Model

University of Petra	جامعة البترا	جامعة البترا - ثلاثون عاما University of Petra
Faculty of Information Technology		كلية تكنولوجيا المعلومات
Department of Computer Science	جامعة البترا	قسم علوم الحاسوب

Advanced Algorithms
601326
Final Exam – 2024 2

Your Name: …….. **Your ID: ……**

Your Instructor Name: ……

Instructions for the Exam:

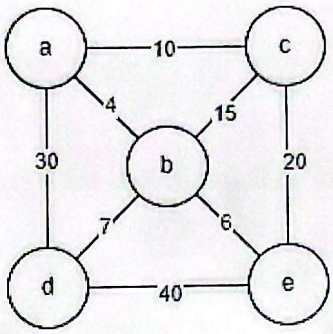
- Write your name and ID number on the exam and answer sheets.
- Write the number of the section that you enrolled in.
- Write the name of your instructor.
- Questions in the exam not allowed.
- Using any type of technology (mobiles, smart watches, etc.) not allowed
- Using extra papers or sheets not allowed

For instructor use only:

Question number	Course ILO	Program ILO	Question weight	Student mark
Q1			5	
Q2			8	
Q3			3	
Q4	I2		4	
Q5			5	
Q6			4	
Q7	I2		6	
Q8			3	
Q9			7	
Total			45	

This exam has 9 Questions. The total mark is 45

Question 1) Apply **Prim's** algorithm to the following graph to find **MST** showing detailed steps.
(5 marks)



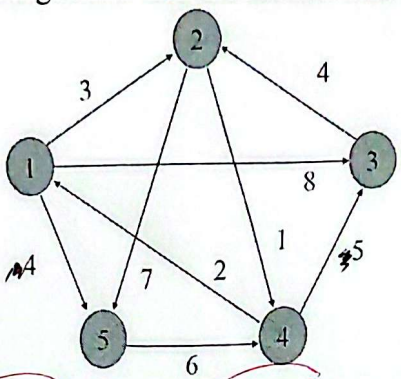
student can start from any node, show the details of the priority queue, and then draw the MST.

Question 2) **(8 marks)**

a. Calculate the shortest path from vertex *1* to each vertex of the below graph using Dijkstra's algorithm

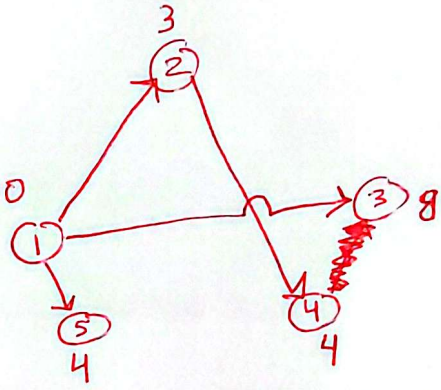
b. Give example shows that Dijkstra's algorithm may not work for a weighted connected graph with negative weights.

c. Discuss the time complexity of the algorithm and the factors affecting it.

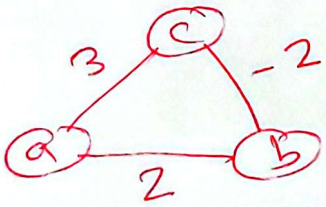


a.

V_T	$V-V_T$			
1(-,0)	2(1,3)	3(1,8)	4(-,∞)	5(1,4)
2(1,3)	3(-,∞)	4(2,4)	5(2,10)	
4(2,4)	3(4,9)	5(4,∞)		
5(1,4)	3(5,∞)			
3(1,8)				



b.



As the shortest path from a to b. Dijkstra's algorithm yields a-b of length 2 which is longer than a-c-b of length 1.

c. The time complexity of Dijkstra's algo. depends on data structure used for implementing the priority queue and for representing input graph itself.

unordered ~~list~~ array + weight matrix $\Longrightarrow O(V^2)$

adjacency list + min heap $\Longrightarrow O(E \log V)$

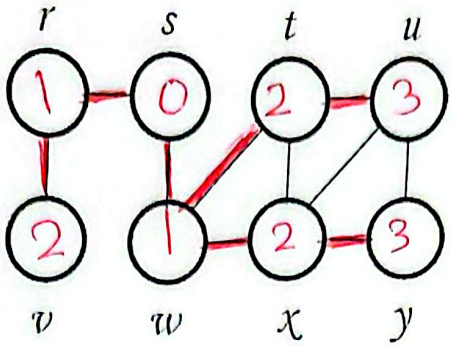
Question 3) Based on your understanding of Floyd's algorithm (for weighted graphs), write a recursive relation for finding R^k_{ij}. **(3 marks)**

$$R^k_{ij} = \begin{cases} w_{ij} & , k = 0 \\ \min\left(R^{k-1}_{ij} , R^{k-1}_{ik} + R^{k-1}_{kj}\right) & , k \geq 1 \end{cases}$$

Question 4) Fill in the bellow table best and worst cases of the following algorithms: **(4 marks)**

	Insertion Sort	Quick Sort	Heap Sort	Merge Sort
Best Case	n	$n \log n$	$n \log n$	$n \log n$
Worst Case	n^2	n^2	$n \log n$	$n \log n$

Question 5) Given the below graph, apply the BFS algorithm to find **distance** from source vertex *s* to all other vertices and find **BFS tree** **(5 marks)**



Q = {s}
= {r, w}
= {w, v}
= {v, t, x}
= {t, x}
= {x, u}
= {u, y}
= {y}
= { }

Question 6) Why can **NOT** the Traveling Salesman Problem (TSP) be solved efficiently using a greedy algorithm? Provide an example to support your answer. **(4 marks)**

Greedy algorithms do not reconsider previous decisions, and in TSP, visiting a near city early might block a better tour that could have been formed later. TSP is global, but the greedy algorithm uses only local information which may miss the best overall route. Greedy algorithms for TSP may quickly find a tour, but there is no guarantee that this tour is optimal. TSP is an NP-hard problem and solving it efficiently requires considering global path optimization, which greedy algorithms do not do. Greedy works locally and that's not enough for TSP.

ation

Question 7) Discuss the difference between the following: **(6 marks)**

- **P** and **NP** Problems
- **Decision** and **Optimization** problems
- **Branch & Bound** and **Brute Force** techniques

- P: problems solvable in polynomial time.
  NP: are problems verifiable in polynomial time.

- Decision problem is a problem that has yes/No answer and it's easier to analyze mathematically than optimization problems, which are problems where we seek the best solution according to some metric (often harder than decision problems).

- Branch & Bound is a systematic way of solving optimization problems by exploring promising paths (branches) and eliminating (bounding) paths that can't lead to better solutions. It's smarter alternative to Brute Force technique that explores all possible solutions.

Question 8) **(3 marks)**

Write the **recursive formulation** for the 0/1 Knapsack Problem for a Dynamic Programming solution

i: row
w: column
b_i: benefit
v_i: value

$$V_{i,w} = \begin{cases} 0 & \text{for } i=0 \text{ or } w=0 \\ V_{i-1,w} & \text{if } w_i > w \\ \max\left(V_{i-1,w}, V_{i-1,w-w_i} + b_i\right) & \text{otherwise} \end{cases}$$

Question 9) **(7 marks)**

Design greedy algorithm for solving the **fractional knapsack** problem:
given weights and values of **n** items, put these items into a knapsack of capacity **W** to get the maximum total value in the knapsack (note: **you can take fractions of an item**).

- Student can calculate $\frac{\text{value}}{\text{weight}}$ ratio for each item for considering both value and weight of each item
- then, sort items by decreasing ratio
- add as much of the item as possible to the knapsack.

1) for each item, compute v_i/w_i $\quad i=1 \rightarrow n$
2) sort all items in decreasing order based on their value-to-weight ratio
3) initialize two variables
   current_weight = 0 $\quad$ total_value = 0
4) repeat for each item in the sorted list:
   if adding the full item keeps the total weight $\leq W$
   - add the full item to the knapsack
   - increase the current_weight by the item's weight
   - increase the total_value by the item's value

   else
   - take the fraction of the item that fits
   - compute the fraction as $(W - \text{current\_weight})/w_i$
   - set current_weight to W (i.e. knapsack is full)
   - Break

Good Luck