WORKSHEET

A5.3 Linear and binary search

Algorithms and complexity · A level · OCR H446 2.3.1, AQA 7517 4.3.4.1, Eduqas A500QS 1.3 · about 25 min

BugBotLab

Name ____________ Class ____________ Date ____________

What this lesson is about

Tracing both searches in pseudocode, recursive binary search, O(n) against O(log n), and when sorting first pays off.

- Linear search
- Binary search
- Binary search by recursion
- Is sorting first worth it?

Questions

6 marks in all

1. What is the worst-case time complexity of binary search? [1 mark]

- ☐ **A** O(log n)
- ☐ **B** O(n)
- ☐ **C** O(1)
- ☐ **D** O(n log n)

2. Binary search looks at the middle item and halves what is left each time. What is the largest number of items it must look at to search a sorted list of 1,000,000 items? [1 mark]

____

3. Which statement about linear search is true? [1 mark]

- ☐ **A** It works on unsorted data and is O(n)
- ☐ **B** It needs sorted data and is O(n)
- ☐ **C** It works on unsorted data and is O(log n)
- ☐ **D** It is O(1) in the worst case

4. This binary search prints each middle index it looks at. What does it print? [1 mark]

```
items = [3, 9, 14, 21, 30, 38, 45, 52, 60, 71, 88]
low, high = 0, len(items) - 1
target = 60
while low <= high:
    mid = (low + high) // 2
    print(mid)
    if items[mid] == target:
        break
    elif target < items[mid]:
        high = mid - 1
    else:
        low = mid + 1
```

____

____

____

5. Recursive binary search has the same time complexity as the iterative version. What is its space complexity? [1 mark]

- ☐ **A** O(log n), because of the stack frames of the waiting calls
- ☐ **B** O(1), like the iterative version
- ☐ **C** O(n), because it copies the list
- ☐ **D** O(n log n)

6. A list of 1,000,000 unsorted readings will be searched just once. Which is the better choice? [1 mark]

- ☐ **A** Linear search, because sorting first costs more than the search saves
- ☐ **B** Sort it and then use binary search, because binary search is always faster
- ☐ **C** Binary search on the unsorted list
- ☐ **D** Neither can search a list that large

The task: recursive binary search

Write `binary_search(items, target, low, high)` **recursively** (it must call itself, with no `while` or `for` loop inside it). It searches the sorted list `items` between indexes `low` and `high` inclusive, finds the middle with `(low + high) // 2`, and **returns a tuple** `(index, looks)`: the index of `target`, or -1 if it is not there, and the number of middle items it looked at. `items` is the 1,000 even numbers 0, 2, 4, ... 1998. Print exactly three lines: - `734: index <index> after <looks> looks` - `735: not found after <looks> looks` - `most looks: <n>`, the largest number of looks needed to find any one of the 1,000 items in the list (search for every item to find out).

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

items = list(range(0, 2000, 2))

def binary_search(items, target, low, high):
    return -1, 0
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a5-3-linear-and-binary-search/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. How many looks does binary search need, at most, for 5,000 items? Work it out, then check it.
2. Change linear search so it stops early on a **sorted** list once it passes where the target would be. What is its worst case now?
3. Write the recursive binary search in AQA pseudo-code.