Question 1

Use the statement counting approach to determine the number of statements executed by the f---

Question 4

Do the answers to question 3 change if the call to `t.toString()` is $\Theta(n)$ rather than $\Theta(1)$? Why or why not?

Solution

Answer: No, the asymptotic complexities remain the same.

Explanation:

When analyzing asymptotic complexity, we consider how the dominant terms

Solution

The best case occurs when the `if`-statement inside the loop is **never** executed—i.e., every character fails the lowercase test. Let $n = s.length()$.

1. **Initialization:**
   - `StringBuilder t = new StringBuilder(s);` → 1 statement
2. **For Loop:**
   - Loop control (`i < s.length()`) and update (`i++`), plus the `if` condition check → 2 statements per iteration
   - Number of iterations: n
   - One extra loop-condition check when the loop exits → +1 statement
   - **Total loop cost:** $2n + 1$
3. **Return:**

Solution

Before we analyze, here is a quick reminder of what each notation means:

- **Big-O (O):** An upper bound on the growth rate of a function. It describes the *worst-case* scenario—how quickly the running time can grow as the input size increases.
- **Big-Omega (Ω):** A lower bound on the growth rate. It describes the *best-case* scenario—the minimum time the algorithm will take as the input size increases.
- **Big-Theta (Θ):** A tight bound. It means the function grows at the same rate in both the best and worst cases; the running time is always proportional to this rate.

- **Worst Case (Big-O):**
  In the worst case, every character in the string is a lowercase letter, so the body of the `if` statement executes every time. The loop runs for every character in the string, and all operations inside the loop are constant-time. Therefore, the worst-case time complexity is
  $$O(n)$$
- **Best Case (Big-Ω):**
  In the best case, none of the characters are lowercase (so the `if` body never executes). However, the loop still iterates over every character, performing the `if` check each time. Thus, even in the best case, the time taken is proportional to the length of the string:
  $$\Omega(n)$$
- **Overall Time Complexity (Big-Θ):**
  Since both the best case and worst case time complexities grow linearly with (

Question 6

Express the overall time complexity of the method in question 5 using the appropriate notation(s) (big-O, big-Θ, big-Ω).

Solution

Answer: Θ(hw)

Explanation:

Since the best case and worst case are identical (as shown in Question 5), we can use Big-Θ notation to express a tight bound.

The exact statement count is 2hw + 2h + 5, where:

- The dominant term is **2hw** (quadratic in the dimensions)
- The linear term **2h** becomes negligible as h and w grow large

Question 8

Express the overall time complexity of the method in question 7 using the appropriate notation(s) (big-O, big-Θ, big-Ω).

Solution

Answer:

- **Worst case:** $O(n^2)$
- **Best case:** $\Omega(1)$
- **No tight bound (Θ) exists**

Explanation:

The time complexity analysis reveals two fundamentally different execution scenarios:

Question 11

Determine the time complexity of the following pseudocode snippet using the active operation approach (**careful! This one is tricky!**).

```
Let G be a weighted graph implemented with an adjacency list;
Let H be a heap of edges ordered by edge weight (initially empty);

for (Vertex v : G.vertices()) {
    if (G.hasEdge(v, 0)) {
        H.insert(new Edge(v, 0));
    }
}
```

Solution

Why This is Tricky

This problem requires careful analysis because **two operations have non-**

Question 12

Determine exact number of statements executed by the following pseudocode function in the worst case (**this one is a bit tricky too!**):

```
public String treeConcat(Node root) {
    String s = root.string();

    if (!root.isLeaf()) {
        for (Node child : root.children()) {
            s += treeConcat(child);
        }
    }

    return s;
}
```

Solution