CAMBRIDGE

Brighter Thinking

GCSE for OCR

COMPUTER SCIENCE

Teacher's Resource

Chapter 5: Searching and sorting algorithms

LEARNING OUTCOMES

By the end of this chapter students should be able to:

- explain why sorted lists are of more value than unsorted lists
- describe the bubble sort, selection sort and merge sort algorithms
- use these algorithms to sort lists into ascending and descending order
- describe the linear and binary search algorithms
- use these algorithms to search sorted and unsorted lists
- write code for the implementation of these algorithms.

What your students need to know

Students should:

- be able to use pseudocode to create variables and display algorithms
- be able to create and use arrays.

Vocabulary

- Ascending order
- Descending order
- Bubble sort
- Insertion sort
- Merge sort
- Sequential
- Linear sort
- Binary search

Common misconceptions and other issues

Students should be encouraged to work through the stages of each sorting and searching algorithm, showing the results of each stage.

It is worth stressing that when sorting into ascending order using a bubble sort, the highest unsorted value will be in its correct position at the end of each pass.

The efficiency of the different algorithms should be stressed, particularly when comparing linear and binary searches.

Skills and coding

- Maths skills:
  - Median (**Activity 5.7** and **Activity 5.8**).
- Coding skills:
  - Use of pseudocode
  - Nested loops
  - Creating and populating arrays
  - Finding the length of an array
  - Using loops to traverse an array
  - Using comparison operators.

Skills and coding for non-specialist teachers

1 Bubble sort

In a bubble sort, the first two items (i.e. items 1 and 2) are compared and are swapped round if they are not in the required order. Then the next pair (items 2 and 3) are compared. This continues until the end of the list.

If they are being sorted into ascending order, the item with the highest value will be in its correct position at the end of the first pass.

Passes are repeated until there are no swaps.

The code for a bubble sort is given after **Activity 5.2** of the Student Book.

This pseudocode will check through the entire list each time. The students could be asked to write the code for a bubble sort and amend it so that it does not check the numbers at the end of the list, which are already in their correct positions each time it carries out a pass. This reinforces the consideration of algorithm efficiency.

The algorithm can be adapted by decreasing the length of the list before the next iteration.

This line should be added at the end of the algorithm:

```
next x
N = N - 1
endwhile
```

Bubble sort is explained in the Student Book and the solution to **Activity 5.2**.

2 Insertion sort

In an insertion sort each item is examined in turn and moved into its correct position. If it is lower than items to its left then those items have to be moved to the right to accommodate it.

Insertion sort is explained in the Student Book and the solutions to **Activity 5.3** and **Activity 5.4**.

Activity 5.4 asks students to create and test a program to carry out an insertion sort.

A possible algorithm in pseudocode is given below.

Pseudocode	Explanation
`S  = List of items` `N = length of list`	The variable S is assigned to the array. The variable N is set to the length of the array.
`for x = 1 to N`	This starts a loop that starts at '1' and ends at the value equal to the length of the array.
`    value = S[x]`	The variable 'value' is assigned the value at the index 'x' of the loop. Arrays start at index 0 and so the loop starts with the second item of the array.
`    position = x`	The variable 'position' is assigned the value of 'x'. So for the first turn of the loop, it will be equal to 1.
`    while position > 0 AND S[position - 1] > value`	This starts a while loop that will continue while the position is greater than 0 so that it does not try to go beyond the start of the array when it compares the number with the one to the left, i.e. it would generate an error if it tried to compare it with the number at an index of 0-1! It will also continue while the number stored in the variable 'value' is less than the value stored in the array at the index to the left, i.e. one less than the index holding the number stored in 'value'.

Pseudocode	Explanation
`S[position] = S[position - 1]`	If the number to the left is greater, then it is moved to the index of the original number, i.e. it is moved to the right.
`position = position - 1`	The variable 'position' is now reduced by one so that the search can continue further to the left.
`endwhile`	The while loop will now run again with the index position set one place to the left, i.e. one place lower.
`S[position] = value`	Once the conditions set in the while loop are not met i.e. the start of the array has been reached or the number in the array at the lower index to the left is actually less than the number stored in 'value', then it is written into the array at this index position.
`next x`	This ends the 'for' loop and it will run again until it reaches the end of the array.

3 Merge sort

Merge sort employs an algorithmic based on recursion. It is said to be, **divide-and-conquer**, as it breaks the problem into sub problems that are similar to the original problem, recursively solves the sub problems, and finally combines the solutions to the sub problems to solve the original problem. Because divide-and-conquer solves sub problems recursively, each sub problem must be smaller than the original problem, and there must be a base case for sub problems. It has three parts:

1. **Divide** the problem into a number of sub problems that are smaller instances of the same problem.
2. **Conquer** the sub problems by solving them recursively.
3. **Combine** the solutions to the sub problems into the solution for the original problem.

Merge sort is explained in the Student Book and the solution to **Activity 5.5**.

4 Sorting algorithms

Linear and binary search algorithms are very straightforward and students should have encountered them in their daily lives.

Comparisons of the best and worst case scenarios provide a good example of discussing algorithm efficiency.

Sorting algorithms are explained in the Student Book and the solutions to **Activity 5.6** to **Activity 5.9**.

The students are not asked to code a binary algorithm but it could be done as an extension activity.

An algorithm, in pseudocode, is shown below.

Pseudocode	Explanation
`target = int(input("Please enter a target: "))`	A variable to store the item to be searched for is declared as 'target'. In this instance a number is the expected input.
`start = 0`	The variable 'start' is set to the index number of the first item
`end = list.length-1`	The variable 'end' is set to the index of the last item of the list.
`found = False`	The Boolean variable 'found' is set to False. This is used to indicate that the search item has not been found.
`while start <= end AND found == False`	A 'while' loop is set up.
`middle = ((start + end)/2)`	The median item is found.

`if list[middle] == target then`
`print(target + " is in the list")`
`found = True`
	If the median number is the target, then 'found' is set to True and the user is informed.

Pseudocode	Explanation
`elseif target < list[middle]` `end = middle - 1`	If the search item is less than the mean, then the variable 'end' is set to the item to the left of the mean – the next item less than the mean.
`else` `start = middle + 1`	If the search item is greater than the mean, then the variable 'start' is set to the item to the right of the mean – the next item greater than the mean.
`endif`	
`endwhile`	

`If found = False then`
`print(target + " is not in the list.")`
`endif`
	The user is informed if the search item has not been found.

Prompting questions

- Why is it important to sort things into an order?
- How many examples can you think of where data has been sorted?
- Bubble sort, insertion sort and merge sort are all different ways to sort data. How do you think these work?

Starters, plenaries, enrichment and assessment ideas

Starters and plenaries

- Provide students with numbered cards in a random order. Tell them to place the cards next to each other and then sort them into order without telling them how. Then ask students how they sorted the data and to write their strategy down. How efficient was their strategy? Could they have done it better? Ask pairs of students to compare their techniques with each other, which one was faster and why? This can then be used to compare against the sorting techniques discussed in the chapter. It works well either as a starter or a plenary.
- Students play a game. With a deck of sorted cards, a player chooses a number secretly. The 'magician' has to work it out using only questions such as 'is it higher or lower?' Another option is to play 20 questions with the class. The ideal questions are those which automatically eliminate at least half the options; such as 'is it male or female?' The popular CS4FN / Teaching London Computing initiative, funded by the Mayor of London, has a detailed outline of activities to teach searching algorithms. **www.cambridge.org/links/kotd4006**

Enrichment activities

- Ask students to investigate the different search and sort algorithms. Can they find the most effective YouTube video that explains the different algorithms and their differences and characteristics?
- Search through **www.cambridge.org/links/kotd4007** and read through the resources. Using inspiration from that style, write an article to explain the different sort algorithms.

Assessment ideas

- Students carry out an investigation to code the different sort algorithms and run them with the same set of data. Can students discover which is the most efficient? Ask them to consider their own criteria for comparison and present their findings and justifications at the end.

Answers

Activity 5.1

Students' own answers

Activity 5.2

20	15	3	13	9	2	6
15	3	13	9	2	6	20
3	13	9	2	6	15	20
3	9	2	6	13	15	20
3	2	6	9	13	15	20
2	3	6	9	13	15	20
2	3	6	9	13	15	20

Activity 5.3

20	15	3	13	9	2	6
15	20	3	13	9	2	6
3	15	20	13	9	2	6
3	13	15	20	9	2	6
3	9	13	15	20	2	6
2	3	9	13	15	20	6
2	3	6	9	13	15	20

Activity 5.4

This algorithm assumes that the array *testResults* has already been initialised and populated and that a *.length* method is available that determines the number of elements in a list. Please see algorithm on page 33.

```
number = testResults.length
for index = 1 to number - 1:
        currentMark = testResults[index]
        position = index
        while position > 0 AND testResults[position - 1] > currentMark
                testResults[position] = testResults[position - 1]
                position = position - 1
        endwhile
        testResults[position] = currentMark
next index
print(testResults)
```

Activity 5.5

20	15	3	13	9	2	6

20/15/3/13. 9/2/6

20/15. 3/13. 9/2. 6

20. 15. 3. 13. 9. 2. 6

15/20. 3/13. 2/9. 6

3/13/15/20. 2/6/9

2/3/6/9/13/15/20

Activity 5.6

This algorithm assumes that the array *popularNames* has already been initialised and populated with the hundred most popular names.

```
found = false
index = 0
name = input('Please enter the name you want to search for: ')
```

```
while found == false AND index < 100
        if name == popularNames[index] then
                found = true
        endif
        index = index + 1
endwhile
if found == true then
        print(name, 'is in the list.')
else
        print(name, 'is not in the list.')
endif
```

Activity 5.7

3	15	21	27	33	39	42	48	56	60	66	67	69
3	15	21	27	33	39							
3	15	21										
21												

Activity 5.8

- Pick median 15
- Too low, so select right hand side, 6 numbers left so choose 56
- Too high, so answer is 45

Activity 5.9

- Find the length of the array
- Start (of search items) equals 0
- End of search items equals length of array – 1
- While Start is less than or equal to End
- Middle equals (start + end) / 2
- If Middle is equal to number entered tell the user and stop the loop
- If middle is less than number entered then Start equals Middle + 1
- If middle is greater than number entered then End equals Middle – 1
- End of while loop
- Inform the user that the number is not present