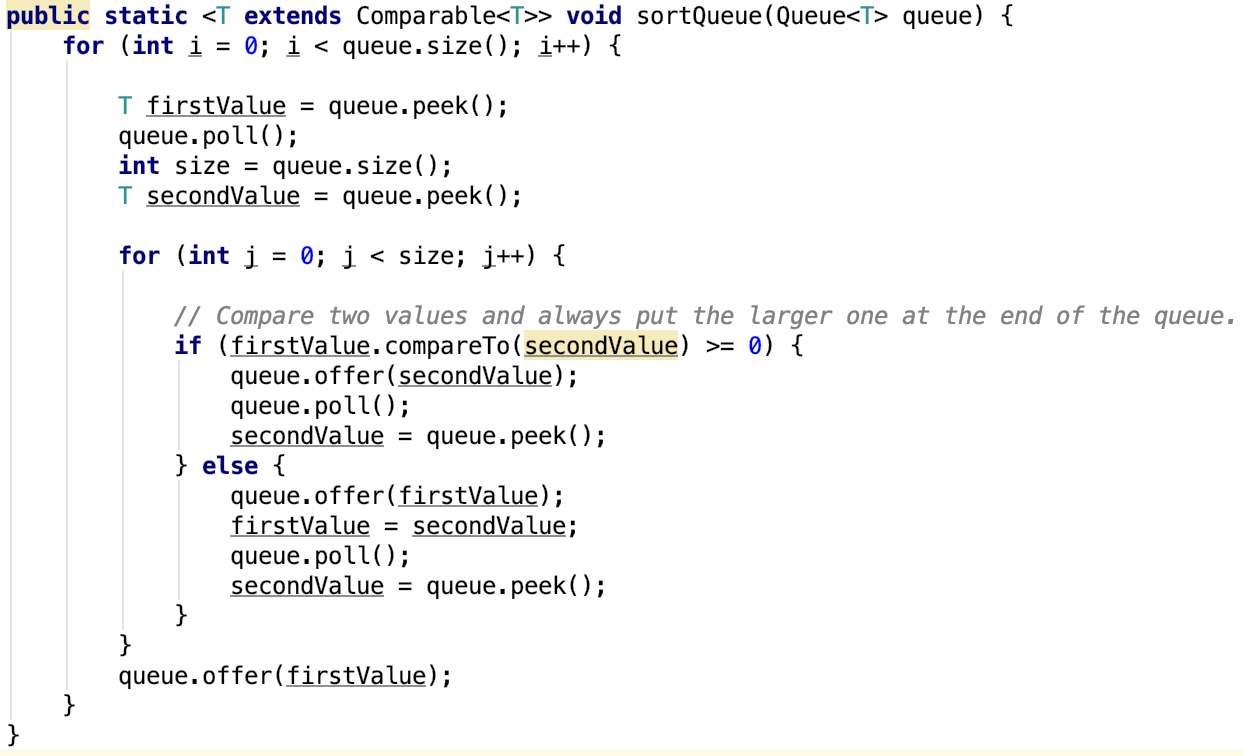
Represent the worst-case running time of your sortQueue algorithm as a mathe- matical function. Then, use the mathematical definition of big-O to determine an asymptotic bound on this function (that is, you should compute the values of c and n0 to show that the bound exists). You may assume the runtime complexities of the standard queue methods are as described in lectures.

“n” means the size of the input.

Use “n” to indicate the mathematical function of running time.

The worst-case runtime:

# Operations

1 + 2(n + 1)

n

n

n

n

n \* (1 + 2(n + 1))

n \* 2n

n \* n

n \* n

n \* n

n \* n

n

Total: 8n2 + 10n + 3

Show that 8n2 + 10n + 3 is *O*(n2)

f(n) = 8n2 + 10n + 3

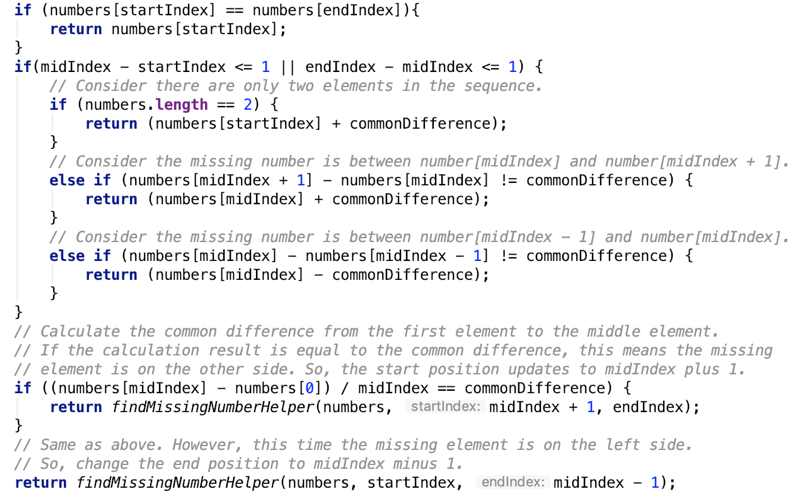
g(n) = n2

8n2 + 10n + 3 ≤ c \* n2

(c – 8)n2 – 10n – 3 ≥ 0

Since we need c and n0 to be positive constants and ,

pick c = 9, n0 = 11

Express the worst-case running time of your findMissingNumber algorithm as a mathematical recurrence. State an asymptotic bound in big-O notation for this recurrence. Explain how you determined this bound.

*O*(1)

“n” means the size of input.

*O*(1) if (n ≤ 3 or (numbers[n-1] – numbers[0] == 0))

*T*(n) =

*T*(n / 2) if (n > 3 and (numbers[n] – numbers[0] != 0))

*T*(n) = *T*()

= *T*()

= *T*()

…­

2, 4, 6, 8, 10, 12, … , n

2, 4, 6, 8 … ,

1 call

2, 4, 6, … ,

2 calls

2, 4, … ,

3 calls

To find the missing number in an arithmetic sequence, we cut off half of the elements after each call. So, the total number of calls will be less than or equal to .

So, we can say *T*(n) is *O*().