University of Iowa | Homework 7 | CS 3330
Department of CS | Fall 2024

Assigned: Friday 11:59 PM, October 25, 2024
Due: Friday 11:59 PM, November 1, 2024

1. [**20 points; A Human Compiler**]: A computer scientist must understand how algorithms work. Compile the following algorithm for the Partition-3-way subroutine of the QuickSort for the given input.

Algorithm 1 Partition-3-Way

Require: Array A and pivot element p
1: Exchange p with the last element in A
2: $i \leftarrow -1$
3: **for** j in range 0 to length of $A - 1$ **do**
4:     **if** $A[j] \leq p$ **then**
5:         $i \leftarrow i + 1$
6:         Exchange $A[i]$ with $A[j]$
7:     **end if**
8: **end for**
9: Exchange p with $A[i + 1]$
10: **return** $A[0 \text{ to } i], A[i + 1], A[i + 1 \text{ till end}]$

Input Array A = [2,4,7,1,3,5,6] and $p = 4$.

(a) [**3 points**] First, show A just after line 2.

(b) [**14 points**] Now demonstrate how A gets updated at the end of each iteration of the for loop using the following table.

Value of j	Value of i	Array
0		
1		
2		
$\vdots$	$\vdots$	$\vdots$
6		

(c) [**3 points**] Show A just after after line 9.

2. [**20 points; Bad-luck sorting**] We discussed Randomized Quicksort Algorithm in class, where we randomly pick a pivot element p in the input array A, and partition the array into left (consisting of elements $\leq p$), right (consisting of elements $> p$) and middle (with only p) sub-arrays. We then recursively sort the left and the right sub-arrays.

(a) [**3 points**] Assuming all elements of A are unique, picking which elements as the pivot leads to the worst partition of the input array A? Why? If multiple elements lead to the worst partition, list them all.

(b) **[7 points]** Let us assume we get unlucky and randomly pick the worst pivot in each recursive call. What is the resulting recurrence for the running time?

(c) **[10 points]**Solve the recurrence in (b) using the substitution method.

3. [**35 points; Solving recurrences**] We looked at recurrence relations when computing running times of divide-and-conquer algorithms. Let us determine a tight asymptotic bound for the recurrence $T(n) = 4T(n/2) + cn$ where c is a constant.

   (a) **[10 points]** Draw the recursion tree. Show the amount of work done at each level, the number of leaves, and the tree's height.

   (b) **[10 points]** Using the tree you just drew, solve for $T(n)$.

   (c) **[15 points]** Verify your solution by the substitution (induction) method.

4. [**25 points; Closest Pair of Points**] In this problem, we are going to design a divide-and-conquer algorithm for the Closest Pair of Points problem.

   In this problem, we are given a set of n points in a 2D plane. We aim to find the closest pair of points (the pair with the minimum Euclidean distance between them). For example, if the given list of points is P = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]. Our algorithm should return the points (2,3) and (3,4).

   (a) **[5 points]** Design a naive $\Theta(n^2)$ algorithm, where n is the number of points. Justify the running time of your algorithm.

   (b) **[4 points]** Now, let us start by sorting the points by their y-axis and dividing the points into two groups Top and Bottom by drawing a horizontal line through (or near) the median y-axis value such that each half has $n/2$ points and recursively finding the minimum distance in each half: m_{top} and m_{bottom}. This is the divide step of the divide-and-conquer approach. Now, I propose taking the minimum between m_{top} and m_{bottom} as the solution for the conquer step. This algorithm is incorrect; give a counter-example to show that it returns the wrong pair of points. Draw your counter-example in a 2D plane.

   (c) **[5 points]** Solve for the running time of the incorrect algorithm in part (b).

   (d) **[5 points]** The above algorithm is erroneous as it misses points that are divided into top and bottom halves but have a shorter distance than $\min(m_{top}, m_{bottom})$. We can fix this by computing the minimum distance m_{mid} between points a and b such that a is in the top half and b is in the bottom half. Assuming we do this by iterating over all such points, write down the resulting recurrence relation and solve it.

   (e) **[5 points]** Now, we want to design an algorithm that is more efficient than the one in part (d). For the rest of the problem, let $m_{min} = \min(m_{top}, m_{bottom})$. It turns out that it is enough to check for pairs of points across the dividing line that might be closer than m_{min}. These points are within a horizontal strip of width 2 m_{min} centered on the dividing line. The points within the strip are sorted by their x-coordinates to efficiently check possible pairs within distance m_{min}. For any point in the strip, it is enough to check its distance against the next 7 points in the strip in sorted order. The smallest distance within the strip is m_{mid}. Now, we can return the minimum among m_{top}, m_{bottom}, and m_{mid}. Write down the pseudocode for the entire divide-and-conquer algorithm using this strategy.

(f) **[1 point]** Instead of sorting the points at each layer, we can maintain a list of points sorted by the x-coordinate and the y-coordinate and use the merge subroutine to combine the list. This ensures that the work done at the level with n points is $\Theta(n)$. Using this information, write down the recurrence relation and give the final running time.