UNIVERSITY OF BIRMINGHAM

School of Computer Science

First Year - BSc Artificial Intelligence and Computer Science
First Year – BSc Computer Science
First Year – MSci Computer Science
First Year – MEng Computer Science Software Engineering
First Year – BSc Computer Science with Study Abroad
First Year – MSci Computer Science with Study Abroad
First Year – BSc Computer Science with Business Management
First Year – BSc Computer Science with Industrial Year
First Year – MEng Computer Science/Software Engineering with Industrial Year
First Year – BSc Artificial Intelligence and Computer Science with Industrial Year
First Year – BSc Computer Science with Business Management with Industrial Year
Second Year – BA/BSc Liberal Arts and Sciences

06 28345

Data Structures and Algorithms

Summer May/June Examinations 2016

Time allowed: 1 hour 30 minutes

[Answer ALL Questions]

No calculator

Answer all questions

1. (a) List and describe a suitable set of primitive constructors, selectors and conditions that can be used to build and manipulate binary trees. [4%]

   (b) Using any of those primitive operators, and a function max(x,y) that returns the maximum of integers x and y, write in pseudocode a recursive procedure height(t) that returns the height of a tree t. [4%]

   (c) Give an inductive definition of a *Binary Search Tree,* and explain how representing data in that form can make searching easier. How is search complexity affected by using a Binary Search Tree rather than simple linear search? [6%]

   (d) Explain what is meant by a *tree rotation* and why it might be useful to perform one. [6%]

   (e) Draw the binary search tree that results from inserting the items [17, 28, 35, 11, 39, 21] in that order into an initially empty tree, and explain how a tree rotation can usefully be applied to that tree. [5%]

2. (a) Give an intuitive argument that demonstrates what the best possible average case time complexity of a comparison-based sorting algorithm can be. [6%]

   (b) Outline the general idea of *divide and conquer* approaches to sorting, and explain how the *Mergesort* algorithm applies that idea to sort a collection of items stored in an array. [8%]

   (c) State in *Big O* notation, with explanations why, the average-case and worst-case time complexities of *Mergesort* in terms of the number of items *n* to be sorted. [5%]

   (d) Describe what it means to say that a given sorting algorithm is *stable*. Explain whether all *Mergesort* algorithms are stable, with reference to any particular aspects of the algorithm that determine their stability. [6%]

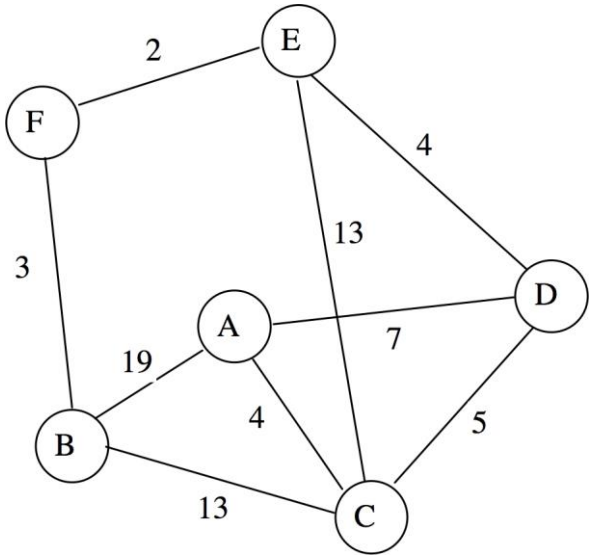
3. (a) Explain what is meant by the terms *hash table*, *hash function* and *hash collision*. [6%]

   (b) Outline why hash tables are useful and what their main disadvantage is. [4%]

   (c) Describe what problem *Direct Chaining* and *Open Addressing* are designed to deal with. Explain how they both work and what advantages and disadvantages they have. [8%]

(d) Illustrate how *Open Addressing* works by designing a simple hash table for storing the student IDs “063982”, “120781”, “149872”, “093574” and “201930”. Show the resulting table, explain any design choices you need to make, and outline how you would scale up your approach to the storage of larger numbers of IDs. [7%]

4. Consider the undirected weighted graph *Graph1* that can be represented as follows:



(a) Explain why *Graph1* is, or is not, a *planar* graph. [3%]

(b) Write down the weight matrix of the *sub-graph* of *Graph1* created by removing all nodes not *adjacent* to node A, and all edges *incident* on the removed nodes. [3%]

(c) Explain how an array-based *Dijkstra’s algorithm* can be used to find the shortest path between two nodes in a given weighted graph. [6%]

(d) The simplest version of Dijkstra’s algorithm has $O(n^2)$ time complexity, where n is the number of nodes in the graph. Explain why, and suggest how that could be improved for sparse graphs. [4%]

(e) Show the content of your Dijkstra’s algorithm arrays at each stage when it is used to find the shortest path from A to B in the above *Graph1*, and state the shortest path found. [9%]