School of Systems Engineering

Assessed Coursework Assignment Brief

**Module code: SE1FA15**

**Lecturer responsible: James Anderson**

**Coursework description: Sorting Floating-Point Objects**

**Work to be submitted on-line via Blackboard by 10:30 am on: Friday 12 February 2016**

**Work will be marked and feedback returned by:**

**[Marked work can be collected from G47 *– delete if returning on-line*]**

**NOTES:**

This coursework should be submitted on-line through Blackboard Learn. **[Microsoft Word document with tracking turned on so that feedback can be given in place.]**

By submitting this work you are certifying that it is all your own work and that use of material from other sources has been properly and fully acknowledged in the text. You are also confirming that you have read and understood the University’s Statement of Academic Misconduct, available on the University web-pages.

If your work is submitted after the deadline, *10%* of the maximum possible mark will be deducted for *each* working day (or part of) it is late. A mark of zero will be awarded if your work is submitted more than 5 working days late. You are strongly recommended to submit work by the deadline as a late submission on one piece of work can impact on other work.

If you believe that you have a valid reason for failing to meet a deadline then you should complete an Extenuating Circumstances form and submit it to the Student Information Centre *before* the deadline, or as soon as is practicable afterwards, explaining why.

**MARKING CRITERIA**

**The assignment details give a list of technical criteria that are to be achieved for a maximum of 5 marks each. There is a bonus of up to 10 marks available to reward any aspect of the work that is particularly well done.**

**ASSIGNMENT DETAILS**

**Implement a program in C++ that sorts a list of floating-point objects. Numbers should be sorted in increasing numerical order. Say how minus zero and Not-a-Number (NaN) objects are to be sorted.**

**Write a document that describes the sorting algorithm you have used. You are free to combine different sorting algorithms in one program. If you use a well-known algorithm, you need only give its name and cite an academic source that describes it. Include your source code and show the input and output of program runs that prove that your program works correctly in each of the test cases listed below. A single program run may satisfy several criteria.**

1. **Give the input and output of a program run that proves it can sort an empty list. [5 Marks]**
2. **Give the input and output of a program run that proves it can sort a list of one element. [5 Marks]**
3. **Give the input and output of a program run that proves it can sort a list of a variable number of elements. [5 Marks]**
4. **Give the output of a program run that proves it can generates minus zero as a floating-point object that is not identical to zero. [5 Marks]**
5. **Give the output of a program run that proves it can generate one NaN. [5 Marks]**
6. **Give the output of a program run that proves it can generate multiple, different NaNs. [5 Marks]**
7. **Describe how your program sorts negative zero relative to zero and the other floating-point numbers. [5 Marks]**
8. **Describe how your program sorts different NaNs relative to each other and to the floating-point numbers. [5 Marks]**
9. **Give the input and output of a program run that proves it can sort multiple repetitions of a single NaN in a list of floating-point numbers. [5 Marks]**
10. **Give the input and output of a program run that proves it can sort multiple repetitions of different NaNs in a list of floating-point numbers. [5 Marks]**
11. **Use any machine method to report the time taken for the sorting program to run on a list of data. [5 Marks]**
12. **Produce a table of list length versus time taken to sort lists. [5 Marks]**
13. **Produce a graph of list length versus time taken to sort lists. [5 Marks]**
14. **Give a recurrence relation that describes the time order for the worst-case of the sorting algorithm. [5 Marks]**
15. **Solve the recurrence relation that gives the time order for the worst-case of the sorting algorithm. [5 Marks]**
16. **Compare the theoretical and actual performance of you program. Is the theory supported by the evidence? If not, what might explain the discrepancy? [5 Marks]**
17. **Test a built-in search procedure. Does it correctly handle minus zero and NaNs? [5 Marks]**
18. **Test a built-in search procedure. What time order does it appear to have? [5 Marks]**
19. **Bonus for anything that is done particularly well. [10 Marks]**