LESSON 6

Basic Sorting and Searching

80 minutes (2 periods) | Foundations of Computer Science | Outcome 1.6 | Outcome 1.7 | Outcome 2.5 | Outcome 2.6 | Outcome 2.7 | Outcome 2.8 | Outcome 2.10

WHAT THIS LESSON DOES

In this lesson, you'll learn the essentials of sorting and searching algorithms, vital for organising and retrieving data efficiently. Follow step-by-step guidance to understand bubble sort, linear search, and binary search, and simulate them using MakeCode for micro:bit.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.6	explain the operation of a variety of algorithms
1.7	develop algorithms to implement chosen solutions
2.5	use pseudo code to outline the functionality of an algorithm
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
2.7	implement algorithms using a programming language to solve a range of problems
2.8	apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm
2.10	explain the common measures of algorithmic efficiency using any algorithms studied

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamental concepts of sorting and searching algorithms in computer science.
- Grasp the mechanics of bubble sort and its step-by-step process for organising data.
- Comprehend the differences between linear and binary search methods for data retrieval.
- Apply sorting and searching algorithms through practical implementation using MakeCode for micro:bit.
- Evaluate algorithm efficiency by comparing time complexities and their real-world applications.

BEFORE THE LESSON

- Open makecode.microbit.org yourself and confirm it loads through the school filter, the two search tasks run there rather than on physical devices.
- Have paper and pens ready for the manual bubble sort task, or confirm students can work it on rough paper.
- Work the bubble sort on [7, 2, 9, 1, 5] yourself first so you can check pass-by-pass answers quickly.

Basic Sorting and Searching

HOW THE LESSON RUNS

Introduction
Understanding Sorting
Bubble Sort
Practical Task: Bubble Sort
Understanding Searching
Linear Search
Practical: Implement Linear Search
Binary Search
Practical: Implement Binary Search
Efficiency Discussion
Wrap Up

TEACHING MOVES

- 2. Understanding Sorting. Anchor sorting in something they already sort: a class list by name, marks by value. Ask why sorting first makes later searching easier before naming any algorithm.
- 3. Bubble Sort. Stress the invariant: after each pass the largest unsorted value is in its final place, so each pass is one shorter. This is what makes it terminate.
- 4. Practical Task: Bubble Sort. Have them write the full list after every pass, not just the final answer. Circulate and check they stop when a pass makes no swaps, that is the sorted signal.
- 6. Linear Search. Point out linear search does not need a sorted list, unlike binary search. Ask when it is the only option available.
- 7. Practical: Implement Linear Search. Walk them through building the blocks exactly as given. Note positions display starting from 1, not 0, so warn early about off-by-one when they read the output.
- 8. Binary Search. Emphasise the precondition: binary search only works on a sorted list. Use the phone book image, then have them predict how many checks 1000 items needs.
- 9. Practical: Implement Binary Search. Have them trace the middle element by hand on [1, 3, 5] before running it, then confirm the simulator matches their trace.
- 10. Efficiency Discussion. Contrast concretely: for 10 items bubble sort takes roughly 100 steps, binary search on a sorted list takes about 4. Let the numbers make the point, not the notation.

WHAT TO WATCH FOR

- Students think binary search is simply faster than linear search in all cases and use it on any list. Show binary search returning wrong results on an unsorted list. Make the sorted precondition a rule they check first, every time.
- Off-by-one confusion when the code reports positions starting from 1 while the list is conceptually indexed from 0. Number the list positions on the board both ways and agree which convention this task uses before they read any output.
- Believing bubble sort needs a fixed number of passes rather than stopping when no swaps occur. Have them add a swap counter in the manual task and see the run finish early once the list is already ordered.

DIFFERENTIATION

- Emerging: Give a shorter three-item list for the manual bubble sort so they see one full sort finish before tackling five. Pair them with a partner to build the MakeCode blocks, one reads the instruction, one places the block.
- Developing: Ask them to predict the number of swaps before doing the manual sort, then check against their count. Have them explain in words why binary search needs a sorted list before running the second task.
- Proficient: Challenge them to rewrite the linear search using an actual loop rather than the unrolled version, and to handle a target that is not present. Ask them to work out and justify the worst-case number of comparisons for binary search on a list of 16 items.