ANSWERS

F5.6 Binary search

Algorithms · GCSE · OCR J277 2.1.3, AQA 8525 3.1.3, Edexcel 1CP2 1.2.6 · about 15 min

What this lesson is about

Halving a sorted list, and why it needs sorted data.

- The algorithm
- Hear it halve
- Why it needs sorted data

Questions

5 marks in all

1. What must be true of a list before binary search can be used? [1 mark]

- ☐ **A** It must be sorted
- ☐ **B** It must be short
- ☐ **C** It must hold numbers only
- ☐ **D** It must have an even number of items

> **Answer:** A. Binary search discards half the list after each check, which only works if the items are in order.

2. Binary search on a sorted list of 1,000 items. What is the most checks it needs? [1 mark]

> **Answer:** 10. Halving 1,000 repeatedly reaches one item after about 10 halvings.

3. Binary search looks for 23 in [2, 5, 8, 12, 16, 23, 38, 56, 72, 91], checking index 4 (16) first. What happens next? [1 mark]

- ☐ **A** It discards 16 and everything before it, and searches the upper half
- ☐ **B** It discards the upper half
- ☐ **C** It checks index 5 next, then 6, then 7
- ☐ **D** It starts again at index 0

> **Answer:** A. 23 is bigger than 16, so the target can only be in the upper half.

4. Why might a programmer use linear search instead of binary search? [1 mark]

- ☐ **A** The data is not sorted
- ☐ **B** Linear search is always faster
- ☐ **C** Binary search cannot find numbers
- ☐ **D** Binary search needs more memory

> **Answer:** A. Sorting first can cost more than a linear search on a short or unsorted list.

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

```
low, high = 0, 9
mid = (low + high) // 2
print(mid)
```

> **Answer:**
>
> ```
> 4
> ```
>
> (0 + 9) // 2 is 4, rounding down.

The task: binary search the markers

The task asks `Which marker?` and answers `56`. Using **binary search** (not `in` or `.index`), find it in the sorted list `[2, 5, 8, 12, 16, 23, 38, 56, 72, 91]`. Beep once for every check, and print `found 56 at index <i> after <n> checks`.

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

ids = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
target = int(input("Which marker? "))
```

The hint students can ask for: Keep a low and a high edge and look at the middle of what is left. Compare the middle value with the target, then move whichever edge rules out half the list. Count and beep once per look.

A solution

```
from bugbot import *
connect()
ids = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
target = int(input("Which marker? "))
low = 0
high = len(ids) - 1
checks = 0
found = -1
while low <= high:
    mid = (low + high) // 2
    checks = checks + 1
    tone(400 + checks * 100, 0.15)
    if ids[mid] == target:
        found = mid
        break
    elif target < ids[mid]:
        high = mid - 1
    else:
        low = mid + 1
print("found", target, "at index", found, "after", checks, "checks")
```

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