THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

2015 SCHOLARSHIP EXAMINATION

PRACTICAL SECTION

DEPARTMENT	Computer Science
COURSE TITLE	Year 13 Scholarship
TIME ALLOWED	Six hours with a break for lunch at the discretion of the supervisor
NUMBER OF QUESTIONS IN PAPER	Three
NUMBER OF QUESTIONS TO BE ANSWERED	Three
GENERAL INSTRUCTIONS	Candidates are to answer ALL THREE questions. All questions are important. Answer as much of each question as you can. Plan your time to allow a good attempt at each question, but be aware that Question 3 is the most difficult and may take considerably longer than the others.
SPECIAL INSTRUCTIONS	Please hand in listings, notes and answers to written questions, and a CD/DVD with your program/computer work for each question. Please make sure that a copy of each program is printed, or stored as a plain text file. You cannot assume that the examiner has available any special software that might be required to read your files. Candidates may use any text or manual for reference during the examination.
CALCULATORS PERMITTED	Yes

TURN OVER

1. **Layout** (Spreadsheet Use)
In this question you are asked to use a spreadsheet to do calculations and to display the results. We expect that the spreadsheet will be used for all calculations unless the question states otherwise - you will be marked down for performing calculations by hand and directly entering the results. Your work will be graded on three criteria.

(i) The accuracy of your results.

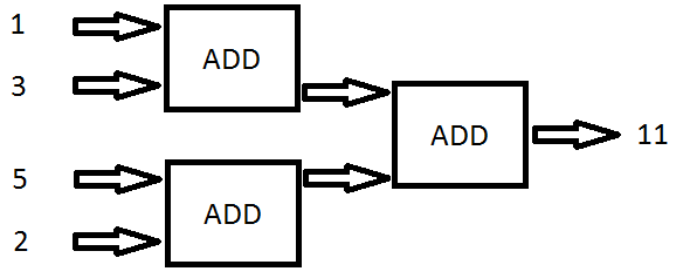
(ii) The skill you show in making use of the capabilities of the spreadsheet.

(iii) The presentation of your results. We have deliberately not provided any instructions concerning layout or formatting and our example graphs lack labels and proper scales.

In this question you are asked to develop a sorting algorithm and to use it in different ways. There are several stages to this problem. Stages may have explanation and all have task(s) for you to perform. We would like to see your answers for each stage – you can put them in separate areas of the same sheet or on different sheets.

Stage A

Algorithms are usually discussed in the context of computer programming. However, algorithms can also be implemented in electronic hardware. In hardware we may have multiple calculation units, then rather than using a loop to perform a repeated calculation, for example, we might wire the output of one unit into the input of the next in a chain. Building on this idea, we can construct an algorithm as a wiring diagram connecting inputs and outputs of calculation units. For example, imagine we have a supply of 'add' units – units that take two numbers as inputs and output their sum. We can wire three of these units to four input numbers and generate the sum of all four as the output.



Note: Don't worry about the details of connecting numbers; clearly the connections are going to be complicated with many real physical wires in each.

Spreadsheets provide an excellent environment for experimenting with this kind of algorithm by 'layout and interconnection'. The addition above becomes

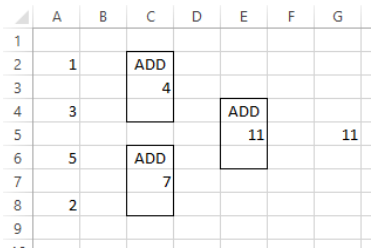
	A	B	C	D	E	F	G
1							
2	1						
3			4				
4	3						
5					11		11
6	5						
7			7				
8	2						
9							

or showing formulae

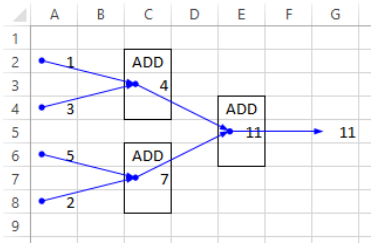
	A	B	C	D	E	F	G
1							
2	1						
3			=A2+A4				
4	3						
5					=C3+C7		=E5
6	5						
7			=A6+A8				
8	2						
9							

(Question 1 – continued on next page)

A little more work with formatting – boxes around the units and 'ADD' labels gives



A helpful spreadsheet may even be able to show the connections



Tasks for Stage A

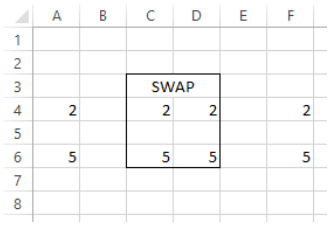
Using a system with 'multiply' units as well as ADD units lay out and test 'programs' in your spreadsheet which

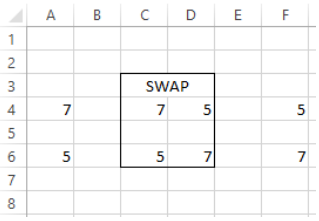
(a) Use two ADD units and one multiply unit to calculate (1 + 3) * (5 + 2)

(b) Calculate 1 + 2 + 3 + 4 + 5 + 6 with a suitable number of ADD units

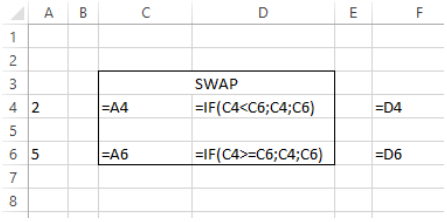
Stage B

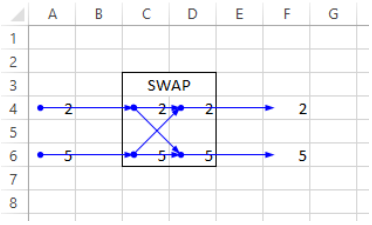
The 'layout and interconnection' approach to expressing algorithms is not limited to direct calculation. In this stage we will work towards making a sorting algorithm. A sorter can be built with two special units that we will call SWAP and COPY. SWAP has two inputs and two outputs. The outputs are the same as the inputs if the values on the inputs are in numeric order, and are a swapped version of the inputs if they are not in order. Here is a first version of SWAP showing how it swaps two numbers if necessary.





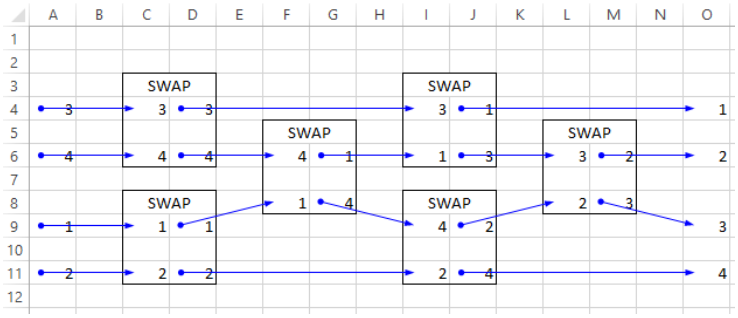
The next images show the formulae used and the 'wiring' both internal and external to the SWAP unit. Note that the formulae inside SWAP reference the input several times, so in order to make it easier to hook up swap units, extra cells have been used to hold copies of the input.





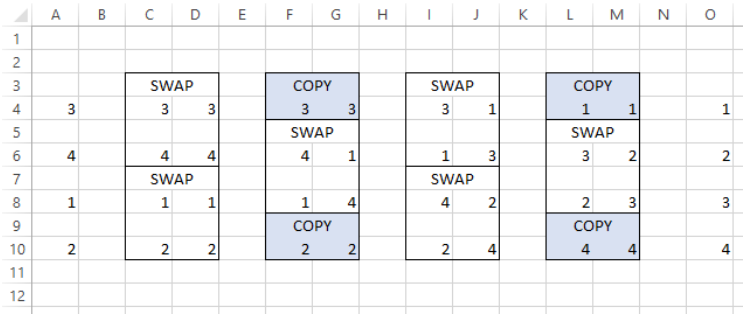
(Question 1 – continued on next page)

Putting together 6 SWAP units allows us to build an algorithm to sort four numbers into order, thus:

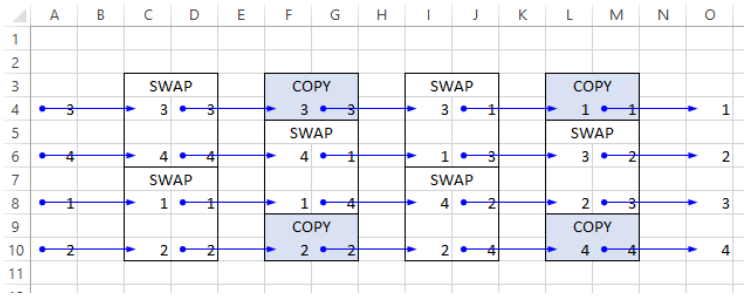


Note that 'hooking up' or 'wiring' two units together is done by putting a formula in an input cell referencing the value we want. For example cell F6 contains =D6.

It would be possible to build a larger sorter with these SWAP units. However, the connections are a bit irregular, so doing all the wiring would take some time. An improvement is to make use of the way spreadsheets access cells – relative to the location of a formula. If we remove the blank lines between SWAP units, then inputs will always come from cells to their left on the same line. Adding a new unit 'COPY' which only copies its input to its output enables us to make sure that SWAP units always take input from cells exactly two columns to their left. The new structure, with COPY units lightly shaded looks like this.

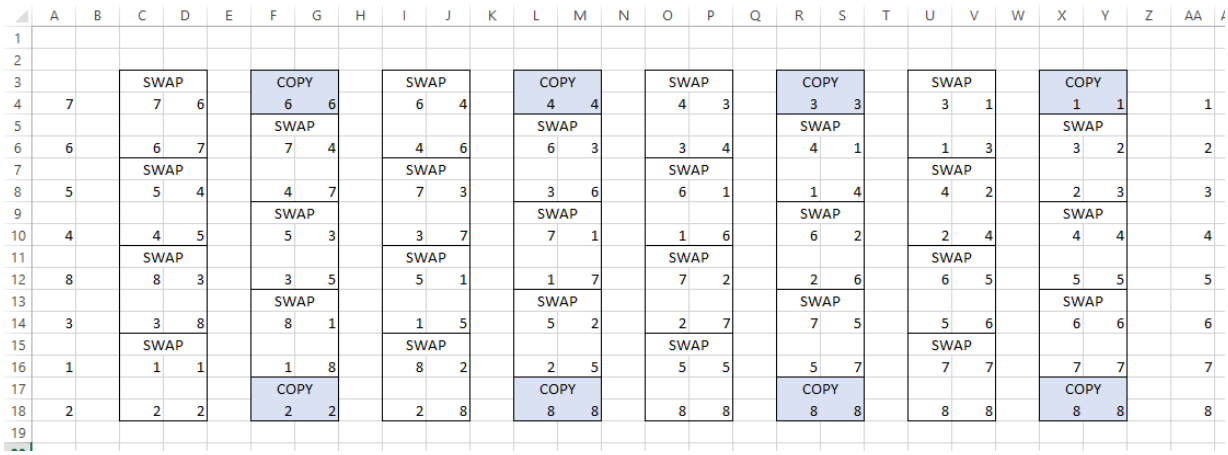


The unit connections are now very regular.



(Question 1 – continued on next page)

The connections are now so regular that we can copy and paste rows and columns of them to build bigger sorters without having to manually put in the connections. Here is a sorter for 8 values.



1																										
2																										
3			SWAP			COPY			SWAP			COPY			SWAP			COPY			SWAP			COPY		
4	7		7	6		6	6		6	4		4	4		4	3		3	3		3	1		1	1		1
5						SWAP						SWAP						SWAP						SWAP			
6	6		6	7		7	4		4	6		6	3		3	4		4	1		1	3		3	2		2
7			SWAP						SWAP						SWAP						SWAP						
8	5		5	4		4	7		7	3		3	6		6	1		1	4		4	2		2	3		3
9						SWAP						SWAP						SWAP						SWAP			
10	4		4	5		5	3		3	7		7	1		1	6		6	2		2	4		4	4		4
11			SWAP						SWAP						SWAP						SWAP						
12	8		8	3		3	5		5	1		1	7		7	2		2	6		6	5		5	5		5
13						SWAP						SWAP						SWAP						SWAP			
14	3		3	8		8	1		1	5		5	2		2	7		7	5		5	6		6	6		6
15			SWAP						SWAP						SWAP						SWAP						
16	1		1	1		1	8		8	2		2	5		5	5		5	7		7	7		7	7		7
17						COPY						COPY						COPY						COPY			
18	2		2	2		2	2		2	8		8	8		8	8		8	8		8	8		8	8		8
19																											

It takes 8 columns of SWAP units to sort 8 items. A similar layout of units works for any sized sorter. The first, third, fifth, etc columns swap 1st and 2nd, 3rd and 4th, 5th and 6th etc. The second, fourth, sixth, etc columns swap 2nd and 3rd, 4th and 5th, 6th and 7th, etc. We need as many columns as we have items to sort. COPY units are put in the gaps where a SWAP won't fit.

Tasks for Stage B

(c) Make a sorter for 6 numbers.

(d) Make a sorter for 3 numbers.

STAGE C

The next stage is make a bigger sorter – one that will sort 32 items. The current layout is quite large. Your goal is to make it smaller. Units could be moved together. Blank lines and columns could be removed. Do units need input cells, or could calculations directly access their inputs? It would be nice to retain some features – boxes around units help us to understand the layout. Colouring different kinds of unit seems helpful. Most importantly we want to be able to make a sorter by copy and paste. Note that you are not required to alter the 'algorithm' here – just use available spreadsheet capabilities.

Task for Stage C

(e) Make as compact a 32 item sorter as you can, and test it.

(Question 1 – continued on next page)

STAGE D

Sorting is a common requirement in computing. However it is rare to just sort numbers. Often there is other data involved. For example a table of students and marks might need to be sorted into mark order.

Smith	58
Chen	46
Jones	78
Brown	98
Green	49
Ngata	71
O'Brien	83

sorted to

Chen	46
Green	49
Smith	58
Ngata	71
Jones	78
O'Brien	83
Brown	98

Tasks for Stage D

(f) Make a new sorter capable of sorting a list of names and marks on the mark values. Your new sorter should follow the pattern of the old one – being based on interconnected units. You should expect to redesign the units.

To hand in, write brief notes on your spreadsheet and print out illustrative screen shots to help with the explanation. You can write this on paper, or use a text editor, or a word processor.

CONTINUED

2. **Time** (Careful and Accurate Programming)

Your programming work in this question will be assessed on two criteria:

(a) Completeness and accuracy of the program.

(b) Good presentation. That is, it should make good use of programming language facilities, be well organised, neatly laid out, and lightly commented.

Software often has to deal with time values – reading, writing and doing calculations may all be required. Consequently, most programming languages have libraries of functions to manage times (eg: time.h accesses time functions in C). Your task for this question is to write a program with your own time handling.

Your program must read a 'start' time and a 'finish' time (in hours, minutes and seconds) from the keyboard, then report the difference (in hours, minutes and seconds). If for example, the inputs were a runner's start and finish times for a marathon, your program would be able to tell them how long they took. Your program must read integers, characters and/or strings from input and **must not use any time facilities from your programming language or libraries**.

A sample run of a version of the program might have input and output like this (note that data input by the person running the program is underlined):

```
Enter start time:  10:45.37
Enter finish time: 12:33.10
Time taken was 1:47.33

Enter start time:  12:33.24
Enter finish time: 12.27.12
You finished before you began
```

The format used for inputting and outputting times is up to you. Think about how you will tell the user what format to use. Think about other issues that might arise and how you will deal with incorrect input, if that can happen.

3. **Parking Cars** (Problem Solving and Programming)

Your programming work in this question will be assessed on two criteria:

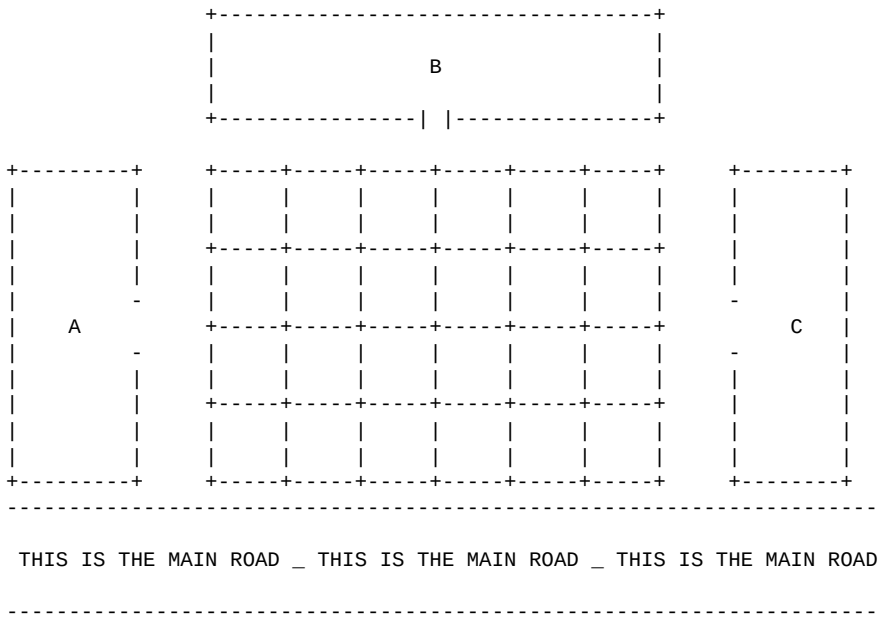
(a) Your approach to the problem. We will be looking at your work for evidence that you found good ways of storing the necessary data, and devised algorithms for finding and displaying the requested results. ***Please hand in any notes and diagrams which describe what you are attempting to program, even if you don't have time to code or complete it. You may include comments in your program, or write a description of your program to hand in.***

(b) The extent to which your program works and correctly solves the problem.

You may find that the programming language you use makes it difficult to produce output as shown in the example implementation steps below. If this is the case, feel free to build your program in a way that suits your circumstances.

Three companies (A, B and C) cooperate in the manufacture, programming and marketing of a range of automatic electric cars. The companies have buildings on three sides of a large carpark. A main road is along the fourth side. The carpark is shared by the staff of the three companies. Each carpark space has a charging station and is surrounded by a security fence that lowers into the ground to open.

Here is a rough map of the area. In the map the car park has 4 rows of 6 car parking spaces. The company buildings are marked A, B and C. Building entrances are in the middle of the sides facing the car park as shown. The squares on the car park show the locations of the parking spaces. The lines between spaces represent roadways along which cars and pedestrians can travel (so the security fences enclose only part of the area of each square).



(Question 3 – continued on next page)

All staff in the companies use company cars and therein lies a problem. Interestingly, whilst the cars navigate roads well enough, their software has severe difficulty in navigating the car park. If two cars attempt to travel in different directions through the narrow driveways of the park they are very likely to get stuck in a standoff situation where neither car can make any progress. For this reason the companies agreed to a parking system as follows.

> When staff arrive in the morning they are dropped off on the side of the main road and their cars then park themselves, avoiding conflict by taking a fixed route through the park and stopping in the first available parking space. At the end of the day staff members walk to their cars and wait in their cars for their turn as the cars leave one by one.

This worked well until winter. The region has high rainfall and staff complained about the distance they had to walk to reach their cars. So an ingenious car reorganization scheme has been devised. The idea is that the cars will reorganize themselves during the day so that they end up parked as close as possible to the building in which their owner works. Your task is to write a program to work out the reorganization schedule.

Following are a series of stages for you to follow in developing your program.

Stage A

Although the program should work with a large carpark (up to 10 by 10 spaces), it will be better to start with a smaller area for program development. Write a program to allow a user to enter a size (number of rows and number of columns) for the car park. There is a requirement that both the number of rows and the number of columns are even numbers (2, 4, 6, etc). Your program should then display a map of just the carpark. It might look like the graphic we showed earlier (but without the buildings).

Stage B

Fill your car park. Each space should have a car. The only thing you need to know for each car is which building the owner of the car works in (A, B or C). You might do this at random. You might choose some other method. Update your display program to show A, B or C in each space. Here is an example with random placement.

```
+-----+-----+-----+-----+-----+-----+
|  A  |  A  |  B  |  C  |  B  |  B  |
|     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+
|  B  |  C  |  A  |  B  |  A  |  A  |
|     |     |     |     |     |     |
**************************-----+-----+
|  A  |  C  |  C  |  B  *  A  |  B  |
|     |     |     |     *     |     |
+-----+-----+-----+-----*-----+-----+
|  C  |  C  |  C  |  A  |  A  |  B  |
|     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+
```

(Question 3 – continued on next page)

Stage C

For each car parking space work out the shortest distance the owner must walk to reach their car. The distance can be measured in 'sides' of a parking space. Assume that the owner starts at the parking space corner nearest to their building entrance and walks along row lines or up or down column lines until they reach a corner of their car space. A suitable route from building A to the car parked in the fifth column of the fourth row is shown in asterisks above. It has distance 5 (the owner walks along 5 parking space sides). Please note that this minimum distance is much simpler to work out than it might appear. Try some examples – basically the route doesn't matter much – just travel far enough left or right and far enough up or down.

Update your display to show minimum walking distances for car owners. A sample might look like this:

```
+-----+-----+-----+-----+-----+-----+
|  A  |  A  |  B  |  C  |  B  |  B  |
|  01 |  02 |  00 |  03 |  01 |  02 |
+-----+-----+-----+-----+-----+-----+
|  B  |  C  |  A  |  B  |  A  |  A  |
|  03 |  04 |  02 |  01 |  04 |  05 |
+-----+-----+-----+-----+-----+-----+
|  A  |  C  |  C  |  B  |  A  |  B  |
|  00 |  04 |  03 |  02 |  04 |  04 |
+-----+-----+-----+-----+-----+-----+
|  C  |  C  |  C  |  A  |  A  |  B  |
|  06 |  05 |  04 |  04 |  05 |  05 |
+-----+-----+-----+-----+-----+-----+
```

Stage D

The car exchange strategy is as follows. Every few minutes the cars all exchange information about building distances. A pair of cars is chosen such that their swapping places will give the best improvement in total walking distance across all cars in the park. These two cars then swap places. If more than one pairs could give the same improvement, it doesn't matter which is chosen. The process is then repeated until there is no pair whose swap can improve walking distance.

To start with, update your program to carry out just one car swap. You should output the locations of cars being swapped. When you are happy that this is working, try repeating car swaps. Do you think that this algorithm will ever stop – will it find a good parking arrangement – will it find the best arrangement?