CS 577 (Epic) # Assignment 6 – Divide and Conquer 2 Spring 2021

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.

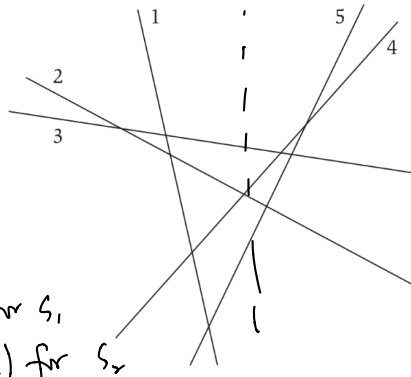
Name: Zhuoyan Xu Wisc id: 7XU444

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 lines in the place labeled $L_1 \dots L_n$. You may assume that no three lines ever meet at the same point. 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.

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



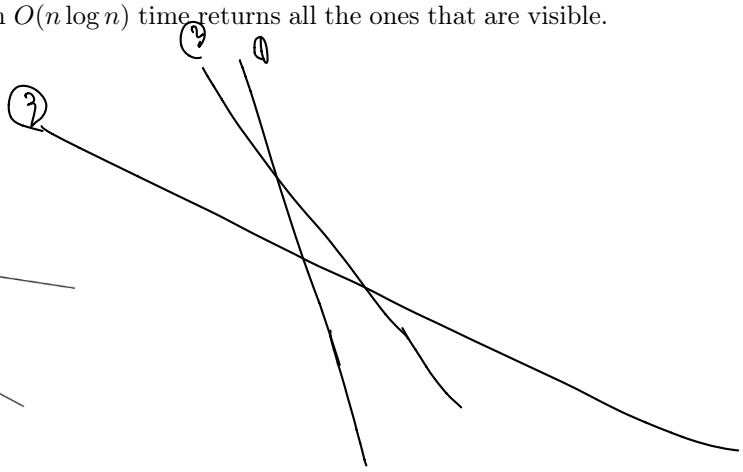


Figure 5.10 An instance of hidden surface removal with five lines labeled *1–5* in the figure. Line segments of lines except for *2* are visible.

end — get $(\tilde{l}_1, \tilde{l}_2)$ for S_1
Similarly: get $(\tilde{l}_3, \tilde{l}_4)$ for S_2
brute-force compute $(\tilde{l}_1, \tilde{l}_2, \tilde{l}_3, \tilde{l}_4)$ get final solutions.

Solution: Since that no 3 lines ever meet at the same point we sort lines by slope and add them one-by-one

Input: n lines Output: solutions

Sort n line based on slope, we have $l_1, \cdots l_n$. Let $S = \{l_1, l_2\}$

let $\tilde{l}_1, \tilde{l}_2$ denote the most recent added lines in S when adding a new line

for $i = 3 \cdots n$
  compute intersect of $\tilde{l}_1, \tilde{l}_2$. $A = (x_A, y_A)$. consider: $l_i: ax+by+c=0$
  while: $ax_A + by_A + c < 0$ (y_A smaller than the y point on l_i with same x_A value)
    remove $\tilde{l}_2$ from S. get new $\tilde{l}_1, \tilde{l}_2$. get new $A(x_A, y_A)$
  end
  add l_i to S.
end

Sorting requires $O(n \log n)$. if we do not enter while loop, which means we do

not remove lines, we just add lines one-by-one, this is $O(n)$. If we enter while loop, since each line can only be remove once, we have $O(n)$. Total is $O(n \log n)$.

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: we have P_x, P_y, P_z as sorted x, y, z values. $O(n \log n)$
find a hyperplane $x = k$ divides space into A (left half of P_x) R (right half of P_x)
then seperate corresponding y, z values to assigned points in A, R ($O(n)$)
divedes based on y, z, and do this recursively. we have $\min\{d(q_0^*, q_1^*), d(r_0^*, r_1^*)\}$
Then use Lemma in lecture, divide positions around hyperplane, check points ($O(n)$)
then total complexity $O(n \log n)$

   (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 use the same algorithm, we consider points on surphare as a subset of larger 3D space. we modify our distance metric $d(x_0, x_1)$ to distance on surphare. Then we use the same algorithm.

   (c) Finally, one way to approximate the surface of a sphere is to take a plane and "wrap" at the edges, so a point at x coordinate 0 and y coordinate MAX is the same as x coordinate 0 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: We treat surface on surphere as a plane. Then we implement regular closest-pair algorithm on this single plane. Then, for checking the points in S (subset of points within δ of L), we also add points within δ of edge of plane into set S. We "padding" the edge use the otherside of points. (We copy the points at lower δ-edge of plane and add to upper of plane).

Then we not only check pairs cross divider but check pairs across edge.

Then the following algorithm is the same.

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 )
```

(2) Then we show correctness. Since the algorithm return the correct result, and algorithm termates in $\lfloor \lg X \rfloor + 1 + \lfloor \lg y \rfloor + 1$. Then the algorithm is correct.

(3) Worst - running time.
The recurrison will stop when $x = y$, In worst - case. it stop at $x = y > 1$.
The running time is $O(\log X + \log y)$

Solution: (1) First we show invariant. Consider k is the greatest common divisor of X and y. ① $X > y$, then $k = X - y$. ② X, y both even. then k is even, and $k/2$ is GCD of $\frac{X}{2}, \frac{y}{2}$. (A is set of CD of (X, y). B is set of CD of $(\frac{X}{2}, \frac{y}{2})$. then $A = 2B$)
③ X is even, y is odd, then k is odd. then any d s.t. X, y is divisible by d, $\frac{X}{2}, y$ is divisible by d. vice versa. then $CD(X, y) = CD(\frac{X}{2}, y)$. we have $GCD(X, y) = GCD(\frac{X}{2}, y)$ ④ X is odd, y is even, we have some results.
⑤ X is odd. y is odd, $X > y$, for d s.t. $X = c_1 d$, $y = c_2 d$. we have $(x - y) = (c_1 - c_2) d$
Since d is odd, we have $\frac{X-y}{2}$ is divisable by d. vice versa. Then $CD(X, y) = CD(\frac{X-y}{2}, y)$. then they have some GCD.
⑥ X is odd, y is odd. $X < y$, we have some result.

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 $n \geq 1$.

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

Solution:

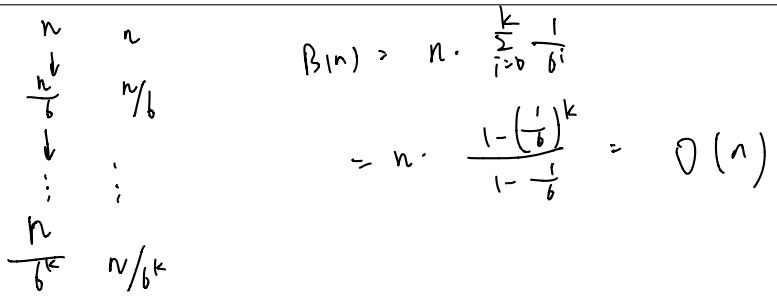
n	1
$\frac{n}{6}$	1
⋮	⋮
$\frac{n}{6^k}$	1

$A(n) \leq \sum_{i=0}^{k} 1 = k = \log_6 n = O(\log n)$

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

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

Solution:

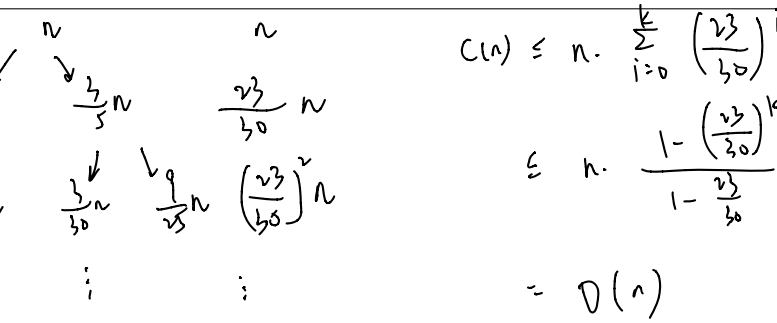


$$B(n) = n \cdot \sum_{i=0}^{k} \frac{1}{6^i} = n \cdot \frac{1-\left(\frac{1}{6}\right)^k}{1-\frac{1}{6}} = O(n)$$

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

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

Solution:

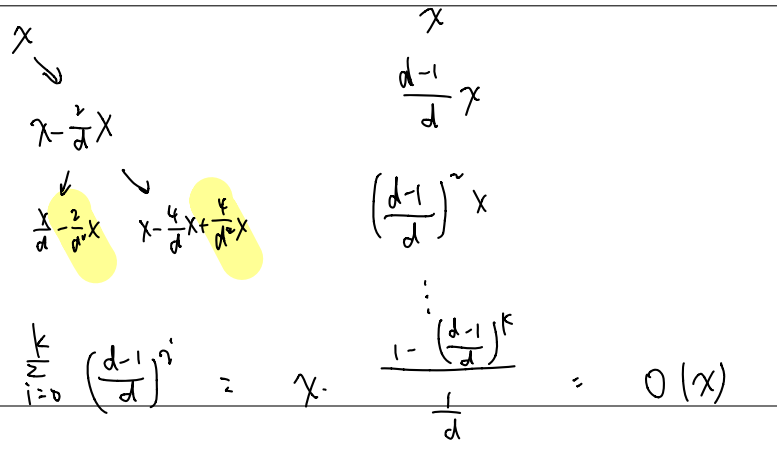


$$C(n) \leq n \cdot \sum_{i=0}^{k} \left(\frac{23}{30}\right)^i \leq n \cdot \frac{1-\left(\frac{23}{30}\right)^k}{1-\frac{23}{30}} = O(n)$$

(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:



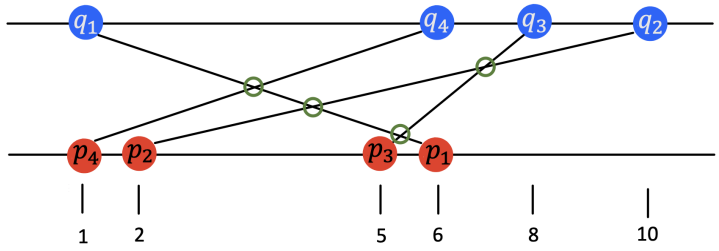
$$D(x) \leq x \cdot \sum_{i=0}^{k} \left(\frac{d-1}{d}\right)^i = x \cdot \frac{1-\left(\frac{d-1}{d}\right)^k}{\frac{1}{d}} = O(x)$$

5. Implement a solution in either C, C++, C#, Java, or Python 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 of a point q_i on the top line. Followed by the final n lines of the instance each containing the location 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.

we sort the top list. and reorder the bottom list by the order of top list



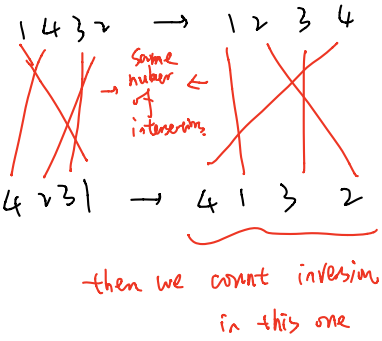


Figure 1: An example for the line intersection problem where the answer is 4

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.

L_1, L_2 is top list

R_1, R_2 is bottom list.

$|L_1| = |R_1|$

$|L_2| = |R_2|$

```
merge(L1, L2, R1, R2).   merge L1, L2 by order.
                         merge R1, R2 by the order of L1, L2.
index1 = 0; index2 = 0;
resq = []; resp = [];   add2 = false
if index1 == L1.length && index2 == L2.length:
    return resq, resp
while true:
    add2 = false;
    if index1 == L1.length:
        add2 = true;
    else if index2 != L2.length && L1(index1) ≤ L2(index2):
        add2 = true;
    if (!add2):
        resq.append(L1(index1));   resp.append(R1(index1))
        index1++;
    else:
        resp.append L2(index2);   resq.append(R2(index2))
        index2++;
```

```
mergeSort(L):
    bin = ⌊L.length/2⌋
    L1 = L[0 : bin]
    L2 = L[bin : L.length]
    L1 = L1.mergeSort;
    L2 = L2.mergeSort;
    return merge(L1, L2)
```

the complexity of mergeSort is $O(n \log n) \leq O(n^2)$

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
```