

# CS180 Exam 1



TOTAL POINTS

**21.9 / 26**

QUESTION 1

Problem 1 10 pts

1.1 Part 1: Asymptotics 1 / 1

✓ - 0 4/6 correct

1.2 Part 2: D&C Master 1 / 1

✓ - 0 Correct answer

1.3 Part 3: D&C Principles 1 / 1

✓ - 0 Correct principle. You mentioned 'divide/split' into subproblems, and combine their solutions.

1.4 Part 4: Recurrence 0.5 / 1

✓ - 0.5 Incorrect answer, you probably used the wrong case from the master's theorem, or didn't use the master's theorem at all.

1.5 Part 5: DFT Definition 1 / 1

✓ - 0 Correct expression, showing the value representation of the polynomial or the matrix representation of DFT calculation

1.6 Part 6: List vs Matrix 0.9 / 1

✓ - 0.1 Correct, answer, but without exact complexities for access and storage

- ☛ Provide the storage/space complexities in terms of  $V$  and  $E$ , instead of a general  $n$  variable.

1.7 Part 7: Dijkstra changing weights 2 / 2

✓ - 0 Correct answer, with an explanation as to why the shortest distances don't change

1.8 Part 8: Modified Dijkstra 1.5 / 2

✓ - 0.5 Correct answer with an invalid example

QUESTION 2

Problem 2 4 pts

2.1 Pattern in matrix Q. 1.25 / 2

✓ - 0.75 Partially correct

2.2 Relation to  $Q_{\lfloor n/2 \rfloor}$  2 / 2

✓ - 0 blank

QUESTION 3

3 Problem 3: Finding majority 3.25 / 4

✓ - 0.75 Unclear/minor error with conquer step

QUESTION 4

Problem 4 4 pts

4.1 BFS Tree 1.5 / 2

✓ - 0.5 Couple of links wrong.

4.2 DFS Tree 1 / 2

✓ - 1 Some ordering intact

QUESTION 5

5 Problem 5 4 / 4

✓ - 0 Correct

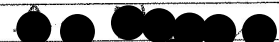

# Mid-term. February 3, 2017

CS180: Algorithms and Complexity  
Winter 2017

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 we proved in class without proofs 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 try to 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.

Problem	Points	Maximum
1		10
2		4
3		4
4		4
5		4
Total		26

Name	
UID	
Section	1A



# 1 Problem

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

- 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]

$f$	$g$	$O$	$\Omega$	$\Theta$
$\log_3 n$	$\log_9 n$	N	Y	N
$3^n$	$6^n$	<del>Y</del>	<del>Y</del>	<del>Y</del>
		Y	N	N

$$\lim_{n \rightarrow \infty} \frac{3^n}{6^n} = \lim_{n \rightarrow \infty} \frac{1}{2^n} = 0 \quad 3^n < 6^n$$

$$\lim_{n \rightarrow \infty} \frac{\log_3 n}{\log_9 n} = \lim_{n \rightarrow \infty} \frac{\log n / \log 3}{\log n / \log 3} = \frac{\log 3}{\log 3} = 1 \Rightarrow \log_3 n = 2 \log_9 n$$

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

$$T(n) = 5T(\lfloor n/2 \rfloor) + O(n^2); \quad a = 5, b = 2, k = \log_2 5 > 2$$

Case 1:  $T(n) = O(n^k) = O(n^{\log_2 5})$

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

- Divide the problem into subproblems
- Solve subproblems recursively.
- Combine the solutions of subproblems.

- What is the solution to the recurrence  $T(1) = 1, T(n) = 2T(n/2) + 100n$ ? [1 point]

$$T(1) = 1; T(2) = 2T(1) + 100 \cdot 2 = 2 + 100 \cdot 2$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 100n$$

$$2\left(2T\left(\frac{n}{2^2}\right) + 100\frac{n}{2}\right) + 100n$$

$$2^2 T\left(\frac{n}{2^2}\right) + 2 \cdot 100n$$



5. Write down the definition of the discrete Fourier Transform for signals of length  $n$  (i.e., write down the expression we used for  $DFT_n(a_0, a_1, \dots, a_{n-1})$ ). [1 point]

$DFT_n(a_0, a_1, \dots, a_{n-1}) = (S_0, S_1, \dots, S_{n-1})$  such that

$$S_i = \sum_{j=0}^{n-1} \left( e^{i \frac{2\pi}{n} jk} \right) a_j$$

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

Adjacency list

Pros: Save space in memory, only store existing edges.

Cons: Take  $O(n)$  time to check existence of an edge between 2 vertices.

Adjacency matrix

Pros: checking existence of an edge is very fast:  $O(1)$ .

Cons: space complexity is  $|V|^2$

7. Let  $G$  be a weighted directed graph with positive weights. Suppose we ran Dijkstra's algorithm starting from a vertex  $s$  to compute the shortest distances from  $s$  to all vertices in  $G$  and let  $T$  be the tree formed by the PARENT links that are computed during the run of the algorithm.

Now suppose we change the weights of the graph as follows: for every edge  $e$  that is **not** part of  $T$ , its weight is doubled, i.e., its weight  $w_e$  is replaced with  $2 \cdot w_e$ . The weights of edges in  $T$  are not changed. This creates a new instance  $G'$  of the problem with the same underlying graph but different costs on all the edges which are **not part of  $T$** .

True or false: "The shortest distances from  $s$  to other vertices in the new instance are the same as they were in the original weighted graph." If true, provide a brief explanation why and if false, provide an example of a graph  $G$  where the statement fails. [2 points]

True.

The weight of each path from  $s$  to ~~all vertices~~ <sup>each vertex  $u$</sup>  in  $G$  in  $T$  is the shortest path from  $s$  to that vertex (by property of Dijkstra's algorithm).

$$\Rightarrow d(s, u) < d(s, u) \text{ that includes } w_e$$

and  $\cancel{w_e} < 2w_e$

$$\Rightarrow d(s, u) < d(s, u) \text{ that includes } 2w_e$$

$\Rightarrow$  The given statement is true.



8. Let  $G = (V, E)$  be a weighted undirected graph with positive weights and let  $s$  be a vertex in  $G$ . Consider a variant of Dijkstra's algorithm where we grow the set of vertices  $S$  by picking the vertex  $v \notin S$  that has the shortest edge to any vertex in  $S$ . That is, consider the following algorithm:

(a) Set  $S = \{s\}$ . Set  $c(s) = 0$  and  $c(v) = \infty$  for all  $v \neq s$ .

(b) While  $S \neq V$ :

i. For each vertex  $v \notin S$ , let  $c'(v) = \min\{\ell_{(u,v)} : u \in S, (u,v) \in E\}$ .

ii. Find the vertex  $v \notin S$  with least  $c'(v)$  and let  $u \in S$  be the corresponding vertex that achieves the minimum in the definition of  $c'(v)$ .

iii. Set  $c(v) = c(u) + \ell_{(u,v)}$ .

*iv. You add  $v$  to  $S$*   
Does the above algorithm compute the lengths of the shortest paths from  $s$  to all other vertices? That is, are the numbers  $c(v)$  computed by the algorithm the distances to  $v$  from  $s$ ? If yes, provide a brief explanation why this may be true. If not, provide an example of a graph  $G$  where the algorithm fails to compute the lengths of the shortest paths. [2 points]



False





## 2 Problem

Let  $n$  be an even integer and let  $Q_n$  denote the  $n \times n$  matrix with rows and columns indexed by  $0 \leq j, k \leq n-1$  and  $Q_n[j, k] = e^{-2\pi i(j \cdot k)/n}$ .

1. Can you identify any repeating pattern in the matrix  $Q_n$  (like the one we saw in our derivation of FFT for computing the discrete Fourier Transform)? [2 points]
2. Can you connect the matrices in the pattern to the matrix  $Q_{n/2}$ ? [2 points]

Let  $w = e^{-2\pi i/n}$

	0	1	2	3	4	...	k	...	n-1
0	1	1	1	1	1	...	1	...	1
1	1	w	w <sup>2</sup>	w <sup>3</sup>	w <sup>4</sup>	...	w <sup>k</sup>	...	w <sup>n-1</sup>
2	1	w <sup>2</sup>	w <sup>4</sup>	w <sup>6</sup>					
3	1	w <sup>3</sup>							
j	1						w <sup>jk</sup>		
n-1	1	w <sup>n-1</sup>							

1) If we let  $w = e^{-2\pi i/n}$ , we have the same pattern as we saw in our derivation of FFT.

2)



### 3 Problem

An array  $A[0, 1, \dots, n-1]$  is said to have a *majority element* if more than half of its elements are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. 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 run in time  $O(n \log n)$ . (You don't have to prove correctness or analyze the time-complexity of the algorithm.) [4 points]

(Hint: Split the array  $A$  into two arrays  $A_L$  and  $A_R$  of half the size each. Does knowing the majority elements of  $A_L$  and  $A_R$  help you figure out the majority element of  $A$ ? If so, you can use a divide-and-conquer approach.)

```
findMajority (array A)
  if size(A) = 2 : if A[0] = A[1] : return A[0] else return "no majority element"
  A_L = A[0 : n/2 - 1]; A_R = A[n/2 : n - 1]
  x = findMajority (A_L)
  y = findMajority (A_R)
  if A_L & A_R have majority elements:
    if x = y:
      return (x) "is the majority element"
    else: return "no majority element"
  if A_L and A_R have no majority element:
    return "no majority element"
  if A_L has a majority and A_R has no majority element:
    count = 0;
    for each element in A:
      if A[i] = x:
        count = count + 1
    if count > floor(n/2):
      return x "is the majority element"
  if A_R has a majority and A_L has no majority element:
    count = 0;
    for each element in A:
      if A[i] = y:
        count = count + 1
    if count > floor(n/2):
      return y "is the majority element"
```



$1 - 2, 3, 4, 5, 6$   
 $2 - 1, 5, 6$   
 $3 - 4, 5, 6$   
 $4 - 3$   
 $5 - 2, 3$   
 $6 - 1, 2, 3$

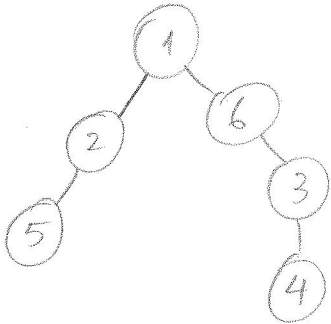
### 4 Problem

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  $[1, 5, 6]$ .

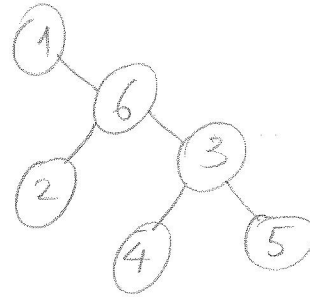
1. Draw the BFS tree that you would get when doing BFS starting from 1. [2 points]
2. Draw the DFS tree that you would get when doing DFS starting from 1. [2 points]

(You don't have to show all the stages of the algorithms just the final trees. Also, keep in mind that you process elements of the adjacency list in increasing order. For example, when doing DFS, you push vertices from an adjacency list onto the stack in increasing order.)

BFS:



DFS:





## 5 Problem

Let  $G = (V, E)$  be an undirected (unweighted) graph and for any two vertices  $u, v \in G$ , let  $distance_G(u, v)$  be the length of the shortest path between  $u, v$  if one exists and  $\infty$  if they are not connected. Define the diameter of a graph  $G$  to be the maximum distance between any two vertices of the graph  $G$ ; that is,  $diameter(G) = \max\{distance_G(u, v) : u, v \in G\}$ . Give an algorithm that given a graph  $G = (V, E)$  (in adjacency list representation) as input, computes the diameter of  $G$ ,  $diameter(G)$ . For full-credit, your algorithm should run in time  $O(|V|^2 + |V| \cdot |E|)$ . [4 points]

Use BFS

(You don't have to prove correctness or analyze the time-complexity of your algorithm.)

Set  $S = \{s\}$ ,  $d(s) = 0$ ,  $d(v) = \infty$  for all  $v \neq s$ ,  $max = d(s)$

While  $S \neq V$ :

for each vertex  $v \notin S$ :

$$d'(v) = \min_{\{u,v\} \in E} (d(u) + l_{(u,v)})$$

find vertex  $w \notin S$  with least  $d'(w)$  and let  $u \in S$  be the corresponding vertex:

add  $w$  to  $S$

set  $d(w) = d'(w)$

add  $e = \{u, w\}$  to  $S$

---

DISCOVERED  $[u] = \text{false}$   $\forall u \in V$ , Tree  $T = \text{empty}$   
DISCOVERED  $[s] = \text{true}$ , Tree  $T \leftarrow s$ ,  $max = 0$   
 $L[0] \leftarrow s$ ,  $i = 0$

While  $L[i]$  is not empty:

$L[i+1] \leftarrow \emptyset$

for each  $u \in L[i]$ :

for each neighbor  $w$  of  $u$ :

if  $\neg$  DISCOVERED  $[w] = \text{false}$ :

set DISCOVERED  $[w] = \text{true}$

add  $e = \{u, w\}$  to Tree  $T$ .

add  $w \leftarrow L[i+1]$

$i \leftarrow i+1$

For each branch of  $s$  in tree  $T$ :

compute  $d = \sum$  all edges of the branch.

if  $max < d$ :  $max = d$ .

return  $max$ .



