To analyze the cyclomatic (McCabe) complexity of the given **Quicksort** algorithm, we need to break it down into its two components: the QUICKSORT recursive function and the PARTITION procedure. Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is calculated by constructing a control flow graph of the program, where **nodes** represent code blocks and **edges** represent control flow. The cyclomatic complexity (V(G)) is given by the formula:

- **$V(G) = E - N + 2P$**
- Where:
- **E** = Number of edges
- **N** = Number of nodes
- **P** = Number of connected components (usually 1 for a single program)

Flowchart of the Algorithm

- **QUICKSORT(A, p, r)**

1. Start
2. Check if $p < r$
   - If false → End
   - If true → Continue
3. Call PARTITION(A, p, r) and store result in q

4. Recursively call QUICKSORT(A, p, q - 1)
5. Recursively call QUICKSORT(A, q + 1, r)
6. End

- **PARTITION(A, p, r)**

1. Set x = A[r]
2. Set i = p - 1
3. Loop from j = p to r - 1
    - If A[j] ≤ x:
        - Increment i
        - Swap A[i] and A[j]
4. Swap A[i+1] and A[r]
5. Return i + 1

Control Flow Graph (CFG)

- We'll denote nodes and edges for both QUICKSORT and PARTITION.
- **Nodes (n):**
- n1: Start QUICKSORT
- n2: Check p < r

- n3: Call PARTITION(A, p, r)
- n4: Call QUICKSORT(A, p, q-1)
- n5: Call QUICKSORT(A, q+1, r)
- n6: End QUICKSORT
- n7: Start PARTITION
- n8: Assign x = A[r]
- n9: Assign i = p - 1
- n10: For loop j = p to r - 1
- n11: If A[j] ≤ x
- n12: i = i + 1
- n13: Swap A[i] ↔ A[j]
- n14: End If
- n15: End For
- n16: Swap A[i+1] ↔ A[r]
- n17: Return i + 1
- **Edges (e):**

  (Edges represent transitions between steps)
- e1: n1 → n2
- e2: n2 (False) → n6

- e3: n2 (True) → n3
- e4: n3 → n4
- e5: n4 → n5
- e6: n5 → n6
- e7: n7 → n8
- e8: n8 → n9
- e9: n9 → n10
- e10: n10 → n11 (loop continues)
- e11: n11 (True) → n12
- e12: n12 → n13
- e13: n13 → n14
- e14: n11 (False) → n14
- e15: n14 → n10
- e16: n10 (loop ends) → n15
- e17: n15 → n16
- e18: n16 → n17

Calculating Cyclomatic Complexity

- Let’s count the number of nodes and edges in the **combined graph**:
- Total nodes (N): 17
- Total edges (E): 18
- Components (P): 1 (single program)
- Using McCabe’s formula:
- **$V(G) = E - N + 2P = 18 - 17 + 2 \times 1 = 3$**
- However, we must consider **decision points** like if and for. A more intuitive way to calculate cyclomatic complexity is:
- **V(G) = Number of decision points + 1**
- In this algorithm:
- One decision in QUICKSORT: if $p < r$
- One for loop in PARTITION: for $j = p$ to $r-1$
- One if statement inside the loop: if $A[j] \leq x$
- Total decision points = 3
- So, **$V(G) = 3 + 1 = 4$**
- This agrees with McCabe’s definition that cyclomatic complexity also measures the **number of linearly independent paths** through the program (McCabe, 1976). Higher

complexity implies more testing effort, potential for errors, and maintenance difficulty (Kan, 2002).

Conclusion

The cyclomatic complexity of the given Quicksort algorithm is **4**. This moderate value indicates that the code has a manageable number of paths, and each needs to be tested to ensure full branch coverage. By understanding and applying cyclomatic complexity, developers can improve code quality through better testing and design decisions.

References:

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), 308–320. https://ieeexplore.ieee.org/document/1702388

Kan, S. H. (2002). *Metrics and Models in Software Quality Engineering*. Addison-Wesley. https://www.amazon.com/Metrics-Models-Software-Quality-Engineering/dp/0201729156

Wordcount: 626