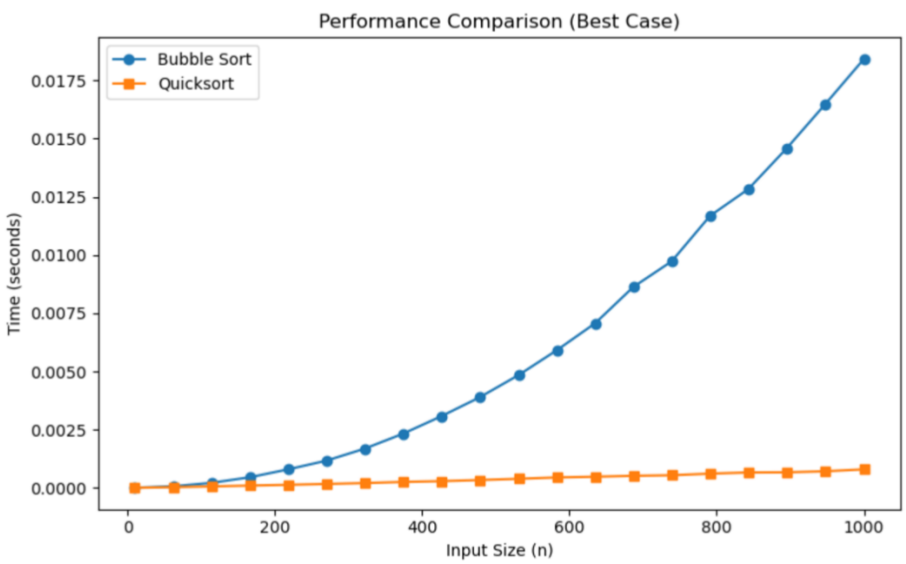
Exercise2

Ex2.3

1. Best case



Bubble Sort:

- Operates on an already sorted list.
- Performs minimal work since it mainly verifies the order.
- Shows near-optimal performance even for small inputs.

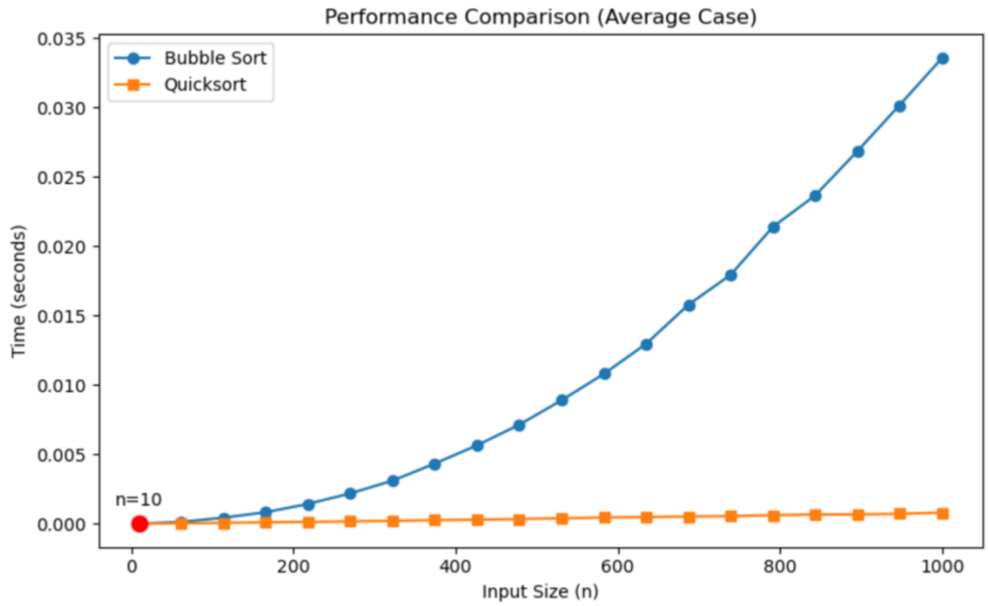
Quicksort:

- Uses the first element as the pivot.
- On sorted data, this leads to highly unbalanced partitions, increasing recursive calls.
- Experiences degraded performance as input sizes grow.

conclusion

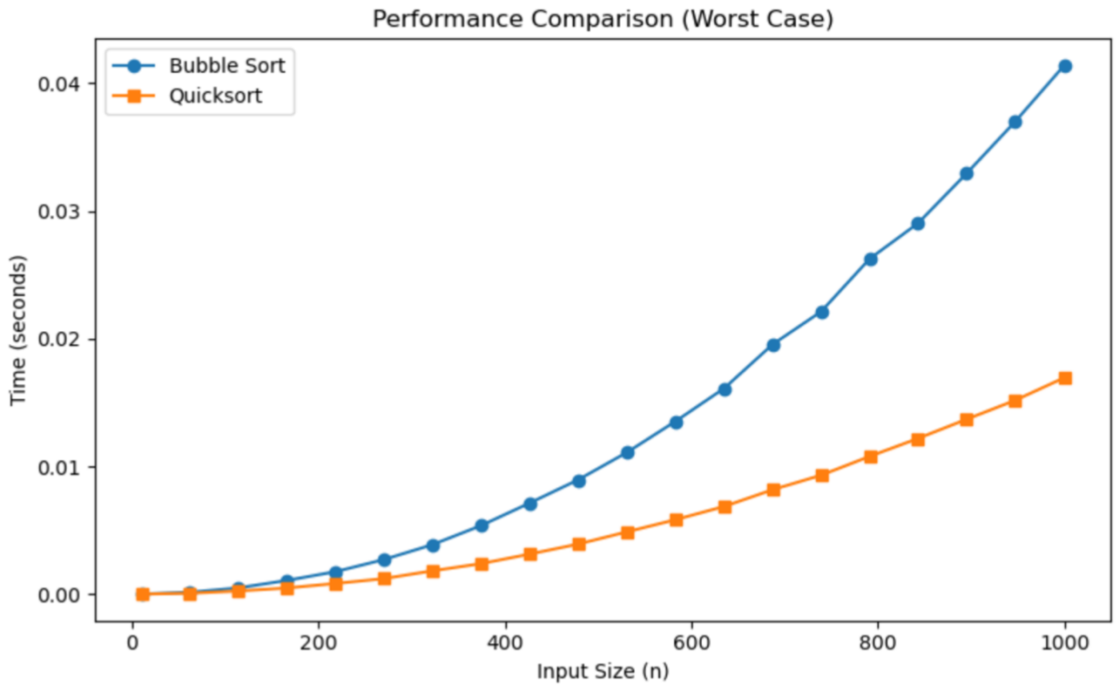
- The plot consistently shows bubble sort's runtime lower than quicksort's for all input sizes.
- This clearly indicates that, in the best-case scenario, bubble sort performs better.

2. Average case



- **Bubble Sort:**
  - Operates on random lists but still suffers from $O(n^2)$ behavior.
  - For small input sizes (e.g., n = 10, 20), the performance difference is marginal.
- **Quicksort:**
  - Also processes random lists and benefits from efficient partitioning.
  - Initially, the overhead of recursion keeps performance differences small.
  - As input sizes increase, the efficient O (n log n) scaling becomes evident.
- **conclusion:**
  - A clear crossover point is observed—around n ≈ 270—where quicksort's runtime becomes lower than bubble sorts.
  - A red marker in the plot emphasizes this threshold, clearly indicating the input size where quicksort begins to perform better.

3. Worst case



Bubble Sort:

- Processes a reverse sorted list, forcing maximum comparisons and swaps.
- Quadratic time complexity ($O(n^2)$) becomes evident as input size increases.

Quicksort:

- Evaluated on a randomly generated list, generally yielding balanced partitions.
- Exhibits O (n log n) performance, which scales much better with larger inputs.

conclusion:

- The runtime curve for bubble sort rises sharply with increasing n, while quicksort's curve remains much lower.
- For larger inputs, the plot distinctly shows that quicksort outperforms bubble sort.

Ex2.4

- The average-case performance plot reveals that quicksort begins to outperform bubble sort at approximately n ≈ 270.
- For inputs smaller than 270, the overhead of quicksort's recursive calls often results in similar or even slightly slower performance compared to bubble sort, classifying these as "small" inputs.
- For inputs of 270 or more, quicksort's $O(n \log n)$ efficiency becomes pronounced, making it significantly faster than bubble sort's $O(n^2)$ behavior.
- Thus, **n ≈ 270 is chosen** as the threshold to differentiate between small and large inputs based on observed performance differences.