STOKE
DAMEREL

Aspire Achieve Thrive

Half Term 1

Computer Science

Year 11

Name: ________________________

Tutor: ________________________

Year 11 Homework Timetable

Monday	Ebacc Option D	Option C	Modern Britain	
Tuesday	English	Tassomai	Option B	Option A
Wednesday	Sparx	Science	Modern Britain	Option C
Thursday	Ebacc Option D	Tassomai	Option B	
Friday	Sparx	Science	English	Option A

Block A	Block B	Block C	Block D
Art	Business Studies	Art	French
Dance	Child Development	Business Studies	Geography
Drama	Catering	Geography	History
Media Studies	Computer Science	Health & Social Care	
Music	Drama	History	
Photography	Health & Social Care	Catering	
	IT	Photography	
	Media Studies	Sport	
	Sociology	Travel & Tourism	
	Sport		

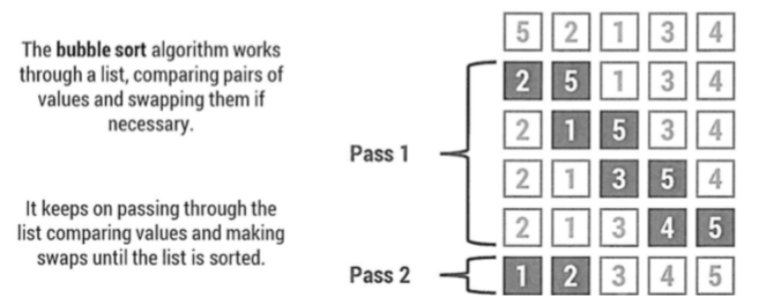
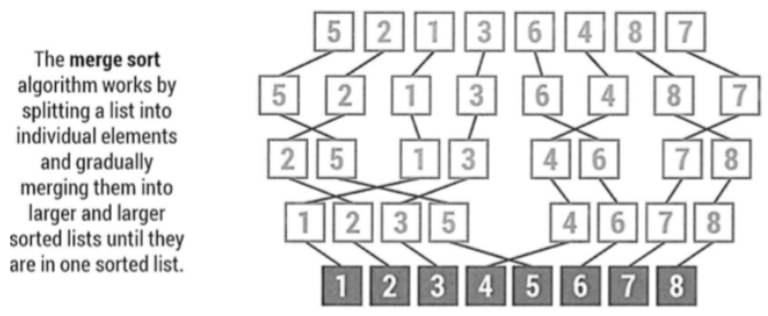
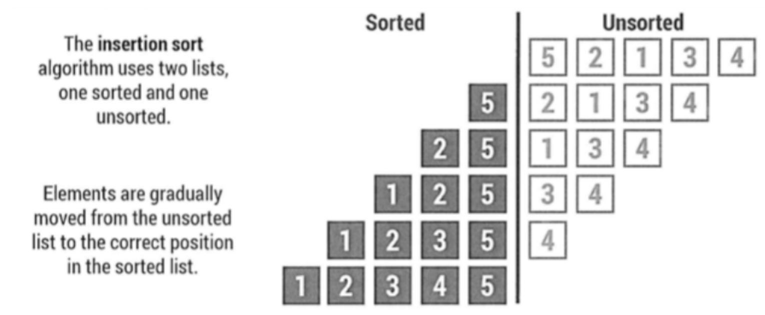
Tassomai - 2 Daily Goals per week
Sparx - 4 tasks of Sparx per week

Year 11 Homework Plan

Week/Date	Homework Task	Examination Question Topic
Week 1 Monday 5th September 2022	**Cornell Notes** Bubble Sort Insertion Sort Merge Sort	Bubble Sort
Week 2 Monday 12th September 2022		Insertion Sort
Week 3 Monday 19th September 2022		Merge Sort
Week 4 Monday 26th September 2022	**Cornell Notes** Defensive Design	Examples of defensive design strategies
Week 5 Monday 3rd October 2022		Functions and procedures
Week 6 Monday 10th October 2022	**Cornell Notes** Testing	Purpose of testing
Week 7 Monday 17th October 2022		Syntax and Logic Errors

QR codes and links to videos are included in each topic.
You should use these videos alongside the Knowledge Organiser to make detailed Cornell Notes / Revision Cards each week.

Week 1, 2 and 3: 2.1.3 Bubble Sort | Merge Sort | Insertion Sort

Bubble Sort	Sorts an unordered list of items by comparing each item with the next one and swaps them if they are out of order. The algorithm finishes when no more swaps need to be made. In effect it 'bubbles up' the largest (or smallest) item to the end of the list with each 'pass'. Very inefficient but easy to implement so can be used for very small data sets.	 The **bubble sort** algorithm works through a list, comparing pairs of values and swapping them if necessary. It keeps on passing through the list comparing values and making swaps until the list is sorted. Pass 1 Pass 2
Merge Sort	Uses a divide and conquer methodology. Data set is repeatedly split in half until each item is in its own list. Then pairs of lists are merged together in the correct order. Very efficient and works well with large data sets.	 The **merge sort** algorithm works by splitting a list into individual elements and gradually merging them into larger and larger sorted lists until they are in one sorted list.
Insertion Sort	Inserts each item into its correct position in a data set, one item at a time. Useful for inserting items into a list that is already sorted.	 Sorted Unsorted The **insertion sort** algorithm uses two lists, one sorted and one unsorted. Elements are gradually moved from the unsorted list to the correct position in the sorted list.

Week 4 and 5: 2.3.1 Defensive Design - Validation, Misuse, Authentication, Maintainability

Input Validation

Input Validation involves checking data that is input by a user meets specific rules or criteria.

Type Check
Data must be the correct data type (integer, string, float, character, etc)

Range Check
Data must fall between a certain range of values. For example, between 1 and 10, or between ‘a’ and ‘z’.

Presence Check
Ensures that some data has been entered.

Format Check
Data is in the correct format. For example, a date is in the format 23/12/2021

Length Check
Data must have a specific number of characters. For example, a password might be required to be at least 8 characters in length.

Anticipating Misuse | Authentication

Division by Zero
The arithmetic logic unit inside a processor cannot compute a division by zero, therefore well-designed code should prevent this from occurring

Communication Error
Online systems required connections to host servers, usually through the Internet. If a connecting is lost or the server is overloaded, the program may crash or hang. Well-designed code should allow users to cancel a connection request or the program should report an error if there is no connection with a remote server.

Printer / other peripheral errors
If a program prints data, the programmer shouldn’t assume the printer was successful and should provide options to ‘reprint’.

Disk Errors
Programs that read and write to files need to handle errors such as:

- File or folder not found
- No space on disk to save
- Corrupt data files
- Reaching the end of a file when reading data.

Authentication
Data used by systems should be secure. Programmers should ensure that users are authenticated (for example, with a login username and password) and data files are encrypted so they can’t simply be opened with another program..

Maintainability

Programmers need to ensure that the code that they write is ‘maintainable’. This includes the following:

Comments
Comments should explain the purpose of the program or a section of code. They may also explain complicated sections of code or unusual approaches.

White Space
Sections of code can be spaced out so that it isn’t cramped together, therefore easier to read.

Indentation
Some programming languages (like Python) forces programmers to indent code so that the flow of the program can be followed.

Identifier Names
Use sensible identifiers for variables when programming so that others can understand what each variable is being used for.

Sub Programs
Use procedures and functions to structure code and eliminate duplicating portions of the same code. As well as reducing the amount of code, if the program needs to be changed, it can be done once in the sub program.

Week 6 and 7: 2.3.2 Testing

The purpose of and types of testing

Four main reasons for testing

- To ensure there are no errors (bugs) in the code.
- To check the program has an acceptable performance and usability.
- To ensure unauthorised access is prevented.
- To check if the program meets the requirements of the user.

Iterative Testing

- Each new module (section of code) is tested as it is written.
- Program branches (if statements or loops) are checked to make sure they work,
- Check new modules don't introduce new errors in existing code.

Final (Terminal) Testing

- Testing all modules work together.
- Testing the program produces the required results.
- Checking the program meets the requirements of the user.

Syntax and Logic Errors

Syntax Errors

A syntax error occurs when code written does not follow the rules of the programming language. A program will not run if it has syntax errors. Any such errors must be fixed first. A good integrated development environment (IDE) will usually point out any syntax errors to the programmer.

Examples of syntax errors:

- misspelling a statement, eg writing pint instead of print
- using a variable before it has been declared
- missing brackets, eg opening a bracket but not closing it

Logic Errors

A logic error is an error in the way a program works. The program simply does not do what it is expected to do. Unlike a syntax error, a logic error will not usually stop a program from running. Instead the program will run but not function as expected.

Examples of logic errors:

- incorrectly using logical operators, eg expecting a program to stop when the value of a variable reaches 5, but using <5 instead of <=5
- incorrectly using Boolean operators
- unintentionally creating a situation where an infinite loop may occur
- incorrectly using brackets in calculations
- unintentionally using the same variable name at different points in the program for different purposes.

Suitable Test Data

When testing software, you should use normal, boundary, invalid and erroneous inputs.

Normal Inputs
Data which should be accepted by a program without causing errors

Boundary Inputs
Data of correct type on the edge of accepted validation boundaries

Invalid Inputs
Data of the correct type but outside accepted validation checks

Erroneous Inputs
Data of the incorrect type which should be rejected by a computer system. This includes no input being given when one is expected.

Refining Algorithms

Code should anticipate all inputs and it should deal with 'bad' data, or missing data, and not crash.

It should ensure prompts to the user are helpful and that the input can only be of the correct type

WEEK 1

Date…………………………………………….

Willow has created a hangman program that uses a file to store the words the program can select from. A sample of this data is shown below:

crime	bait	fright	victory	nymph	loose

Show the stages of a bubble sort when applied to the data shown

[4 marks]

WEEK 2

Date…………………………………………….

The following names of students are stored in an array with the identifier studentnames.

studentnames = ["Rob", "Anna", "Huw", "Emma", "Patrice", "Iqbal"]

The names of students are sorted into ascending alphabetical order using an insertion sort.

Complete the following diagram to show the stages an insertion sort would take to complete this task.

Each row represents one pass of the insertion sort algorithm. You may not need to use all empty rows.

Rob	Anna	Huw	Emma	Patrice	Iqbal

[4 marks]

WEEK 3

Date…………………………………………….

A library gives each book a code made from the first three letters of the book title in upper case, followed by the last two digits of the year the book was published.

For example, “Poetry from the War”, published in 2012 would be given the code POE12.
The library sorts their books based on the book code.

Show the steps that a merge sort would take to put the following list of book codes into ascending alphabetical order (from A to Z).

POE12	BAC97	FLY77	JAV16	TAL86	AND18	ZAR09	HOP86

Show the stages of a bubble sort when applied to the data shown

[4 marks]

WEEK 4

Date…………………………………………….

Elliott plays football for OCR FC. He wants to create a program to store the results of each football match they play and the names of the goal scorers. Elliott wants individual players from the team to be able to submit this information.

Describe two examples of defensive design that should be considered when developing this program.

WEEK 5

Date…………………………………………….

A library gives each book a code made from the first three letters of the book title in upper case, followed by the last two digits of the year the book was published.

For example, “Poetry from the War”, published in 2012 would be given the code POE12.
Functions and procedures are both examples of sub programs.

Describe one difference between a function and a procedure.

[2 marks]

Describe **two** benefits to a programmer of using sub programs.

[4 marks]

WEEK 6

Date…………………………………………….

A computer program has been written to control a vending machine. The vending machine is tested before it is released.

Explain the purpose of testing the vending machine.

[2 marks]

Describe the difference between iterative testing and final testing.

[2 marks]

WEEK 7

Date…………………………………………….

Louise writes a program to work out if a number entered by the user is odd or even. Her first attempt at this program is shown.

```
01        num = input("enter a number")
02        if num MOD 2 >= 0 then
03                  print("even")
04        else
05                  pritn("odd")
06        endif
```

The program contains a logic error on line 02.

i. State what is meant by a logic error.

[1 mark]

ii Give a corrected version of line 02 that fixes the logic error.

[1 mark]

The program contains a syntax error on line 05.

i State what is meant by a syntax error.

[1 mark]

ii Give a corrected version of line 05 that fixes the syntax error.

[1 mark]

STEP 2: CREATE CUES

What: Reduce your notes to just the essentials.

What: Immediately after class, discussion, or reading session.

How:

- Jot down key ideas, important words and phrases
- Create questions that might appear on an exam
- Reducing your notes to the most important ideas and concepts improves recall. Creating questions that may appear on an exam gets you thinking about how the information might be applied and improves your performance on the exam.

Why: Spend at least ten minutes every week reviewing all of your previous notes. Reflect on the material and ask yourself questions based on what you've recorded in the Cue area. Cover the note-taking area with a piece of paper. Can you answer them?

STEP 1: RECORD YOUR NOTES

What: Record all keywords, ideas, important dates, people, places, diagrams
and formulas from the lesson. Create a new page for each topic discussed.

When: During class lecture, discussion, or reading session.

How:

- Use bullet points, abbreviated phrases, and pictures
- Avoid full sentences and paragraphs
- Leave space between points to add more information later

Why: Important ideas must be recorded in a way that is meaningful to you.

STEP 3: SUMMARISE & REVIEW

What: Summarise the main ideas from the lesson.
What: At the end of the class lecture, discussion, or reading session.
How: In complete sentences, write down the conclusions that can be made from the information in your notes.
Why: Summarising the information after it's learned improves long-term retention.

Date / / Topic **WEEK 1**

Questions	**Notes**

Summary

Date / / Topic **WEEK 2**

Questions	Notes

Summary

Date / / Topic **WEEK 3**

Questions	**Notes**

Summary

Date / / Topic **WEEK 4**

Questions	Notes

Summary

Date / / Topic **WEEK 5**

Questions	**Notes**

Summary

Date / / Topic **WEEK 6**

Questions	**Notes**

Summary

Date / / Topic **WEEK 7**

Questions	Notes

Summary

Revision Page
