Midterm1

● Graded

Student

Dev Pratap Singh

Total Points

69 / 100 pts

Question 1

MCQ

33 / 36 pts

✔ **+ 3 pts** Q1 is correct.
Correct answer: *set a:* **B**, *set b:* **B**, *set c:* **D**

✔ **+ 3 pts** Q2 is correct.
Correct answer: *set a:* **C**, *set b:* **A**, *set c:* **A**

✔ **+ 3 pts** Q3 is correct.
Correct answer: *set a:* **A**, *set b:* **D**, *set c:* **B**

✔ **+ 3 pts** Q4 is correct.
Correct answer: *set a:* **B**, *set b:* **D**, *set c:* **B**

✔ **+ 3 pts** Q5 is correct.
Correct answer: *set a:* **B**, *set b:* **B**, *set c:* **A**

✔ **+ 3 pts** Q6 is correct.
Correct answer: *set a:* **C**, *set b:* **B**, *set c:* **A**

+ 3 pts Q7 is correct.
Correct answer: *set a:* **D**, *set b:* **B**, *set c:* **A**

✔ **+ 3 pts** Q8 is correct.
Correct answer: *set a:* **C**, *set b:* **D**, *set c:* **A**

✔ **+ 3 pts** Q9 is correct.
Correct answer: *set a:* **B**, *set b:* **C**, *set c:* **C**

✔ **+ 3 pts** Q10 is correct.
Correct answer: *set a:* **D**, *set b:* **C**, *set c:* **D**

✔ **+ 3 pts** Q11 is correct.
Correct answer: *set a:* **D**, *set b:* **D**, *set c:* **C**

✔ **+ 3 pts** Q12 is correct.
Correct answer: *set a:* **A**, *set b:* **C**, *set c:* **B**

+ 0 pts None of the MCQ is correct

Question 2

Question 13

12 / 17 pts

- ✔ **+ 2 pts** 2 rows correct
- **+ 3 pts** 3 rows correct
- **+ 4 pts** 4 rows correct
- **+ 5 pts** 5 rows correct/5 vertices correct
- **+ 6 pts** 6 rows correct/6 vertices correct
- **+ 7 pts** all rows correct

- ✔ **+ 3 pts** (b) order correct
- ✔ **+ 5 pts** (c) tree correct
- ✔ **+ 2 pts** (d) answer yes
- **+ 0 pts** all incorrect

Question 3

Question 14

6 / 15 pts

Part a)

✔ **+ 5 pts** Correct

+ 4 pts Significant progress made.

+ 2 pts Incorrect attempt but used right kind of proof.

+ 0.5 pts (Mostly wrong) 10% points.

+ 0 pts Did not attempt or left blank.

Part b)

+ 9 pts Overall, mostly correct but the return value is not a position.

+ 2 pts Part 1 is correct.

+ 1 pt Part 1 off by 1 error.

✔ **+ 1 pt** Part 1 is a little off (the return value is not a position).

+ 0 pts Part 1 is incorrect or left blank.

+ 8 pts Part 2 is correct.

+ 7.5 pts Part 2 almost correct (return y used instead of return m)

+ 7.5 pts Part 2 almost correct (off by one)

+ 7 pts Part 2 almost correct (1 and n is used instead of i and j)

+ 6 pts Part 2 correct divide and conquer but not enough detail shown.

+ 5 pts Part 2 one case is missed.

+ 5 pts A correct O(n) or O(nlogn) algorithm is given.

+ 4.5 pts Part 2 is partially correct (algorithm runs in O(logn), if/else statement used).

+ 4 pts An incorrect O(logn) algorithm is given.

+ 3.5 pts An incorrect O(n) or O(nlogn) algorithm is given.

+ 3 pts Incomplete solution but the idea of divide and conquer is used.

+ 2.5 pts One return is correct

+ 1 pt (Mostly wrong) 10% points.

✔ **+ 0 pts** Part 2 is incorrect or left blank.

Question 4

Question 15

13 / 16 pts

15.(a)

+ 4 pts Proof 1: Correctly proves that universally reachable vertices must belong to the sink component.

+ 3 pts Proof 1: Mentions that universal-reachable vertices must belong to the sink component, but the proof has minor errors.

+ 2 pts Proof 1: Correctly proves that there must be only one sink component.

+ 1 pt Proof 1: Mentions that there must be only one sink component, but the proof has minor errors.

✔ **+ 5 pts** Proof 2: Correctly explain that any of the universal-reachable vertices can be reached by other universal researchable vertices. So all of them are in a same connected component.

+ 1 pt Proof 2: Correctly prove that there can not be other vertices (non-universal reachable vertices) in this connected component. (maximal)

+ 0 pts Incorrect proof / blank.

+ 0.6 pts I don't know how to answer this question.

15.(b)

+ 8 pts Correct $O(|V| + |E|)$ algorithm

✔ **+ 3 pts** Partial credit for finding connected components of G.

+ 2 pts Partial credit for checking whether there is only one sink component, returning 0 if there isn't, and returning the number of vertices in the sink component if there is.

+ 1 pt Partial credit for checking whether there is only one sink component (though with errors).

✔ **+ 3 pts** Partial credit for finding the sink component.

+ 2 pts Partial credit for finding the sink component (though with errors).

+ 3 pts Partial credit for an $O(|V| \times (|V| + |E|))$ or even slower algorithm.

+ 0 pts Incorrect algorithm / blank.

✔ **+ 2 pts** Correct time complexity: mentions that finding connected components (O(|V| + |E|)) dominates the time complexity.

+ 1 pt Partial credit for mentioning the time complexity of other non-dominant parts that lead to O(|V|) or O(|E|) time complexity.

+ 1 pt I don't know how to answer this question.

+ 0 pts Incorrect time complexity / blank.

Question 5

Question 16

5 / 16 pts

16a)

+ 0 pts All entries incorrect

+ 1 pt 1/4 entries correct

+ 2 pts 2/4 entries correct

+ 3 pts 3/4 entries correct

✔ + 4 pts All entries correct

+ 0.4 pts I don't know answer.

16b)

+ 8 pts Correctness of Algorithm: Algorithm correctly computes shortest distance from a to u in reverse graph G^R and a to v in graph G, and sums it up.

+ 4 pts Partial Correctness of Algorithm: Algorithm correctly computes exactly one of: (i) shortest distance from a to v in G (ii) a to u in reverse graph G^R or (iii) provides partial explanation

✔ + 0 pts Incorrect Algorithm / No Algorithm

+ 2 pts Correct Time complexity: O(V^2 + VE) is mentioned with proper reasoning

✔ + 1 pt Partially correct Time Complexity: O(VE) is mentioned or O(V^2 + VE) is mentioned without valid reasoning

+ 1 pt non-optimal time complexity provided (with proper reasoning).

+ 0.2 pts I don't know answer. (Runtime part)

+ 0 pts Incorrect/No Time Complexity provided

+ 2 pts Valid Correctness Explanation provided

+ 1 pt Partially Valid Correctness Explanation (for example, explains why minimum distance d_a(u,v) can be achieved using d(u,a) + d(a,v) but algorithm is wrong)

+ 0.2 pts I don't know answer. (Correctness part)

✔ + 0 pts Incorrect Correctness Explanation / No Explanation

+ 1.2 pts "I don't know the answer" or the answer is very off.

CMPSC 465 Data Structures & Algorithms
Fall 2024 Mingfu Shao and Debarati Das

Mid-Term 1

Time: 8-10PM, Tuesday, Oct. 8th, 2024

Your Name: Dev Pratap Singh

Your PSU Access ID: dxs6169

Your Recitation: 008R

INSTRUCTIONS:

1. Please clearly write your full name, your PSU Access User ID (i.e., xyz1234), and the recitation you are in (001–010) in the box above.
2. This exam contains 16 questions.
3. For questions 1–12 (i.e., multiple-choice questions) exactly one choice is correct.
4. Your answer to questions 1–12 MUST be recorded in the table on top of page 2.
5. For questions 13–16, aim to complete your answer within the space provided on the current page and the next page. If additional space is needed, use the last 4 pages (pages 13–16), and clearly indicate which problem your answer corresponds to.

THE TABLE BELOW IS FOR GRADERS USE ONLY:

Question	1–12	13	14	15	16	Total
Points						

Your answer to questions 1–12:

Question	1	2	3	4	5	6	7	8	9	10	11	12
Your Answer	D	A	B	B	A	A	D	A	9	D	C	B

1. (3 pts.) What is the time complexity of the following procedure?

```
1: for i ← 1 to n do
2:     j ← 1
3:     while j ≤ n do
4:         j ← j × 3
5:     end while
6: end for
```

A. $\Theta(n)$
B. $\Theta(n^2)$
C. $\Theta(n^3)$
D. $\Theta(n \cdot \log n)$

2. (3 pts.) Let S be an array with n distinct integers. Similar to the selection algorithm, we partition S into $n/19$ subarrays, each of which contains 19 numbers. Let x be the median of the medians of the $n/19$ subarrays. How many numbers in S that are guaranteed to be less than or equal to x? You may assume that n is divisible by 38.

A. $5n/19$
B. $11n/38$
C. $6n/19$
D. $13n/38$

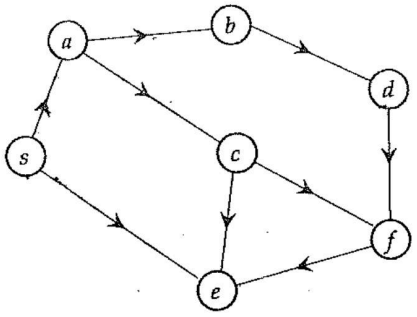
3. (3 pts.) Consider the two recursive functions: $f(n) = 4f(n/2) + n^3$, $g(n) = 6g(n/2) + n^2 \log n$. Which of the following is correct regarding the growth rates of $f(n)$ and $g(n)$?

A. $f = O(g)$ but $f \neq \Theta(g)$
B. $f = \Omega(g)$ but $f \neq \Theta(g)$
C. $f = \Theta(g)$

4. (3 pts.) Adding an edge to a DAG creates a cycle.

A. Always
B. Sometimes
C. Never

5. (3 pts.) How many linearizations does the following directed acyclic graph (DAG) have?



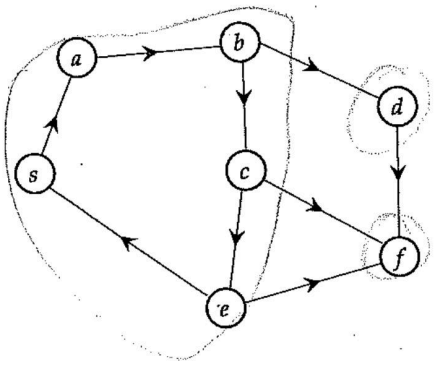
A. 3

B. 4

C. 5

D. 2

6. **(3 pts.)** How many edges in the meta-graph of the following graph?



A. 3

B. 4

C. 5

D. 2

7. **(3 pts.)** Consider a binary min-heap represented using an array A. It is known that the priority of one element has changed so that the min-heap-property does not hold. Currently $A = [15, 20, 16, 14, 22, 17]$. Which one of the following procedure can restore its heap-property? Assume A is indexed from 1.

A. bubble-up(A, 4);

B. bubble-up(A, 6);

C. sift-down(A, 1);

D. sift-down(A, 2);

8. **(3 pts.)** Consider running DFS-with-timing on a directed graph $G = (V, E)$. Assume that u is explored before v. Assume also that v can reach u but u cannot reach v in G. Which one of the following is always true?

   A. $pre[u] < pre[v]$ and $post[u] < post[v]$.
   B. $pre[u] < pre[v]$ and $post[u] > post[v]$.
   C. $pre[u] > pre[v]$ and $post[u] < post[v]$.
   D. $pre[u] > pre[v]$ and $post[u] > post[v]$.
   E. None of the above is always true.

9. **(3 pts.)** Consider the following variation of merge-sort: instead of partitioning the given array A into two subarrays of equal size, we divide A into two subarrays such that the first one consists of only one number, i.e., $A[1]$, and the other one consists of the remaining numbers, i.e., $A[2..n]$. What is the time complexity of this variation of merge-sort (where n is the size of the input array)?

   A. $O(n)$
   B. $O(n \cdot \log n)$
   C. $O(n^2)$
   D. $O(n^2 \cdot \log n)$

10. **(3 pts.)** Let $G = (V, E)$ be a directed acyclic graph (DAG). Which of the following implies that the vertex v is a sink in G?

   A. The vertex v with the smallest post number when performing DFS-with-timing on G.
   B. The vertex v that appears last in any linearization of G.
   C. The vertex v with the largest post number when performing DFS-with-timing on G_R, i.e., the reverse graph of G.
   D. All of the above.

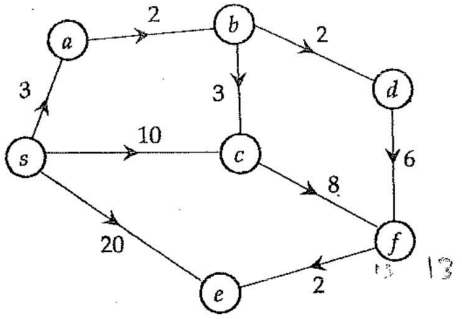
11. **(3 pts.)** Which of the following does NOT satisfy $f = O(g)$?

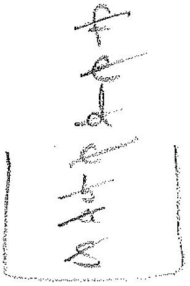
   A. $f(n) = n^3 \cdot 2^n$, $g(n) = n^2 \cdot 3^n$
   B. $f(n) = \sum_{k=1}^{n} 1/k$, $g(n) = \log n$
   C. $f(n) = \log(n^{\log n})$, $g(n) = 2^{\log \log n}$
   D. $f(n) = \log(n!)$, $g(n) = n^2$

12. **(3 pts.)** Let $G = (V, E)$ be a directed graph. Assume that G consists of just one connected component, i.e., all vertices in V form a single connected component. Assume also that $|V| \geq 2$. Which of the following is NOT always true?

   A. $|E| \geq |V|$.
   B. $|E| \leq |V|^2/2$.
   C. G cannot be linearized.
   D. Let $u \in V$ be an arbitrary vertex. Then u can reach every other vertex in G.

13. (7 + 3 + 5 + 2 pts.) Consider the directed graph G given below.





(a) Run Dijkstra's algorithm on G, starting at the given source vertex s. Whenever there is a choice of vertices with the same dist value, always pick the one that is alphabetically first. Draw a table where each row shows the dist array at each iteration of the algorithm.

(b) Give the order of vertices following which they are removed from the priority queue.

(c) Draw one shortest path tree of G with respect to source s.

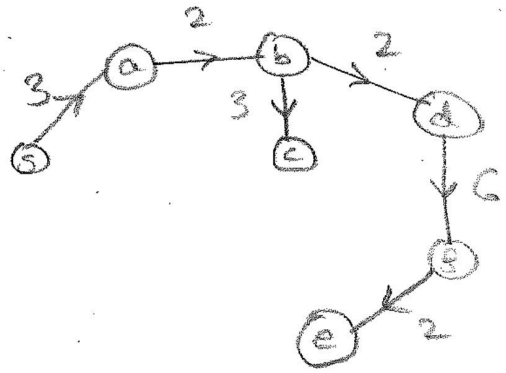
(d) Is the shortest path tree of G unique? You just need to answer Yes or No.

b] s, a, b, d, c, f, e

a]

node	s	a	b	c	d	e	f
0	∞	∞	∞	∞	∞	∞	∞
1	0	3	∞	∞	∞	∞	∞
2	0	3	5	∞	∞	∞	∞
3	0	3	5	∞	7	∞	∞
4	0	3	5	8	7	∞	∞
5	0	3	5	8	7	∞	13
6	0	3	5	8	7	15	13

c]



d] Yes

1 3 2 ^ 1 2 3 4

14. (5 + 10 pts.) Let $A[1..n]$ be an array with n positive integers, $n \geq 3$. Assume that $A[1] < A[2]$ and $A[n] < A[n-1]$. A location k is said to be a *peak* if $A[k] \geq A[k-1]$ and $A[k] \geq A[k+1]$. For example, there are five peaks in the following array; the peaks are in boldface.

$$\mathbf{5}, 6, \mathbf{6}, 2, \mathbf{3}, \mathbf{7}, 5, \mathbf{4}, \mathbf{8}, 3, \mathbf{3}, 4, \mathbf{10}, 6$$

(a) Prove that, in such an array A, peak always exists.

(b) Now we aim for designing an algorithm to find the location of a peak. If there are multiple peaks, we only need to find the location of one. The framework of the algorithm is given below, where the recursive function find-peak(A, i, j) is defined to return the peak location in $A[i..j]$. You are asked to fill in the missing two parts in it (2 + 8 pts). The entire algorithm should run in $O(\log n)$ time.

n = 3
A[3] < A[2]

1 5 6 6 8 7

function find-peak($A[1..n]$, i, j)

1: If $i + 2 \geq j$: **# FILL IN MISSING PART 1**
2: $m \leftarrow \lceil (i + j)/2 \rceil$
3: $x \leftarrow A[m-1]$
4: $y \leftarrow A[m]$
5: $z \leftarrow A[m+1]$
6: **# FILL IN MISSING PART 2**

~~a) the element after A[2] has to be less than A[2]~~
~~A[2] as A[n] < A[n-1]~~

a] Proof by contradiction, assume peak doesn't exist.
For a peak to not exist, we would need a monotonous array [increasing or decreasing] but that is not possible as A[n] < A[n-1] and A[1] < A[2] would contradict each other.

If we have increasing order,

A[1] < A[2] < ... A[n-1] < A[n]
right — wrong condition

If we have decreasing order,

A[1] > A[2] ... A[n-1] > A[n]
wrong condition — right

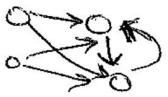
And if we have neither increasing nor decreasing, we would find a peak somewhere as the shift from inc. to decreasing in any part of array would form a peak.

Example → [5 6 7 8 3 6 7 5]

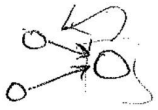
b]

#1 return ;

2



15. **(6 + 10 pts.)** You are given a directed graph $G = (V, E)$. We define a vertex $v \in V$ to be *universal-reachable* if every other vertex $u \in V \setminus \{v\}$ can reach v. The task is to identify all universal-reachable vertices in G.



(a) Prove that, if universal-reachable vertices exists, then they form a connected component of G,

(b) Given $G = (V, E)$, design an algorithm to find all universal-reachable vertices in G. Describe your algorithm (8 pts) and analyze its running time (2 pts). Your algorithm should run in $O(|V| + |E|)$ time. (*Hint: think which connected component that universal-reachable vertices belong to.*)

a] Proof by contradiction, assume that vertex $v \in V$ does not form a connected component of universal-reachable vertices.

a] Proof by contradiction, assume that universal-reachable vertices $v_1, v_2 \ldots v_k \in V$ do not form a connected component. Assuming universal-reachable vertices exist. ☆

Base Case → if there's only 1 universal-reachable vertex v_i.

If $u \in V - \{v_i\}$ can reach v_i as it is a universal-reachable vertex, we have $(u, v_i) \in E$ for every vertex $\in |V|$.

So we have paths $u_1 \to v_i$, $u_2 \to v_i$, ...

Since, we have assumed that it is not part of a

If we have paths $u_1 \to v_1, u_1 \to v_2, \ldots$

$u_2 \to v_1, u_2 \to v_2, \ldots$

$u_k \to v_1, u_k \to v_2 \ldots u_k \to v_k$

} Since, they are a universal-reachable vertex

where $v_i \in V$, $1 \le i \le k$

*continued ↗

We also have paths and vice-versa.

$v_1 \to v_2, v_3 \to v_2, v_4 \ldots v_k \to v_2$

$v_2 \to v_1, v_3 \to v_1, \ldots v_k \to v_1$

$v_1 \to v_3, v_2 \to v_3 \ldots$

} All the universal-reachable vertices reach other universal-reachable vertices as they are obviously universal-reachable vertices.

What we get are paths going from

$v_1 \rightleftarrows v_2 \rightleftarrows v_3 \rightleftarrows \ldots v_k \rightleftarrows v_1$

as each vertex has to reach the other for it to be universal-reachable.

All these paths form a connected component as they are all interconnected and can reach other from any vertex which is the definition of a connected component.

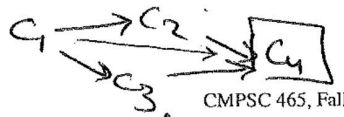
Any vertex in this arrangement will be included in the component.

b) For finding all the universal-reachable vertices, we should run a special-order DFS which separates all the different connected components of G. While running DFS, we should keep a counter (u_r_v (universal-reachable)) initialised at 0 & incremented when all the conditions of such a vertex are passed by the vertex to count the number of such vertices. While running DFS, we would keep track for every vertex (reached = 0) if the reached counter = |V| − 1 (except itself so −1)

We would do this by incrementing it in the explore f^n where it checks if it has visited this particular node before, so if

```
for edge (v_i, v_j) ∈ E:
    if v[j] = 0:
        explore(G, v_j);
    elif v[j] > 0:
        reached += 1;
```

b) We would find the meta graph of G and with the new graph G_M, find the sink connected component as the connected components of G would have their sink as the universal-reachable vertices.

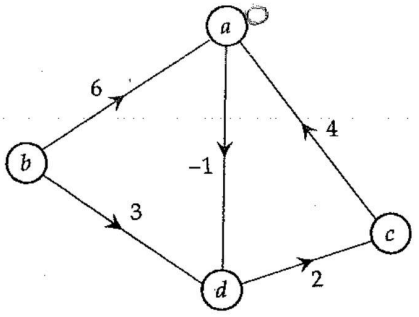


☆ 15.5] continued on blank pages →

16. **(4 + 12 pts.)** Let $G = (V, E)$ be a directed graph with possibly negative edge length, but without negative cycle. Let $a \subset V$. Define $distance_a(u, v)$ as the length of the shortest path from u to v that goes through vertex a.

(a) See an example given below. Fill out the last row of the matrix, i.e., calculating $distance_a(d, v)$ for each $v \in \{a, b, c, d\}$.

(b) Given $G = (V, E)$ and $a \in V$, design an algorithm to calculate $distance_a(u, v)$ for all pairs of vertices in G. Describe your algorithm (8 pts), analyze its running time (2 pts), and briefly prove the correctness of your algorithm (2 pts). Your algorithm should run in $O((|V| + |E|) \cdot |V|)$ time. (*Hint: consider running Bellman-Ford algorithm twice (on two different graphs).*)



Input: G, vertex a

a]

	a	b	c	d
a	0	∞	1	−1
b	6	∞	7	5
c	4	∞	5	3
d	6	∞	7	5

Output: $distance_a(u, v)$, for all pairs (u, v), represented as a matrix

b] Do Bellman - Ford twice

Once on G

Once on G_R

this

Runtime would be $O((|V|+|E|) \cdot |V|)$

as

★ 15.6] Continued.

Example → 1 | 2 | 1 | 2 | 3 | 4 | 4 | 4

Max value of special vertex visited currently.

So, 1, 2, 1 are reachable.

If the meta-graph has it's other components

We need to check if the vertex is universal reachable which we can do by keeping a counter reached = 0 and incrementing

We need to check if the universal connected component is present or not. So, we check if a particular component is reachable from other components by running DFS on meta. $\Theta(|V|+|E|)$

If there is such a component present such that is universally reachable, it would contain the universal-reachable vertices.

The logic is that if you have a component where every other component can reach it, it would have all the vertices of components reaching all the vertices of the last (sink) component and is also forming a component so it is all interconnected according to the definition.

DFS with meta would take $O(|V|+|E|)$ time as it explores all vertices & edges, running the increment of universal-reachable vertices would have a runtime way smaller than the $|V|+|E|$ as only a few compared to every vertex would be u-r.

If every vertex would be u-r, we would find it in $|V|+|E|$ tim