Accredited

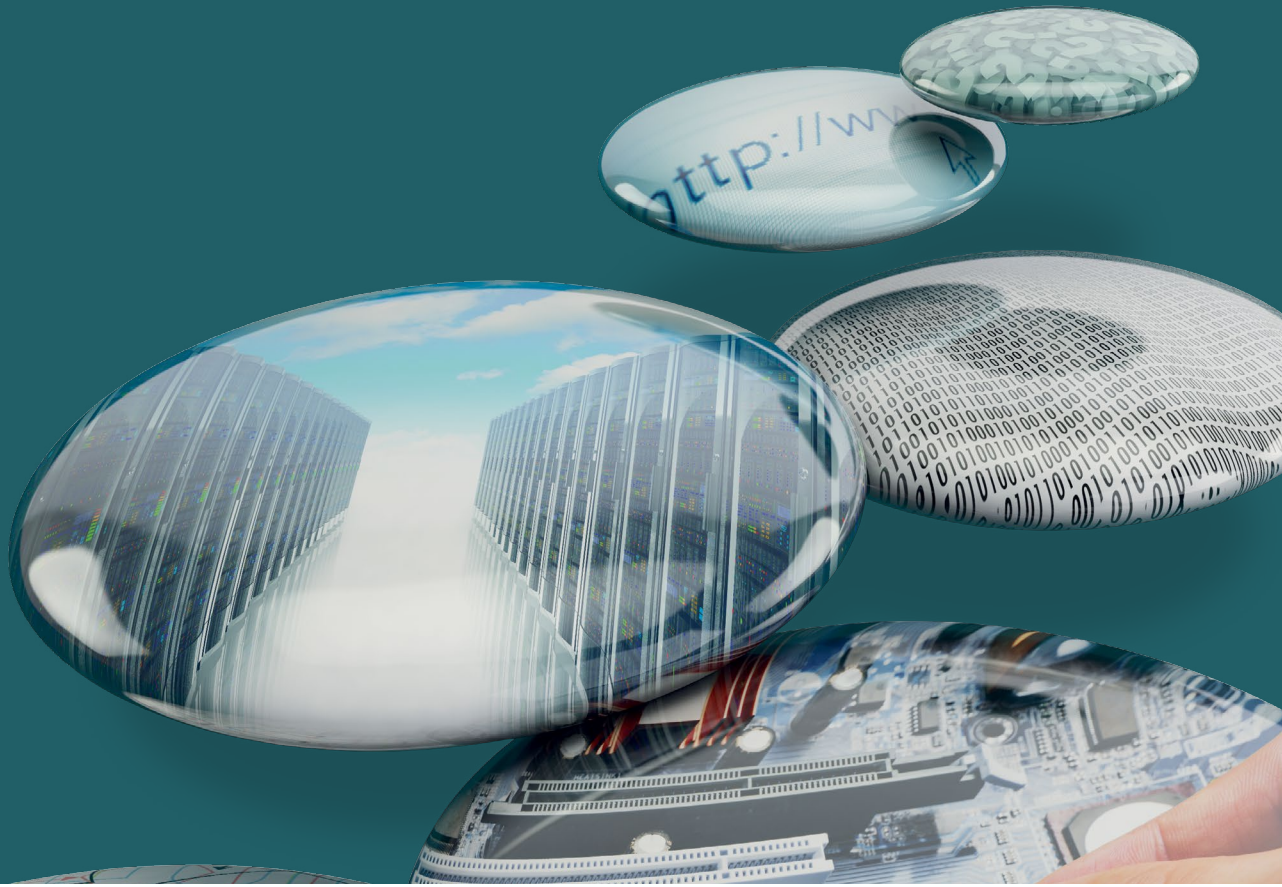
A LEVEL
Delivery Guide

H446

COMPUTER SCIENCE

Theme: Computational Techniques

September 2015



CAMBRIDGE UNIVERSITY PRESS
www.cambridge.org

OCR
Oxford Cambridge and RSA

We will inform centres about any changes to the specification. We will also publish changes on our website. The latest version of our specification will always be the one on our website (www.ocr.org.uk) and this may differ from printed versions.

Oxford Cambridge and RSA Examinations is a Company Limited by Guarantee.
Registered in England. Registered company number 3484466.

Registered office: 1 Hills Road
Cambridge
CB1 2EU

OCR is an exempt charity.

CONTENTS

Introduction

Delivery guides are designed to represent a body of knowledge about teaching a particular topic and contain:

- Content: A clear outline of the content covered by the delivery guide;
- Thinking Conceptually: Expert guidance on the key concepts involved, common difficulties students may have, approaches to teaching that can help students understand these concepts and how this topic links conceptually to other areas of the subject;
- Thinking Contextually: A range of suggested teaching activities using a variety of themes so that different activities can be selected which best suit particular classes, learning styles or teaching approaches.

If you have any feedback on this Delivery Guide or suggestions for other resources you would like OCR to develop, please email resourcesfeedback@ocr.org.uk.

KEY

Click to view associated resources within this document.

Click here — Click to view external resources.

AS Level only — AS Level content only.

Curriculum Content

Content (from A level)

a) Features that make a problem solvable by computational methods.

b) Problem recognition.

c) Problem decomposition.

d) Use of divide and conquer.

e) Use of abstraction.

f) Learners should apply their knowledge of:

- backtracking
- data mining
- heuristics
- performance modelling
- pipelining
- visualisation to solve problems.

H.B. Meyer Backtracking algorithm animation (http://www.hbmeyer.de/backtrack/backtren.htm) is a great way to visualise backtracking process. There are also links to other backtracking algorithms but the use of Google translate (from German) might be necessary.

http://www.sorting-algorithms.com/ contains a very visual comparison of different sorting algorithms. We can change parameters and see the differences in performance depending on array sizes and how sorted they are.

Thinking Conceptually

Common misconceptions or difficulties students may have

To make a problem solvable computationally, the analysis and design stages of product development will convert a 'story' that a customer has to a specific and unambiguous set of objectives and assumptions/limitations. Activity and data flow diagrams will identify type of variables, data structure and the computational stages that will be required. The word 'solution' refers to the required output, while the 'problem' can refer to instances of input data – a file to parse, a string to encrypt, etc. A learner has a set of tools, which can include knowledge of 2d arrays, recursion, API's and so on. The purpose is to map the problem to the tools, so that the learner could write out a list of tools that will be needed for a particular problem. This process of mapping will inevitably split the main problem into smaller simpler sub-tasks – the process known as 'decomposition'. This process also allows us to compare the sub-tasks and group them by type, known as 'abstraction'. Recognising that multiple sub-tasks can be solved in a similar way will prompt the learner to write a self-contained piece of code, a function or a sub, that can be reused on multiple sub-tasks. Often, the sub-tasks are quite similar, for example reading another line in a file, so they can be automated through iteration. If the sub-tasks are really similar but only differ in a parameter (e.g. search upper half of the list vs search the lower half of the list), recursion becomes a powerful tool and gives us the ability to branch while iterating , known as the 'divide and conquer' technique used in algorithms like Quicksort. In such algorithms, efficiency can be improved if we can skip some of the processing-'prune' it. Learners need to be aware that the simplest approach of running through every possible combination of inputs can often be avoided through backtracking – which involves setting up a constraint and eliminating obvious dead-end solutions based on that constraint. This is similar to multiple-choice technique in tests where a pupil eliminates obviously wrong options to concentrate on the ones that are possible answers. Backtracking is not a trivial concept but can be made easier through visualisation, for example through graphs and binary trees, or solving popular puzzles like Sudoku or Solitaire. Backtracking together with heuristics helps with particularly hard problems where enumeration (considering every possibility) is not practical. Understanding of heuristics leads us to the solutions that work but we might suspect are not the best; in other words, they satisfy all constraints but are not optimal. Data mining refers to a search for patterns in a set of data. Efficient data mining can utilise heuristics to find patterns; for example, an antivirus program scans all files on a computer looking for virus signatures (patterns); by scanning sensitive system files first, or the Downloads folder, the antivirus program can improve its chances of finding a virus – by using previous knowledge of where viruses tend to enter the system. Visualisation is another way of looking for patterns and the use of infographics and sophisticated graphing theory makes data mining much easier.

Thinking Conceptually

Speaking of large sets of data, as an algorithm has to deal with increasing volume of data, it might slow down disproportionately with the increase in data volume. This needs to be predicted with performance modelling before there is a problem. The use of pipelining identifies which processing stages can be carried out concurrently and which ones can't.

Santa's Dirty Socks (Computer science unplugged, http://csunplugged.org/divideAndConquer) is another way to illustrate an algorithm.

Rosettacode.org has implementations of most algorithms in different programming languages, so pupils can try a variety of competing algorithms and time them to see how efficient they are.

Thinking Contextually

Activity 1: Divide and conquer	Resources
Task A: How does Quicksort use the so-called divide and conquer strategy? [Solution] Quicksort uses a pivot to split/move the data into smaller parts, the ones larger than pivot get moved to the part above the pivot, the ones that are lower get moved below pivot, and these parts which are recursively further split into smaller parts, thus avoiding making comparisons between all the list items. [End solution] Task B: Using RosettaCode.org site find Quicksort implementation in your favoured language. Copy it out and supply with annotations that explain what each line does and especially how recursion makes it possible. (http://rosettacode.org/wiki/Sorting_algorithms/Quicksort is available under GNU Free Documentation License 1.2.) [Solution using Python]	Learner Resource 1.1 Click here

```
def Quicksort(arr): #start a recursive function with unsorted array passed in
    less = [] #empty list for the less-than-pivot values
    pivotList = [] #empty list for the sorted values
    more = []#empty list for the less-than-pivot values
    if len(arr) <= 1: #terminating condition of an unsorted array #having a length of 1
       return arr #terminates the function and the recursion
    else:
        pivot = arr[0] #set the pivot value to the first array #element
        for i in arr: #iterate through the array
            if i < pivot:#compare each element to the pivot
                less.append(i)#if value is smaller, move to the #lesser sub-array
            elif i > pivot:#if element is larger than the pivot
                more.append(i)#move to the larger sub-array
```

Activity 1: Divide and conquer	Resources

```
            else:#if element equals to pivot
                pivotList.append(i) #add to sorted sub-array
        less = Quicksort(less) #divide and conquer via recursion
#this will return the sorted sub-array of items smaller than pivot
        more = Quicksort(more) #divide and conquer via recursion
#this will return the sorted sub-array of items larger than pivot

        return less + pivotList + more #combines the sub-arrays back

a = [4, 65, 2, -31, 0, 99, 83, 782, 1]#set up the unsorted list
b = Quicksort(a) #assign the sorted version of a to b
print(b)#print b
```

[End solution]

Thinking Contextually

Activity 2: Backtracking	Resources
Task A: After looking at the definition of computational 'backtracking', put the definition in plain English terms and identify a list of three real-life (non-computing) applications of backtracking. [Solution] Backtracking is a step in solving a computing problem after all possibilities have been identified, for example all combinations of a chess move. Some of these possibilities can be eliminated if checked against a constraint. Real-life application 1: Answering multiple-choice questions – they rely on elimination of first, obviously wrong alternatives, and then once the options are narrowed, eliminating the options that are true some times (but not all the time), living the only true answer. Real-life application 2: Police detectives compile a list of suspects and then eliminate those against a constraint which could be an alibi or the lack of a motive. Real-life application 3: Employers recruiting new workers have a stack of CVs to go through. By setting a constraint (e.g. an experience with a certain machine), they can eliminate a whole subset of applicants. Could also consider: Navigating a maze, performing genetic selection – creating a dog breed with particular attributes, applying 'gut feelings' (e.g. this doesn't look right, too good to be true, just run), behavioural experiments. [End solution] Task B: Which computational problems can assisted through visualisation? [Solution] Binary search can be helped by constructing a binary tree. [End solution] Binary search of a sorted list – if a central array element (pivot) is larger than our criteria, we can eliminate the whole half of the array.	

Thinking Contextually

Activity 3: Visualising data and data mining	Resources

Pilot users were asked to rate the new interface for the upcoming update of the office software which was meant to address the complaints that the previous version was confusing the novices. The users who participated in the study were asked two questions:

'On the scale of 1–10 (where 1 is unusable and 10 is fully usable without training or reading the manual) rate the ease of use of the interface.'

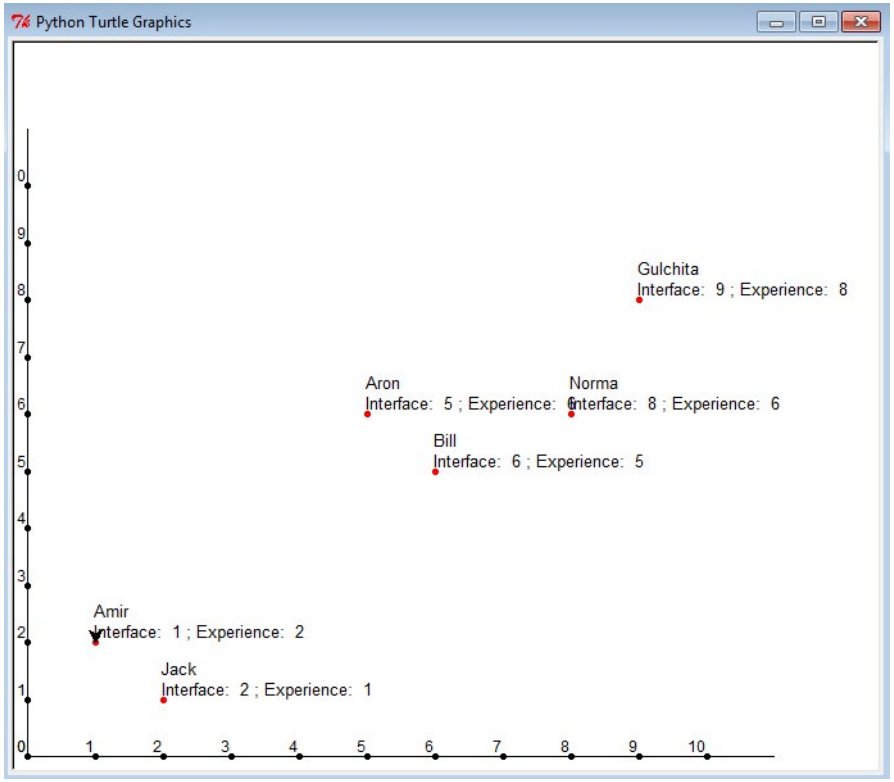
'On the scale of 1–10 (where 1 is not comfortable with computers at all and 10 is expert) rate the level of your general computer knowledge.'

The software supplier is trying to get a sense of how the level of experience correlates with the perception of the new interface.

The responses were as follows:

Name	Rating of Interface	Level of Computer Experience
Jack	2	1
Bill	6	5
Norma	8	6
Gulchita	9	8
Aron	5	6
Amir	1	2

Thinking Contextually

Activity 3: Visualising data and data mining	Resources
Task: Develop a program that plots the responses, so that a visual correlation can be established (correlation is stronger when data points are arranged in a narrow cloud and is weaker when that cloud is rounder). You can use this diagram as a guide: 	

Activity 3: Visualising data and data mining	Resources

[Solution]
 	
Learners can even use Scratch if Turtle or Small Basic is not available.
 	
They need to work out the algorithm for drawing the axis (optional) and plotting the dots. The dots look better to the casual user if they have labels on them, for example respondents' names and scores.
 	
In this solution, for simplicity, three arrays were used, instead of a 2d array. The arrays hold respondents' names, rating of the new interface and level of computer experience, respectively.
 	
`#set up actual data`
`names=["Jack","Bill","Norma","Gulchita","Aron","Amir"]`
`iface=[2,6,8,9,5,1]`
`exper=[1,5,6,8,6,2]`
`scale1=40 #set up spacing between the dots`
 	
`#set up the graphics via the turtle module`
`import turtle`
`s=turtle.Screen()`
`s.setworldcoordinates(0,0,500,500)#rescale to avoid going off screen`	

Activity 3: Visualising data and data mining	Resources

```
a=turtle.Turtle(); a.pu()
#draw vertical axis
a.setpos(0,0); a.pd(); a.setheading(90)
for j in range(0,11):
    a.dot(5,"black")
    a.write(j,font=("Arial",9),align="right")
    a.fd(40)
#draw horizontal axis
a.setpos(0,0); a.pd(); a.setheading(0)
for j in range(0,11):
    a.dot(5,"black")
    a.write(j,font=("Arial",9),align="right")
    a.fd(40)

a.pu()
for i in range(len(names)):
    a.setpos(iface[i]*scale1,exper[i]*scale1)
    a.tilt(45)
    a.dot(5,"red")
    a.write(" ".join([names[i],"\nInterface: ",str(iface[i]),"; Experience: ",
                      str(exper[i])]),move=False,font=("Arial",10),align="left")
s.exitonclick()
```

[End solution]

Activity 4: Shuffling a list	Resources

Task: Given this list, how would you shuffle it to randomise the position of the elements? (Elements must be unchanged.)
ar=[5,2,8,9,5,7,3]
[Solution]
There are multiple approaches to designing this algorithm but the simplest approach is to 'pop' (remove) elements from random indices and add them to the end of the list – just like the process of real-life card shuffling. First, we can see how to do it once, then extend to a loop.
`#how to shuffle these numbers?`
`print(ar)`
`import random`
`#shuffle once`
`a=ar.pop(random.randint(0,len(ar)-1))`
`ar.append(a)`	

Activity 4: Shuffling a list	Resources

```
print(ar)
now, let's shuffle it x times:
ar =[5,2,8,9,5,7,3]

#shuffle 20 times
for i in range(20):
    a=ar.pop(random.randint(0,len(ar)-1))
    ar.append(a)
    print(ar)
[7, 3, 8, 5, 5, 9, 2]
[7, 3, 8, 5, 5, 9, 2]
[7, 3, 8, 5, 5, 2, 9]
[7, 3, 8, 5, 5, 2, 9]
[7, 3, 8, 5, 2, 9, 5]
[7, 3, 8, 2, 9, 5, 5]
[7, 3, 2, 9, 5, 5, 8]
[7, 3, 9, 5, 5, 8, 2]
[7, 3, 9, 5, 8, 2, 5]
[7, 3, 5, 8, 2, 5, 9]
[7, 3, 5, 8, 2, 5, 9]
```

[End solution]

Learner resource 1.1

Activity 1: Divide and conquer.

Task A: How does Quicksort use the so-called divide and conquer strategy?

Task B: Using RosettaCode.org site find Quicksort implementation in your favoured language. Copy it out and supply with annotations that explain what each line does and especially how recursion makes it possible. (http://rosettacode.org/wiki/Sorting_algorithms/Quicksort, available under GNU Free Documentation License 1.2.)

Activity 2: Backtracking.

Task A: After looking at the definition of computational 'backtracking', put the definition in plain English terms and identify a list of three real-life (non-computing) applications of backtracking.

Task B: Which computational problems can be assisted through visualisation?

Activity 3: Visualising data and data mining.

Pilot users were asked to rate the new interface for the upcoming update of the office software which was meant to address the complaints that the previous version was confusing the novices. The users who participated in the study were asked two questions:

'On the scale of 1–10 (where 1 is unusable and 10 is fully usable without training or reading the manual) rate the ease of use of the interface'

'On the scale of 1–10 (where 1 is not comfortable with computers at all and 10 is expert) rate the level of your general computer knowledge.'

Learner resource 1.1

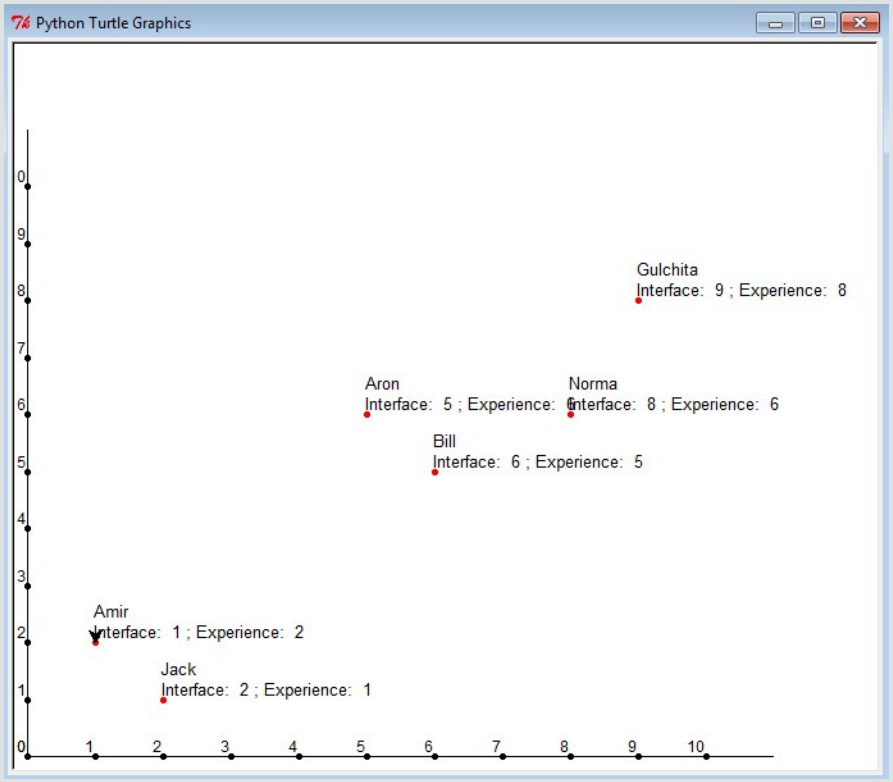
The software supplier is trying to get a sense of how the level of experience correlates with the perception of the new interface.

The responses were as follows:

Name	Rating of Interface	Level of Computer Experience
Jack	2	1
Bill	6	5
Norma	8	6
Gulchita	9	8
Aron	5	6
Amir	1	2

Learner resource 1.1

Task: Develop a program that plots the responses, so that a visual correlation can be established (correlation is stronger when data points are arranged in a narrow cloud and is weaker when that cloud is rounder). You can use this diagram as a guide:



Activity 4: Shuffling a list.

Task: Given this list, how would you shuffle it to randomise the position of the elements? (Elements must be unchanged.)

ar = [5,2,8,9,5,7,3]


We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

If you do not currently offer this OCR qualification but would like to do so, please complete the Expression of Interest Form which can be found here: www.ocr.org.uk/expression-of-interest

OCR customer contact centre
General qualifications
Telephone 01223 553998
Facsimile 01223 552627
Email general.qualifications@ocr.org.uk