frac{(1+(d-2))}{d}}$

$= \frac{x}{\frac{d-1+d+2}{d}}$

$= \frac{dx}{2d+1} = O(x)$

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.

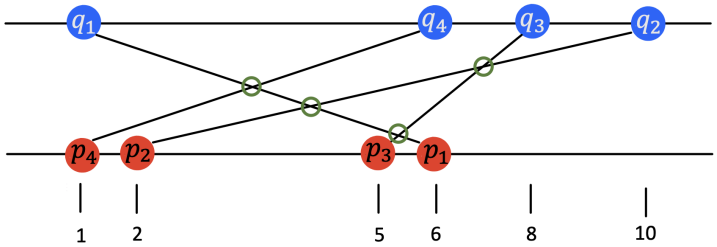


Figure 1: An example for the line intersection problem where the answer is 4

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