Practical 1: Quite Good Numbers

Task

A *perfect* number is an integer that is equal to the sum of its divisors (where 1 is considered a divisor). For example, 6 is perfect because its divisors are 1, 2, and 3, and 1 + 2 + 3 is 6. Similarly, 28 is perfect because it equals 1 + 2 + 4 + 7 + 14.

A *quite good* number is an integer whose *badness*—the size of the difference between the sum of its divisors and the number itself—is not greater than a specified value. For example, if the maximum badness is set at 3, there are 12 quite good numbers less than 100: 2, 3, 4, 6, 8, 10, 16, 18, 20, 28, 32, and 64.

Your task is to write a C++ program that determines numbers within a designated maximum badness that are less than a limiting value. The limiting value and maximum badness are provided as command line arguments when the program is executed. Command line arguments can be specified within CLion by editing the *build configuration*.

SVN check out

Before you begin, you will need to check out the practical directory from the topic repository. This can be achieved within CLion by selecting Get from Version Control on the welcome screen. In the window that appears, select Subversion from the dropdown list. If you have not previously connected CLion to the repository, click the add button (**+**) and paste the following URL into the text field (replacing **`FAN`** with your FAN):

```
https://topicsvn.flinders.edu.au/svn-repos/COMP2711/FAN
```

If you are prompted to login, use your University FAN and password. Expand the repository listing and select the **quitegood** directory. Click Check Out then select a location to store your project (preferably in a practicals directory) and click Open. From the destination list, choose the second option to create a project directory named **quitegood** and then click OK to complete the check out. A dialogue may appear confirming the subversion working format; if so, 1.8 is fine. Finally, when prompted if you would like to open the project, click Yes.

Automated marking

This practical is available for automatic marking via the quiz **Practical 1**, which is located in the Assessment module on FLO. To assess your solution, first commit your changes to the topic repository. This can be completed within CLion via the Commit option under the VCS menu (or alternatively with ⌘+K on macOS or Ctrl+K on Windows).

You should adhere to good development habits and enter a commit message that describes what you have changed since your last commit. When ready, click Commit to upload your changes and receive a revision number. Enter this revision number into the answer box for the relevant question (level).

There are no penalties for incorrect solutions, so if you do not pass all test cases, check the report output, modify your solution, commit, and try again. Remember to finish and submit the quiz when you are ready to hand in. You may complete the quiz as many times as you like—your final mark for the practical will be the highest quiz mark achieved. If you do start a new quiz attempt, ensure you reassess any levels you have previously completed.

Background

A simple strategy is to try all possible numbers from 2 up to the specified limit. For each candidate number, calculate the sum of the candidate's divisors. If the difference between the sum and the candidate is not greater than the maximum badness, the number is considered good enough.

Level 1: Small perfect numbers

Begin by writing a program that prints perfect numbers (those with a badness of 0). Separate the output of each number by a single space. Read the limiting value as a command line argument. For example, **`quitegood 100`** should print **`6 28`**. For this level, you can assume that the limiting value will not exceed 10,000.

Level 2: Small near-perfect numbers

Extend the program so that the badness limit can be specified as a second command line argument. For example, **`quitegood 100 3`** should print **`2 3 4 6 8 10 16 18 20 28 32 64`**.

Level 3: Large numbers

Extend the program so that it will work efficiently for values up to at least 1,000,000. For example, **`quitegood 1000000 1`** should print **`2 4 6 8 16 28 32 64 128 256 496 512 1024 2048 4096 8128 8192 16384 32768 65536 131072 262144 524288`**.

Level 4: Convert either way

Extend the program so that a negative badness value causes the program to output a count of the quite good numbers rather than the numbers themselves. For example, **`quitegood 1000000 -1`** should print **`23`** because there are 23 numbers less than 1,000,000 with a badness no more than 1.