SOFT153
Algorithms, Data Structures and Mathematics

ALGORITHMS ASSIGNMENT

Hand-in Date: 13 May 2019, 12:00 noon

Hand-in of coursework is by submission on the SOFT 153 DLE pages

(Note that there are two submission points on the DLE, one for this part of the coursework and one for the Mathematics part)

Module Leader: Thomas Wennekers
thomas.wennekers@plymouth.ac.uk
School of Computing, Electronics and Mathematics

This assignment is worth 50% of the total coursework mark. (The other 50% come from the Maths part.)

This is an individual assignment: You should complete it entirely on your own.

If you have queries regarding this assignment, please direct them to Thomas Wennekers (twennekers@plymouth.ac.uk) or ask in the labs.

Format of submission:

Implement one project for task 1 and a second project for task 2. For each of the two projects use only a single C# file to implement the requested functions/sub-routines. You can either zip up the two files with the code only, or zip the entire solutions and upload them on the DLE.

Where explanations or tables are asked for, put them in a word file and submit it together with the code.

Do not code graphical user interfaces -- use a console application. GUIs will not attract more marks, but likely make the code less readable. This could reduce marks.

Do not use C#-programming constructs on top of the basic "C-core". The C definition in EBNF form can be found on the SOFT153 DLE pages. Everything that has an equivalent in C# can be used, like basic types, arrays, for, while, if-then-else, arithmetics, logic, plain structs or method-less classes, (static) functions, type casts. Note that C does not have strings, only character arrays. You can use ReadLine and WriteLine.

As a rule of thumb, if you are uncertain about a construct, think about how many algorithmic steps it would need to implement the functionality the construct provides. If that is more than one step you should probably implement the construct yourself (if it is really needed). For example, copying a string needs at least as many steps as the string has characters.

Keep your code simple and clear.

Late assignments policy

You are reminded about the University's policy for late assignments. You can submit for up to 24 hours after the hand-in date, but in this case the assignment can only receive a maximum of pass-level marks (40%). Later assignments will be awarded zero marks. You are, therefore, encouraged to submit the assignment as early as possible before the deadline. The DLE system allows to replace previously submitted files.

This assignment assesses the following learning outcomes:

- Recognise and explain the importance of algorithmic design in optimising use of computing resources.
- Identify suitable structures and algorithms to implement programming tasks.
- Synthesize the solution to a real-world task as a combination of two or more standard

algorithms.

Plagiarism: students are warned that the University takes plagiarism very seriously. You are advised to read the section in your student handbook on Examination and Assessment Offences. As a brief guide, plagiarism is the deliberate use of another person's work without attribution or acknowledgement. DO NOT construct your assignments from code taken from another student or from the Internet. There is no objection to you discussing your solution with other people, but the code you hand in must be your own.

Please read carefully the specifications for the tasks given below – you will lose marks if your coursework does not adhere to what is specified.

Task 1. Doubly linked list with sorting

A doubly linked list is a list with pointers or links in forward and backward direction. It can be traversed in both directions.

Implement sub-tasks a) to c) in one project.

a) Doubly linked list

[30 Marks]

Starting from the singly linked list implementation provided in the lectures and practicals about linked lists, implement a doubly linked list with the following operations

insertBeginning: inserts a node at the beginning of the list

insertEnd: inserts a node at the end of the list

insertAfter: inserts a node after another one that is already in the list

listLength: returns the number of nodes in the list

findNode: finds a node given the reference of the node; returns true if the node has been found and false else.

removeBeginning: removes the first node and returns a reference to it

removeNode: remove a node from the list given the node reference

swapNodes: swap two nodes in the list

printList: print the data in the list

appendLists: append a list at the end of another list

The implementation in a) to c) can use integers as data.

b) Insertion sort and Quicksort

[20 Marks]

Implement insertionSort and quickSort using your implementation from part a). Do not use arrays for the sorting but do the sorting directly on the list data structure. Do not copy values from one node into another node. You will have to compare the values of course.

Quicksort appears in a later lecture. Explanations are also easy to find on the WWW.

c) Run-time Complexity

[10 Marks]

Measure the run-time complexity of your implementations from b) similar as it was done in the practical about run-time complexity. That means, initialise lists of increasing size with random values and measure the run-times for sorting. The run-times should be presented as a table for increasing list size N. You need to provide the code for these tests in your solution.

Compare your run-times with results from implementations of insertion sort and quicksort that use arrays. Use the implementations provided in the labs for this. You do not need to include the code for these tests with arrays, only the run-times, which you should put into the same table as for the list sorting results.

Explain why some algorithms are faster or slower than others are. What is the complexity class in big-O notation of each of these four algorithms?

Task 2: Reading data from a file into a list (and some ‘string’ acrobatics)

[20 Marks]

For this task, use a second project. Duplicate only those functions from Task 1 that are essential to solve Task 2.

Assume you have data records in a file with lines of the form

firstname, lastname, ID

(An example file is provided on the DLE together with these assignment specs.)

Write a program that reads the data in the file character-by-character and puts the data into a doubly linked list of data records.

For this task use character arrays to represent the name-strings and integers for the IDs.

Write one function that sorts the list by either of the record fields. (A function argument could select which sort order is requested.) You can implement helper-functions for sub-tasks you need.

You have to use a StreamReader, StreamReader.Read() and .EndOfStream. You can use type casts, e.g. (char) . Do not use String methods, like Split() or .length. Do not implement exception handling.

The **remaining 20 marks out of 100** are for code efficiency (space and time complexity, 10 marks) and good programming style (10 marks), such as proper layout, identifier names, commenting, code modularisation.

Note: Make your solution as efficient as possible given the specs. Do not implement exception handling, but make sure your functions are reasonably safe to use and do not hang up too easily. Do some reasonable testing in separate test-functions for the different parts of the tasks. These functions you can call in Main() but do not implement a GUI or text-menu or similar; they do not gain marks but easily make the code messy. Layout and comment your code according to principles of good programming practice. Don't use object oriented programming other than method-free classes for the data containers, e.g., called Node, List, etc. There are many examples in the lectures and labs that show how to implement data structures from scratch.