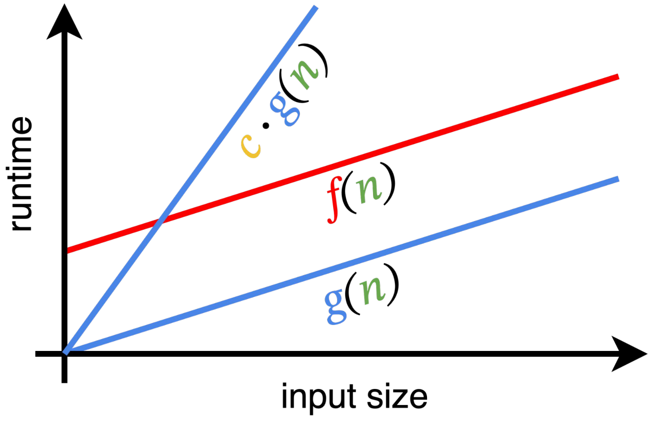
Section 2

Runtime

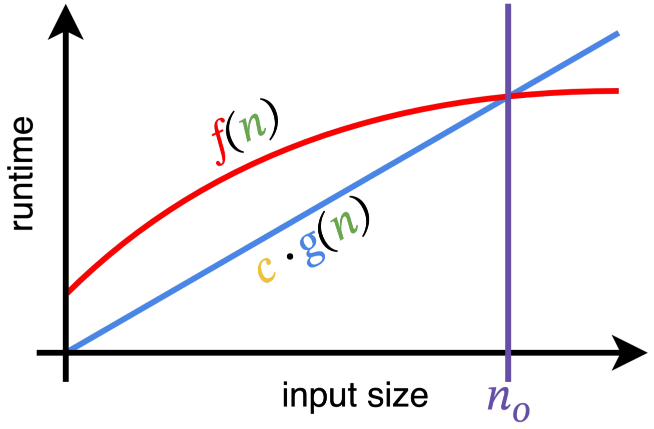
Big-Oh Review

- Definition:
  - If $f: \mathbb{N} \to \mathbb{R}$ and $g: \mathbb{N} \to \mathbb{R}$ are two functions, then we say $f(n) \in O(g(n))$ provided there exists positive constants c and n_0 such that $f(n) \le c \cdot g(n)$ for all values of $n \ge n_0$.

Why is c important?
Ensures that we focus on the ***shape*** *of the runtime curve's* ***growth****.*



Why is n_0 important?
Ensures that we focus on the input size n as it ***goes to infinity*** *by allowing us to ignore small values*



Main ideas:

•*In Big-O, we focus on the* ***growth*** *of the runtime as the input size n* ***goes to infinity****.*

•*Big-O represents an* ***upper bound*** *on the algorithm runtime. Not necessarily tight!*

Big-Omega and Big-Theta

- Definition of Ω:
  - If $f: \mathbb{N} \to \mathbb{R}$ and $g: \mathbb{N} \to \mathbb{R}$ are two functions, then we say $f(n) \in \Omega\big(g(n)\big)$ provided there exists positive constants c and n_0 such that $f(n) \geq c \cdot g(n)$ for all values of $n \geq n_0$.
  - *We also focus on the **growth** of the runtime as the input size n **goes to infinity**.*
  - *Ω represents a **lower bound** on the algorithm runtime. Not necessarily tight either.*

- Definition of Θ:
  - If $f: \mathbb{N} \to \mathbb{R}$ and $g: \mathbb{N} \to \mathbb{R}$ are two functions, then we say $f(n) \in \Theta\big(g(n)\big)$ provided that both $f(n) \in O\big(g(n)\big)$ and $f(n) \in \Omega\big(g(n)\big)$
  - *Ω represents a **tight/exact bound** on the algorithm runtime.*

Practice

- Suppose that $f(n) \in O(n)$. Indicate if each statement is guaranteed to be True, guaranteed to be False, or might be either.
  1. $f(n) \in O(n^2)$
  2. $f(n) \in O(4n)$
  3. $f(n) \in O(\log n)$
  4. $f(n) \in O(n \log n)$
  5. $f(n) \in O(1)$
  6. $f(n) \in \Omega(n)$
  7. $f(n) \in \Omega(\log n)$
  8. $f(n) \in \Omega(n^2)$

Practice

- Suppose that $f(n) \in O(n)$. Indicate if each statement is guaranteed to be True, guaranteed to be False, or might be either.

1. $f(n) \in O(n^2)$	1. Always True
2. $f(n) \in O(4n)$	2. Always True
3. $f(n) \in O(\log n)$	3. Sometimes True
4. $f(n) \in O(n \log n)$	4. Always True
5. $f(n) \in O(1)$	5. Sometimes True
6. $f(n) \in \Omega(n)$	6. Sometimes True
7. $f(n) \in \Omega(\log n)$	7. Sometimes True
8. $f(n) \in \Omega(n^2)$	8. Always False

Practice

- Suppose that $f(n) \in O(n)$. Indicate if each statement is guaranteed to be True, guaranteed to be False, or might be either.
  1. $f(n) \in O(n^2)$
     1. Always True
  2. $f(n) \in O(4n)$
     1. Always True
  3. $f(n) \in O(\log n)$
     1. Sometimes True
  4. $f(n) \in O(n \log n)$
     1. Always True
  5. $f(n) \in O(1)$
     1. Sometimes True
  6. $f(n) \in \Omega(n)$
     1. Sometimes True
  7. $f(n) \in \Omega(\log n)$
     1. Sometimes True
  8. $f(n) \in \Omega(n^2)$
     1. False

Worksheet problems

1. Prove that $f(n) \in O(g)$

a) $f(n) = 7n$, $g(n) = n/10$
b) $f(n) = 1000$, $g(n) = 3n^3$
c) $f(n) = 2^n$, $g(n) = 3^{2n}$
d) $f(n) = 7n^2 + 3n$, $g(n) = n^4$
e) $f(n) = n + 2n\log(n)$, $g(n) = n\log(n)$

How to Approach these Problems

When trying to prove something like $f(n) \in O(g)$, $f(n) \in \Omega(g)$, or $f(n) \in \Theta(g)$, you need to find a c and n_0.

- The **proof**, or final solution, for the problem should simply declare the values of c and n_0 and should plug them in/explain why they make the inequality true.
- The proof should **not** explain how to *solve for* c and n_0- that would be your own work.

1a) $f(n) = 7n, g(n) = \frac{n}{10}$

- We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $7n \leq c\frac{n}{10}$

$$7n \leq c\frac{n}{10}$$
$$70n \leq cn$$
$$70 \leq c$$

This is ok only because we said $n \geq n_0 > 0$

- Meaning that this inequality holds for all values of $n > 0$ so long as $c \geq 70$, so we can select $c = 70$ and $n_0 = 1$.

1a) Proof - Final Solution

1b) $f(n) = 1000, g(n) = 3n^2$

- We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $1000 \leq c \cdot 3n^2$.
- Here we can select $c = 1$ and then make sure n_0 is large enough so $1000 \leq 3n^2$
- If we select $n_0 = 20$ then $3n^2 = 1200$
- Definition of O holds for $c = 1$ and $n_0 = 20$.

1b) Proof - Final Solution

1 c) $f(n) = 2^n$, $g(n) = 3^{2n}$

We need to find positive constants c and n_0 so that for all $n \geq n_0$
we have that $2^n \leq c * 3^{2n}$
$2^n \leq c * 3^{2n}$
$2^n \leq c * 9^n$

If we select $n_0 = 2$ and $c = 1$ then:
$2^n \leq 1 * 9^n$
certainly holds true $\forall_n \geq 2$

1c) Proof - Final Solution

1 d) $f(n) = 7n^2 + 3n,\ g(n) = n^4$

- We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $7n^2 + 3n \leq c\, n^4$

$$7n^2 + 3n \leq c\, n^4$$

$$7n^2 + 3n \leq 7n^2 + 3n^2 \leq c\, n^4$$

As longs $n \geq 1$

$$10n^2 \leq c \cdot n^4$$

- Because $10n^2 \leq c \cdot n^4$ for $c = 10$ and every $n \geq 1$, we can satisfy $f(n) \in O\big(g(n)\big)$ for $c = 10$ and $n_0 = 1$

1d) Proof - Final Solution

Let $c = 10$ and $n_0 = 1$, and consider an arbitrary $n \geq n_0$.

We have $7n^2 \leq 7n^4$ (since $n \geq 1$) and
$3n \leq 3n^4$ (since $n \geq 1$).
Adding these inequalities we have
$7n^2 + 3n \leq 10n^4$, as required.

1 e) $f(n) = n + 2n\log_2 n, \ g(n) = n\log_2 n$

- We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $n + 2n\log_2 n \leq c \cdot n\log_2 n$

$$n + 2n\log_2 n \leq c \cdot n\log_2 n$$

$$n + 2n\log_2 n \leq 3n\log_2 n \leq c \cdot n\log_2 n$$

As long s $n \geq 2$

- Because
  - $3n\log_2 n \leq c \cdot n\log_2 n$ for $c = 3$ and every $n \geq 1$,
  - and $n + 2n\log_2 n \leq 3n\log_2 n$ for every $n \geq 2$,
- we can satisfy $f(n) \in O\big(g(n)\big)$ for $c = 3$ and $n_0 = 2$

1e) Proof - Final Solution

Let $c = 3$ and $n_0 = 2$. For an arbitrary $n \geq n_0$, we have:

$$n \leq n\log n \quad (since\ n \geq 2, \log n \geq 1)$$

$$2n \log n \leq 2n \log n$$

Adding these inequalities, we have $n + 2n\log n \leq c \cdot n \log n$, as required. So we have f(n) $\in$ O(g(n)).

Worksheet problems

2. We provide functions $f(n)$ and $g(n)$. Prove that $f(n) \in \Theta(g)$

a) $f(n) = 7n$, $g(n) = n/10$

b) $f(n) = n^3 + 10n$, $g(n) = 3n^3$

2a) $f(n) = 7n,\ g(n) = \frac{n}{10}$

- $f(n) \in O\big(g(n)\big)$ shown in 1a
- Now we show $f(n) \in \Omega\big(g(n)\big)$
- We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $7n \geq c\frac{n}{10}$

$$7n \geq c\frac{n}{10}$$
$$70n \geq cn$$
$$70 \geq c$$

- Meaning that this inequality holds for all values of $n > 0$ so long as $c \leq 70$, so we can select $c = 70$ and $n_0 = 1$.

2a) Proof - Final Solution

Again, we let $c = 70$ and $n_0 = 1$, let $n \geq n_0$ be an arbitrary integer.

We start with the left-hand-side of the inequality. Observe that

$$7n = 70 \cdot \frac{n}{10} = c \cdot \frac{n}{10}$$

Thus $7n \geq c \cdot \frac{n\tilde{\tilde{o}}}{10}$ for all $n \geq n_0$.

We thus have $7n \in \Omega\left(\frac{n}{10}\right)$, by definition. Since we have both O and Ω bounds, we conclude $7n \in \Theta\left(\frac{n}{}\right)$.

2 b) $f(n) = n^3 + 10n,\ g(n) = 3n^3$

- To show O:
  - We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $n^3 + 10 \leq c \cdot 3n^3$.
  - $n^3 + 10 \leq 2n^3$ as long as $n \geq 3$
  - And $2n^3 \leq c \cdot 3n^3$ as $c \geq \frac{2}{3}$
  - We can select $c = 1$ and $n_0 = 3$
- To show Ω:
  - We need to find positive constants c and n_0 so that for all $n \geq n_0$ we have that $n^3 + 10 \geq c \cdot 3n^3$.
  - Let's select $c = \frac{1}{3}$
  - $n^3 + 10 \geq n$^3 for every choice of n
  - Definition of Ω holds for $c = \frac{1}{3}$ and $n_0 = 1$

2b) Proof - Final Solution **Part 1**

First we will show $n^3 + 10$ belongs to $O(3n^3)$.

Let $c = 1$ and $n_0 = 3$ and let $n \geq n_0$ be an arbitrary integer.

Note that $n^3 + 10 \leq n^3 + n^3 = 2n^3$(as $n \geq 3$ gives $n^3 \geq 27$)

We thus have $n^3 + 10 \leq 2n^3 \leq 1 \cdot 3n^3$.
So $n^3 + 10 \in O(3n^3)$

2b) Proof - Final Solution **Part 2**

Next, we show $n^3 + 10 \in \Omega(3n^3)$.

Consider $c = \frac{1}{3}$ and $n_0 = 1$. For an arbitrary $n \geq n_0$ we have:

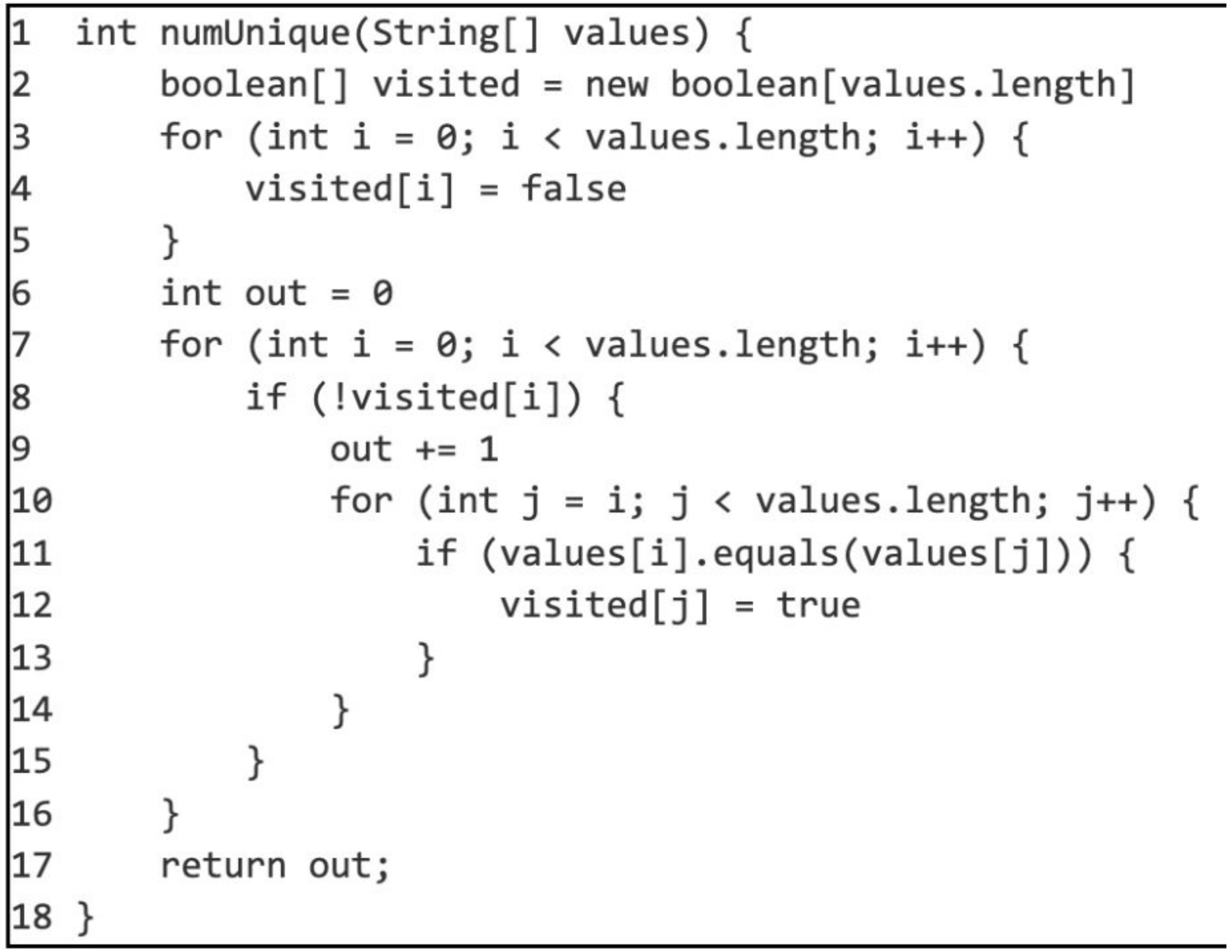
$$n^3 + 10 \geq \frac{1}{3} \cdot 3n^3 + 10 > \frac{1}{3} 3n^3 = c3n^3$$

We thus have $n^3 + 10 \in \Omega(3n^3)$

- Since we have matching O and Ω bounds, we conclude

$n^3 + 10 \in \Theta(3n^3)$.

Worksheet problems: Q3



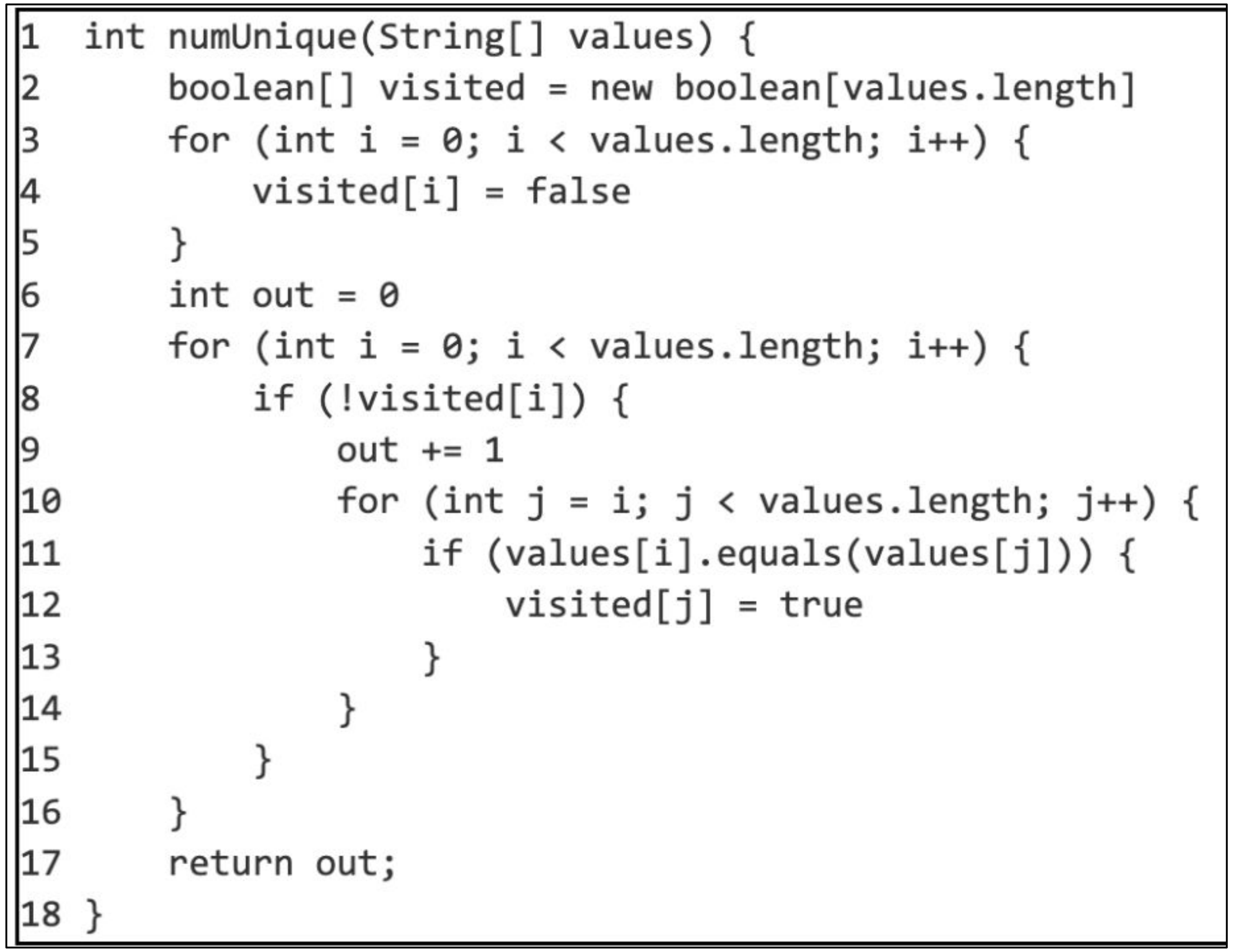
```
int numUnique(String[] values) {
    boolean[] visited = new boolean[values.length]
    for (int i = 0; i < values.length; i++) {
        visited[i] = false
    }
    int out = 0
    for (int i = 0; i < values.length; i++) {
        if (!visited[i]) {
            out += 1
            for (int j = i; j < values.length; j++) {
                if (values[i].equals(values[j])) {
                    visited[j] = true
                }
            }
        }
    }
    return out;
}
```

Get the $\Theta(\cdot)$ bound of each function below.

a) $f(n)$ = worst case running time
b) $g(n)$ = best case running time

We will construct equations for each function and then simplify them to get a closed form.

3a) f(n) = worst case running time



```
1  int numUnique(String[] values) {
2      boolean[] visited = new boolean[values.length]
3      for (int i = 0; i < values.length; i++) {
4          visited[i] = false
5      }
6      int out = 0
7      for (int i = 0; i < values.length; i++) {
8          if (!visited[i]) {
9              out += 1
10             for (int j = i; j < values.length; j++) {
11                 if (values[i].equals(values[j])) {
12                     visited[j] = true
13                 }
14             }
15         }
16     }
17     return out;
18 }
```

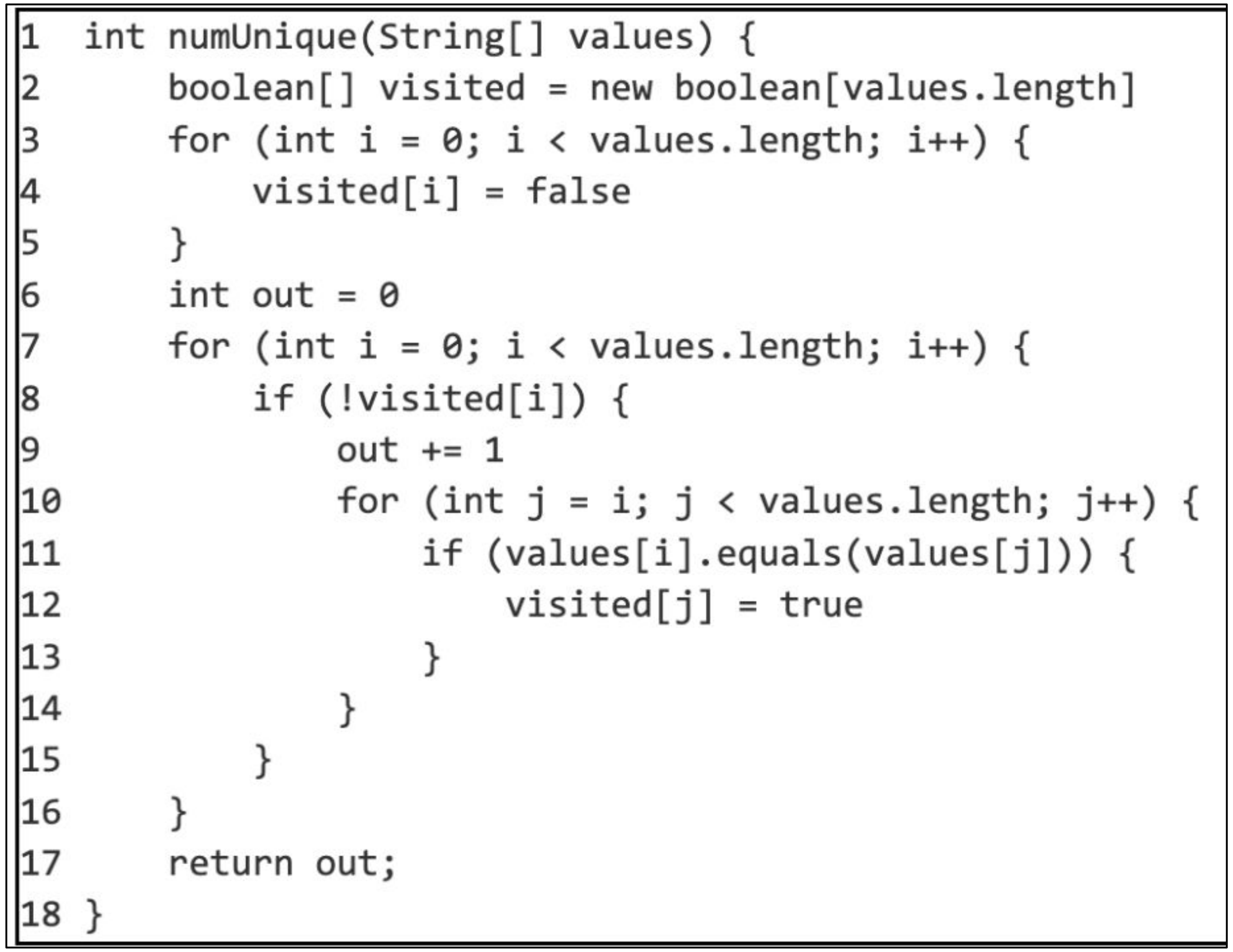
First function (lines 3-5) always runs once.

Worst case for lines 7-16 is if every value in array is unique. This means for every iteration of the outer loop, the inner loop will iterate through the rest of the array. Total number of iterations is n + (n-1) + (n-2) + … + 1.
There is a formula for this sum: n * (n+1) / 2

f(n) = n + n(n+1)/2 => **n^2. This is quadratic running time**

Do something similar to Q2 proofs to prove that $n + n(n+1)/2 \in \Theta(n^2)$.

3b) g(n) = best case running time



```
1  int numUnique(String[] values) {
2      boolean[] visited = new boolean[values.length]
3      for (int i = 0; i < values.length; i++) {
4          visited[i] = false
5      }
6      int out = 0
7      for (int i = 0; i < values.length; i++) {
8          if (!visited[i]) {
9              out += 1
10             for (int j = i; j < values.length; j++) {
11                 if (values[i].equals(values[j])) {
12                     visited[j] = true
13                 }
14             }
15         }
16     }
17     return out;
18 }
```

First function (lines 3-5) always runs once.

Best case for lines 7-16 is if every value in array is the same. This means the inner loop will only run once: for the first iteration of the outer loop. Then, it will not run because every index will be visited after the first time.

g(n) = n + n => **n**
This is linear running time

Do something similar to Q2 proofs to prove that n + n ∈ Θ(n).