CS180 Midterm

DOAN, HARRIS

TOTAL POINTS

75 / 100

QUESTION 1

1 Problem 1 18 / 20

- **0 pts** Correct
- **1 pts** (a) Incomplete about stable matching problem statement or no valid solution.
- **2 pts** (a) missing

✓ **- 2 pts** *(b) lack of enough justify/explanation of time complexity.*

- **2 pts** (b) need to explain all data-structure used.
- **4 pts** (b) miss/wrong runtime complexity
- **8 pts** (b) missing
- **2 pts** (c) partial prove
- **4 pts** (c) not a complete prove
- **10 pts** (c) missing

QUESTION 2

2 Problem 2 10 / 20

- **0 pts** Correct
- **2 pts** 1 order wrong
- **4 pts** 2 order wrong
- **6 pts** 3 order wrong
- **8 pts** 4 order wrong

✓ **- 10 pts** *5 order wrong*

- **12 pts** all order error.
- **4 pts** wrong and without explaination.
- **20 pts** missing

QUESTION 3

3 Problem 3 16 / 20

- **0 pts** Correct

✓ **- 1 pts** *(a) Minor mistake (i.e. wrote ABDCEFG instead)*

- **2 pts** (a) Major mistake but showed some correct work
- **3 pts** (a) Major mistake with no work shown
- **5 pts** (a) No attempt

✓ **- 1 pts** *(b) Minor mistake (i.e. correct distances, wrong order)*

- **2 pts** (b) Major mistake but showed some correct work
- **3 pts** (b) Major mistake with no work shown
- **5 pts** (b) No attempt

✓ **- 1 pts** *(c) Minor mistake*

- **2 pts** (c) Major mistake but showed some correct work
- **3 pts** (c) Major mistake with no work shown
- **5 pts** (c) No attempt

✓ **- 1 pts** *(d) Minor mistake*

- **2 pts** (d) Major mistake but showed some correct work
- **3 pts** (d) Major mistake with no work shown
- **5 pts** (d) No attempt

QUESTION 4

4 Problem 4 14 / 20

- **0 pts** Correct
- **1 pts** (a) Minor mistake (i.e. wrong time complexity analysis)
- **2 pts** (a) lack justification

✓ **- 3 pts** *(a) Correct algorithm for the problem but not optimal*

- **5 pts** (a) Incorrect algorithm
- **7 pts** (a) No attempt
- **1 pts** (b) Minor mistake (i.e. wrong time complexity analysis)
- **2 pts** (b) lack justification

✓ **- 3 pts** *(b) Correct algorithm for the problem but not optimal*

- **5 pts** (b) Incorrect algorithm
- **7 pts** (b) No attempt

QUESTION 5

5 Problem 5 **17 / 20**

- **0 pts** Correct
- **10 pts** (a) no effort
- **2 pts** (a) did not finish the algorithm
- **1 pts** (a) the complexity is not correct

✓ **- 3 pts** *(a) not efficient enough*

- **4 pts** (a) did not prove correctness
- **10 pts** (b) No effort
- **3 pts** (b) not an efficient algorithm
- **1 pts** (b) the time complexity is not correct
- **8 pts** (b) is not correct
- **3 pts** (b) did not finish

Problem 1 (a) (2pt) Briefly describe the problem statement for *stable matching* between n men and n women and what constitutes a valid solution.

(b) (8pt) State the asymptotic run-time complexity of the Gale-Shapely algorithm for n men and n women. Explain all data-structures used. Justify running time by counting the data-structures used and their complexity.

(c) (10pt) Prove that if we run the Gale-Shapely algorithm twice, once when men are proposing and once when woman are proposing, and the matching is the same for both runs, then it must be a unique stable matching for this problem instance (i.e. there is no other stable matching that exists).

a) Gale-Shapely seeks to pair n men and n women in what's called a stable matching. This algorithm famously guarantees a bachelar finds his girl. Both men and women create a preference list in which they rank perspective suitors. Starting with Day 1 men will propose to their #1 ranked women. If there are no coaflicts in which two men have the same #1 we continue. If two males have the same #1, we then go off the women's ranking. The winning male stays, losing male waits for the next day. The process repeats until we have all happy couples.

Ex: m_1 : w_1, w_2, w_3 w_1 : m_2, m_1, m_3

m_2 : w_3, w_2, w_1 w_2 : m_3, m_2, m_1

m_3 : w_3, w_1, w_2 w_3 : m_1, m_2, m_3

Day 1: Day 2: Day 3:

w_1 : m_1 w_1 : m_1, ~~m_3~~ w_1 : m_1 Done!

w_2 : w_2 : w_2 : m_3 (w_1, m_1)

w_3 : m_2, ~~m_3~~ w_3 : m_2 w_3 : m_2 (w_2, m_3)

(w_3, m_2)

For this algorithm, at termination we are guaranteed a stable matching for all n men and women.

b.) Gale-Shapely: $O(n^2)$

We use arrays of length n for men and women. So that means our algorithm has to go through both data structures, $O(n^2)$

c.)

$m_1: w_1, w_2, w_3$
$m_2: w_3, w_2, w_1$
$m_3: w_3, w_1, w_2$

$w_1: m_2, m_1, m_3$
$w_2: m_3, m_2, m_1$
$w_3: m_1, m_2, m_3$

Boys Proposing:
(w_1, m_1)
(w_2, m_3)
(w_3, m_2)

Women Proposing:

(1)

$m_1: w_3$
$m_2: w_1$ $\Rightarrow$
$m_3: w_2$

Women:
(m_1, w_3)
(m_2, w_1)
(m_3, w_2)

Results:

Women Proposing: $(m_1, w_3), (m_2, w_1), (m_3, w_2)$

Men Proposing: $(w_1, m_1), (w_2, m_3), (w_3, m_2)$

$\rightarrow$

As we can see based on a simple change of who proposes, we were lead to different and unique pairings. This is due to the fact G-S has men proposing, therefore there is a bias towards men. Men will reciere their best possible partner and women their pessimal partner. This is due to the original picking being based on the males preference and not the womens. When roles are reversed, women get their best matching and men their pessimal. That means for each unique case of men, women, and who proposes $\exists$ a single unique solution.

1 Problem 1 18 / 20

- **0 pts** Correct
- **1 pts** (a) Incomplete about stable matching problem statement or no valid solution.
- **2 pts** (a) missing

✓ **- 2 pts** *(b) lack of enough justify/explanation of time complexity.*

- **2 pts** (b) need to explain all data-structure used.
- **4 pts** (b) miss/wrong runtime complexity
- **8 pts** (b) missing
- **2 pts** (c) partial prove
- **4 pts** (c) not a complete prove
- **10 pts** (c) missing

Problem 2 (20pt) Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $f(n)$ is immediately before $g(n)$ in your list, then it should hold that $f(n)$ is $O(g(n))$. (AKA slower-growing functions first before faster-growing functions.)

You don't need to formally prove each sequential pair in your list, but you should have solid justification for the ordering. You can assume all given log terms are base 2. Write the ordering of functions using the function numbers in the list below.

$n = 16$

$f_1(n) = \frac{n^4}{(\log n)^{13}}$ → poly

$f_2(n) = \frac{2^{n-(\log n)^{13}}}{82,589,933}$ → exp

$f_3(n) = \frac{n^2}{13}$ → poly

$f_4(n) = 2^{3 \log n}$ → exp

$f_5(n) = n(\log n)^{42}$ → log

$f_6(n) = 2^{2 \log n - \log \log n}$ → exp

$f_5(n), f_3(n), f_1(n), f_2(n), f_6(n), f_4(n)$ smallest

exp > poly > log

Exp: $f_2(n), f_4(n), f_6(n) \Rightarrow 2^{n-c}, 2^{3\log n}, 2^{2\log n - \log\log n}$

Poly: $f_1(n), f_3(n)$ n^4 vs n^2 $n^4 > n^2$

$f_1(n) > f_3(n)$

log: $f_5(n)$

$n = 2$

(1) $2^{2-(\log 2)^{13}} \Rightarrow 2^1 \Rightarrow 2$

(3) $2^{3\log(2)} \Rightarrow 2^3 = 8$

(6) $2^{2\log(2) - \log(\log 2)} = 0$

(2) $2^{2(1-4)} = 2^{-8} = \frac{1}{4}$

Order:

$f_5(n), f_3(n), f_1(n), f_2(n), f_6(n), f_4(n)$

2 Problem 2 **10 / 20**

- **0 pts** Correct
- **2 pts** 1 order wrong
- **4 pts** 2 order wrong
- **6 pts** 3 order wrong
- **8 pts** 4 order wrong

✓ **- 10 pts** *5 order wrong*

- **12 pts** all order error.
- **4 pts** wrong and without explaination.
- **20 pts** missing

Problem 3 (a) (5pt) List the order in which Breadth-First Search explores the graph of Figure 1 starting from vertex A. (When arbitrarily choosing which edges from a node to look at first, choose the edges that go to the letter earlier in the alphabet first.)

(b) (5pt) Execute Dijkstra's algorithm on the graph of Figure 1 starting from vertex A. List the vertices and their corresponding shortest path distances (to vertex A) in the order popped from the priority queue.

(c) (5pt) Execute Prim's algorithm on the graph of Figure 1 assuming all edges are undirected starting from vertex A. The vertex with the lower alphabetical order comes first if there are any ties. List the edges in the order in which they are added to the tree.

(d) (5pt) Execute Kruskal's algorithm on the graph of Figure 1 assuming all edges are undirected starting from vertex A. List the edges in the order in which they are added to the forest.

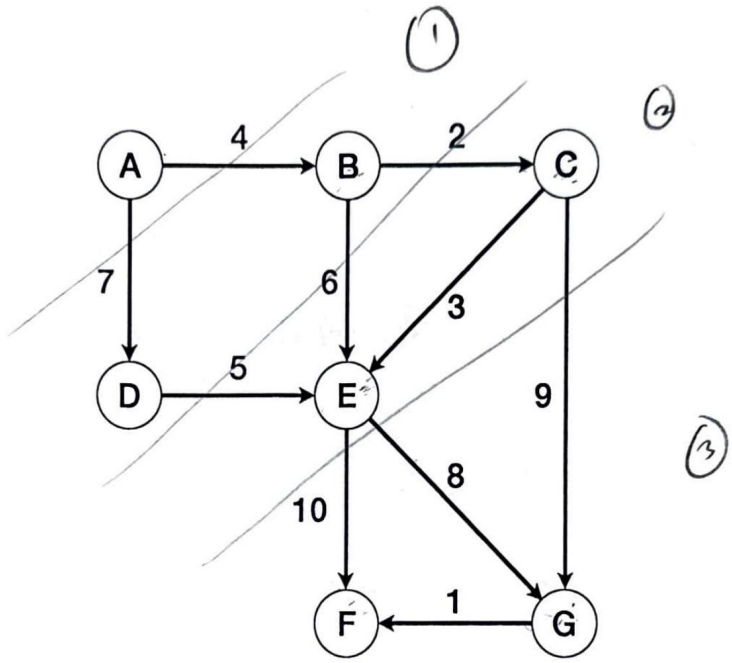


Figure 1: Graph for problem 3.

a.) BFS goes by layer

A → B B → C D → E E → F

A → D B → E E → G

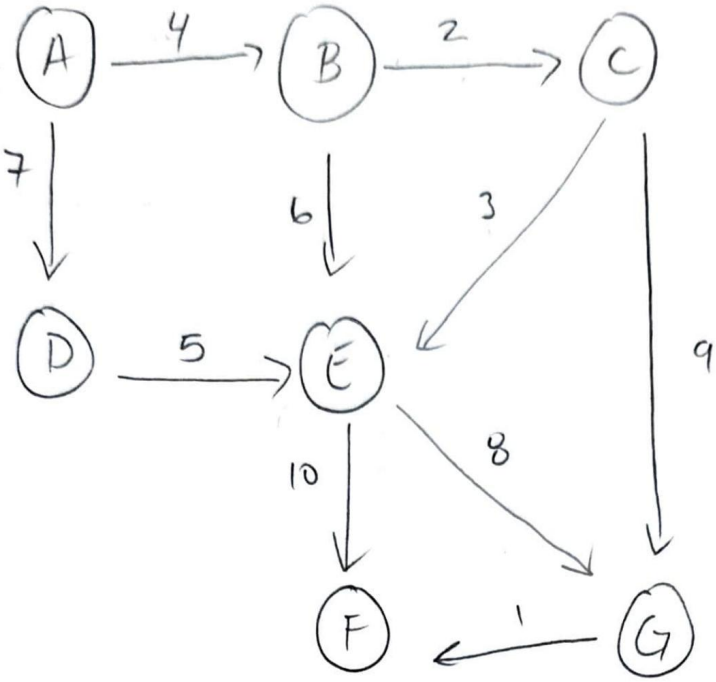
O(V+E)

A, B, D, C, E, F, G

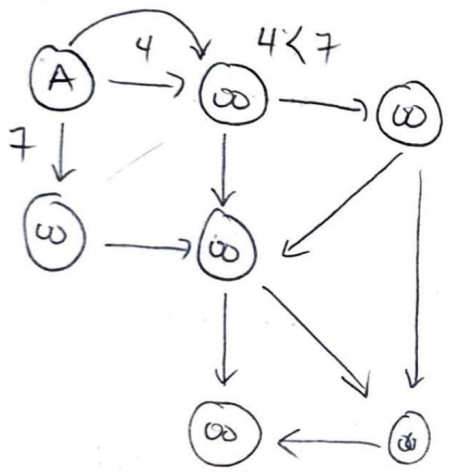
Name: He

④ →

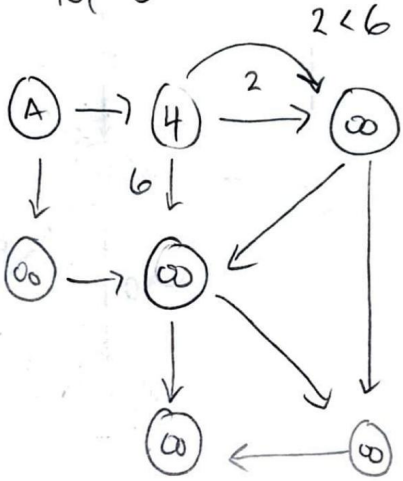
b.)



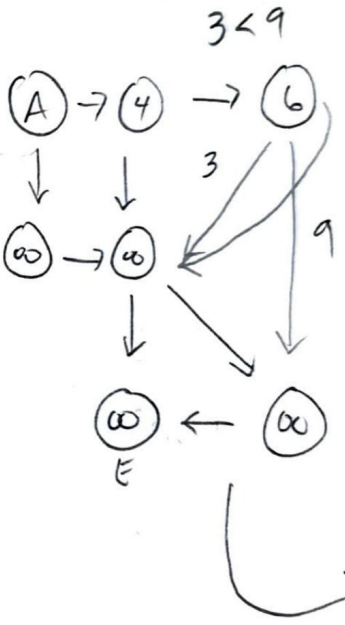
Dijkstra: A - F



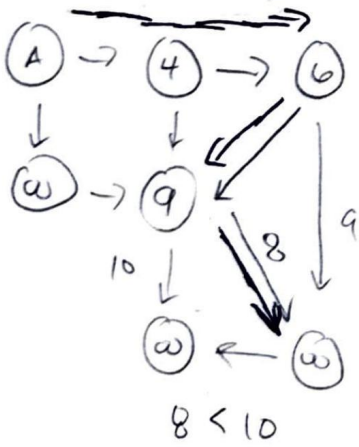
Pop B:



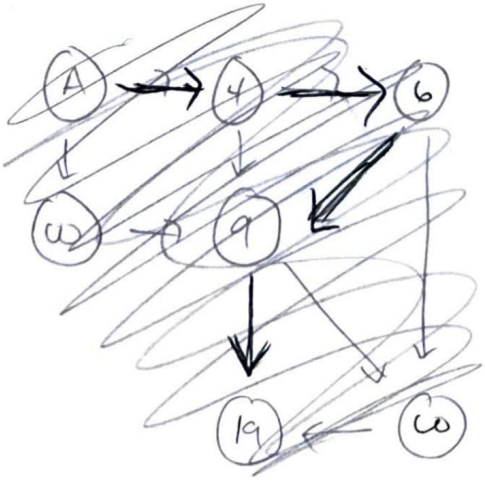
Pop C:

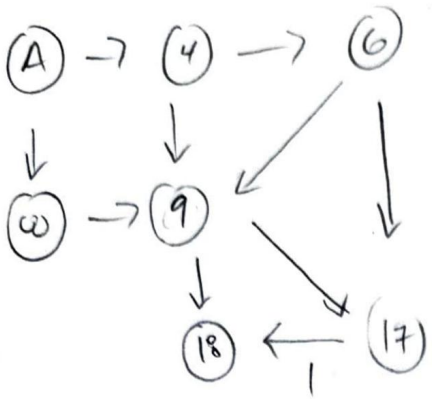


Pop E:



Pop G:

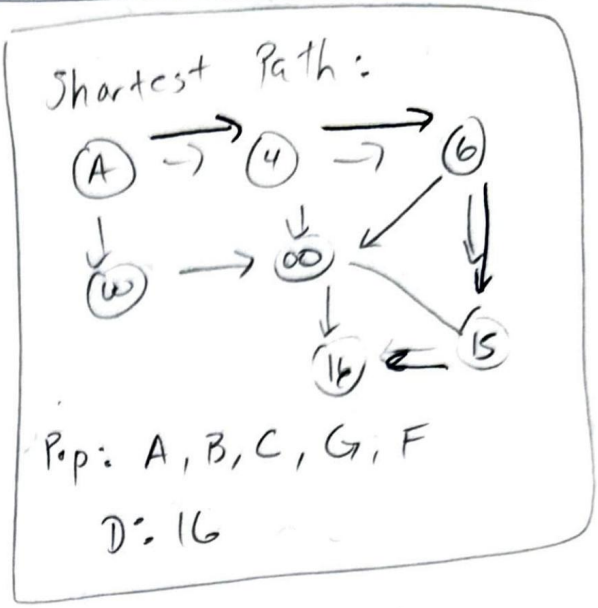




D: 18

Pop F: But

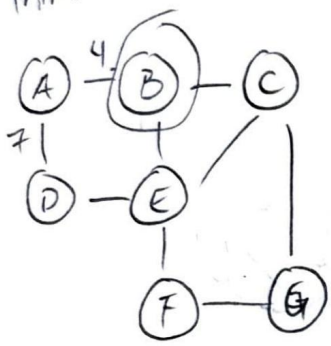
b.)

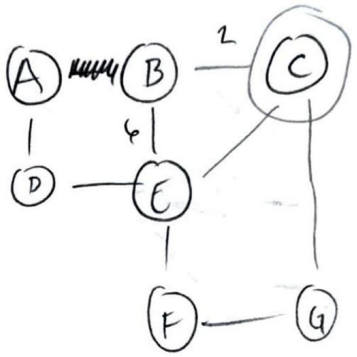


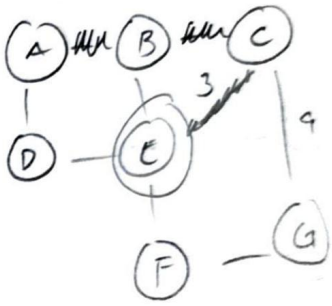
Pop: A, B, C, G, F

D: 16

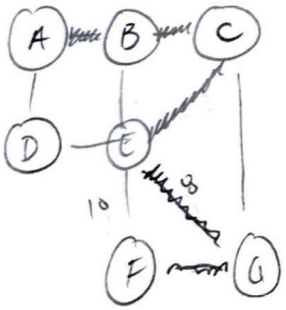
c.) Prim's: 4 & 7

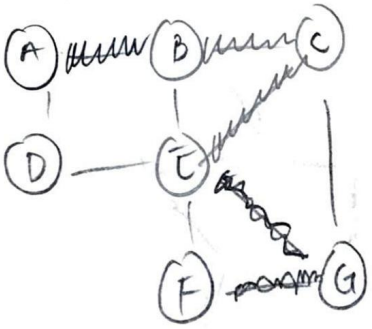






MST:





Edges:

(A, B), (B, C), (C, E)

(E, G), (G, F)

↳ O(E log V)

D.) Kruskal's

First sort By Weight:

4 (A, B) = 4

7 (A, D) = 7

5 (D, E) = 5

6 (B, E) = 6

2 (B, C) = 2

3 (C, E) = 3

10 (E, F) = 10

8 (E, G) = 8

9 (C, G) = 9

1

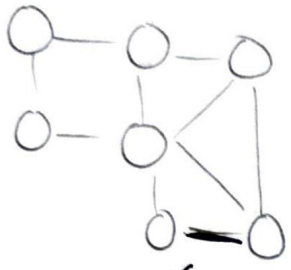
(G, F) = 1

Sorted Order:

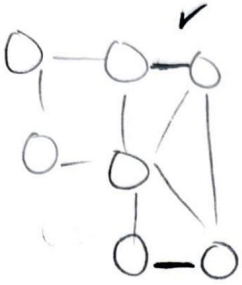
(G,F), (B,C), (C,E), (A,B), (D,E), ~~(B,E)~~, (A,D), (E,G)

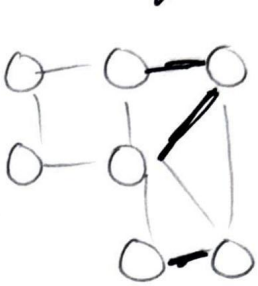
~~(C,G)~~, ~~(E,F)~~

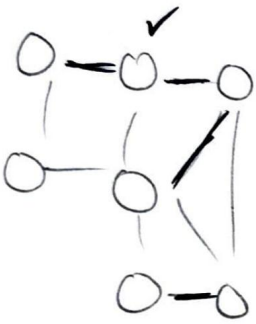
↳ Now Add in order, remove if cycle forms

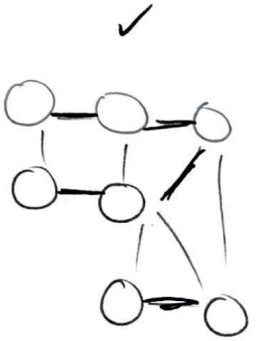
① 

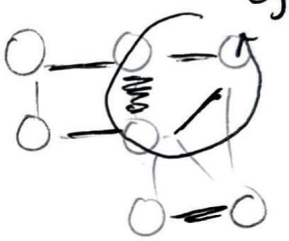
(G,F) ✓

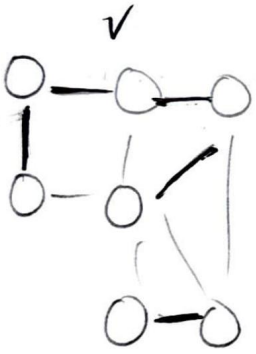
② 

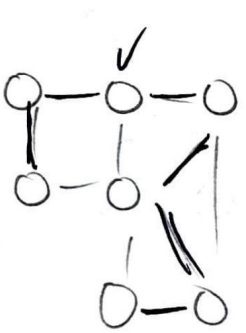
③ 

④ 

⑤ 

⑥ 

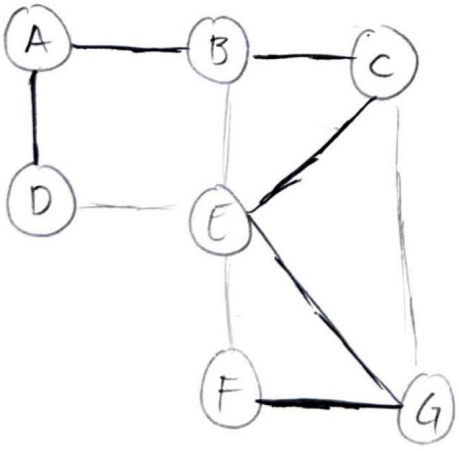
⑦ 

⑧ 

(G,G) = cycle!

(E,F) = cycle!

Kruskal:



Edges:

(G,F), (B,C), (C,E), (A,B), (D,E)

(A,D), (E,G)

3 Problem 3 **16 / 20**

- **0 pts** Correct

✓ **- 1 pts** *(a) Minor mistake (i.e. wrote ABDCEFG instead)*

- **2 pts** (a) Major mistake but showed some correct work
- **3 pts** (a) Major mistake with no work shown
- **5 pts** (a) No attempt

✓ **- 1 pts** *(b) Minor mistake (i.e. correct distances, wrong order)*

- **2 pts** (b) Major mistake but showed some correct work
- **3 pts** (b) Major mistake with no work shown
- **5 pts** (b) No attempt

✓ **- 1 pts** *(c) Minor mistake*

- **2 pts** (c) Major mistake but showed some correct work
- **3 pts** (c) Major mistake with no work shown
- **5 pts** (c) No attempt

✓ **- 1 pts** *(d) Minor mistake*

- **2 pts** (d) Major mistake but showed some correct work
- **3 pts** (d) Major mistake with no work shown
- **5 pts** (d) No attempt

Problem 4 We are given an array of n continuous, distinct, positive, integers, ranging in value from 1 to $n+1$, with one integer missing from the array.

For this problem, propose the most efficient means of determining the missing integer given the following conditions. Briefly justify the correctness and time complexity of your solution. All solutions should not be greater than a linear runtime (i.e. $O(n)$):

(a) (10 pt) If the array is sorted. $O(n \log n)$

(b) (10 pt) If the array is unsorted. $O(n^2)$

a.) If the array is sorted, we can just iterate linearly across the array, until the missing integer pops up.

array: n=3 => range: 1 to 4

1	2	3

=> 4 is missing.

```
for int i = 0; i < array size; i++ ← O(n) complexity
    if array[i] != i+1;
        return i
```

is since its sorted, array[i] should equal the index +1.

array[0] = 1 1 = 0+1 ✓

index: 0

1	2	3
[0]	[1]	[2]

Name:

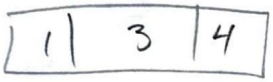
b.) For an unsorted array:

Similar to Kruskal's algorithm, we can just initially sort the array. Start by sorting the array with any of our sort method: bubble sort as an example.

Where:

$n = 3$

Sort Alg: $O(\log n)$ complexity:



Apply our original algorithm, which has $O(n)$.

We have a final algorithm with $O(n \log n)$.

$O(\log n)$ → sort(num) ← add the additional sort

```
for i in range (len(num)):
    if num[i] != i+1
        return 1
```

$O(n \log n)$

Our original

$O(n)$

4 Problem 4 **14 / 20**

- **0 pts** Correct
- **1 pts** (a) Minor mistake (i.e. wrong time complexity analysis)
- **2 pts** (a) lack justification

✓ **- 3 pts** *(a) Correct algorithm for the problem but not optimal*

- **5 pts** (a) Incorrect algorithm
- **7 pts** (a) No attempt
- **1 pts** (b) Minor mistake (i.e. wrong time complexity analysis)
- **2 pts** (b) lack justification

✓ **- 3 pts** *(b) Correct algorithm for the problem but not optimal*

- **5 pts** (b) Incorrect algorithm
- **7 pts** (b) No attempt

Problem 5 (a) **Graph Coloring** (10 pt) We are given an undirected graph $G = (V, E)$. We say that G has *max-degree* Δ if any vertex in G has at most Δ edges. Your algorithm should color all vertices in V using a set of S different colors, where each vertex gets assigned a unique color from S. We call a coloring *valid* if all vertices are colored and for any $(u, v) \in E$, the assigned color of u is different from assigned color of v. Show an as efficient as possible algorithm that finds a valid coloring of any G with max degree Δ using $\Delta + 1$ colors. Prove correctness of your algorithm and state its running time.

(b) **DAG Reachability** (10 pt) Given a Directed Acyclic Graph $G = \{V, E\}$, your task is to design an algorithm that finds the minimum number of "starting" nodes $S \subset V$ so that you can reach any node in G starting from one of the nodes in S and following edges of G. Given an adjacency list representation of G, describe in detail your algorithm (including all data-structures) and its running time. Your algorithm should be as efficient as possible. The algorithm does not need to output all the starting nodes, only the *cardinality* of $|S|$.

a.) For graph coloring, we are trying to assign each unique vertex with another unique color. This is similar to our Gale-Shapely problem! Since we have a maximum Δ vertex, we can have a max Δ colors. For each vertex, have them rank by color, doesn't really matter. And then perform Gale-Shapely. Since Gale-Shapely guarantees that if a stable matching is possible, it will be found at the end of the algorithm.

Ex: v_1: c_1, c_2, c_3
v_2: c_2, c_3, c_1
v_3: c_1, c_3, c_2

c_1: v_3, v_2, v_1 ↙ Uses arrays
c_2: v_2, v_3, v_1
c_3: v_3, v_1, v_2

(1)
c_1: ~~v_1~~, v_3
c_2: v_2
c_3:

(2)
c_1: v_3
c_2: v_2, ~~v_1~~
c_3:

(3)
c_1: v_3 ↙
c_2: v_2
c_3: v_1

(c_1, v_3)
(c_2, v_2)
(c_3, v_1)

As you can see from our algorithm, we formed a unique matching.

↳ $O(n^2) = O(\Delta^2)$

To prove correctness: non => $\Delta \cdot \Delta$

Assume GS does NOT produce a unique matching. We know that we have a set of Δ vertices and Δ colors. By Pigeonhole Principle: $\frac{\lceil \Delta \rceil}{\Delta} = 1$

This means for every Δ vertex there is AT most only 1 color that can be assigned to it. This means that there cannot $\exists$ a $(u,v) \in E$ such that the vertex u comes from and the vertex in which v accepts u that they have the same color. This contradicts our assumption, since Pigeonhole Principle proves there can only be 1 color per vertex.

→ ←

5.) For this I belicve the approach of a tree traversal of DFS would be proper. Since its directed and not cyclic, if we traverse the graph until we hit our desired node, we can simply return the queue that

Stores the nodes visited. Starting at our root node, we can have a queue of explored and unexplored nodes. Since DFS, prioritize visiting nodes over edges, the amount of nodes visited from the root node to desired node can be returned, this would also be the minimum nodes. From then we can simply trace the path from the root node to desired node, and return the amount of nodes visited.

Proof of Correctness:

DFS starts at a root node and visits adjacent nodes layer by layer, taking into account the minimum weight. Let's assume for the sake of contradiction, that DFS does NOT visit every node. Since DFS, visits every node, taking into account the layers, weight, and path involved, it is certain that when DFS terminates the algorithm would have visited all possible nodes. Since DFS terminates when the maximum tree depth is reached signifying that there are no more nodes able to visit,

it is a contradiction that DFS does not follow every node. →←

Since it traverses the entire tree, there must ∃ a path from root node to desired node in which we can count and return the minimum node from root to desired.

↳ $O(V+E)$

↳ just like BFS. Depends on # V and # E

5 Problem 5 17 / 20

- **0 pts** Correct
- **10 pts** (a) no effort
- **2 pts** (a) did not finish the algorithm
- **1 pts** (a) the complexity is not correct

✓ **- 3 pts** *(a) not efficient enough*

- **4 pts** (a) did not prove correctness
- **10 pts** (b) No effort
- **3 pts** (b) not an efficient algorithm
- **1 pts** (b) the time complexity is not correct
- **8 pts** (b) is not correct
- **3 pts** (b) did not finish

CS 180: Introduction to Algorithms and Complexity
Midterm Exam

2:00pm – 3:50pm, May 1, 2023

Name	Harris Doan
UID	605317270
Section	1C

★ **Print your name, UID and section number in the boxes above, and print your name at the top of every page.**

★ **There are 5 problems. Please check to make sure your test contains all 5 problems.**

★ **At the end of the test, you will be instructed to stop all writing, take out your phone, and use it to scan and upload your test to Gradescope. After you are done, turn in the physical test to the instructor. If for whatever reason you are unable to upload your test after time is up, verbally tell the instructor when turning in your physical test.**

- Only write your answers on the provided pages. Scratch paper will be provided by request. If you include parts of your solution on scratch paper, you must tell the instructor verbally when turning your test in.
- Your answers are supposed to be in a simple and understandable manner. Sloppy answers are expected to receive fewer points.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Partial credits will be given. (e.g., for correct algorithms that do not meet the expected time complexity, answers that show evidence of mostly-correct reasoning with a few mistakes, etc.)