CS2516 SAMPLE OPEN BOOK EXAM

|  |  |
| --- | --- |
| **Instructions to Candidates** | Answer all questions  Total Marks: **80**  This is an Open Book exam - you are free to consult lecture slides, lab solutions, textbooks, and online sources. However, as always, the work you submit must be your own, and plagiarism rules still apply.  The answers will be submitted on Canvas using Microsoft Word, as per University instructions. When entering your answers into Word, don't waste time using fancy formatting, or colours or font sizes, or tables or equation editors, and don't try to draw pictures using the picture editor - apart from the length of time it will take, the tools often behave strangely, shifting things around and off the page, and sometimes it crashes. Just use plain text, unless you are specifically asked to use some formatting. In CS2516, this is only likely to be highlighting some specific characters, numbers, words or phrases in **bold** or *italics* or underlined*.* |
| **Duration of Paper** | 90 minutes, plus University-mandated 'uploading time' |
| **Special Requirements** | The use of electronic calculators is permitted |

**Q1. 40 marks**

(i) What are the worst-case time complexities of the following algorithms, in terms of *n*, the length of the input?

(a) *InsertionSort*

(b) *HeapSort*

(c) *QuickSort*

*(6 marks)*

(ii) For the in-place versions of each of the algorithms in part (i), sort the list below. For each algorithm, show the intermediate list that results after each main iteration on a separate line, with each item that is guaranteed by the algorithm to be in its final position marked in bold, and report the number of comparisons between list items for each iteration at the end of the line in brackets. At the end, report the total number of comparisons.

For Heapsort, show the results after each addition or selection of a value.

For QuickSort, do not use any randomisation, and highlight the pivot in italics.

*8 15 3 23 7 19 25 4 12 18*

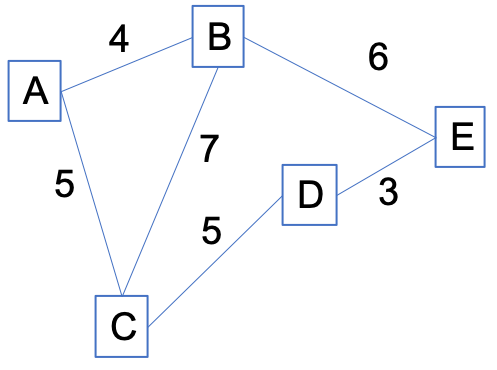
*(26 marks [6 for InsertionSort, 10 for Heap, 10 for Quick])*

(iii) Consider a list of *n* distinct integers. We want to identify the two integers in the list which are closest together in value – that is, find the pair of integer values *x* andy in the list that minimise |*x – y*|. Write an efficient algorithm which will answer this question for any size of input list. What is the worst-case complexity of your algorithm? Justify your answer. Answers will be judged on algorithm clarity, efficiency, and the complexity justification.

*(8 marks)*

**Q2. 40 marks**

(i) Give a complete representation of the following graph using the *AdjacencyList* implementation of the *Graph* ADT (Abstract Data Type).



Make sure you represent the entire graph as a single thing - e.g. if there are *n* vertices, don't give *n* independent lists - enclose them in some suitable data structure.

Give a brief description of any other class or ADT that you use inside to represent information like labels or weights, and briefly explain the notation you will use to represent them inside other structures.

*(8 marks)*

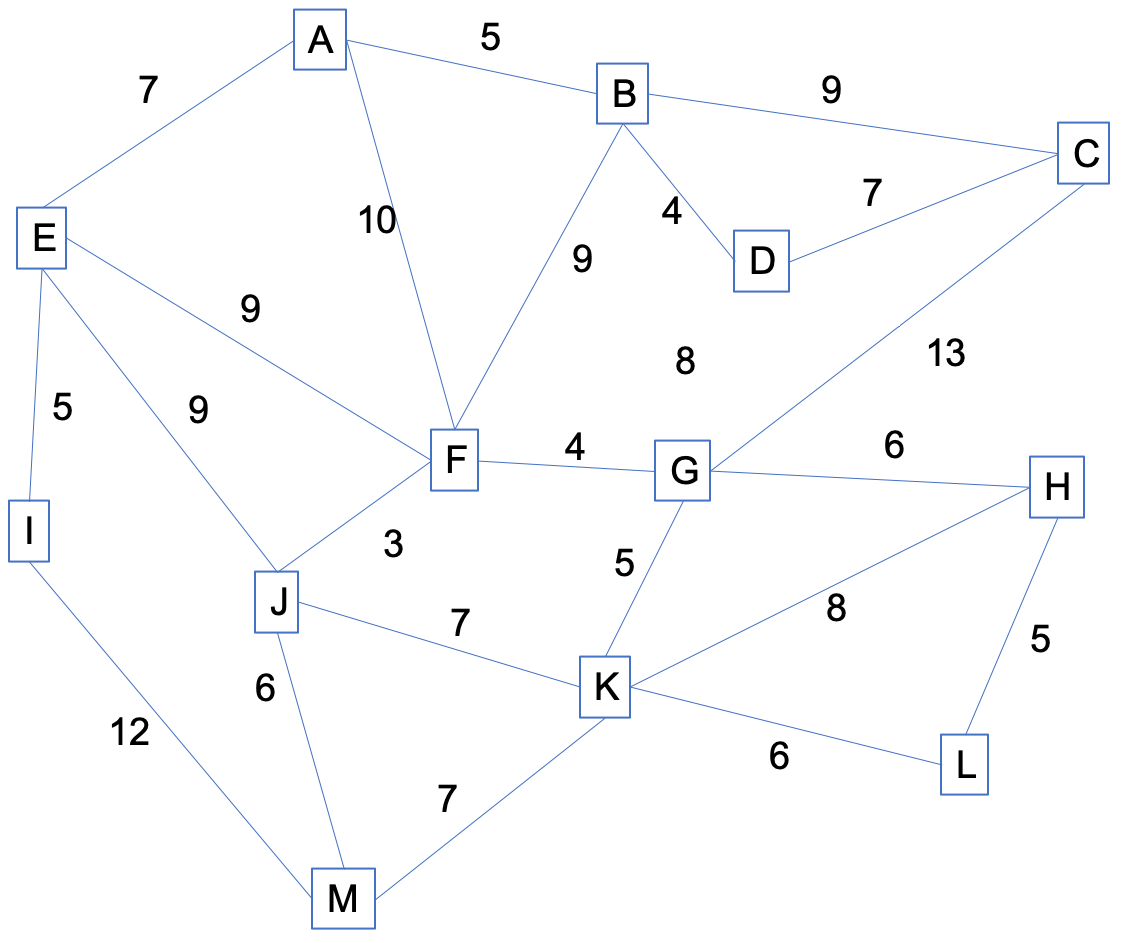
(ii) Explain how you would use your Graph representation of part (i) to obtain the following, specifying any method calls or pseudocode that you need:

(a) a vertex object with label matching *x*

(b) a list of all edges incident on a vertex *x*

*(4 marks)*

(iii) For the following graph, show the progress of *Kruskal's algorithm* for finding a minimum spanning tree.



At each stage, show the how the relevant data structures have changed. If you use an APQ, you should display it as a sorted list (regardless of how it would be implemented underneath). If you use any notation for data structures other than that given in previous questions, you must explain it, and make sure that enough details are shown that would allow someone else to trace through your execution. Explain how to re-construct the final answer from the information you have displayed.

*(10 marks)*

(iv)

An *articulation* vertex in a graph is a vertex such that if it was removed, the graph would no longer be connected. Give clear pseudocode for an algorithm which determines whether or not a connected undirected graph has an articulation vertex. What is the worst-case complexity of your algorithm? Justify your answer. Answers will be judged on algorithm clarity, efficiency, and the complexity justification.

*(8 marks)*

**END OF PAPER**