CS180 Exam 1

Justin Ma

TOTAL POINTS

22 / 22

QUESTION 1

Problem 1 8 pts

1.1 Asymptotic notation **1 / 1**

✓ **- 0 pts 6 out of 6**

- 0.25 pts 5 out of 6

- 0.4 pts 4 out of 6

- 0.5 pts 3 out of 6

- 0.6 pts 2 out of 6

- 0.75 pts 1 out of 6

- 0.75 pts 0 out of 6

1.2 True or False: DC **1 / 1**

✓ **- 0 pts Correct**

- 0.4 pts Wrong answer but correct formula formed

- 0.5 pts Wrong answer with wrong formula

- 0 pts Correct but wrong explanation

1.3 Principles of DC **1 / 1**

✓ **- 0 pts Correct**

- 0.4 pts divide not mentioned

- 0.4 pts merge not mentioned

1.4 Solving recurrence **1 / 1**

✓ **- 0 pts used master theorem**

- 0 pts Used expansion

- 0.5 pts wrote the master theorem components but wrong reasoning

- 0.75 pts master theorem components are wrong

- 0.5 pts used expansion but wrong answer

- 0.75 pts wrong attempt for expansion

1.5 Karatsuba trick **1 / 1**

✓ **- 0 pts Correct**

- 0.5 pts wrong formation of trick

- 0.75 pts no usage of trick at all

1.6 List vs Matrix representations **1 / 1**

✓ **- 0 pts Correct**

- 0.5 pts no mention of space

- 0.5 pts no mention of edge access time

- 0.75 pts missing considerations of space and edge access times

1.7 Definition of path **1 / 1**

✓ **- 0 pts Correct**

- 0.5 pts Incorrect definition / not generic

1.8 Checking if graph is connected **1 / 1**

✓ **- 0 pts Correct**

- 0.7 pts Wrong Answer

- 0.5 pts Did not check if all vertices are discovered

- 0.5 pts Did not check if all vertices are connected/discovered. Just checked one.

QUESTION 2

2 Sorting sorted arrays **4 / 4**

✓ **- 0 pts Correct**

- 1.5 pts using mergesort to combine 2 sorted arrays. Gives runtime O((nk) log(nk)) - more than allowed.

- 1 pts unclear merge step

- 1.5 pts heap ops should be stated and clarified as these were not covered in class.

- 1.75 pts reasonable attempt but missing crucial details and/or not correct.

- 2.25 pts Missing crucial details and/or not correct.

- 3 pts attempt something relevent

- 3.5 pts attempt something irrelevent

- 4 pts empty

- 3 pts Solution runs in time O(n k^2) time, much more than the O(nk log k) the problem was looking for.

QUESTION 3

3 Finding plurality elements **4 / 4**

✓ **- 0 pts Correct**

- **0.5 pts** no base case
- **1.5 pts** no/wrong run-time analysis or no recurrence relation of the time complexity
- **1.5 pts** no/wrong counting of returned elements from the recursion in the merge part
- **1.75 pts** reasonable attempt but not returning all plurality elements
- **2.25 pts** reasonable attempt with an algorithm running in time O(n^2) or worse.
- **2.5 pts** attempt missing many details and not correct.
- **3.25 pts** not a reasonable attempt
- **4 pts** no answer

QUESTION 4

4 Closest pair L4-distance **4 / 4**

✓ **- 0 pts** **Correct**

- **0 pts** You check way too many points for S_y and didn't show how you derived the number. Try to simplify your strip construction./ Or show how you derive this number
- **2.25 pts** reasonable attempt but missing many crucial details and/or not correct.
- **2.5 pts** moderate attempt but missing many crucial details and/or not correct.
- **1.5 pts** Didn't state how to compute/how to organize the points in the strip S. (for example, "sort by y coordinate" or including which points in strip or the width/height of grid) or Wrong way to construct the strip and grid.
- **1.5 pts** Didn't mention how many points to look up for each S_y in the strip
- **1.5 pts** Didn't Identify the divide-conquer high-level steps correctly
- **4 pts** No answer
- **1.5 pts** wrong number of points to look up

QUESTION 5

5 BFS trace **2 / 2**

✓ **- 0 pts** **Correct**

- **1 pts** Extra lists than needed (You have mostly not considered the edges {4,6} {5,6} in line 2 of the Question)
- **1 pts** Extra lists than needed
- **1 pts** L[2] has extra elements
- **0.75 pts** L[2] order of elements wrong
- **0.5 pts** L[1] order of elements wrong

Exam 1. April 25, 2018

CS180: Algorithms and Complexity
Spring 2018

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so.
- Write your solutions clearly and when asked to do so, provide complete proofs. You may use results and algorithms from class without proofs or details **except for Problem 4** as long as you state what you are using.
- I recommend taking a quick look at all the questions first and then deciding what order to tackle to them in. Even if you don't solve the problems fully, attempts that show some understanding of the questions and relevant topics will get reasonable partial credit.
- You can use extra sheets for scratch work, but you can only use the white space (it should be more than enough) on the exam sheets for your final solutions.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly and any cheating reported with the score automatically becoming zero.
- Write clearly and legibly. All the best!

Problem	Points	Maximum
1		8
2		4
3		4
4		4
5		2
Total		22

Name	Justin Ma
UID	604805167
Section	1E

$n = O(n^2) = o(n)$

1 Problem

The answers to the following should fit in the white space below the question.

1. For each pair (f, g) below indicate the relation between them in terms of O, Ω, Θ. For each missing entry, write-down Y (for YES) or N (for NO) to indicate whether the relation holds (no need to justify your answers here). For example, if $f = O(g)$ but not $\Omega(g)$, then you should enter Y in the first box and N in the other two boxes. Similarly, if $f = \Theta(g)$, then you should enter Y in all the boxes. [1 point]

f	g	O	Ω	Θ
n^2	$n^2 - 2n + 2$	Y	Y	Y
$\log_2 n$	$(\log_{100} n)^2$	Y	N	N

2. Is the following True or False: Consider a divide and conquer algorithm which solves a problem on an instance of length n by making six recursive calls to instances of length $\lfloor n/3 \rfloor$ each, and combines the answers in $O(n^2)$ time. Then, the time-complexity of the algorithm is $O(n^2)$. [1 point]

$\boxed{\text{True}}$ $T(n) = 6T(\frac{n}{3}) + O(n^2)$ $6 < 3^2$

3. State the principles behind the divide and conquer technique for designing algorithms. [1 point]

- State the base case
- If base case not met, divide problem into smaller sets & conquer each smaller set recursively
- merge the seperate solutions into one solution.

4. What is the solution to the recurrence $T(1) = 1$, $T(n) = 2T(n/2) + 10n$? [1 point]

$O(n \log n)$ $2 = 2$

$T(1) = 2T(0.5) + 5$

5. Let a_0, a_1, b_0, b_1 be four integers that are k bits long. Write down Karatsuba's trick (that we used in class for fast integer multiplication) to compute the four products $a_1 \cdot b_1, a_1 \cdot b_0, a_0 \cdot b_1, a_0 \cdot b_0$ using only three multiplications and some additions and subtractions.

$$a_1 \cdot b_1 = \overbrace{a_1 \cdot b_1}^{X}$$

$$a_0 \cdot b_0 = \underbrace{a_0 \cdot b_0}_{Y}$$

$$a_0 \cdot b_1 + a_1 \cdot b_0 = \underbrace{(a_1 + a_0)\cdot(b_1 + b_0)}_{Z} - \underbrace{a_1 \cdot b_1}_{X} - \underbrace{a_0 \cdot b_0}_{Y}$$

6. Write down some pros and cons of the adjacency-list and adjacency-matrix representations of graphs. [1 point]

adjacency list: + low memory cost
- takes $O(n)$ to determine if 2 vertices have edge b/w them

adjacency matrix: + takes $O(1)$ to determine if 2 vertices have edge b/w them
- high memory cost

7. Write down the definition of a path in a graph $G = (V, E)$. [1 point]

a path between vertices A and B in G means there exist edges that connect A and B with an arbitrary number of vertices between (can be 0 in which case edge {A, B} exists.)

8. How can we efficiently check if a graph given in adjacency-list representation is connected? (You can refer to algorithms done in class without writing them out fully.) [1 point]

Pick an arbitrary vertex in the array of vertices and use BFS to find all connected points. Refer to the discovered array, and if every element is discovered, the graph is connected. Otherwise, it is not connected.

2 Problem

You are given k sorted arrays, each with n numbers in them. Give an algorithm for merging these arrays into a single sorted array of numbers that runs in time $O(nk \log k)$. You don't have to analyze the running time or prove correctness. [4 points]

(You can assume that the solution to the following recurrence is $O(nk \log k)$: $T(1) = O(1)$, $T(k) \leq 2T(k/2) + O(n \cdot k)$.)

ksort (1 : k)

- If k = 1, return array as is
- ~~If k = 2, merge arrays using the merge step in merge-sort and return final array~~
- Else, recursively sort both halves of the array

  $L = \text{ksort}\left(1 : \lfloor \frac{k}{2} \rfloor\right)$

  $R = \text{ksort}\left(\lfloor \frac{k}{2} \rfloor + 1 : k\right)$ $\qquad 2T\left(\frac{k}{2}\right)$

- merge arrays L, R using the merge step of merge-sort $\qquad O(k \cdot n)$
- return final array

$$T(k) = 2T\left(\frac{k}{2}\right) + O(k \cdot n)$$

$$\Rightarrow \boxed{O(nk \log k)}$$

3 Problem

Given an array $A[0, 1, \ldots, n-1]$, an element $A[i]$ is said to be a *pluraility element* if more than $\lfloor n/3 \rfloor$ of its elements equal elements of A. For example, the array $A = [1, 11, 2, 4, 2, 2, 1, 2, 4]$ has one plurality element 2; the array $A = [1, 1, 2, 4, 2, 2, 1, 2, 1]$ has two plurality elements 1, 2; the array $A = [1, 11, 2, 1, 2, 1, 11, 2, 11]$ has no plurality elements.

Given an array as input, the task is to design an efficient algorithm to tell whether the array has any plurality elements and, if so, to find all the plurality elements. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as mp3 files, say; so in particular, you cannot sort the elements.) However you can answer questions of the form: "is $A[i] = A[j]$" in constant time.

Give an algorithm to solve the problem. For full-credit, your algorithm should be correct and run in time $O(n \log n)$ and you should bound the run-time of the algorithm. (You don't have to prove correctness.). [4 points]

Create structure that holds: 2 arrays

plurality-rec(A):

- If length of A is 3, or less, brute force add every unique element to array S and assign a count to each element (# occurances). Then create an exact copy of S called S*, return struct that holds both arrays $O(n)$
- Else divide A into thirds and recursively solve each third.

  $x = \text{plurality-rec}(A[0] : A[\lfloor \frac{n}{3} \rfloor])$

  $y = \text{plurality-rec}(A[\lfloor \frac{n}{3} \rfloor + 1] : A[\lfloor \frac{2n}{3} \rfloor])$ $3T(\frac{n}{3})$

  $z = \text{plurality-rec}(A[\lfloor \frac{2n}{3} \rfloor + 1] : A[n])$

- Merge the S arrays of x, y, z, taking only unique elements, into final array B $O(n)$
- Add the counts of similar elements.
- For each element in B if count $> \lfloor \frac{n}{3} \rfloor$ add element to array S* $O(n)$
- return struct containing both B and S*

$T(n) = 3T(\frac{n}{3}) + O(n)$

$\Rightarrow \boxed{O(n \log n)}$

The list of plurality elements is S*. This algorithm basically finds the # of occurances of every element.

4 Problem

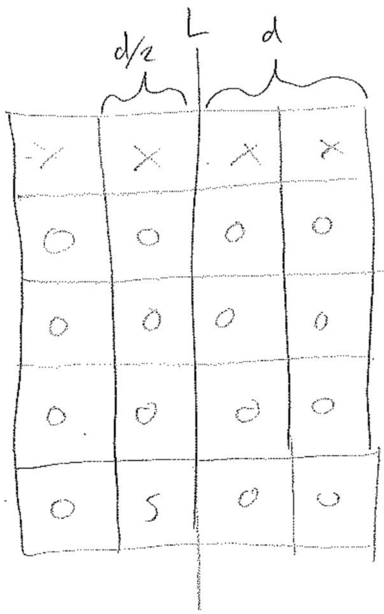
Given a set of points $P = \{p_1, \ldots, p_n\}$ in the plane, give an algorithm for finding a pair of points with the smallest possible L4-distance among the points where L4-distance between two points is defined by $d_4((x, y), (x', y')) = (|x - x'|^4 + |y - y'|^4)^{1/4}$.

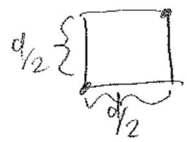
For full-credit your algorithm should be correct and run in time $O(n \log n)$. You don't have to prove correctness or analyze the run-time of the algorithm. You should describe all the steps in the algorithm at a level of detail similar to what was done in class (however, you don't have to describe how to manipulate the sorted lists). [4 points]

- Sort P into P_x (sorted by x-coord) and P_y (sorted by y-coord) using merge-sort

dist (P_x, P_y)
- If size of P is less than or equal to 3, find shortest possible distance using brute force and return that distance.
- Divide P into Q and R of size n/2 horizontally using P_x to find the midpoint.
- Sort Q into Q_x and Q_y and R into R_x and R_y
- recursively find the shortest distance in Q and R
  $q = \text{dist}(Q_x, Q_y)$
  $r = \text{dist}(R_x, R_y)$
- Let $d = \min(q, r)$
- Let the rightmost point of Q be x^*
- Let L be a vertical line going through x^*
- Sort all points d away from L into an array S with the bottommost point first
- For every point in S, compute the distance, d^*, to the next 15 points above it. if $d^* < d$, Let $d = d^*$
- return d

** Proof of why 15 points on back





There can only be atmost 1 point per block. Since points on opposite corners are

$$\sqrt[4]{\left(\frac{d}{2}\right)^4 + \left(\frac{d}{2}\right)^4}$$

$$\sqrt[4]{\frac{2d^4}{32}} = \frac{d}{2} \cdot \sqrt[4]{2} < d$$

For a point in S, any block more than 3 blocks up will be more than d away since $3 \cdot \frac{d}{2} = \frac{3d}{2} > d$

There are a total of 15 o's so You must search at least 15 points up.

5 Problem

Let $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{\{1,2\}, \{1,6\}, \{2,5\}, \{2,6\}, \{3,4\}, \{3,5\}, \{3,6\}, \{4,6\}, \{5,6\}\}$. Suppose that G was given to you in adjacency list representation where the elements in the adjacency list are ordered in increasing order. For example, the adjacency list of vertex 2 would be $[1, 5, 6]$. Run the BFS algorithm on G starting from the vertex 1. It suffices to show the step-by-step evolution of the lists $L[0], L[1], \ldots$ as we described in class. [2 points]

$L[0] = \{1\}$

$L[1] = \{2, 6\}$

$L[2] = \{5, 3, 4\}$

$L[3] = \phi$