CS180 Exam 1

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	
UID	
Section	

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]

$T(n) \le 6T(n/3) + O(n^2)$ $k = \log_3 6$ $3^x = 6$ ✓

True

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

// base case
// split up
// merge

1. Split up the input
2. Solve the split up parts recursively ✓
3. Combine the parts

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

$T(n) \le 2T(n/2) + O(n)$ $k = \log_2 2$

$2^x = 2$ $x = 1$

$O(n \log n)$

$(a_1+b_0)(a_0+b_1) = a_1a_0 + a_1b_1 + b_0a_0 +$

$(a_1+a_0)(b_1+b_0) = a_1b_1 + a_1b_0 + a_0b_1 + a_0b_0$

5. ~~Let a_0, a_1, b_0, b_1 be four integers that are k bits long. Write d~~own 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.

✓

Karatsuba's trick only uses three multiplications

original: $2^n(a_1 \cdot b_1) + 2^{n/2}(a_1b_0 + a_0b_1) + a_0b_0$ — 4 multiplications in original!

Karatsuba: $2^n(a_1 \cdot b_1) + 2^{n/2}\left((a_1+a_0)(b_1+b_0) - a_1b_1 - a_0b_0\right) + a_0b_0$

since $a_1b_0 + a_0b_1 = (a_1+a_0)(b_1+b_0) - a_1b_1 - a_0b_0$

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

✓

Adjacency List: Pros: Saves Space. Can represent in $O(|V|+|E|)$

Cons: To see if two vertices are connected need to traverse through degree(v) which is worse than $O(1)$

Adjacency Matrix: Pros: To see if an edge exists between vertices takes $O(1)$

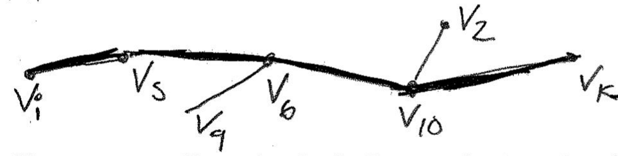
Cons: Takes $O(|V|^2)$ to represent relationships between vertices.

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

Words: Path in a graph is a connection of edges $(v_1, \dots v_k)$ that connects two vertices v_1 and v_k.

✓

Drawing:



— Path from v_1 to v_k

If there are k vertices in the path, at least k-1 edges

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]

We can check if a graph given in adjacency list representation is connected by starting at an arbitrary vertex, and setting vertices as discovered if they are visited by the BREADTH FIRST SEARCH (algorithm that we did in class)

After, we would go through the list of vertices again and if any vertex is NOT marked as discovered, the graph is NOT connected. Or else it is!

$\log_2 2 = 1$

n^1

$O(nk\log k)$

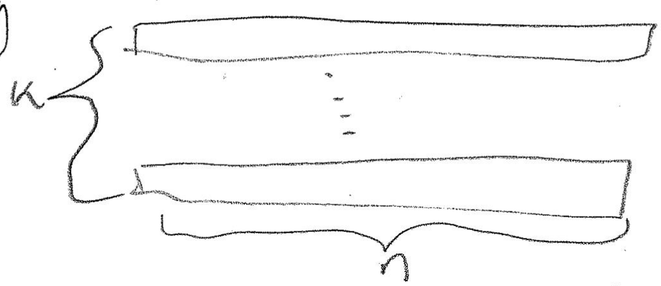
$T(k) \le 2T\left(\frac{k}{2}\right) + O(n \cdot k)$

2 Problem — Runtime Complexity analysis: $O(1) + T(k/2) + T(k/2) + O(1) + O(k \cdot n)$ $= 2T(k/2) + O(kn)$ works!

✓ 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 $O(nk \log k)$: $T(1) = O(1)$, $T(k) \le 2T(k/2) + O(n \cdot k)$.) ← base case!

This algorithm would follow Case 2 of the Masters Theorem.



12345
678910
1112131415

merging one of them would be $O(n)$

$O(n^2)$

```
def merge arrays(k sorted arrays with n numbers).
    if k equals 1:
        return k          // it is sorted                } O(1)
    first_half_of_k_sorted = mergearrays(⌈k/2⌉ arrays)   → T(k/2)
    last_half_of_k_sorted = mergearrays(⌊k/2⌋ arrays)    → T(k/2)
initialization {
    final_array = [ ]
    i=0                                                  } O(1)
    j=0
    while i is NOT equal to the length of first_half_of_k_sorted      O(k·n) merging
        OR j is NOT equal to the length of first_half_of_k_sorted:
        if first_half_of_k_sorted[i] ≤ last_half_of_k_sorted[j]:
            final_array.append(first_half_of_k_sorted[i])
        else
            final_array.append(last_half_of_k_sorted[j])
    Go through and add remaining elements (could be none!) of first_half_of_k_sorted to final_array
    Go through and add remaining elements of last_half_of_k_sorted to final_array
                                  (could be none!)
    return final_array
```

Peurality elerent = if more than n/3 eqial elements of A
majority element = if more than n/2 equal elements of A
(in homework)

3 Problem

KEY: There can ONLY BE AT MOST TWO plurality elements

Given an array $A[0, 1, \ldots, n-1]$, 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. 4/9 4/9

4/9

all 3/9

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(nlogn) comes from $3T(n/3) + O(n)$ → $n^{\log_3 3}$ O(nlogn)

```
def findPluralityElement(array A):
    if length of A < 5:   ← constant
        brute force return all plurality elements (max 2)    } all that really matters is this is constant O(1)
    plurality_elements = []                                   → O(1)
    A1 = findPluralityElement(bottom ⌊n/3⌋ of array A)        → T(n/3)
    A2 = findPluralityElement(middle ⌈n/3⌉ of array A)        → T(n/3)
    A3 = findPluralityElement(top ⌈n/3⌉ of array A)           → T(n/3)
    // A1, A2, A3 can each return 1, 2, or no plurality elements in that array
    if (A1 == no plurality [] AND A2 == no plurality [] AND A3 == no plurality []):
        return no plurality ([]) ← empty list                 → O(1)
O(1) if (A1's plurality elements == A2's plurality elements == A3's plurality elements)
        return A1's plurality elements   ← arbitrary A1, A2, A3
O(1) test plurality = A1's plurality and A2's plurality and A3's plurality   ← length this at most 6
O(1) if test plurality length is 1:
        return test plurality
O(n) for i in A:
        keep count of each plurality in test plurality
O(1)  for i in test plurality:          → keep in mind at most 6!
O(1)     if count(i)/n > 1/3
            plurality_elements.append(i)
    return plurality_elements
```

$T(n) \leq 3T(n/3) + O(n)$

1. 3 recursive calls
2. split into 3

\# lets say we find the plurality element for 3 sub arrays. Now need to find answer in O(n)

There are many different cases to consider:

A1 A2 A3

1. all of them don't have ~~n/3~~
2. 1 of them has n/3 → count
3. 2 of them has n/3 → do two counts
4. ~~all of them have plurality element~~

By master theorem case 2, O(nlogn)

$P = \{P_1, \ldots P_n\}$

$2T(n/2) + O(n)$ $\left(|x-x'|^4 + |y-y'|^4\right)^{1/4} = \sqrt[4]{|x-x'|^4 + |y-y'|^4}$

4 Problem

want $O(n \log n)$ BIG KEY: WE DONT HAVE TO PROVE CORRECTNESS → meaning we don't need to prove the main lemma!

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]

def setup:
  Make P_x: P sorted by x coordinate → $O(n \log n)$
  Make P_y: P sorted by y coordinate → $O(n \log n)$
  find L4 Distance(P_x, P_y) → $T(n)$

def find L4Distance(P_x, P_y): → $T(n)$
  If number of points < (5) ← constant factor } $O(1)$ - constant factor
    brute force search smallest L4 distance
  Q_x = first half of P_x inclusive $\lfloor P_x/2 \rfloor$ → ~~T(n/2)~~ $O(n)$
  Q_y = first half of P_y inclusive $\lfloor P_y/2 \rfloor$
  R_x = second half of P_x inc. $\lceil P_x/2 \rceil$ → ~~T(n/2)~~ $O(n)$
  R_y = second half of P_y " $\lceil P_y/2 \rceil$
  (q_1^*, q_2^*) = find L4Distance(Q_x, Q_y) // returns 2 points → $T(n/2)$
  (r_1^*, r_2^*) = find L4 Distance(R_x, R_y) // returns 2 points → $T(n/2)$
  ← now, need to "combine" the two → $O(1)$
  Set $\delta = \min\left(d_4((q_{1x}^*, q_{1y}^*), (q_{2x}^*, q_{2y}^*)), d_4((r_{1x}^*, r_{1y}^*), (r_{2x}^*, r_{2y}^*))\right)$
  X^* = biggest x coordinate in Q_x → basically last element in Q_x → $O(1)$
  L = line made from $X = X^*$ equation $O(1)$
  S_y ~~t~~ all the points δ away from L! $O(1)$
  for point(i,j) in S_y: → $O(n)$
    check i,j with the next ~~1279~~ points (if possible) and if the distance between one of those comparisons is less than δ, return that point $O(1)$
  if $d_4((q_{1x}^*, q_{1y}^*), (q_{2x}^*, q_{2y}^*)) < d_4((r_{1x}^*, r_{1y}^*), (r_{2x}^*, r_{2y}^*))$: $O(1)$
    return (q_1^*, q_2^*)
else

5 Problem

9

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]

Input:

1 → 2 → 6 → ∅
2 → 1 → 5 → 6 → ∅
3 → 4 → 5 → 6 → ∅
4 → 3 → 6 → ∅
5 → 2 → 3 → 6 → ∅
6 → 1 → 2 → 3 → 4 → 5 → ∅

9×2

L[0] = [1]
L[1] = [2, 6]
L[2] = [5, 3, 4]
L[3] = [ ] //STOP!