

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	Weijia Shi
UID	104757423
Section	1A

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

→

$$y = \log_2 n \quad z = n$$

$$z = (\log_{100} n)^2 \cdot \sqrt{z} = \log_{100} n$$

$$100^{\sqrt{z}} = n$$

$k = \log_2 n$

$$m = (\log_{100} n)^2 = \frac{\log_2 n}{\log_2 100} = \frac{\log_2 n}{\log_2 10^2} = \frac{\log_2 n}{2 \log_2 10} = \frac{\log_2 n}{2 \cdot 3.32} \approx \frac{\log_2 n}{6.64}$$

$100^{(\log_{100} n)^2} = \log_{100} n$

$n^{\log_{100} n}$

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]

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

$$k = \log_3 6 < 2$$

$$O(n^2)$$

True

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

① Divide a problem into subproblems

② solve the subproblems

③ combine the solution of subproblems to get the final solution

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

$$k = \log_2 2 = 1$$

case 2: $T(n) = O(n \log n)$

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.

$$\begin{aligned}
 a &: a_0 a_1 & a_0 b_1 + a_1 b_0 &= (a_0 + a_1)(b_0 + b_1) \\
 b &: b_0 b_1 & &= \underline{a_0 b_0} + \underline{a_0 b_1} + \underline{a_1 b_0} + \underline{a_1 b_1} \\
 a &= 2^{\frac{k}{2}} a_0 + a_1 & a \cdot b &= 2^k a_0 b_0 + a_1 b_1 + 2^{\frac{k}{2}} (a_0 b_1 + a_1 b_0) - a_0 b_0 - a_1 b_1 \\
 b &= 2^{\frac{k}{2}} b_0 + b_1 & &
 \end{aligned}$$

$$a \cdot b = 2^k a_0 b_0 + a_1 b_1 + 2^{\frac{k}{2}} (a_0 b_1 + a_1 b_0)$$

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

Adjacency list:

pro: Storage efficient $O(|E| + |V|)$

con: determine if an edge exists between 2 nodes need $O(V)$ time

Adjacency matrix:

pro: quick retrieval of edge $O(1)$ time

con: storage takes $O(V^2)$

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

path: is a sequence of vertices

$(v_1, v_2, \dots, v_{k-1}, v_k)$ such that:

for all $v_i, \dots, v_k \in V$

for any $i \in \{1, \dots, k-1\}, (v_i, v_{i+1}) \in E$

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]

Use breadth-first search

if after breadth-first search, all vertices in graph are

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

- Base case:
if A has only 1 sorted array
return A

A is k sorted array
Merge-recur(A)
- Divide the input of k sorted arrays into 2 groups

$O(nk)$ {
 $A_1 = \text{first } \lfloor \frac{k}{2} \rfloor \text{ sorted array}$
 $A_2 = \text{the rest sorted array}$

$T(\frac{k}{2})$ - $B_1 = \text{Merge-recur}(A_1)$

$T(\frac{k}{2})$ - $B_2 = \text{Merge-recur}(A_2)$

- Merge: $k_1 = 0$, $k_2 = 1$, Array $q = \emptyset$
for i in range $1 : \frac{k}{2}$
if $A_1[k_1] < A_2[k_2]$
Add $A_1[k_1]$ to q
 $k_1 = k_1 + 1$
else
Add $A_2[k_2]$ to q
 $k_2 = k_2 + 1$
end if
return q

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]

$\geq \lfloor n/3 \rfloor$
 find-plura (A)
 - Base case:
 If $|A| = 1$
 return True, A
 - Divide A into 3 groups
 $A_1 =$ first $\lfloor n/3 \rfloor$ elements in A
 $A_2 =$ second $\lfloor n/3 \rfloor$ elements in A
 $A_3 =$ the rest elements in A.
 $O(n)$ {
 $A_1 =$ find-plura(A_1)
 $A_2 =$ find-plura(A_2)
 $A_3 =$ find-plura(A_3)
 put a_1, a_2, a_3 into a new array g
 - Declare an array c of size $|g|$ to store the counts for element e in A
 for elements i in range $0 \sim |g|-1$
 if $(e == g[i])$
 $c[i] = c[i] + 1$
 $i = i + 1$
 End for
 end for

we know for any array A
 A can have at most 2 plurality elements:
 Proof: if A has more than 2 plurality elements a, b, c
 a has $> \lfloor n/3 \rfloor$ element
 b has $> \lfloor n/3 \rfloor$ element
 $|a| + |b| > \lfloor \frac{2n}{3} \rfloor$
 $|c| < \frac{1}{3}n$
 Therefore c is not a plurality element.

the largest size of g is 6.

next page

ans 10:

Out of for loop

```
O(n) } <= b {
    Define output as an array
    output = {}
    for i in 0 ~ size(k) - 1
        if c[i] > floor(n/3)
            add 'g[i]' to output
    end for
```



$$T(n) = 3T(n/3) + O(n)$$

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

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]

Sort P based on x : P_x

Sort P based on y : P_y

find - closest (P_x, P_y)

- Base case:

if $|P| \leq 3$

Brute force search the closest pair
return the distance

- Divide P_x into Q, R (Q has first $\lfloor \frac{n}{2} \rfloor$ elements, R has the rest)

- Sort Q to Q_x, Q_y

Sort R to R_x, R_y

- $d_1 = \text{find - closest}(Q_x, Q_y)$

$d_2 = \text{find - closest}(R_x, R_y)$

- Let x^* = the point with largest x in Q_x

- $L = \{(x^*, y)\}$

- for point S in P

if X of S within δ from x^*

- sort L to L_y

- declare $\delta = \infty$

- for point L in L

calculate distance d_3

(which is the minimum distance of point L to next 19 points in

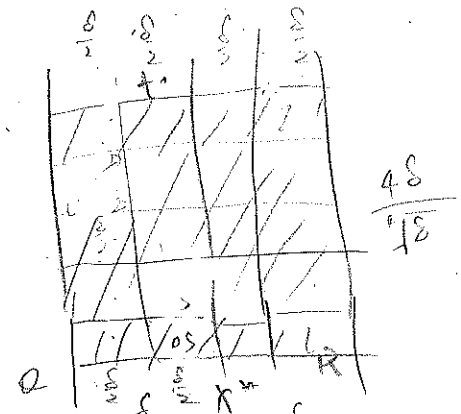
L_y)

if $d_3 < \delta$

$\delta = d_3$

end for

next pair \rightarrow



Observ: there are no 2 points in the same block

d_{\max} - dist in one block

$$\textcircled{1} \leq \sqrt[4]{\left(\frac{\delta}{2}\right)^4} = \frac{\delta}{2}$$

$$\textcircled{2} < \sqrt[4]{\left(\frac{\delta}{2}\right)^4 + \left(\frac{\delta}{2}\right)^4} < \frac{\delta}{\sqrt[4]{8}} < \delta$$

Since $\frac{\delta}{\sqrt[4]{8}} < \delta$ no 2 points in 1 block

Observe the points we need to check with 19 points in P_y

Proof: $\delta_{S \rightarrow T} > \delta$

next page \rightarrow

```
if ( d1 < d2 )
```

```
    d = d1
```

```
else
```

```
    d = d2
```

```
if ( d < S )
```

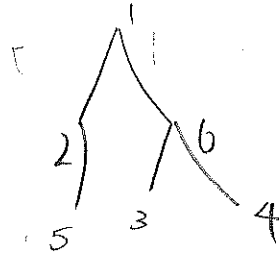
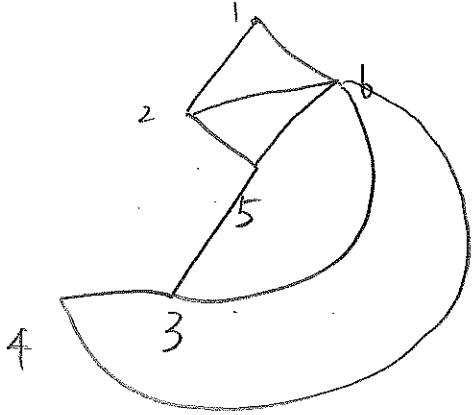
```
    return d
```

```
else
```

```
    return S
```

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]



$L[i]$	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
$L[2] = 5, 3, 4$	T	T	T	T	T	T
$L[3] = \emptyset$						

