Midterm1

Graded

Student

Krishna Nitin Pagrut

Total Points

84.5 / 100 pts

Question 1

MCQ

Resolved **27** / 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

Regrade Request **Submitted on: Oct 20**

Q10 Requesting Regrade
n^2 log n is the fastest growing function out of all the Options and since Big-O is bounding above. The n^2 log n above bounds every other function given in the option.
n^2 log n > n^2 which means n^2 = O(n^2 log n)

it does not reflect the actual time complexity derived from solving the recurrence relation for this specific problem.

Reviewed on: Oct 25

Question 2

Question 13

17 / 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

11.5 / 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

14 / 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

15 / 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: Krishna Pagrut

Your PSU Access ID: Knp5451

Your Recitation: 8R

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	B	A	D	D	C	A	B	D	C	A	D	C

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

$j = j \times 3$ $\quad 3^k = n$ $\quad k = \log_3 n$ $\quad n^3$ $\quad a=6,\ b=2,\ d=2$ $\quad n^2$ $\quad s=1$ $\quad n^3$ $\quad n^{\log_2 6}$ $\quad \log_2 6 > 2$

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

$a=4,\ b=2,\ d=3,\ s=0$ $\quad 3$ $\quad \log_2 4 = 2$ $\quad 2 <$ $\quad \delta(n) = \Theta(n^3)$ $\quad d > \log_b a$ $\quad n^d \log$

3. (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. $11n/38$
B. $13n/38$
C. $6n/19$
D. $5n/19$

4. (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. $f(n) = \log(n!),\ g(n) = n^2$
D. $f(n) = \log(n^{\log n}),\ g(n) = 2^{\log \log n}$

$f < c \cdot g$ $\quad \frac{f}{g} = 0$ $\quad n^3 \ \frac{n}{(1.5)^n} = \frac{1}{}$ $\quad n \log n - n$ $\quad n(\log n - 1)$

5. **(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.

15, 20, 16, 14, 22, 17
1 2 3 4 5 6

A. bubble-up(A, 6);

B. bubble-up(A, 4);

✓C. sift-down(A, 2);

D. sift-down(A, 1);

6. **(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?

(pre, post)

u → v

u v v u

[[ ]]

✓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 $post[u] < post[v]$.

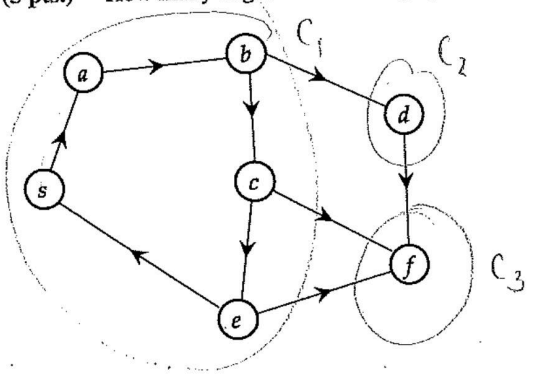
D. $pre[u] > pre[v]$ and $post[u] > post[v]$.

E. None of the above is always true.

post[u] < post[v]

pre[v] < pre(u)

7. **(3 pts.)** How many edges in the meta-graph of the following graph?



C1 → C2

C1 → C3

C2 → C3

A. 2

✓B. 3

C. 4

D. 5

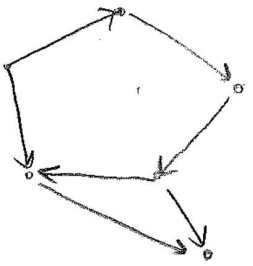
8. **(3 pts.)** Let $G = (V, E)$ be a directed acyclic graph (DAG). Which of the following implies that the vertex v is a sink in G?

✓A. The vertex v that appears last in any linearization of G.

B. The vertex v with the largest post number when performing DFS-with-timing on G_R, i.e., the reverse graph of G.

✓C. The vertex v with the smallest post number when performing DFS-with-timing on G.

✓D. All of the above.



9. (3 pts.) Adding an edge to a DAG creates a cycle.

A. Always

B. Never

C. Sometimes

10. (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. $O(n^2 \cdot \log n)$

B. $O(n \cdot \log n)$

C. $O(n^2)$

D. $O(n)$

$T(n) = T(1) + T(n-1) + \Theta(n \log n)$

$T(n) = T(n-1) + \Theta(n \log n)$

$n-1$

11. (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?

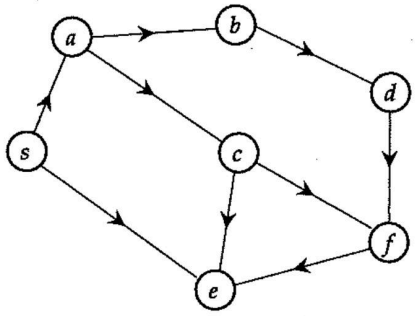
A. G cannot be linearized.

B. Let $u \in V$ be an arbitrary vertex. Then u can reach every other vertex in G.

C. $|E| \geq |V|$.

D. $|E| \leq |V|^2/2$.

12. (3 pts.) How many linearizations does the following directed acyclic graph (DAG) have?



s a b d f e (3)

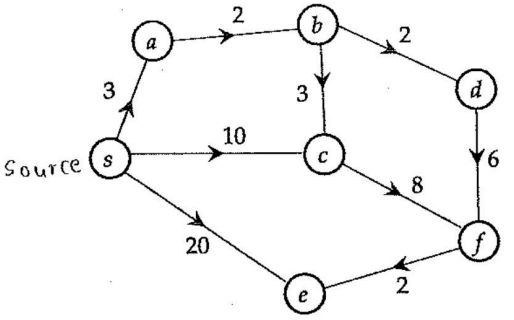
A. 5

B. 4

C. 3

D. 2

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



Alphabetical order

Dijkstra's possible since no l(e) <

(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.

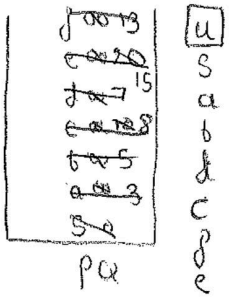
(b) Give the order of vertices following which they are removed from the priority queue.

(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.

(a)

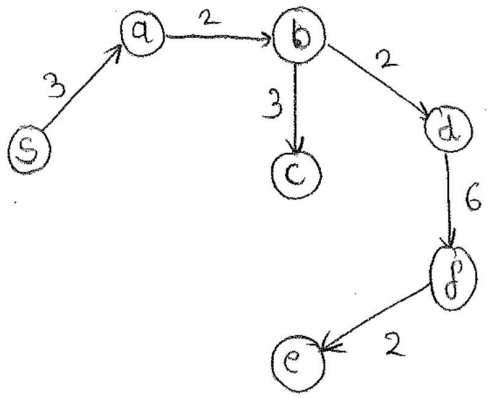
iterations	s	a	b	c	d	e	f
init	0	∞	∞	∞	∞	∞	∞
1	0	3	∞	10	∞	20	∞
2	0	3	5	10	∞	20	∞
3	0	3	5	8	7	20	∞
4	0	3	5	8	7	20	13
5	0	3	5	8	7	20	13
6	0	3	5	8	7	15	13
7	0	3	5	8	7	15	13



(b) Order of Removal ⇒ s a b d c f e

Continued on next page (6) →

(c)



Shortest path Tree

(d) Yes it is unique for this graph G

One graph, can have multiple ways to reach a node with the same amount of least time. But this graph does not.

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.

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

(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.

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) Proof: Consider array $A[1 \cdots n]$ where $n \geq 3$. Let's use Induction.

Base Case ($n=3$): When $A[1,2,3]$ array exists then according to the definition of array A:
$A[1] < A[2]$ & $A[2] > A[3]$, making $A[2]$ a peak.

Inductive Hypothesis ($n=k$): Let us assume in $A[1 \cdots k]$ where $3 \leq k$, there exists at least one peak. We know that $A[k] < A[k-1]$

Inductive Step: ($n=k+1$): Now consider $A[1 \cdots k+1]$ where we know that $A[k+1] < A[k]$

elif ~~$y > z$~~ $y \geq x$!
~~$x > y$~~

5 6 6 2 3 7 5 8 3 3 4 10 6
i m j

x y z
5 6 6 2 3 7 5
i m j

2 3 7 5

(a) Proof: Let's proceed by proof by Contradiction, and let's assume Given array $A[1 \cdots n]$, $n \geq 3$ there exists no peak.

Now, we know since there exists no peak ~~is~~ means

$\forall i$, $3 < i < n$ we will either have $A[i-1] < A[i] < A[i+1]$

or $A[i-1] > A[i] > A[i+1]$.

Without Loss of generality Lets consider ascending order.

$\forall i, 3 < i < n$, $A[i-1] \leq A[i] < A[i+1]$

when $i = n-1$ we get $A[n-2] < A[n-1] < A[n]$

But from Def of array A we know that $A[n] < A[n-1]$

which is a contradiction & our assumption is wrong.

Thus, in A at least one peak exists □

b) func find_peak (A[1 ··· n], i, j)

```
if i+2 ≥ j : return 'Peak not Found';
m = ⌈(i+j)/2⌉
x = A[m-1]
y = A[m]
z = A[m+1]
if y ≥ x and y ≥ z :
    return 'Peak Found';
elif x > y:
    find_peak (A[i ··· m], i, m);      // i, j have to hold start, end of
elif z > y:                                the new matrix A.
    find_peak (A[m ··· j], 1, j);
endif
```

end func

(This runs in $O(\log n)$ time)

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: Consider a graph G with 'K' universal vertices in it.
$G = (V, E)$ where $V = [u_1, u_2 \cdots u_n, v_1, \cdots v_K]$

~~For~~ $\forall i, 1 \le i \le K$, all vertices in $V - \{v_i\}$ can reach v_i.
v_i are universal vertices.
All the v_i vertices can reach each other so they form a connected component, we now prove that $CC_U = \{v_1, v_2 \cdots v_K\}$ is maximal, where CC_U = CC of universal vertices

Consider a vertex $u^* \in V$ that is not a universal vertex.
For the sake of proof by contradiction Lets assume $u^* \in CC_U$ which implices that u^* is reachable via all v_i universal vertices. All v_i are reachable by all $u_j \in V, 1 \le j \le n$, by extention then all u_j can reach u^*. This would make u^* a universal vertex as it is reachable from all v_i and u_j. This is a contradiction.

So we can say that $v_i, \forall i, 1 \le i \le K$ form a maximal CC. □

(b) Explanation:

All the universally reachable vertices in graph G will form a connected component and in G_M all connected components will have an edge to the CC with all the universal vertices.

func find_universal (G=(V,E)):

① Run DFS on the special order (PostListR) to get the connected componets and form the meta-graph G_M;

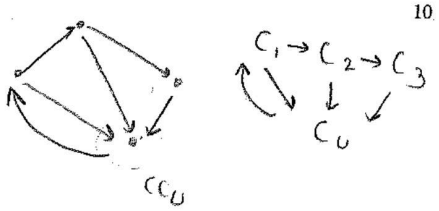
② We now just have to find the node in G_M that is reachable from all other nodes.
So we perform func_Linearization_of_DAG which will give you the Linearization of the G_M (x) the universal component has to be at the end of X since it is reachable from every node in G_M

③ Return Last node of X.

Time Complexity:

Step ① runs in $\Theta(|V|+|E|)$ time since it is DFS
step② also runs in $\Theta(|V|+|E|)$ time

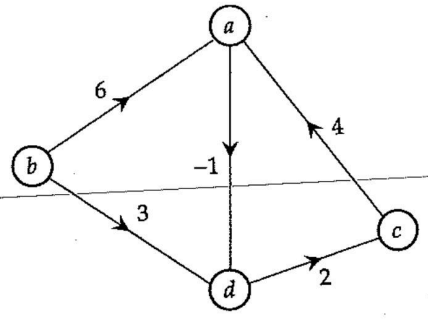
So time complexity → $\Theta(|V|+|E|)$



16. **(4 + 12 pts.)** Let $G = (V, E)$ be a directed graph with possibly negative edge length, but without negative cycle. Let $a \in V$. 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	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) def distance_a (G=(V,E), L(e) ∀ e∈E, a∈V):

① Perform Bellman ford from source vertex a to get shortest distance from 'a' to all vertices in array dist_from_a

② Reverse the graph G to G_R and perform Bellman for once again with source 'a' to get min distance of a node in G from a in array dist_to_a.

③ the sum of the proper values from the two arrays of size |V| will give us the distance$_a$(u,v)

Running Time :

Step ① Runs in $\Theta(|V| \cdot |E|)$ time

Step ② Runs in $\Theta(|V| \cdot |E|)$ for bellman ford and Reversing G to G_R takes $\Theta(|V|)$

Step ③ Runs in $\Theta(|V|)$ time

Total run time is $\Theta((|V|+|E|) \cdot |V|)$

Correctness :

For any path in the graph $p = u, v$ the path has to pass through 'a' for our required results.

Let P_1 = Path $(u, a) = l_1$, P_2 = path $(a, v) = l_2$ as path costs

Bellman-ford on G & source a provides us with value of l_2 for the path P_2

Bellman-ford on G_R & source a provides us with values of l_1 on the reverse path P_1 (a, u) however reversing this does not affect the value of l_1

So we get $p(u, v) = l_1 + l_2$ the correct shortest path since in G_R only the direction of $e \in E$ change not their Lenghts.