Assignment 3 – Divide and Conquer

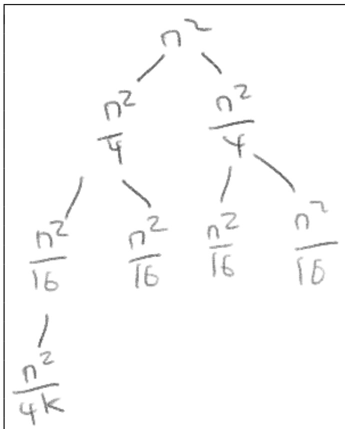
Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Alan Liang   Wisc id: 908 427 1296

Divide and Conquer

1. *Erickson, Jeff. Algorithms (p.49, q. 6).* Use recursion trees to solve each of the following recurrences.

   (a) $C(n) = 2C(n/4) + n^2$; $C(1) = 1$.



$2^0 \cdot \frac{n^2}{2^0} = n^2$

$2^1 \cdot \frac{n^2}{4^1} = \frac{n^2}{4}$

$2^2 \cdot \frac{n^2}{4^2} = \frac{n^2}{4}$

$\frac{n^2}{4^k} = 1$

$n^2 = 4^k$

$n = 2^k$

$\lg n = k$

$$\sum_{i=0}^{k} 2^i \cdot \frac{n^2}{4^i} = n^2 \sum_{i=0}^{k} \left(\frac{1}{2}\right)^i = n^2 \frac{\frac{1}{2}^{k+1} - 1}{\frac{1}{2} - 1} = n^2 \frac{\frac{1}{2}^{(\lg n)+1} - 1}{\frac{1}{2} - 1}$$

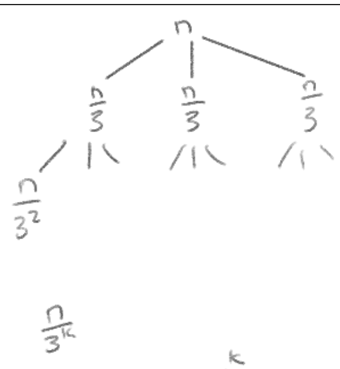
$$= n^2 \frac{\frac{1}{2n} - 1}{\frac{1}{2} - 1} = n^2 \frac{\frac{1}{2n} - 1}{-\frac{1}{2}}$$

$$= n^2 \frac{\frac{1}{n} - 2}{-1}$$

$$= n^2\left(-\frac{1}{n} + 2\right)$$

$$= -n + 2n^2 \in O(n^2)$$

   (b) $E(n) = 3E(n/3) + n$; $E(1) = 1$.



$3^0 \cdot \frac{n}{3^0}$

$3^1 \cdot \frac{n}{3^1}$

$3^2 \cdot \frac{n}{3^2}$

$\frac{n}{3^k} = 1$

$n = 3^k$

$k = \log_3 n$

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

$$(\log_3 n + 1)\, n \in O(n \log n)$$

2. *Kleinberg, Jon. Algorithm Design (p. 246, q. 1).* You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains n numerical values—so there are $2n$ values total—and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we will define here to be the nth smallest value.

   However, the only way you can access these values is through queries to the databases. In a single query, you can specify a value k to one of the two databases, and the chosen database will return the kth smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

   (a) Give an algorithm that finds the median value using at most $O(\log n)$ queries.

Algorith
Let A and B be the two databases with n values each.
FindMedian$(n, 0, 0)$
end

FindMedian(size, a, b)
if size=1, return the smallest element $\min\left(A_{a+\lceil \frac{size}{2} \rceil}, B_{b+\lceil \frac{size}{2} \rceil}\right)$
if the value at $(a + \lceil \frac{size}{2} \rceil)$ in A is greater than the value at $(b+\lceil \frac{size}{2} \rceil)$
return FindMedian$(\lceil \frac{size}{2} \rceil, a+\lceil \frac{size}{2} \rceil, b)$
else
return FindMedian$(\lceil \frac{size}{2} \rceil, a, b+\lceil \frac{size}{2} \rceil)$
end

   (b) Give a recurrence for the runtime of your algorithm in part (a), and give an asymptotic solution to this recurrence.

$F(n) = c + F(\frac{n}{2})$; $c \geq 0$

$\frac{n}{2}$ c $\frac{n}{2^k} = 1$

| +

$\frac{n}{2^1}$ c $k = \lg n$

| +

$\frac{n}{2^k}$ c

$k \cdot c = \lg n \cdot c \in O(\log n)$

(c) Prove correctness of your algorithm in part (a).

Base case: size=1

The algorithm returns the smallest value between A and B which have one value each.

Induction step:

Assume FindMedian returns the median of the two improper subsets of A and B that is passed into it. Moreover, this median of the two smaller sets is the median for the larger set since the number returned and recombined with the set will be smaller than the size/2 numbers of one set and larger than the size/2 numbers of the other set, which maintains its status as a median.

3. *Kleinberg, Jon. Algorithm Design (p. 246, q. 2).* Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, ..., a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, this measure is very sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$.

(a) Give an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.

Input: a sequence of n distinct numbers
Output: a sorted sequence of the input and the number of inversions, V.

FindInversions

```
if |A|=1, return (A,0)
(A1, V1) = FindInversions (First half of A)
(A2, V2) = FindInversions (Second half of A)
(A,V) = MergeSort (A1,A2)
return (A, V1+V2+V)
```

Input: two sequences of comparable nums
Output: One sequence with values from both in sorted order and number of significant inversions.

MergeSort

```
Let S be an empty list, C be 0
while A || B is not empty
    pop & prepend max(back of A, back of B) to S.
    if prepended item is from A and is more than twice of
        back of B, increase C by |B|.
    end
end
return (S, C)
```

(b) Give a recurrence for the runtime of your algorithm in part (a), and give an asymptotic solution to this recurrence.

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

$\rightarrow$ Merge Sort $\rightarrow$ $O(n\log n)$

(c) Prove correctness of your algorithm in part (a).

Find Inversions:

Base case:

A set with 1 element has no inversions.

Induction Step:

Assume A_1 and A_2 are sorted sets and v_1 and v_2 are their respective inversion counts. We proceed to merge A_1 and A_2 using MergeSort. Add the largest element of A and B to the combined. If $a \in A$ is less than $2\times$ $b \in B$, proceed regularly. If $a \in A$ is greater than double of $b \in B$, then we know a is also greater than $2\times$ the rest of the elements left in B (since its in ascending order), so we increase the inversions by such. Adding the inversion counts together gives the total inversions of $A = A_1 + A_2$.

4. *Kleinberg, Jon. Algorithm Design (p. 246, q. 3).* You're consulting for a bank that's concerned about fraud detection. They have a collection of n bank cards that they've confiscated, suspecting them of being used in fraud.

   It's difficult to read the account number off a bank card directly, but the bank has an "equivalence tester" that takes two bank cards and determines whether they correspond to the same account.

   Their question is the following: among the collection of n cards, is there a set of more than $\frac{n}{2}$ of them that all correspond to the same account? Assume that the only feasible operations you can do with the cards are to pick two of them and plug them in to the equivalence tester.

   (a) Give an algorithm to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.

num occurences

Input: A collection, S, of cards

Output: A card, the card with the most occurences.

CountEquivalence

Base

```
C == null
if |S| = 1, then return S
if |S| = 2
| if s1 equals s2, then return s1
C = CountEquivalence(first half of S)
if C is not null
| counter = 0
| Check each element with S. If C matches, increment counter.
| if counter > |S|/2, then return C
C = CountEquivalence(second half of S)
if C is not null
| counter = 0
| Check each element with S. If C matches, increment counter.
| if counter > |S|/2, then return C
return C; // this happens if there does not exist a max element of either sets
```

   (b) Give a recurrence for the runtime of your algorithm in part (a), and give an asymptotic solution to this recurrence.

$T(n) = 2T(n/2) + cn \rightarrow O(n\log n)$

(merge sort)

(c) Prove correctness of your algorithm in part (a).

Base case $|S| = 1$

If there is only one element, then it is majority of the set

Base case $|S| = 2$

If two elements match, then that card is the majority and is returned. Else, there is no majority and nothing is returned

Induction step.

Assume countEquivalence correctly returns the majority element for S_1 & S_2 where S_1 = first half of S and S_2 = second half of S. There are 3 cases

1. Both S_1 & S_2 have a majority
   Check if the majority element exists for more than 50% of the elements, and will return either the majority of S_1, S_2 or null as desired.

2. S_1 has a majority, S_2 does not have a majority (WLOG covers S_2 has and S_1 doesn't)
   If S_1 exists in more than half of the elements of S, then it gets returned.
   Otherwise return nothing as desired.

3. Neither have majority.
   If neither have a majority element, then nothing is returned which is correct since the majority element of S must be majority in at least one of the two sets. [if it is not majority in either, then the occurences of the card will not exceed $n/2$, or majority).

5. *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.

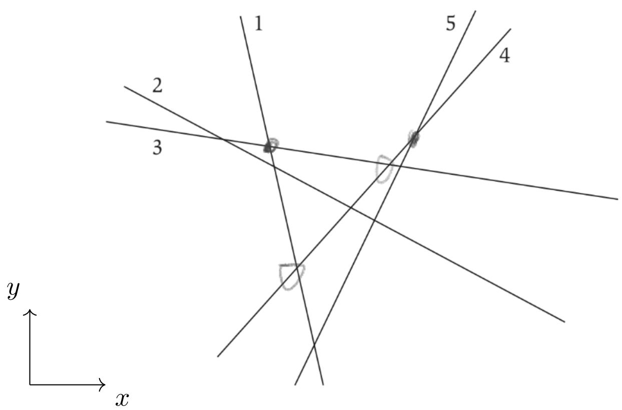


Figure 5.10 An instance of hidden surface removal with five lines (labeled *1–5* in the figure). All the lines except for *2* are visible.

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

Input: A set of lines sorted by slope, S
Output: A set of visible lines, and intersection points

FindVisible

If |S| ≤ 3
    Bruteforce search for visibility and return.
(S_1, L_1) = FindVisible(First n/2 lines of S)
(S_2, L_2) = FindVisible(Second n/2 lines of S)
(S_3, L_3) = Merge(S_1, S_2, L_1, L_2)
return (S_3, L_3)

Input: Two sets of lines, two lists of intersections
Output: One set of lines (merged) and intersections

Merge

S = empty set
Q = $Q_1 + Q_2$ (points)
while (Q contains elements)
    C = pop first element of Q.
    n = line that is uppermost at C from S_1
    m = line that is uppermost at C from S_2
    add elements from S_1 $\{L_1, \dots L_n\}$ to S
    add elements from S_2 $\{L_m, \dots, L_{n-1}, L_{max}\}$ to S
Find where each line intersects next line and add to Q.
return (S, Q)

(b) Write the recurrence relation for your algorithm.

$T(N) = 2(T/2) + n \in O(n \log n)$

$+ n \log n$ preprocessing time

(c) Prove the correctness of your algorithm.

Base case:

When there are only 3 elements in order of increasing slope, the first & third will always be visible. The second will be visible if the intersection with line 1 is less than line 1's intersection with line 3.

Induction Step:

Suppose FindVisible returns the correct set of visible lines and intersection points for (S_1, L_1) and (S_2, L_2). Merge will find the areas in which the visible lines from the two sets intersect and add the ones on the left in S_1 and the ones on the right in S_2 since everything is in increasing slope. These intersections of visible lines is the only point in which visibility may change.

6. 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.

   Divide 3d are using a plane

   instead of just checking the closest 15 points, we must check the points within the 4x4x4 cube (max of 63 other points)

   (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.

   Sort by longitude instead of x?

   (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.

   I barely even know how to do the sphere. I don't know how to split the torus up evenly with each recursive call and how to even calculate $\delta/2$ since I don't know how the coordinate system works.

7. *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 )
```

Base Case:

If $x = y$ then they are each other's greatest common divisor.

Induction step:

If x and y are both even, the GCD will indeed be 2 times GCD$(x/2, y/2)$

If x or y is even but the other is odd, we know 2 is not a factor of the odd, and can factor it out and proceed.

I don't know how to prove the case where x & y are both odd

I don't know how to prove the recurrence since there are so many options for each recursive call. I don't know what to put in the T( ) parameter.

8. Use recursion trees or unrolling to solve each of the following recurrences.

   (a) Asymptotically solve the following recurrence for $A(n)$ for $n \geq 1$.

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

$A(n) \longrightarrow 1$

$A(n/6) \longrightarrow 1$

$A(n/36)$

$A(n/6^k)$

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

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

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

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

$n = 6^k$

$k = \log_6 n$

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

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

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

$$n \to n/6 \to n/36 \to$$

$$\frac{n}{6^0} \quad \frac{n}{6^1} \quad \frac{n}{6^2} \quad \frac{n}{6^k}$$

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

$$6^k = n$$

$$k = \log_6 n$$

$$\sum_{i=0}^{k} \frac{n}{6^i} = n \sum_{i=0}^{k} \left(\frac{1}{6}\right)^i = n \frac{\frac{1}{6}^{k+1} - 1}{\frac{1}{6} - 1} = n \frac{\frac{1}{6}^{\log_6 n + 1} - 1}{\frac{1}{6} - 1}$$

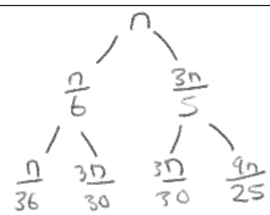
$$= n \frac{\frac{1}{6n} - 1}{\frac{1}{6} - 1}$$

$$= n \frac{1/n - 6}{1 - 6} = n \frac{\frac{1}{n} - 6}{-5}$$

$$= -\frac{1}{5}(1 - 6n) \in O(n)$$

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

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



$$n$$

$$\frac{5n}{30} + \frac{18n}{30} = \frac{23n}{30} = \left(\frac{23}{30}\right)^1 n$$

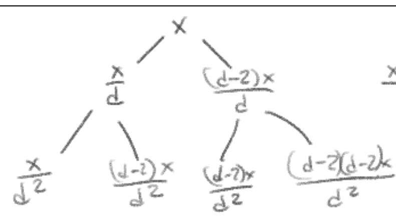
$$\left(\frac{23}{30}\right)^2 n$$

$\left(\frac{23}{30}\right)^k n$ at each level

$$n \sum_{i=0}^{k} \left(\frac{23}{30}\right)^i \leq n \sum_{i=0}^{\infty} \left(\frac{23}{30}\right)^i = \frac{n}{1 - \frac{23}{30}} = \frac{n}{\frac{7}{30}} = \frac{30n}{7} \in O(n)$$

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

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



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

$$x\left(\frac{(d-1)}{d}\right)^k$$

$$x \sum_{i=0}^{k} \left(\frac{d-1}{d}\right)^k \leq x \sum_{i=0}^{\infty} \left(\frac{d-1}{d}\right)^k = \frac{x}{1 - \frac{d-1}{d}} = \frac{dx}{d - d^2 - d} = \frac{dx}{d^2} = \frac{x}{d}$$

$$O\left(\frac{x}{d}\right)$$

Coding Questions

9. **Inversion Counting:**
Implement the optimal algorithm for inversion counting in either C, C++, C#, Java, Python, or Rust. Be efficient and implement it in $O(n \log n)$ time, where n is the number of elements in the list.

The input will start with an positive integer, giving the number of instances that follow. For each instance, there will be a positive integer, giving the number of elements in the list.

Note that the results of some of the test cases may not fit in a 32-bit integer.

A sample input is the following:

```
2
5
5 4 3 2 1
4
1 5 9 8
```

The sample input has two instances. The first instance has 5 elements and the second has 4. For each instance, your program should output the number of inversions on a separate line. Each output line should be terminated by a newline. The correct output to the sample input would be:

```
10
1
```

10. **Line Intersections:**
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, your code needs to run in $O(n \log n)$ time.
Hint: How does this problem relate to 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.

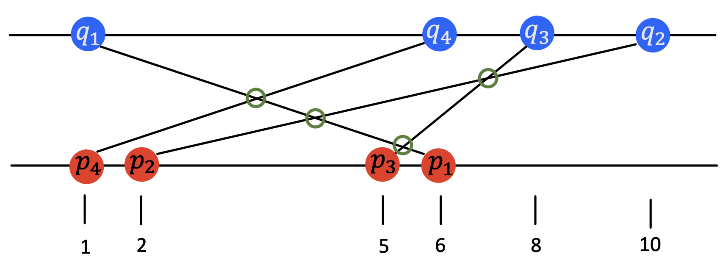


Figure 1: An example for the line intersection problem where the answer is 4

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 the results of some of the test cases may not fit in a 32-bit integer.

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
```