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	Kristi Richter
UID	904787968
Section	F

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

$\log n \quad (\log n)^2$

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]

$6T(n/3) + O(n^2) \quad \log_3 6 = 1.\text{something}$

$\log_3 6 < 2 \quad$ case 3

True

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

① split your problem into sub problems (they must be smaller than your original problem)

② solve each subproblem recursively

③ merge the results of your subproblems to get the final answer

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

$k = \log_2 2 = 1 \qquad O(n^1) \qquad$ case 2

$T(n) = (n \log n)$

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.

$a_1 b_1$ = Karatsuba (a_1, b_1)

$a_0 b_0$ = Karatsuba (a_0, b_0)

$a_1 b_0 + a_0 b_1$ = Karatsuba $(a_1 + a_0, b_1 + b_0) - (a_1 \cdot b_1) - (a_0 \cdot b_0) = (a_1 b_0 + a_0 b_1)$

↳ proof $(a_1 + a_0)(b_1 + b_0) = a_1 b_1 + a_0 b_1 + a_1 b_0 + a_0 b_0$

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

List

pros: space efficient $O(|V| + |E|)$

cons: longer access time to traverse neighboring vertices list $O(|E|)$

matrix

pros: fast access $O(1)$

cons: large space $O(|V|^2)$

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

a path is a set of vertices $V = \{V_1 \dots V_n\}$ where the edges $\{(V_1\ V_2)(V_2\ V_3) \dots (V_{n-1}\ V_n)\}$ exist between them.

in other words one can follow the edges between each vertex in the path to get from the start to the end of the path

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]

you can use a breadth first search algorithm which starts at one vertex, checks the list for neighbors and adds them to a queue and marks vertices as discovered along the way. accessing the next item in the queue is $O(1)$ b/c it's an array and then the list is traversed again to check for new neighbors.

Check the other vertexes discovered flag to see if they are connected.

Time complexity is $O(|V| + |E|)$

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 $O(nk \log k)$: $T(1) = O(1)$, $T(k) \leq 2T(k/2) + O(n \cdot k)$.)

$O(K)$ → split the K arrays into 2 groups K_1 and K_2

K_1 has the first $\frac{K}{2}$ elements (arrays) and K_2 has the last $\frac{K}{2}$ elements (arrays)

$T(\frac{K}{2})$ → merge K_1 recursively

$T(\frac{K}{2})$ → merge K_2 recursively

$O(n \cdot K)$ → merge K_1 & K_2 using the technique for merging we saw in class (comparing each element in K_1 & K_2 and constructing the merged list).

traversing n elements for every K arrays $= K \cdot n$

$T(K) \leq 2T(\frac{K}{2}) + O(n \cdot K)$

$= O(nK \log K)$ by case 2 of master theorem

* Base case: if K is 1, return K as the "merged" sorted array

If b is plurality ele of all 3 it's the plurality of A
of 2 → manually count
of 1 → manually count
of none → not plurality element

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]

O(n) → split array A into 3 pieces A_1 A_2 and A_3
A_1 contains the first $n/3$ elements, A_2 the middle, and A_3 the last

$T(\frac{n}{3})$ → recursively return the plurality element(s) of A_1 in an array A_{1P}
(to return multiple maybe return an array A_{1P})

$T(\frac{n}{3})$ → recursively return the plurality element(s) of A_2 (array of PE's is A_{2P})

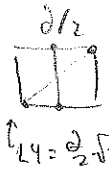
$T(\frac{n}{3})$ → recursively return the plurality element(s) of A_3 (array of PE's is A_{3P})

* each array A can have max 2 plurality elements by definition
this also means A_1 A_2 A_3 can each have max 2 PE's too
there are maximum 6 elements to be checked for plurality

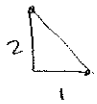
~~for each PE in A_{1P}, if it is in A_{2P} and A_{3P} return it as a PE of A~~
~~remove this element from A_{1P} A_{2P} and A_{3P}~~
~~if it is in either A_{2P} or A_{3P} but not both~~
~~go through the other array and count the times the PE appears~~
~~if it appears $> \frac{n}{3}$ times then return it as a PE~~
~~remove that element from either A_{2P} or A_{3P}~~
~~if it is in neither A_{2P} nor A_{3P}~~
go through both arrays and count #occurrences

O(n) → for each element in A_{1P} A_{2P} and A_{3P} (maximum 6 iterations) ← so it stays O(n)
if PE is in all 3, store it in the PE array → if it's a PE of all 3, it's a PE of the entire array
traverse A_1 A_2 and A_3 and count occurrences to determine
if it's a plurality element ($\geq \frac{n}{3}$ occurrences)
store the element in the PE array if it is
if size(PE array) == 2 (the max # of plurality elements) break;
return the PE array

run time $\leq 3 \cdot T(\frac{n}{3}) + O(n)$ case 2 = $O(n \log n)$



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



2.25 $4+1$ $2^4 + 1^4$ $\sqrt[4]{16+1}$ $\sqrt[4]{32}$ $\sqrt[4]{2^4 \cdot 2}$ $2\sqrt{2}$

$1\sqrt{2}$ $1+1$ $\sqrt[4]{2}$

L_4 ★ actual

$\sqrt[4]{x^4+y^4} < \sqrt[2]{x^2+y^2}$

$\sqrt{x^4+y^4} < x^2+y^2$

$x^4+y^4 < x^4+y^4+2x^2y^2$

L_4 is an underestimate of the actual euclidean distance

$\sqrt{\frac{2\partial^4}{4^4}} = \frac{\partial}{4}\sqrt{2}$

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]

→ base case: if ≤ 3 points in P, use brute force to compute closest pair

→ sort set of points P by x coord (P_x) and y coord (P_y)

→ using P_x create a list of points Q that contains the first $\frac{n}{2}$ points in P_x and R with the last $\frac{n}{2}$ points

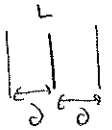
→ recursively find the closest pair of points in each of those lists call them $(q_0^* q_1^*)$ and $(r_0^* r_1^*)$

→ compute the L4 distance between each of these 2 points and take ∂ to be the minimum of the two

$$\partial = \min(L_4(q_0^* q_1^*), L_4(r_0^*, r_1^*))$$

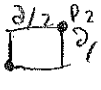
→ make a line L that is the x-coordinate of the highest x-coord point in Q

→ define a region of ∂ on either side of this line and have all the points in this region be placed in set S



→ sort S by y-coord (from P_y)

→ make a grid in the region of squares $\frac{\partial}{2} \times \frac{\partial}{2}$ each square is guarenteed to have at most 1 point b/c L_4 is an underestimate of the actual distance and (see ★ above) in the edge case where points are at the diagonals



$L_4(p_1, p_2) = \partial/2 \cdot \sqrt{2}$ which is $< \partial$

→ There are 4 boxes across the region and given a point s in S, any point s' with $L_4(s, s') < \partial$, s' can only be 3 boxes above s at most (b/c $3 \cdot \frac{\partial}{2} > \partial$), the region we check is ∴ 16 boxes (4×4)

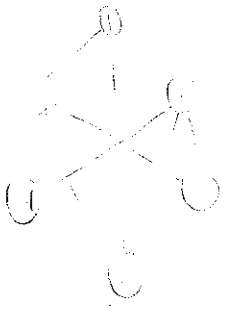
→ Therefore 2 points s and s' will be within 15 spots of eachother in the list S_y

→ compute the $L_4(s,s')$ for these next 15 spots, if any is $<$ min return (s, s') else return the min $(L_4(q_0^*, q_1^*), L_4(r_0^*, r_1^*))$

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]

start = V_1



	Discovered 1	2	3	4	5	6
$L[0] = \{1\}$	T	F	F	F	F	F
$L[1] = \{2, 6\}$	T	T	F	F	F	T
$L[2] = \{5, 3\}$	T	T	T	F	T	T
$L[3] = \{4\}$	T	T	T	T	T	T

$L[4] = \emptyset$

STOP