

# CS180 Exam 1



TOTAL POINTS

**19.5 / 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 relevant
- **3.5 pts** attempt something irrelevant
- **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 2.5 / 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)

✓ - **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

#### 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 1 / 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

# 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





# 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  $O, \Omega, \Theta$ . 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]

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

$f(N) = O(g(N))$   
 if  $\exists$  a constant  $x > 0$   
 s.t.  $f(N)$  eventually stays  
 at most  $c \cdot g(N)$

$$\frac{\log N}{\log 2} \left( \frac{\log N}{\log 100} \right)^2$$

$$x \ll x^2$$



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]

$$T(N) \leq 6T\left(\frac{N}{3}\right) + O(N^2)$$

$$a = 6 \quad b = 3 \quad f(N) = O(N^2)$$

$$\log_3 6 < 2$$

True by case 3 of Master Thm

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

- ① divide a problem into smaller subproblems
- ② recursively solve the subproblems
- ③ combine sub solutions to form overall solution

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

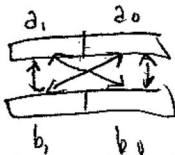
$$a = 2 \quad b = 2 \quad f(N) = 10N$$

$$k = \log_2 2 \quad c = 1$$

=

$N \log N$  by case 2 of Master Thm





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.

TRICK  $\Rightarrow a_0 b_1 + a_1 b_0 = (a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0$  only 1 mult

$a_1 b_1 + a_1 b_0 + a_0 b_1 + a_0 b_0$  2 mults

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

ADJACENCY LIST		ADJACENCY MATRIX	
PROS	CONS	PROS	CONS
$O( V  +  E )$ space complexity	$O(N)$ to find if a given edge exist	$O(1)$ access to edges	$O(N^2)$ space complexity

much better than  $\curvearrowright$

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

$\{v_1, \dots, v_k\}$  a sequence of vertices  $\in V$  that are connected

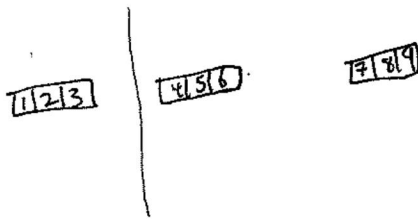
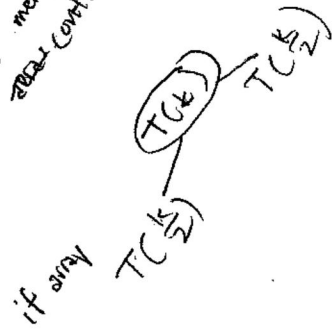
~~with  $\{v_i, v_{i+1}\}$~~

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]

Breadth first search

all vertices end up "T" in DISCOVERED by the end

input:  $k$  sorted arrays of size  $N$   
 output: 1 merged array in sorted order  
 array containing  $k \cdot N$  elements



~~Mergesort~~

Divide arrays in half  
 if 1 array return that array

if 2 arrays input  
 go thru each one  
 comparing  $2N$  elements  
 and copy in ascending  
 order into new array

left call merge Arrays on first  $k/2$  arrays  
 right call merge Arrays on second  $k/2$  arrays

merge (left, right)



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

mergeArrays (A: list of arrays, of size  $k$ ) {

if  $k=1$ , return the single sorted array

left  $\leftarrow$  mergeArrays (first  $k/2$  arrays in A)

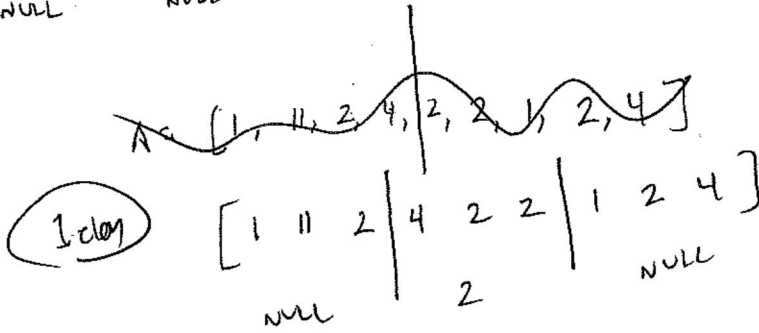
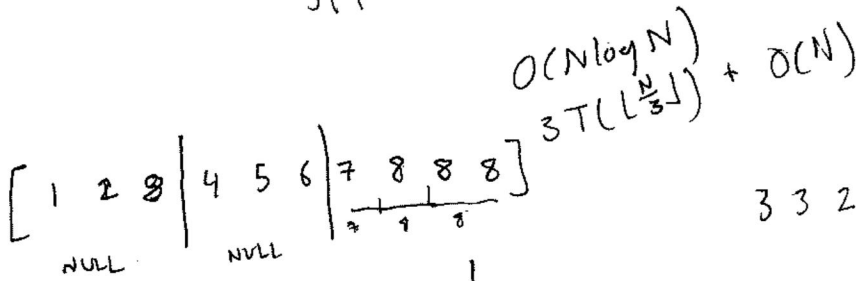
right  $\leftarrow$  mergeArrays (second  $k/2$  arrays in A)

compare each element from left and right, copying into new array B in ascending order

return B

}

$$3 \sqrt[3]{10} \quad 3 \sqrt[3]{4}$$



2 2 2  
1 ..

A [3] has  
0 or 1 plurality elem

**BASE**

if A has < 4 elems  
find if / which plurality elem  
return any (all) plurality elems, or NULL

call findPlurals on  $A(0, \lfloor \frac{N}{3} \rfloor)$  → plurality check 1 if

" " " "  $A(\lfloor \frac{N}{3} \rfloor, \lfloor \frac{2N}{3} \rfloor)$  → plurality check 2

" " " "  $A(\lfloor \frac{2N}{3} \rfloor, \text{end})$  → plurality check 3

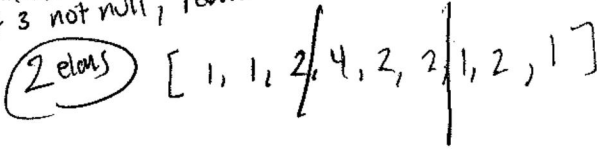
if all 3 null, return null  
if at least 2 out of 3 not null, return

traverse the remaining elems  
returning address to

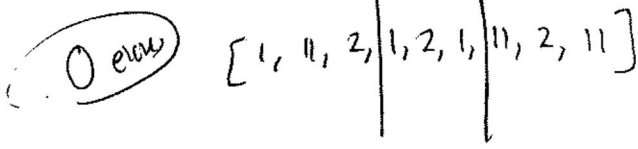
[1, 2, 3]

if A has ≤ 3 elems, brute force  
return plural elem ~~if 1 or 2 elems~~ OR NULL

plurals n.1 = find plurals on first  $\frac{N}{3}$  elems of A  
plurals n.2 = find plurals on second  $\frac{N}{3}$  elems of A  
plurals n.3 = find plurals on third  $\frac{N}{3}$  elems of A



if all null, return null



if at least 1 or more NOT null  
else traverse thru NULL elem  
thirds of A to check if they  
contain plurality element from the  
lucky third

### 3 Problem

Given an array  $A[0, 1, \dots, n-1]$ , an element  $A[i]$  is said to be a *plurality 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]

findPlurals (A) {

—— if A has  $\leq 3$  elements, brute force (either has or does not have a plurality element)  
return the plurality element, how many times it ~~appears~~ occurs / its count  
OR return NULL if none found

——  $p_1 \leftarrow$  findPlurals recursively on first  $\frac{1}{3}$  of A

——  $p_2 \leftarrow$  findPlurals recursively on second  $\frac{1}{3}$  of A

——  $p_3 \leftarrow$  findPlurals recursively on third  $\frac{1}{3}$  of A

Let  $P =$  array of plurality elements initialized to  
~~A p1, p2, p3~~ ~~return NULL~~

—— if count of plurality element <sup>is given</sup> in non-NULL  $P$  call return  
PLUS count of that element found in other two pieces of A  
is greater than  $\frac{\text{size of A}}{3}$ , add to plurality elements array

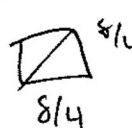
—— return plurality elements array

}

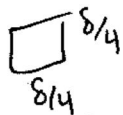
P

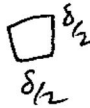
$$\sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2}$$

$$\frac{\delta^2}{4} + \frac{\delta^2}{4} \leq \delta$$

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


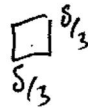
$$\left(\frac{\delta}{4}\right)^4 + \frac{\delta}{2}$$



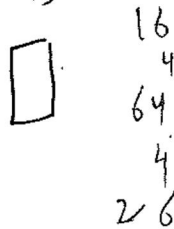


$$\sqrt[4]{\left(\frac{\delta}{4}\right)^4 + \left(\frac{\delta}{4}\right)^4}$$

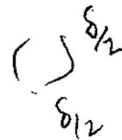
$$\frac{\delta^4}{256} + \frac{\delta^4}{256}$$



$$\sqrt[4]{\frac{2\delta^4}{256}}$$



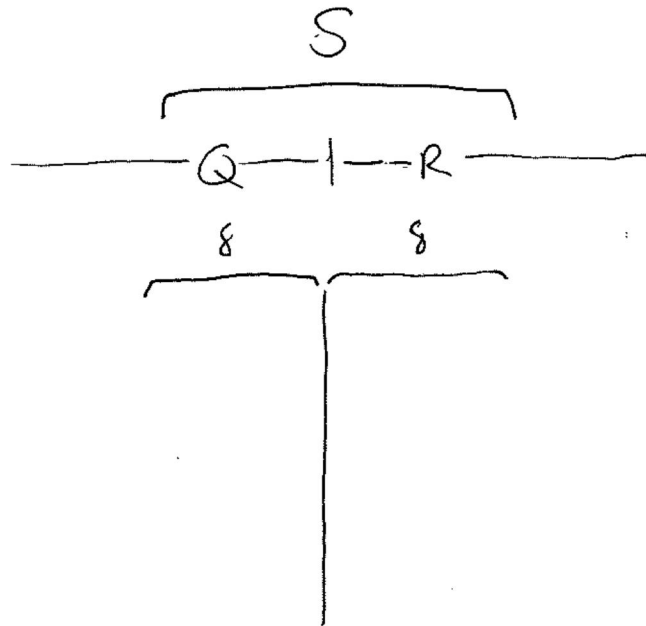
$$\frac{\sqrt[4]{2}\delta}{4}$$



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

$$\sqrt[4]{\frac{2\delta^4}{16}}$$

$$\frac{\sqrt[4]{2}\delta}{2}$$



$$d((x,y), (x',y')) = \sqrt{(x-x')^2 + (y-y')^2}$$

$$d_2((x,y), (x',y')) = |x-x'| + |y-y'|$$

$$d_4((x,y), (x',y')) = (|x-x'|^4 + |y-y'|^4)^{1/4}$$

$$d_4 = \sqrt[4]{(x-x')^4 + (y-y')^4}$$

#### 4 Problem

Given a set of points  $P = \{p_1, \dots, p_n\}$  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  $d_4((x,y), (x',y')) = (|x-x'|^4 + |y-y'|^4)^{1/4}$ .

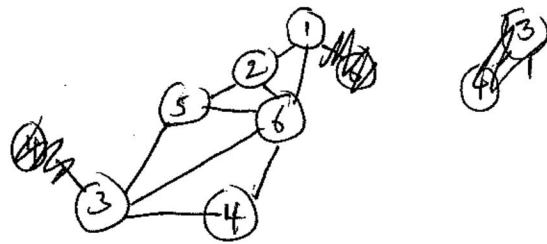
For full-credit your algorithm should be correct and run in time  $O(n \log n)$ . You don't have to prove correctness or analyze the run-time of the algorithm. You should describe all the steps in the algorithm at a level of detail similar to what was done in class (however, you don't have to describe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

- WORK DONE PRIOR TO RECURSION
- construct sets  $P_x, P_y$  where  $P_x$  is  $P$  sorted by x-coord &  $P_y$  is  $P$  sorted by y-coord
  - closest-pair-rec( $P_x, P_y$ )
  - if  $|P| \leq 3$ , brute force find closest pair in  $P$
  - construct  $Q, R$  where  $Q$  is the first half of  $P_x$  and  $R$  is the second half
  - construct  $Q_x, Q_y$  using  $Q$  and  $P_y$
  - construct  $R_x, R_y$  using  $R$  and  $P_y$
  - store closest-pair-rec( $Q_x, Q_y$ ) in  $(q_0^*, q_1^*)$
  - store closest-pair-rec( $R_x, R_y$ ) in  $(r_0^*, r_1^*)$
  - let  $\delta = \min(L4dist(q_0^*, q_1^*), L4dist(r_0^*, r_1^*))$
  - let  $S =$  set of all points at most  $\delta$  distance away from  $L = \{x \in P : x^* \leq x^*\}$  where  $x^*$  is the max x-value in  $Q_x$
  - construct  $S_y$  using  $P$  where  $S_y$  is  $S$  sorted according to y-coordinate
  - Traverse through  $S_y$ , checking if L4distance between each point in  $S_y$  and the next 15 points is less than  $\delta$  (if so, return those two points)
  - else if  $(q_0^*, q_1^*) = \delta$  return this pair of points
  - else return  $(r_0^*, r_1^*)$

NOTE  
 The Lemma proven in class with Euclidean distance should still apply to L4-distance.  
 We can still split up the squares in the strip,  $S$ , by creating  $\delta/2$  by  $\delta/2$  boxes

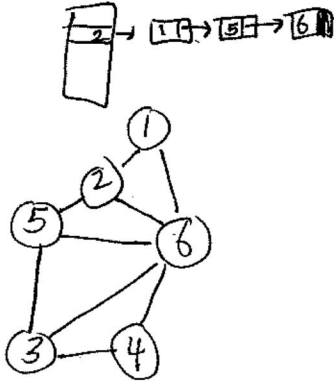
$$\# \text{ of points in box} \leq \frac{4\sqrt{2}\delta}{2} < \frac{\delta}{2}$$





### 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  $L[0], L[1], \dots$  as we described in class. [2 points]



	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
$L[2] = [6, 5]$	T	T	F	F	T	T
$L[3] = [3, 5, 3, 4]$	T	T	T	T	T	T
$L[4] = [3, 4]$	↓	↓	↓	↓	↓	↓
$L[5] = [4]$	↓	↓	↓	↓	↓	↓
$L[6] = [ ]$	↓	↓	↓	↓	↓	↓
ALGO STOPS	_____					

