University of Petra		
Faculty of Information Technology		**كلية تكنولوجيا المعلومات**
Department of Computer Science		**قسم علوم الحاسوب**

Advanced Algorithms
601326
Final Exam – 2025 1

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

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

Question number	Course ILO	Program ILO	Question weight	Student mark
Q1			5	
Q2			5	
Q3			2	
Q4	I2		4	
Q5			5	
Q6			5	
Q7	I2		4	
Q8			3	
Q9			7	
Total			40	

This exam has 9 Questions. The total mark is 40

Question 1) Choose the correct answer for each of the following: (5 marks)

1. What is time complexity of TSP using Branch and Bound in the worst case? (where n is number of vertices and E is number of edges)
   a) O(1)
   b) O(n*E)
   c) $O(n^2)$
   d) O(n!)

2. What is the space complexity of the Insertion Sort algorithm?
   a) O(1)
   b) $O(n^2)$
   c) O(n)
   d) O(n log n)

3. Given Build_Max_Heap Algorithm:

```
Build_Max_Heap(A){
    n= length(A)
    for i= n/2 down to 1
        max_heapify(A, i, n)
}
```

Based on your understanding of the algorithm, why does i start from n/2 ?

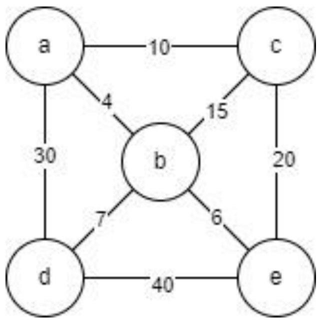
   a) to get value of non-leaf vertices
   b) to get index of non-leaf vertices
   c) to divide A into two sub-arrays and sort each one separately
   d) to call max_heapify only twice
   e) none of the above

4. Which of the following algorithms design techniques support backtracking:
   a) Divide and conquer
   b) Greedy programming
   c) Dynamic programming
   d) None of the above

5. What is the time complexity of the Dynamic algorithm of 0/1 Knapsack Problem? (where n is number of items, and W is knapsack capacity)
   a) O(n+W)

b) O(n*W)
c) O(n log W)
d) O(n^2)

Question 2) Apply **Prim's** algorithm to the following graph to find **MST** showing detailed steps. **(5 marks)**



Answer:
A student can start from any vertex randomly, show priority queue, details of adding one vertex at a time until ending with the MST.

Question 3) Based on your understanding of Floyd's algorithm (for weighted graphs), write a recursive relation for finding R^k_{ij} . **(2 marks)**

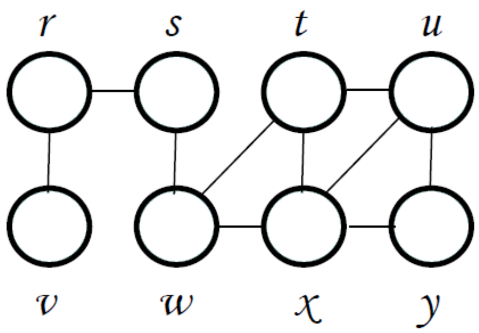
Answer:

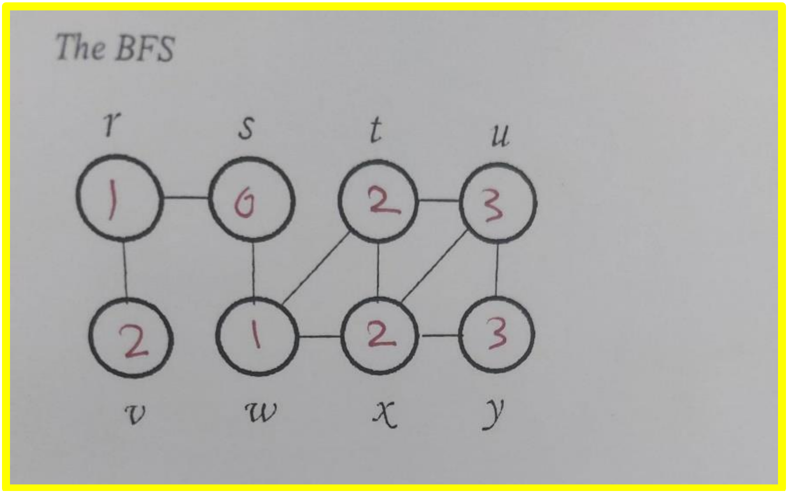
$$\mathbf{R^k_{ij}} = \begin{cases} w_{ij} & ,k = 0 \\ \blacksquare & \\ min(R^{k-1}_{ij},\ R^{k-1}_{ik} + R^{k-1}_{kj}) & ,k \geq 1 \end{cases}$$

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

	Breadth-First Search	**Quick Sort**	**Selection Sort**	**Recursive Towers of Hanoi**
Best Case	**O(V+E)**	**O(nlogn)**	**O(n^2)**	**O(2^n)**
Worst Case	**O(V+E)**	**O(n^2)**	**O(n^2)**	**O(2^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)**





Question 6) Given the following set of characters and their frequencies: **(5 marks)**

Character	A	B	C	D	E	F
Frequency	5	9	12	13	16	45

(a) Construct the Huffman coding tree for the given characters by clearly showing details

(b) From the constructed tree, determine the binary Huffman code for each character.

(a) Huffman coding tree construction:

1. **Pick lowest two:**
   A(5) and B(9) ⟶ merge ⟶ **(AB)(14)**

2. **Queue now:**
   C(12), D(13), (AB)(14), E(16), F(45)

3. **Pick lowest two:**
   C(12) and D(13) ⟶ merge ⟶ **(CD)(25)**

4. **Queue now:**
   (AB)(14), E(16), (CD)(25), F(45)

5. **Pick lowest two:**
   (AB)(14) and E(16) → merge → **(ABE)(30)**

6. **Queue now:**
   (CD)(25), (ABE)(30), F(45)

7. **Pick lowest two:**
   (CD)(25) and (ABE)(30) → merge → **(ABCDE)(55)**

8. **Final merge:**
   (ABCDE)(55) and F(45) → **Root (100)**

(b) Huffman Codes

Character	**Huffman Code**
F	0
C	100
D	101
A	1100
B	1101
E	111

Question 7) Discuss **briefly** the **difference** between the following: **(4 marks)**

- **P vs NP problems:**
P problems can be solved in polynomial time, while **NP** problems can be verified in polynomial time.

- **Decision vs Optimization problems:**
Decision problems ask for a yes/no answer, whereas **optimization** problems ask for the best (minimum or maximum) solution.

- **Branch & Bound vs Brute Force techniques:**
Brute Force explores all possible solutions, while **Branch & Bound** prunes subproblems using bounds to avoid unnecessary exploration.

- **Traveling Salesman Problem vs Hamiltonian Cycle:**
TSP seeks the minimum-cost Hamiltonian cycle, whereas a **Hamiltonian Cycle** only checks for existence without considering cost.

Question 8) **(3 marks)**

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

Let:

- n = number of items
- W = knapsack capacity
- wi = weight of item i
- vi = value (profit) of item i

$$K(i,w) = \begin{cases} 0, & \text{if } i = 0 \text{ or } w = 0 \\ K(i-1,w), & \text{if } w_i > w \\ \max\left(K(i-1,w),\ v_i + K(i-1,w-w_i)\right), & \text{if } w_i \le w \end{cases}$$

Question 9) **(7 marks)**

Consider the problem of scheduling n jobs of known durations t1, ..., tn for execution by a single processor. The jobs can be executed in any order, one job at a time. You want to find a schedule that minimizes the total time spent by all the jobs in the system. (The time spent by one job in the system is the sum of the time spent by this job in waiting plus the time spent on its execution.)

a) Design a greedy algorithm for this problem.
b) Does your greedy algorithm always yield an optimal solution?

a. Sort the jobs in nondecreasing order of their execution times and execute

them in that order.

b. Yes, this greedy algorithm always yields an optimal solution. Indeed, for any ordering (i.e., permutation) of the jobs i1, i2, ..., in, the total time in the system is given by the formula ti1 + (ti1 + ti2) + ... + (ti1 + ti2 + ... + tin) = nti1 + (n − 1)ti2 + ... + tin. Thus, we have a sum of numbers n, n−1,...,1 multiplied by “weights” t1, t2, ...tn assigned to the numbers in some order. To minimize such a sum, we have to assign smaller t’s to larger numbers. In other words, the jobs should be executed in nondecreasing order of their execution times.

Good Luck