Homework 04: Repetition

Release date: 2/04
Due date: 2/11 by 11:59 pm

Goals:

- Familiarity with the binary search algorithm
- Using loops for calculations/approximations

Introduction:

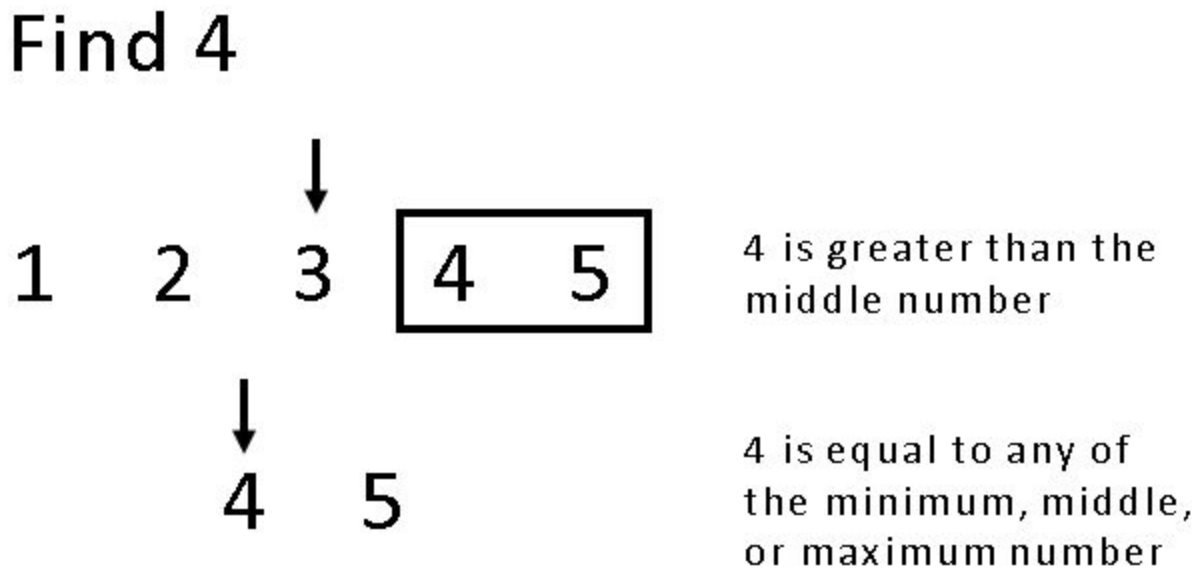
For this homework, you will be making a program that will approximate the square root of a number from user input. Users should be able to decide how many decimal points they want the square root approximated to. For the purposes of this lab, the number of decimal points will be limited from 1 to 14 and all numbers will be non-negative.

First, let's think about some properties of square roots. Think about the guess and check method of solving a math problem. If our goal is to find the square root of real number "x", then for some real number "y", y*y = x. Therefore, theoretically we should be able to approximate the square root of x by trying various values of y and squaring them. While y*y does not equal x to the desired precision, we can adjust and continue trying different values of y. But how can we efficiently try values of y to calculate x?

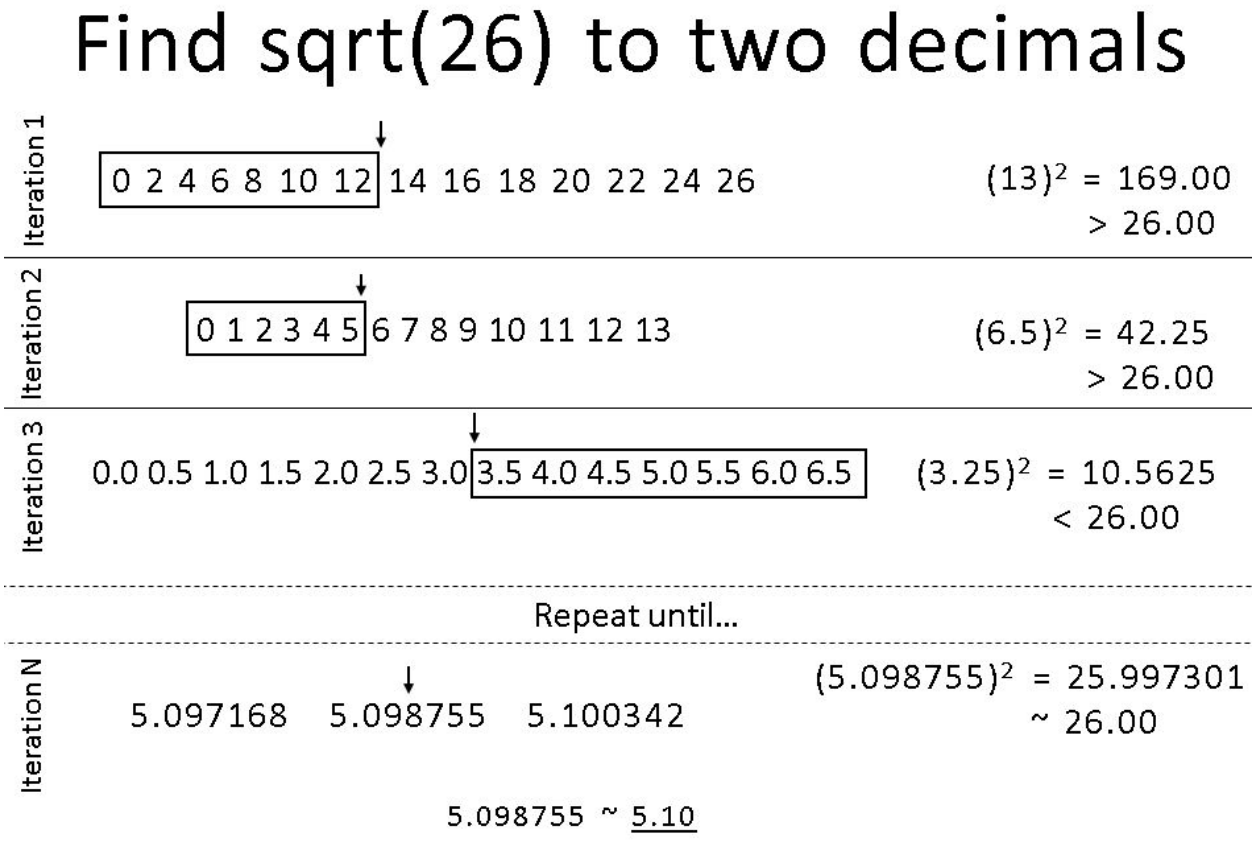
As a starting point, let's look at something called the binary search algorithm. You will need to implement a modified version of this algorithm for the homework. We will provide the binary search algorithm and it will be your task to figure out how to modify and implement it to approximate the square root of a number.

The binary search algorithm provides an efficient way to search for certain value in an ordered sequence. If there are n values in the sequence, the value at the $n/2_{th}$ [middle] position will be our first guess (round up to the n+1th position if n is odd). Think about the first number in the sequence as defining the minimum of the search space, and the last number as defining the maximum position of the search space. Since we know the sequence is ordered, if the value at the middle position is less than the value we are searching for, then we know we can now confine our search to only the values at positions greater than our initial guess. Now, we can redefine the minimum position of the search

space as being the position defined by the index of the middle position plus 1, and the maximum position stays the same. If the initial guess was instead greater than the value we were searching for you would instead adjust the maximum position. Now, we can redefine the middle value and continue repeating the process until we have found the value we are searching for (we conclude the value does not exist in the sequence if the min position is equal to the max position). See the next image for finding 4.



In a number line, while there are an infinite number of values in between any two numbers, there are a finite number of values between any two numbers to a defined precision. This is what allows us to use a modified binary search algorithm to approximate the square root below (note that ~ means "approximately equal to", or "close enough to" according to the defined precision).



Explanation:

1. Start with a range of numbers from 0 to 26.

2. Calculate the middlemost number and square it. For the first iteration, we found 13, which squares to 169.

3. Compare the squared number to 26.00. 169 is greater than 26.00, so we will look for another number that is less than 13, but bigger than 0. *If* the square were larger, we would look at numbers between 13 and 26.

4. Repeat steps 1 through 3 until we find a number that has a square of about 26.00 when rounded. Note that the answer is **not exactly** the square root of 26: this is only an estimate precise to two decimal places. Reducing the precision of the result from 5.098755 to 5.10 may also change the accuracy of your result. **For the purpose of this assignment, this is okay**. Just don't use this algorithm for your finances.

An example run of the program is below. You should model the functionality of your program on the one shown using the algorithm discussed above.

```
Enter a positive number to find the square root of: [type quit to exit]
-1
Enter a positive number
Enter a positive number to find the square root of: [type quit to exit]
26
Enter the precision of the estimator (number of digits after decimal point):
2
Square Root of 26: 5.10
Enter a positive number to find the square root of: [type quit to exit]
26
Enter the precision of the estimator (number of digits after decimal point):
3
Square Root of 26: 5.099
Enter a positive number to find the square root of: [type quit to exit]
quit
```

****NOTE: You may NOT use any pre-existing square root functions for this task (e.g. Math.sqrt or Math.pow). You will receive a ZERO and use a submission****

Submission:

Items required for submission via Blackboard:

- **SqrtEstimator.java**

Rubric:

- **SqrtEstimator** class — 100 points total
  - Program continues running until user enters "quit" command – 20 points (required for other criteria).
  - Error checking for invalid (non-positive) input/prompting while not correct – 20 points
  - Approximation displays correct precision – 10 points
  - Approximation is mathematically correct – 50 points