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

Advanced Algorithms

601326

Final Exam – 2024-2

Your N...

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

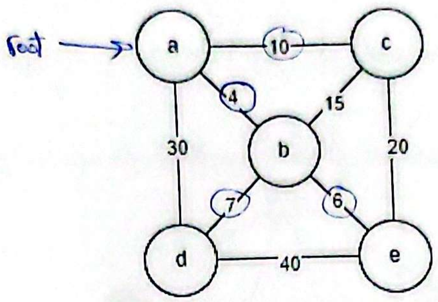
<u>For instructor use only:</u>

Question number	Course ILO	Program ILO	Question weight	Student mark
Q1			5	5
Q2			8	7.25
Q3			3	1
Q4	I2		4	3.75
Q5			5	5
Q6			4	4
Q7	I2		6	4.25
Q8			3	2.25
Q9			7	7
Total			45	39.5

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) 5



Q = { }

at a: Q = {4, 10, 30}

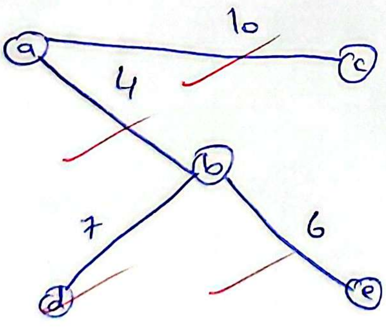
at b: Q = {6, 7, 10, 15, 30}

at e: Q = {7, 10, 15, 20, 30, 40}

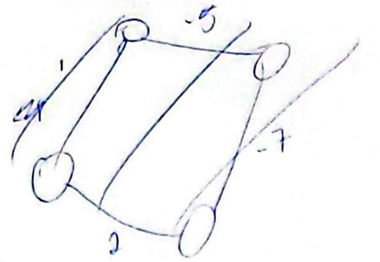
at d: Q = {10, 15, 20, 30, 40}

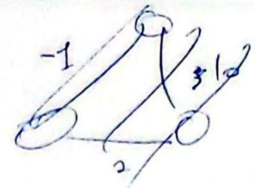
at c: Q = {15, 20, 30, 40} → all will produce cycles.

MST:



Cost = 10 + 4 + 6 + 7 = 27





7.25

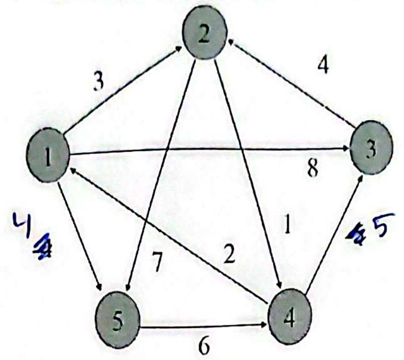
Question 2) **(8 marks)**

3.75

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.

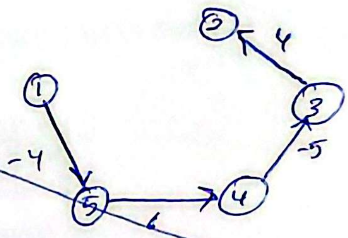
c. Discuss the time complexity of the algorithm and the factors affecting it.



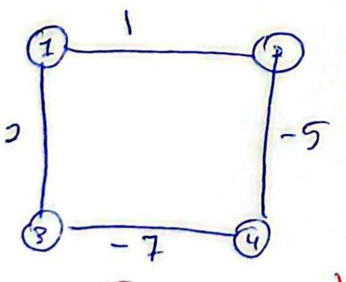
V N

a)

1 (-, -)	2(1,3), 3(1,8), 4(1,∞), 5(1,-4)
5 (1,-4)	2(5,∞), 3(5,∞), 4(5,2)
4 (5,2)	2(4,∞), 3(4,-3)
3(4,-3)	2(3,1)
2(3,1)	



b)



the negative weights will cause an endless loop between nodes 2,4,3 causing inaccurate results & errors.

1.5

c) Using min-heaps and adjacency list Dijkstra's will have a time complexity of O(E log V), however using adjacency matrix and fibbonacci heap it will have a time complexity of O(V²).

2

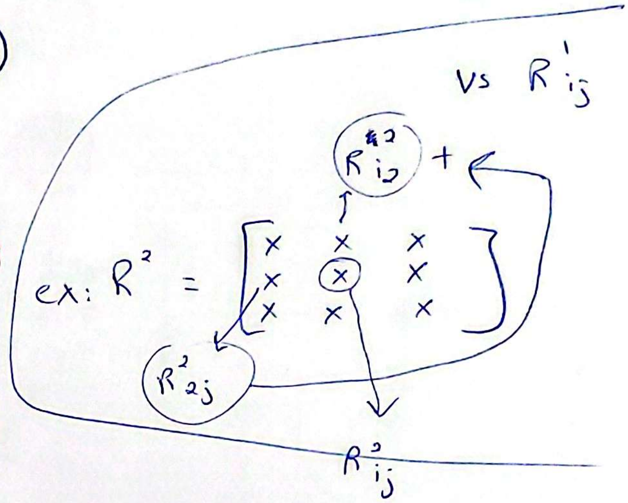
Question 3) Based on your understanding of Floyd's algorithm (for weighted graphs), write a recursive relation for finding R^k_{ij}. **(3 marks)**

1

$$T_n = \min\left(R^{k-1}_{ij},\ [R^k_{ik} + R^k_{kj}]\right)$$

?!

$$T_n = \begin{cases} \infty \ ?! \ , & R^{k-1}_{ij} = \infty \text{ OR } R^k_{ik} = \infty \text{ OR } R^k_{kj} = \infty \\ \min\left(R^{k-1}_{ij},\ [R^k_{ik} + R^k_{kj}]\right., & \text{otherwise} \end{cases}$$



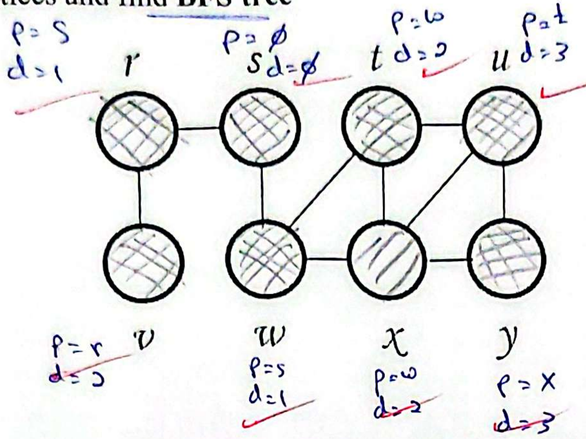
Question 4) Fill in the bellow table best and worst cases of the following algorithms: **(4 marks)**

3.75

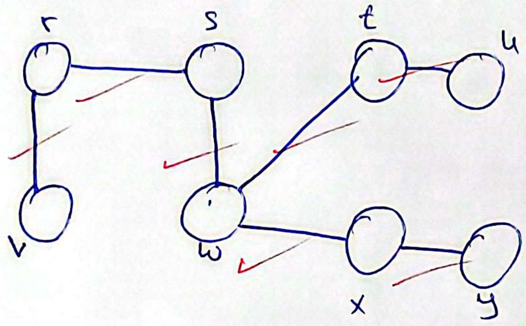
	Insertion Sort	Quick Sort	Heap Sort	Merge Sort
Best Case	$O(n)$	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Worst Case	$O(n^2)$	$O(n^2)$	$O(n\log n)$	$O(n)$

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)**

5



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

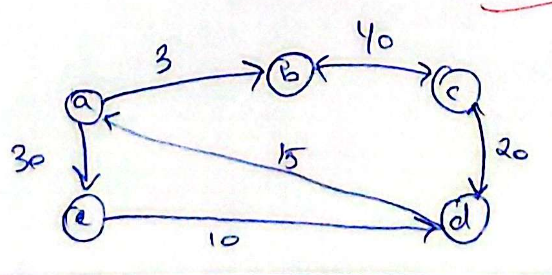


n	p	d
s	∅	0
r	s	1
w	s	1
v	r	2
t	w	2
x	w	2
u	t	3
y	x	3

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)**

4

If the traveling salesman takes the locally optimal (immediate shortest) path he might start on a path where going back to a previous city (node) would be impossible without going through the starting point, therefore ending the tour before visiting all cities.



→ applying greedy technique the salesman will visit b and be unable to return to e.

4.25

(6 marks)

Question 7) Discuss the difference between the following:

a- • **P** and **NP** Problems
b- • **Decision** and **Optimization** problems
c- • **Branch & Bound** and **Brute Force** techniques

a) P problems are problems that can be solved using algorithms in polynomial time, while NP problems do not have a solution in polynomial time yet, making them non-deterministic polynomial problems. 1.75

b) Decision problems are answered by either yes or no, for example: does a graph have a hamiltonian ~~cycle~~ Path? (yes/no), whereas Optimization problems ~~answer~~ provide ~~a~~ more elaborate answers where a solution is needed, for example: find the hamiltonian cycle for a particular graph. 1.5

c) Branch & bound technique tries all possible paths to a solution before picking the most optimal one, whereas brute force technique insists on taking the first solution and keeps going until reaching an output. 1

(3 marks)

Question 8)

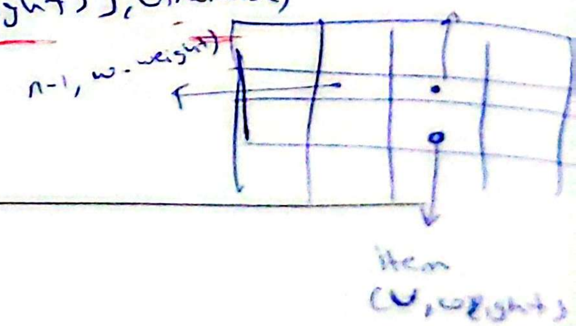
Write the **recursive formulation** for the 0/1 Knapsack Problem for a Dynamic Programming solution 2.25

$$T(n) = \{ \max(T(n-1, w), [i.weight + T(n-1, w - i.weight)])$$

?! when?!

$$T_n = \begin{cases} 0 & , T(n-1, w) = 0 \ \& \ item.weight > Capacity \\ \max(T(n-1, w), [i.weight + T(n-1, w - i.weight)]) & , otherwise \end{cases}$$

0.5 0.75 (n-1, w) 1



ce

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**).

7

```
Fractional knapsack ( A[i(w,v) ... n(w,v)], capacity )
  items_added = {} ;  ----> empty array for items we used
{ while (capacity > 0)
   {
     max_value =
        -item  A[i(w,v)];
        max_value = A[i].value;
     for (index=0; index < A.length; index++)
       {
         if (A[index].value > max_value)
          {
            max_value = A[index.value];
            max_value_item = A[index (w,v)];
          }
       }
     if (max_value_item.weight ≤ capacity)
      {
        items_added.append (max_value_item);
        capacity = capacity - max_value_item.weight ;
        A.remove (max_value_item);
      }
     else if (max_value_item.weight /2 ≤ capacity)
      {
       A.remove (max_value_item);
       capacity = capacity - (max_value_item.weight /2);
       items_added.append (max_value_item [w/2, v/2]; }

     else
       {
         A.remove (max_value_item);
       }
   }
}
```

...ore ...n ...th ...x value ...gardless ...f weight. → ...reedy.

Capacity = 7

item 1: (3, 2)
item 2: (4, 5)
item 3: (2, 1)

greedy: highest value item no matter weight cost

...f the item with the max value fits in the knapsack, add it to added items list and remove from possible items (can only be used once) and decrease max capacity

otherwise tests to see if half the item fits, if it does we take it

otherwise remove item from possible items to take because it is too large

Good Luck