# **PEX 1: I Can Factor That Number In *x* Seconds**

### *Due: Lesson 29*

#### (100 Points)

|  |
| --- |
| **Help Policy**  **AUTHORIZED RESOURCES:** Any, except another cadet’s program.  **NOTE:**   * Never copy another person’s work and submit it as your own. * Do not jointly create a program. This is neither a team project nor a group project. Although you may discuss the assignment with any cadet enrolled in CS431, you may not jointly implement a program, you may not be coached or walked through a program, and you may not use another cadet's program as a source of help. You must do the assignment yourself and submit your own work for a grade. * You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook). * **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.** |

|  |
| --- |
| **Documentation Policy** |
| * You must submit a documentation statement documenting all help received on this assignment. |
| * The documentation statement may describe the assistance received or simply refer to comments contained in the program code. |
| * The documentation statement must specify **WHAT** assistance was provided, **WHERE** in the code assistance was provided, and **WHO** provided the assistance. |
| * If **no help** was received on this assignment, the documentation statement must state “NONE.” |

1. Requirements

You have been hired by a super-secretive national-level intelligence organization to develop software to crack codes. Your new office is particularly interested in efficient algorithms for factoring large numbers. After discovering that you had CS431 at the Air Force Academy, they have asked you to begin your job by evaluating the relative performance of several well-known factoring methods. You decide to develop a program to do the following:

1. Display the name of the program and your name.
2. Prompt the user to enter an integer, *n*,which they desire to factor.
3. Output the statistics for factoring *n* (ie: the time elapsed in finding a factor as well as the identified factor itself) using the 3 algorithms specified below:
   1. Brute force (smartly choose which values you test for full credit)
   2. Pollard’s rho algorithm
   3. Dixon’s algorithm
4. Repeatedly ask for another integer from the user until user chooses to quit.

A typical run of your program should look something like the following (user input in **bold**):

PEX1 - Factoring! - by Capt Coolio

CS431

Enter a number to factor: **124076833**

Brute force factoring

Found a factor = 1531

It took 13.26 seconds.

Pollard Rho

Found a factor = 1531

It took 0.77 seconds.

Dixon’s Algorithm

Enter # of factors in factor base: **10**

Done generating factor base.

1 80982557 === 100188000 5 2 3 0 2 0 0 0 1 0

2 107169634 === 83010048 9 1 0 0 1 0 3 0 0 0

...

10 107382214 === 111815418 1 1 0 0 1 1 0 4 0 0

Found a factor = 1531

It took 24.54 seconds.

…and so forth. Your program should always allow the user to enter another number until the user chooses to quit (by entering a zero, for instance).

1. **Programming Language:**

You may implement this PEX in any language you like, but it will need to have support for large, variable length integers (> 64 bits). Java has this with the BigInteger class from java.math, however, other languages have support for arbitrary precision arithmetic as well, check [here](http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic#Arbitrary-precision_software) for details.

1. **Supporting Files:**

Here are a few predefined routines to perform Gaussian Elimination mod 2 that may be useful for implementing Dixon’s Algorithm:

* 1. [Java](file://dfcs2/v_drive/courses/CS431/S11/Web/PEXs/mod2%20GE.java)
  2. [Python](file:///C:/Courses/cs431/12/PEX1/mod2GaussianElim.py)

1. Suggested Plan of Attack:
2. Implement brute force first. Confirm it correctly finds a factor and stops. To determine the upper limit of factors to try, you will need a way to take the square root of a BigInteger. See below for a hint of how to implement that in Java.
3. Add the code to time how long the algorithm takes. Be careful to only time the actual algorithm.
4. Next, implement Pollard’s rho. Use the algorithm at the bottom of page 91 (Section 3.2.2) of the *Handbook of Applied Cryptography*. You will likely have to modify the gcd algorithm you wrote for HW 2 (Extended Euclidean Algorithm) to work with arbitrary precision numbers, or else find a library routine. Once you have checked for correctness, add the timing mechanisms.
5. Finally, implement Dixon’s algorithm. See the hints document for details on getting this to work. Check for correctness and add the timing mechanisms.
6. Turn In Requirements

Deliverables

1. E-mail an **electronic copy** of all your PEX files (.zip please) to your instructor. Make sure you include a documentation statement. Include sufficient information to enable your instructor to compile and run your program. Remember turn-ins may be in different languages, so explain in detail what run-time system or IDE is needed.
2. Provide a **hard copy** of your files, including the attached **cut sheet**, and a **screenshot** of your code factoring the below test cases. (This is a backup in case I am unable to compile your code – give me SOMETHING to grade!)
3. 1002731 – use a factor base of 8
4. 2163841 – use a factor base of 10
5. 1722896477 – use a factor base of 25

**PEX 1 Cut Sheet (100 Points)**

|  |
| --- |
| **Name:**  **Section:**  **Grade: /100** |

|  |  |  |
| --- | --- | --- |
| **Requirements:** |  |  |
|  | **Some semblance of design** | **(5)** |
|  |  |  |
|  | **Implementation** |  |
|  | Brute force works correctly. (15) | **(15)** |
|  | Pollard’s rho works correctly. (30) | **(30)** |
|  | Dixon’s algorithm works correctly. (45) | **(45)** |
|  |  |  |
|  |  |  |
|  | **Adherence to Programming Standards** | **(5)** |

|  |  |  |
| --- | --- | --- |
| **Penalties:** |  |  |
|  | **Vague/Missing Documentation (-5):** |  |
|  | **Missing Grade Sheet (-5):** |  |

|  |  |  |
| --- | --- | --- |
| **Total:** |  | **(100)** |