Confidential

Faculty of Engineering and IT
School of Information Technologies
COMP 2007: Algorithms and Complexity

Student Details (to be filled in by the candidate)	
seat number	
full name	
other names you use	
SID	

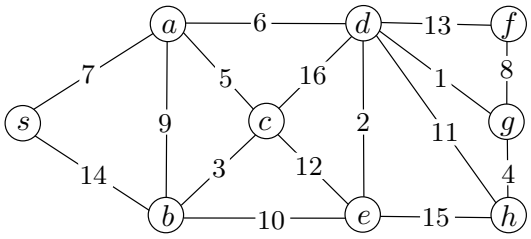
Exam information and instructions:

- 10 minutes reading time, 2:30 hours exam
- the paper comprises 11 pages
- you are not allowed to use any electronic devices
- this exam paper must not be removed from exam room

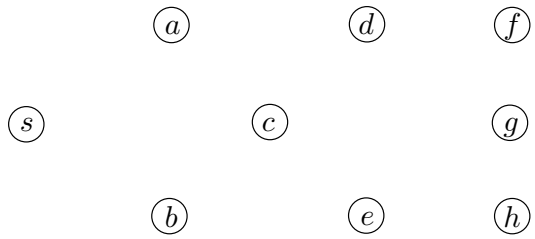
Results (for office use only)						
Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	Total
/10	/10	/10	/10	/10	/10	/60

- **Question 1 (10 points): Graphs**

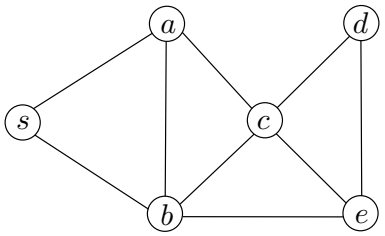
(a) Draw a minimum spanning tree of the weighted graph using Prim's algorithm, starting at node s. Indicate the order in which the algorithm adds the edges to the solution.



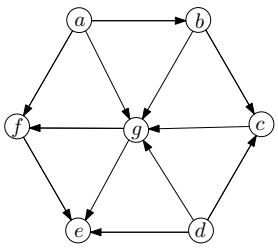
Answer:



(b) Show a minimum vertex cover of the below graph.



(c) Draw a topological order of the below graph.



Answer:

- **Question 2 (10 points): Interval scheduling**

  (a) Describe a greedy algorithm that solves the interval scheduling problem.

  Answer:

  (b) Argue why the greedy algorithm outputs a correct solution.

  Answer:

Question 2(c) →

(c) Given the input instance shown below (Fig. 1(c)), state the optimal schedule produced by your algorithm for Question 2(a).

Answer:

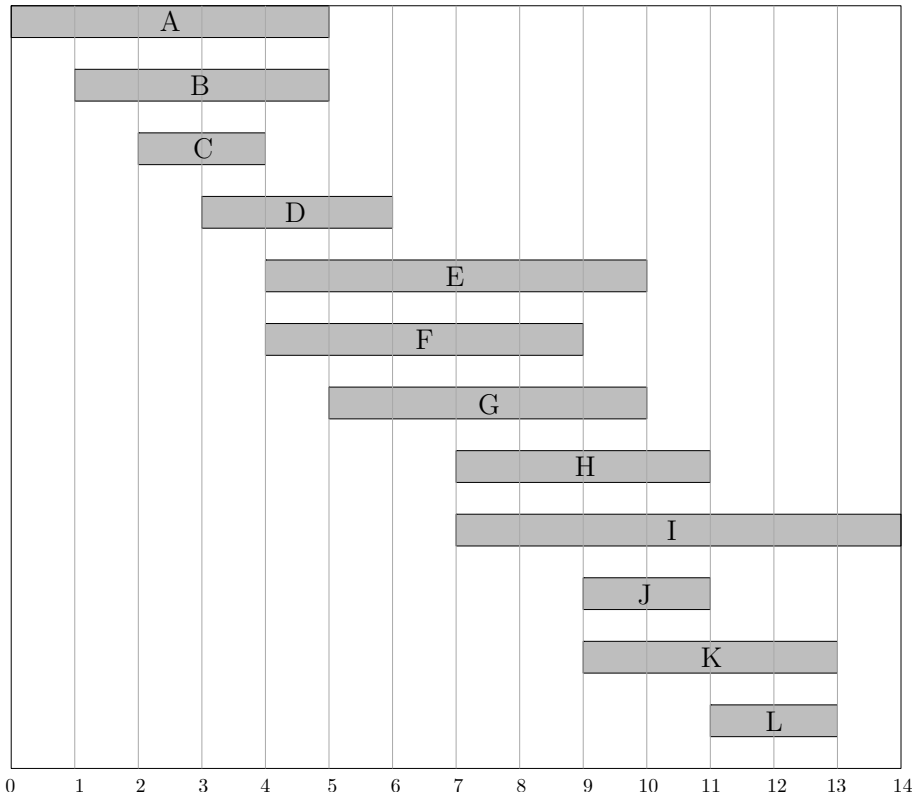


Figure 1: The input instance to Question 2.

- **Question 3 (10 points): Divide and Conquer**

Given an array $A[1..n]$ of integers, an element x in A is said to have the *majority* if and only if the number of x's in A is greater than $n/2$. Consider the following algorithm (also given in pseudocode below). We split the array A into two subarrays A_L and A_R of half the size. We choose the majority element M_L of A_L and the majority element M_R of A_R, if they exist. After that we check if M_L is a majority element of A. If not then we check if M_R is a majority element of A. If none of these are true then the algorithms returns *'no majority'*.

Algorithm 1 MAJORITY

```
1:  function MAJORITY(A[1..n])          ▷ A is an array of n integers
2:      if n = 1 then
3:          return A[1]
4:      end if
5:      Let A_L be the first half of A
6:      Let A_R be the second half of A
7:      M_L = MAJORITY(A_L)
8:      M_R = MAJORITY(A_R)
9:      if M_L is a majority element of A then
10:         return M_L
11:     end if
12:     if M_R is a majority element of A then
13:         return M_R
14:     else
15:         return "no majority"
16:     end if
17: end function
```

(a) State and solve the recurrence of the algorithm. You can assume lines 9 and 11 requires $O(n)$ time.

Answer:

Question 3(b) →

(b) Argue the correctness of the MAJORITY algorithm.

Answer:

- **Question 4 (10 points): Knapsack**

Consider the knapsack problem as we discussed in class. You are not given the actual input, instead you are given the trace of the execution of the knapsack algorithm. The dynamic programming table B is given below. Recall that $B[k, w]$ is the optimal solution that can be obtained using only the first k items and a maximum allowed total weight of w.

$k \setminus w$	0	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	25	25	25	25
2	0	0	15	15	15	15	15	25	25	40	40
3	0	0	15	20	20	35	35	35	35	40	45
4	0	0	15	20	20	35	36	36	51	56	56

(a) What is the number of items in the input instance?

Answer:

(b) What is the maximum weight limit of the knapsack?

Answer:

(c) What is the weight of item 1?

Answer:

Question 4(d-f) →

(d) What is the value of item 2?

Answer:

(e) What is the value of the best packing having a total weight of at most 8.

Answer:

(f) Which items are included in an optimal solution for $w = 10$?

Answer:

- **Question 5 (10 points): NP-completeness**

  Consider a set $A = \{a_1, a_2, \ldots, a_n\}$ and a collection $B_1, B_2, \ldots, B_m$ of subsets of A, $B_i \subseteq A$. We say that a set $H \subseteq A$ is a hitting set for the collection $B_1, \ldots, B_m$ if H contains at least one element of from each B_i. The hitting set problem is defined as follows: Given a set $A = \{a_1, a_2, \ldots, a_n\}$ and a collection of subsets $B_1, B_2, \ldots, B_m$, and a positive integer k decide whether there exists a hitting set $H \subseteq A$ for $B_1, B_2, \ldots, B_m$ so that the size of H is at most k. Prove that the hitting set problem is NP-complete. Hint: Use the Vertex Cover problem for your reduction.

- **Question 6 (10 points): Dynamic Programming**

  Let $G = (V, E)$ be an undirected graph with edge weights $w : E \to Z^+$ (positive integers). Recall that a matching $M \subseteq E$ is a subset of edges such that no two edges in M are incident on the same vertex. The maximum weight matching problem is to find a matching M maximizing the sum of the weights of the edges in M, that is, maximize $\sum_{e \in M} w(e)$.

  Your task is to design a polynomial time algorithm for solving the maximum weight matching on *binary* trees using dynamic programming. Remember to:

  (a) Clearly define your DP states.

  Answer:

  (b) State the recurrence (base and recursive cases).

  Answer:

Question 6(c) →

(c) Analyze the time complexity of your algorithm.

Answer:

(end of exam)