CS375 Assignment 2 (Fall 2025)
(Due Sept 27, 2025 by 11:59pm)

Objective:

(1) Design and analyze an algorithm with D&C or recursion.
(2) Establish recurrence equation and solve it.

There are two parts in this assignment: (A) Theory part and (B) programming part

[Part A] Theory [78%]:

1. (6%) We have a problem that can be solved by a direct (non-recursive) algorithm that operates in N^2 times. We also have a recursive algorithm for this problem that takes NlgN operations to divide the input into two equal pieces and lgN operations to combine the two solutions together. Show whether the direct or the recursive version is more efficient. (Note: For the recursive algorithm, the base case is T(n) = 1 if n=1.)

2. [10%]

   a. [3%]Write the recurrence equation for the code below. Use the number of comparisons as your barometer operation (The *min* operation requires 1 comparison and the *max* operation requires 1 comparison): *): (Note: you can count min and max as the comparison operation and skip the rt-lt <=1)*

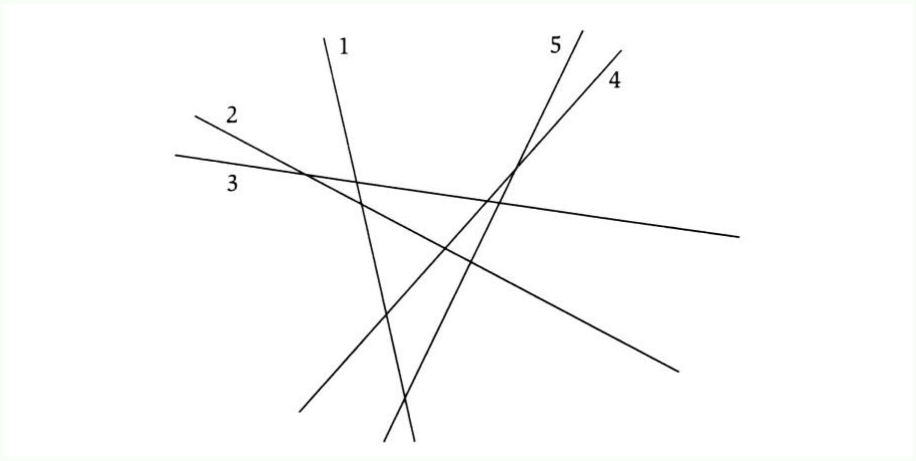
```
MinMax(A,lt,rt)
// return a pair with the minimum and the maximum
   if (rt - lt ≤ 1)
       return (min(A[lt], A[rt]), max(A[lt],A[rt]));
   (min1, max1) = MinMax(A,lt, ⌊(lt+rt)/2⌋ );
   (min2, max2) = MinMax(A, ⌊ (lt+rt)/2 ⌋ +1,rt);
   return (min (min1, min2), max(max1, max2));
```

   b. [4%] Show the recursion tree and solve the recurrence equation for this code. For simplicity assume n = 2^k.

   c. [3%] Prove the solution using induction

3. (8%) Design an algorithm for visible lines detection: Given *n* non-vertical lines in a plane, labeled *L1…., Ln*, with line equation *y=ai*x +bi (i=1, …n).* We can make an assumption that no three of the lines all meet at a single point.

Definition: line *Li* is uppermost at a given x-coordinate *x0* if its y-coordinate at *x0* is greater than the y-coordinates of all the other lines at *x0*. Line *Li* is visible if there is some x-coordinate at which it is uppermost – intuitively, some portion of it can be seen if you look from infinity of y (*y = ∝)*; For example: following figure is an instance of visible lines (labeled 1-5). All the lines except for 2 are visible.



Give an algorithm that takes n lines as input and in no more than O(n^3) time returns all of the ones that are visible. Show the time complexity.

4. (9%) Given a sorted array of distinct integers A[1, …, n], you want to find out whether there is an index i for which A[i]=i. Give a divide-and-conquer algorithm to solve this problem. Derive the time complexity. (Note: the running time much be less than O(n)).

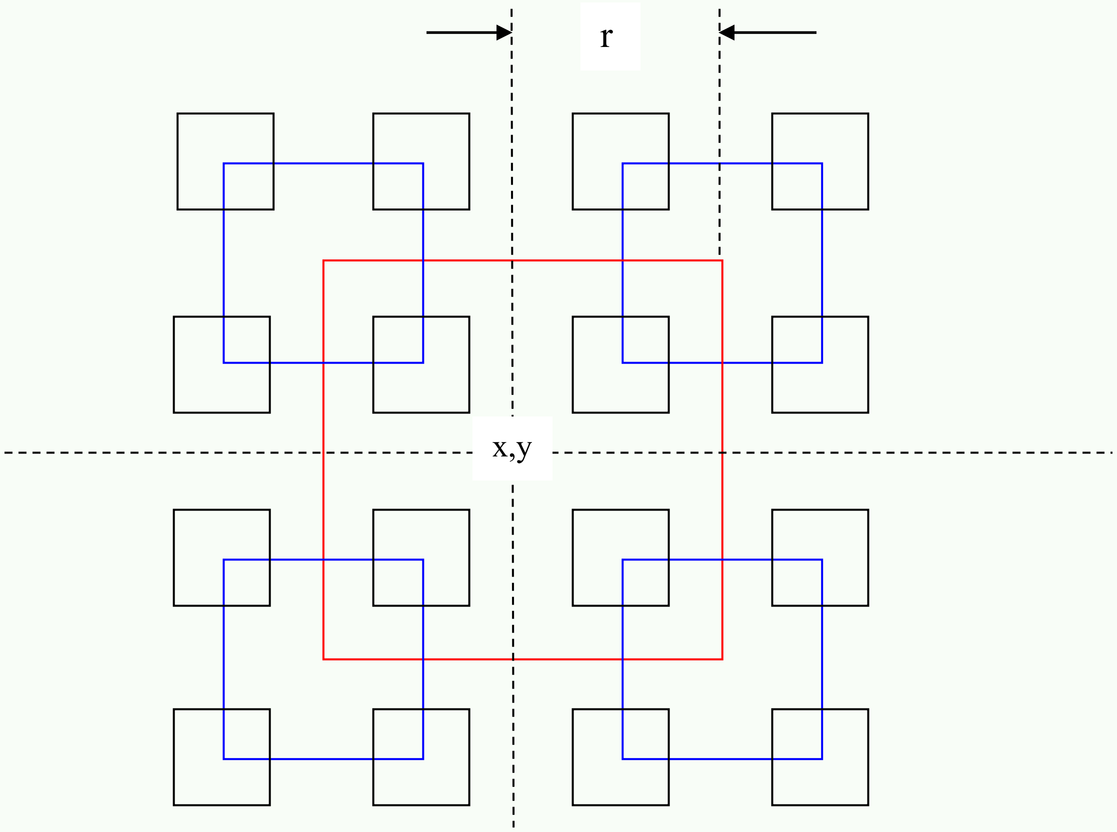
5. [10%] Write a piece of pseudo-code to plot the following graph. Assuming that the plotting function has been provided as DrawSquare(x, y, r), which draw a square of size 2r with center (x, y).

(1) Pseudo-Code to plot following graph. (4%)

(2) Write the recurrence equation for your algorithm. (3%)

(3) Plot the recursion tree to derive the solution of T(r). (3%)

(Note: r is the input size, which is the power of 2. The time complexity for drawing one square is O(1)). (Hint: The input of function can be: int x, int y, int r)



6. (9%) An Array A[1,…,n] is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. Show how to solve this problem in O(nlgn) time.(Hint: Split the array A into two arrays A1 and A2 of half the size. Does knowing the majority elements of A1 and A2 help you figure out the majority element of A?) Note that it is required to use a divide-and-conquer approach with O(nlgn) time complexity.

7. [6%] Find the asymptotic bound of the divide and conquer recurrence T(n) using master method:

    1. $T(n) = 9*T(n/3) + n^2+4$
    2. $T(n) = 6*T(n/2) + n^2-2$
    3. $T(n) = 4*T(n/2) + n^3 +7$

8. [9%] Design an algorithm to rearrange elements of a given array of n real numbers so that all its negative elements precede all its positive elements. Your algorithm should be both time- and space-efficient. (Hint: using the idea of partition with D&C).

9. [5%] Solve the following recurrence equation using methods of characteristic equation.

T(n)=8T(n-1)-21T(n-2)+18T(n-3) for n>2
T(0)=0
T(1)=1
T(2)=2

*Hint: if there are two same roots, the solution template is T(n) = C1*r1^n + C2*r2^n + C3*n*r3^n*

10. [6%] Solve the following recurrence equations using recursion tree technique discussed in class. You may make any assumptions about *n*, such as to assume that *n* is an exact power of 2. (hint: Base case: T(0) = T(1) = Theta (1))

T(n)=T(n/2)+T(n/4)+T(n/8)+n

[Part B]: Divide and Conquer Programming (22%)

1. [12%] Implement the algorithm for finding the closest pair of points in two dimension plane using divide and conquer strategy.

A system for controlling air or sea traffic might need to know which are the two closest vehicles in order to detect potential collisions. This part solves the problem of finding the closest pair of points in a set of points. The set consists of points in R^2 defined by both an x and a y coordinate. The "closest pair" refers to the pair of points in the set that has the smallest Euclidean distance, where Euclidean distance between points $p_1=(x_1,y_1)$ and $p_2=(x_2,y_2)$ is simply $sqrt((x_1-x_2)^2+(y_1-y_2)^2)$. If there are two identical points in the set, then the closest pair distance in the set will obviously be zero.

Input data: n points with coordinates:

X coordinates: p[0].x, p[1].x, p[2].x,…., p[n].x

Y coordinates: p[0].y, p[1].y, p[2].y,…., p[n].y

Output: minimum distance between points p[i] and p[j] (index i and j should be identified)

Input data for test:

```
n = 10000;
for(i=0; i<n; i++)
 {
   p[i].x= n- i;
   p[i].y= n- i;
    }
```

Other Input data for test:

```
n = 10000;
for(i=0; i<n; i++)
 {
   p[i].x= i*i;
   p[i].y= i*i;
    }
```

Or:

Input:

If X>0
X=1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, ……,19995, 19997, 19999   (odd numbers)
Y=int($\sqrt{9999^2 - (X - 10000)^2}$)

Else
X=0, -2, -4, -6, -8, -10, -12, ……, -19996, -19998, -20000 (even number)
Y= int($\sqrt{10000^2 - (X + 10000)^2}$)

2. (10%) Use the brute-force approach to solve the above problem. Compare the two implemented algorithms in terms of time complexity. Print out the time cost during the execution of these two algorithms.

3. (Optional: Extra 10%) apply the close-pair algorithm to the three dimensional case.

Note 1:

Your report (.doc) of the programming part should follow the following format:

(1) Algorithm description

(2) Major codes

(3) Running results

(4) Report of any bugs

Note 2:

2.1 For Part B, your code package includes your code, makefile, executable file, and write-up (.doc).

2.2 Your program will read an input file and write an output file.

2.3 Your program should be invoked like this…

```
prompt>submission inputFile.txt outputFile.txt
```

where inputFile.txt is referring to an input file,
ouputFile.txt is referring to an output file.