Prime Numbers

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Can you spot a pattern?
Is \( n \) Prime? – Facts

- If \( n \) is not divisible by any number smaller than \( n \), then \( n \) is prime.
- To prove \( n \) is not prime, only one factor needs to be found.
- If \( n \) is not prime, then there exists a pair of integers, \( a \) and \( b \) such that \( n = ab \).
- If \( n = ab \), then either \( a \) or \( b \) must be less than or equal to the square root of \( n \).
- Therefore, starting from 2 and counting up, if no factors have been found from 2 through \( \sqrt{n} \), then \( n \) must be prime.

Is \( n \) Prime? – The Algorithm

1. Given \( n \), calculate the largest possible first factor: \( \text{largestPossibleFirstPrime} = \sqrt{n} \)
2. Let \( \text{testDivisor} = 2 \).
3. If \( \text{testDivisor} \) is greater than \( \text{largestPossibleFirstPrime} \), then stop checking because \( n \) is prime.
4. If \( \text{testDivisor} \) is a factor of \( n \), then \( n \) is not prime.
5. If \( \text{testDivisor} \) does not a factor of \( n \), then increment \( \text{testDivisor} \) and return to step 3.
Is \( n \) Prime? By \textbf{for} Loop

```java
public static void main(String[] args) {
    int n = 64;
    int largestPossibleFirstPrime = (int)Math.sqrt(n);
    boolean foundDivisor = false;

    for (int k=2; k <= largestPossibleFirstPrime; k++)
    {
        System.out.println("k="+k);
        if (n % k == 0)
        {
            foundDivisor = true;
        }
    }

    if (foundDivisor)
    {
        System.out.println(n + " is NOT Prime.");
    }
    else System.out.println(n +" is Prime.");
}
```

Is \( n \) Prime? By \textbf{while} Loop

```java
public static void main(String[] args) {
    int n = 64;
    int largestPossibleFirstPrime = (int)Math.sqrt(n);
    boolean foundDivisor = false;
    int k=2;

    while(k <= largestPossibleFirstPrime)
    {
        System.out.println("k="+k);
        if (n % k == 0)
        {
            foundDivisor = true;
            break;
        }
        k++;
    }

    if (foundDivisor)
    {
        System.out.println(n + " is NOT Prime.");
    }
    else System.out.println(n +" is Prime.");
}
```
**for** Loop verses **while** Loop

```java
public static void main(String[] args)
{
    for (int i=0; i<12; i=i+3)
    {
        System.out.print("i=");
        System.out.print(i);
    }
    System.out.println();

    int k=0;
    while (k<12)
    {
        System.out.print("k=");
        System.out.print(k);
        k=k+3;
    }
}
```

*Output:*

```
i=0, i=3, i=6, i=9,
k=0, k=3, k=6, k=9,
```

**Prime Factors**

- $9 \rightarrow 3 \cdot 3$
- $100 \rightarrow 10 \cdot 10 \rightarrow (5 \cdot 2) \cdot (5 \cdot 2)$
- $13 \rightarrow 13$
- $420 \rightarrow 10 \cdot 42 \rightarrow (5 \cdot 2) \cdot (7 \cdot 6) \rightarrow 5 \cdot 2 \cdot 7 \cdot (2 \cdot 3)$
- $100 \rightarrow 2 \cdot 50 \rightarrow 2 \cdot (2 \cdot 25) \rightarrow 2 \cdot 2 \cdot (5 \cdot 5)$

The above is a non-systematic method.
Finding Prime Factors - Algorithm

1. Let \( n \) be the number of which to find the prime factors.
2. Start with 2 as a test divisor.
3. If the test divisor is greater than the square root of \( n \), then the test divisor is either 1 or prime, and we are done.
4. If the remainder of \( n \) divided by the test divisor is zero, then:
   a) The test divisor is a prime factor of \( n \). Print it.
   b) Replace \( n \) with \( n \) divided by the test divisor.
   c) If the new \( n \) equals one, then all the divisors have been found.
5. If the remainder of \( n \) divided by the test divisor not zero, then:
   a) Increment the test divisor.
6. Loop to step 3.

Prime Factors: Example \( n=100 \)

- Start with \( n=100 \), and testDivisor = 2;
- 2 divides 100, so print 2 an set \( n=100/2=50 \).
- 2 divides 50, so print 2 and set \( n=50/2=25 \).
- 2 does not divide 25, so increment testDivisor to 3.
- 3 does not divide 25, so increment testDivisor to 4.
- 4 does not divide 25, so increment testDivisor to 5.
- 5 divides 25, so print 5, and set \( n=25/5=5 \).
- 5 divides 5, so print 5, and set \( n=5/5=1 \).
- Since \( n \) is equal to 1, stop.
Lab 2: Prime Factors

- Write a Java program that hard-codes a positive integer value `final int ORIGINAL_NUMBER`, and uses the given prime factor algorithm to correctly print that `ORIGINAL_NUMBER` is prime, or to print its prime factors.
- All repeated factors must be printed.
- The prime factors must be printed in order from smallest to largest.
- Your code must work for all positive integer values of `ORIGINAL_NUMBER` from 1 through 2 billion.
- Your output must include the value of `ORIGINAL_NUMBER` and must be clearly represented.

Lab 2: Grading Rubric

- Total points: 20
- Your code will be run with 12 test cases of unknown legal values of `ORIGINAL_NUMBER`. You will receive one point for each correct (that matches the specs) result.
- Proper adherence to the CS-259 coding standard, including comments, is worth 5 points. You start with those 5 points and each error levies a -1 up to the maximum of 5 points.
- 3 Points: your code is efficient (i.e. does not retest many values, ...), and your code is easy to read.
- -2 points for each compiler warning.
- Due Date: Midnight on Monday, Aug 31.