Fast Facts

Teachers Teaching with Technology™
Professional Development from Texas Instruments

Answers & Teacher Notes

7 8 9 10 11 12

TI-Nspire | Coding | Student | 60 min

Prerequisite Resources

The Texas Instruments website provides for a series of free, online, self-paced learning modules titled "10 Minutes of Code". Students can work through these modules at their own pace, teacher notes are also available. For this activity it is recommended that students have completed Unit 3 – Conditional Statements and Unit 4 – Loops.

https://education.ti.com/en-au/activities/ti-codes/nspire/10-minutes

Finding Factors

There are many ways to determine the quantity of factors for a specified number. The most common method is to test the divisibility for every number up to the specified number, however this is a slow process and many of the numbers being tested are not necessary.

Example: Determine the quantity of factors for the number 45.

45 ÷ 1 = 45	Factor ✓	45 ÷ 2 = 24 rem 1	Not a factor	45 ÷ 3 = 15	Factor ✓

Testing from these first three numbers provides some possible short cuts.

- 45 is not divisible by 2, therefore it cannot be even. If the number is not even there is no purpose checking divisibility by 4, 6, 8 … An efficient algorithm (program) should therefore check first to see if the number being tested is even, potentially removing 50% of future testing.
- 45 is divisible by 3 since: 3 x 15 = 45. With the exception of perfect squares, factors occur in pairs so once one number in the pair has been identified the other can be found by division rather than additional searching. This approach means that a much smaller testing threshold can be established.

45 ÷ 1 = 45	Factor ✓	45 ÷ 2 = 24 rem 1	Not a factor	45 ÷ 3 = 15	Factor ✓
45 ÷ 4 = 11 rem 1	Not a factor	45 ÷ 5 = 9	Factor ✓	45 ÷ 6 = 7 rem 3	Not a factor
45 ÷ 7 = 6 rem 2	Not a factor				

- In the example above no further testing is required as the divisor (7) is now greater than the quotient (6). Further increases in the divisor will only continue to reduce the quotient locating factor partners that have already been established. The threshold at which this occurs is the square-root of the original number. $\sqrt{45} \approx 6.71$, so searching for factors of 45 can cease at 6.

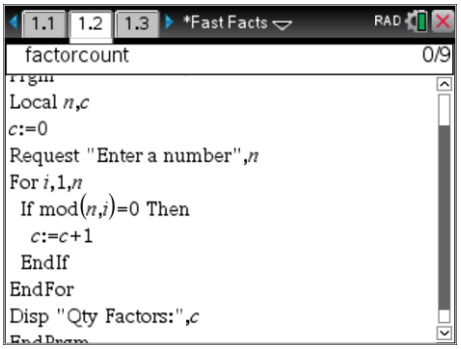
Author: P. Fox

Instructions

Open the TI-Nspire document: Fast Facts.

Navigate to page 1.2: "factorcount".

This is the program listing for the "factorcount" program that 'requests' a number and returns the total number of factors for that number.



Navigate to page 1.3, use the [**Var**] key to access the factorcount program, select and run the program. When prompted enter the number 360. The program should return: 24, signifying there are 24 factors for the number 360.

Question: 1.

Use a stop watch to record how long the calculator takes to count the quantity of factors for each of the following numbers:

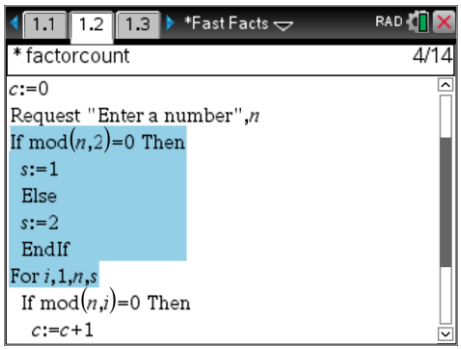
Number	100	101	1,000	1,001	10,000	10,001	100,000	100,001
Time	< 1.0s	<1.0s	1 to 2s	1 to 2s	≈ 15s	≈ 15s	≈ 150s	≈ 150s

Answers will vary slightly depending on student response times. Students should notice that an increase in a multiple of 10 for the original number also increases the time for the calculator program to run by a similar ratio.

Return to page 1.2 and edit the program. The required edits and changes have been highlighted.

When you have finished editing press **Ctrl** + **B** to save and compile the programming code.

Return to page 1.3 so the program can be run again and tested.



Question: 2.

What is the "**IF**" statement checking when it tests **mod(n,2)=0**?

"mod" performs 'modular' arithmetic, better known to students as 'remainder'. Mod(n,2) returns the remainder when the quantity 'n' is divided by 2. If the result is zero the original number is a multiple of 2, an even number. [If it is not even, then it must be odd is implied.]

Question: 3.

The **FOR** syntax is: **For** *variable, start, finish, step*. What possible values can the step size take according to the previous **IF** statement?

If the original number is even (… mod(n,2)=0 …) the step size is 1 (… s:=1 …). This means that every number will be checked for divisibility.

If the original number is odd, (… Else …) the step size is 2 (… s:=2 …). This means every second number, starting at 1 (… 1, 3, 5, 7 …) will be tested for divisibility.

Author: P.Fox

Question: 4.

Run the program and check how long it takes to count the factors for the numbers in Table 1 of Question 1, compare your results with those obtained previously.

Number	100	101	1,000	1,001	10,000	10,001	100,000	100,001
Time	< 1.0s	<1.0s	1 to 2s	≈ 1s*	≈ 15s	≈ 8s	≈ 150s	≈ 75s

* Students may not detect the time difference for this calculation.

Students should notice that the odd numbers, particularly the larger ones run in approximately half the time as only every second number is being tested.

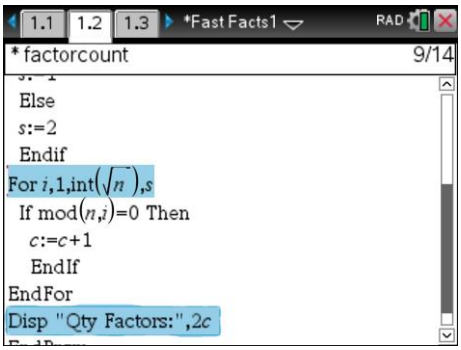
Further algorithms to bypass other multiples of primes would make the program faster again, however it is much hard to come up with a simple rule for the step size.

Return to page 1.2 and edit the highlighted section once again.

The "INT" statement removes the decimal values from a calculation guaranteeing that the FOR loop has a whole number to count up to.

To understand the inclusion of the Square-root calculation, read back through the introduction "Finding Factors"

Notice also that the factor count result is doubled.



Question: 5.

Run the program and check how long it takes to count the factors of 10,001 and 10,000. Compare your results to those obtained in Question 1.

Number	100	101	1,000	1,001	10,000	10,001	100,000	100,001
Time	< 1.0s	<1.0s	<1.0s	<1.0s	<1.0s	<1.0s	<2.0s	<1.0s

Most of these times are virtually impossible to detect accurately as the counting algorithm is now too fast for students to accurately measure. Students should acknowledge that this new algorithm (program) is significantly more efficient!

Question: 6.

An important aspect of writing code is to test it, to make sure it is working properly. Write down the factors for each of the following numbers: 18, 32, 37, 45, 50, 100 and 144. Check if your program returns the correct quantity of factors for each number. If there are any discrepancies, suggest a possible cause.

According to the program:

18 … has 6 factors: {1, 2, 3, 6, 9, 18}
32 … has 6 factors: {1, 2, 4, 8, 16, 32}
37 … has 2 factors: {1, 37}
45 … has 6 factors: {1, 3, 5, 9, 15, 45}
50 … has 6 factors: {1, 2, 5, 10, 25, 50}
100 … has 10 factors … this however is incorrect: {1, 2, 4, 5, 10, 20, 25, 50, 100}
144 … has 16 factors … this is also incorrect: {1, 2, 3, 4, 6, 8, 9, 12, 16, 148, 24, 36, 48, 72, 144}

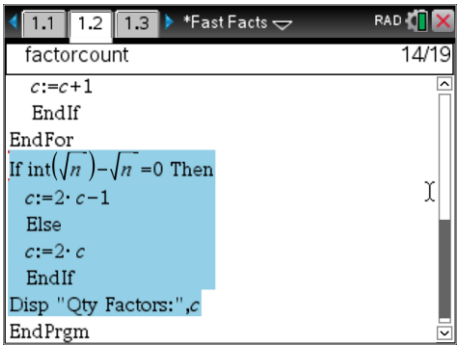
Students must identify that 100 and 144 have the incorrect quantity of factors, they however may not realise the reason, both numbers perfect squares! From a programming perspective the

Author: P.Fox

question is designed to highlight the importance of the 'testing phase', to ensure your program is doing what it is designed to do, accurately and efficiently count factors.
The problem is that doubling the number of factors found up to and including the square-root of the original number duplicates counting the square-root of the number. The time efficiency of the algorithm however makes consideration of exceptions worthwhile. A small algorithm can correct the factor count by testing if the original number was a perfect square.

Return to page 1.2 and edit the highlighted section once again.

This final check corrects a problem that occurs for a special group of numbers, the inclusion of this final step however does not add much computational time to the program because it is not contained within the search loop and is therefore executed just one time at the end of the program.



Question: 7.

Write down at least 10 different numbers. Write down the factors for each of the numbers and use your program to check the factor count.

Answers will vary depending on the numbers chosen. If students realise that the only problem with the previous factor count was to do with perfect squares, they should test a considerable proportion of perfect squares.

Question: 8.

The number 21 is the product of exactly two prime numbers: 7 and 3.

a. Write down the factors of 21. {1, 3, 7, 21}

b. The number: 6,397 is prime, so too 9,397. The product of these two prime numbers is equal to: 60112609. (60 million, 112 thousand, 609) Predict how many factors there are for 60112609 then use the FactorCount program to check the quantity of factors.

Prediction: 4 factors: {1, 6397, 9397, 60112609}

Calculator Program: 4 factors … and counted in less than 30 seconds!

c. The number 67,629,137 is prime, so too is 73,939,133. The product of these two prime numbers is: 5,000,439,755,318,221. (5 Quadrillion, 439 billion, 755 million, 318 thousand and 221). Predict the quantity of factors for this very large number.
Do **NOT** test this with your calculator program!

This number is of the order 10^8 times bigger than the one in part B. It would take the calculator approximately 3 days to return a result. Students however can reason, using factor trees that the result would consist of 4 factors: {1, 67629137, 73939133, 5000439….}

Note: If students do attempt to run the program on this very large number, and they probably will … , hold down the ON key for approximately 3 seconds to halt the program.

Author: P.Fox

Encryption techniques used by banks and other organisations that transmit private or secure data over the internet rely on the fact that it is very time consuming to find the factors of a very large number, particularly where that large number is the product of very large prime numbers. As factorising techniques are improved and computer processing power increases, the prime numbers being used for secure transactions must get bigger and bigger in order to keep your data safe.

Author: P.Fox