WORKSHEET

A5.6 Quick sort

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

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

Pivots and in-place partitioning, O(n log n) on average and O(n^2) at worst, and the four sorts compared.

- The idea
- Partitioning in place
- Analysis
- The four sorts side by side

Questions

5 marks in all

1. What is the worst-case time complexity of quick sort? [1 mark]

- ☐ **A** $O(n^2)$
- ☐ **B** O(n log n)
- ☐ **C** O(n)
- ☐ **D** $O(2^n)$

2. Quick sort always uses the last item as its pivot. Which list makes it slowest? [1 mark]

- ☐ **A** A list that is already sorted
- ☐ **B** A list in random order
- ☐ **C** A list with all different values
- ☐ **D** A short list

3. Which are true of quick sort? [1 mark]

Tick every answer that is true.

- ☐ **A** After partitioning, the pivot is in its final position
- ☐ **B** It is stable
- ☐ **C** Its average case is O(n log n)
- ☐ **D** It needs O(n) extra space for merged lists

4. This partitions the list around its last item. What does it print? [1 mark]

```
items = [7, 2, 9, 4, 5]
pivot = items[-1]
i = 0
for j in range(len(items) - 1):
    if items[j] < pivot:
        items[i], items[j] = items[j], items[i]
        i = i + 1
items[i], items[-1] = items[-1], items[i]
print(items, i)
```

5. How does quick sort differ from merge sort in where it does its work? [1 mark]

- ☐ **A** Quick sort works while dividing (partitioning); merge sort works while combining (merging)
- ☐ **B** Quick sort works while combining; merge sort works while dividing
- ☐ **C** Both do all their work while dividing
- ☐ **D** Neither is a divide and conquer algorithm

The task: place the pivots

Write `quick_sort(items, low, high)` using the in-place partition from this lesson: the pivot is `items[high]`, items **strictly less** than the pivot go to its left, and the function sorts the left part before the right part. It sorts `items` between indexes `low` and `high` inclusive, and **returns the number of comparisons** with a pivot that it made. Each time a partition puts a pivot in its final place, print `pivot <value> placed at index <index>`. Parts of fewer than 2 items are not partitioned and print nothing. Then print, in this order: - `sorted:` followed by `readings` after sorting - `readings: <n> comparisons` - `ordered: <n> comparisons`, sorting `ordered`, which is already in order (print no pivot lines for this one: give the function a way to switch printing off, such as a parameter `show` that defaults to `True`)

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

readings = [38, 12, 71, 45, 9, 83, 27, 60, 50]
ordered = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

def quick_sort(items, low, high):
    return 0
```

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

Do it on the robot

www.bugbotlab.com/learn/a5-6-quick-sort/

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

Challenges

1. Change the partition to use the middle item as the pivot (swap it to the end first). How many comparisons does the already sorted list take now?
2. Trace quick sort on `[5, 5, 5, 5, 5]`. What happens, and why is it slow?
3. Show with a list of `(name, time)` records that quick sort is not stable.