ANSWERS

A2.9 Computational methods

Recursion and computational thinking · A level · OCR H446 2.2.2, AQA 7517 4.4.1.1, Eduqas A500QS 1.3 · about 25 min

What this lesson is about

Problem recognition, divide and conquer, backtracking, heuristics, performance modelling, data mining and visualisation.

- Is it a problem a computer can solve?
- Divide and conquer
- Backtracking
- Heuristics
- Performance modelling
- Data mining and visualisation

Questions

6 marks in all

1. What is a heuristic? [1 mark]

- ☐ **A** A rule of thumb that finds a good enough answer quickly, without guaranteeing the best
- ☐ **B** An algorithm that always finds the optimal answer
- ☐ **C** A way of storing results to avoid recalculating them
- ☐ **D** A diagram of a solution

> **Answer:** A. Heuristics trade a guarantee of the best answer for speed, when an exact method would take too long.

2. Which method builds a solution step by step and, on reaching a dead end, undoes the last choice and tries another? [1 mark]

- ☐ **A** Backtracking
- ☐ **B** Pipelining
- ☐ **C** Data mining
- ☐ **D** Visualisation

> **Answer:** A. Mazes, sudoku and the queens puzzle are classic backtracking problems.

3. A brute force program checks every order of visiting 8 places. How many orders is that? [1 mark]

> **Answer:** 40320. 8! = 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 40,320. Estimating this before building is performance modelling.

4. A supermarket searches millions of receipts and finds that people who buy barbecue charcoal often buy burgers. Which method is this? [1 mark]

- ☐ **A** Data mining
- ☐ **B** Divide and conquer
- ☐ **C** Backtracking
- ☐ **D** Heuristics

> **Answer:** A. Data mining finds patterns in large data sets that were not known in advance.

5. What does this program print? [1 mark]

```
def biggest(items, lo, hi):
    if lo == hi:
        return items[lo]
    mid = (lo + hi) // 2
    a = biggest(items, lo, mid)
    b = biggest(items, mid + 1, hi)
    return a if a > b else b

print(biggest([12, 47, 5, 33, 29], 0, 4))
```

Answer:

```
47
```

Divide and conquer: the largest of each half is found recursively, and the larger of the two is returned.

6. Which features make a problem suitable for solving by computational methods? [1 mark]

Tick every answer that is true.

- ☐ **A** It can be stated precisely, with a clear test of a correct answer
- ☐ **B** It can be solved by a finite sequence of steps
- ☐ **C** An algorithm can solve it in a reasonable time for the sizes that matter
- ☐ **D** It depends on personal taste

Answer: A, B, C. A question of taste has no precise test of a correct answer.

The task: mining the run log

The starter holds a log of 12 runs, each a dictionary with the keys `"robot"`, `"speed"` (a whole number) and `"bumps"` (how many times it hit something). Look for a pattern between speed and bumps. - Find the different speeds that appear in the log, from the data (do not type them into your code), and sort them from smallest to largest. - For each speed, print `speed <speed>: <runs> runs, <mean> bumps per run`, with the mean to 2 decimal places, for example `speed 40: 4 runs, 0.25 bumps per run`. - Finally print `most bumps: speed <speed>`, the speed with the highest mean.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

runs = [
    {"robot": "Ada", "speed": 40, "bumps": 0}, {"robot": "Bolt", "speed": 60, "bumps": 1},
    {"robot": "Cog", "speed": 80, "bumps": 2}, {"robot": "Ada", "speed": 40, "bumps": 1},
    {"robot": "Bolt", "speed": 60, "bumps": 0}, {"robot": "Cog", "speed": 80, "bumps": 3},
    {"robot": "Ada", "speed": 40, "bumps": 0}, {"robot": "Bolt", "speed": 60, "bumps": 1},
    {"robot": "Cog", "speed": 80, "bumps": 2}, {"robot": "Ada", "speed": 40, "bumps": 0},
    {"robot": "Bolt", "speed": 60, "bumps": 2}, {"robot": "Cog", "speed": 80, "bumps": 1},
]
```

The hint students can ask for: First find which speeds appear in the log, without assuming. Then, for each one, pick out the runs at that speed and total their bumps. Keep track of the best mean seen so far as you go.

A solution

```
from bugbot import *
connect()

runs = [
    {"robot": "Ada", "speed": 40, "bumps": 0}, {"robot": "Bolt", "speed": 60, "bumps": 1},
    {"robot": "Cog", "speed": 80, "bumps": 2}, {"robot": "Ada", "speed": 40, "bumps": 1},
    {"robot": "Bolt", "speed": 60, "bumps": 0}, {"robot": "Cog", "speed": 80, "bumps": 3},
    {"robot": "Ada", "speed": 40, "bumps": 0}, {"robot": "Bolt", "speed": 60, "bumps": 1},
    {"robot": "Cog", "speed": 80, "bumps": 2}, {"robot": "Ada", "speed": 40, "bumps": 0},
    {"robot": "Bolt", "speed": 60, "bumps": 2}, {"robot": "Cog", "speed": 80, "bumps": 1},
]
speeds = []
for run in runs:
    if run["speed"] not in speeds:
        speeds.append(run["speed"])
speeds.sort()
best_speed = None
best_mean = -1
for speed in speeds:
    count = 0
    bumps = 0
    for run in runs:
        if run["speed"] == speed:
            count = count + 1
            bumps = bumps + run["bumps"]
    mean = bumps / count
    print(f"speed {speed}: {count} runs, {mean:.2f} bumps per run")
    if mean > best_mean:
        best_mean = mean
```

```
        best_speed = speed
print("most bumps: speed", best_speed)
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.