Practical 2: Complexity Analysis

Background

When we study algorithms, we are interested in characterizing them according to their efficiency. We are interested in the order of growth of the running time of an algorithm, not in the exact running time. We call this the asymptotic running time. We need to develop a way to talk about the rate of growth of functions so that we can compare different algorithms. Asymptotic notation gives us such a method.

So if someone tells you that their algorithm runs in $O(n^3)$, then that means their code has n^3 operations or less. For this very reason Big O notation is said to give you upper bounds on an algorithm's performance. **Big O tells you that your algorithm is at least this fast or faster - it won't run slower.**

In general, you would not say a function runs in Big O of n^2 if you can prove that it runs in Big O of n. If someone showed you the print Hello() function above, in an interview and asked you to find the complexity of it, if you answer $O(n^2)$, more than likely they would disqualify you. Even though it is technically correct, the answer they would expect is O(n).

What am I doing today?

1. Compute an integer multiplication by hand using the Russian Peasant's algorithm
2. Convert a pseudo-code implementation of the algorithm into java
3. Test the performance of the algorithm on inputs of various sizes

Instructions

Try all the questions. Ask for help from the demonstrators if you get stuck.

Some of the solutions will be posted afterward.

***To get started, pull the latest update from the class github repository that provides helper functions, data and so on.**

Grading: Remember if to add this sheet, any code, graphs and any accompanying files to your github repo as part of your algorithm portfolio for the course.

1. Russian Peasant's algorithm

The *Russian Peasant's Algorithm* is an algorithm for multiplication that uses doubling, halving, and addition. This was an algorithm or a tool that was used before computers by people to multiply two numbers. One advantage it possesses over the standard method of multiplication that is taught in many schools (i.e. the Standard Algorithm) is that you do not need to have previously memorized multiplication tables to use the algorithm. In practice, the Russian Peasant's Algorithm was likely calculated with the aid of small stones or beads to represent the units.

Read more here:
https://en.wikipedia.org/wiki/Ancient_Egyptian_multiplication#Russian_peasant_multiplication

The Russian Peasant's Algorithm Instructions:

The basic idea is that to multiply x by y, we can compute instead:

While (number1 != 0){

If number1 is even, then add nothing

If number1 is odd, add number 2 to the running total

number1/2

number2*2

}

Example: Multiply 13 x 238

For example, to multiply 238 (number 1) by 13 (number 2) , the smaller of the numbers (to reduce the number of steps), 13, is written on the right and the larger on the left (the order does not matter, it's a matter of convenience). The left number is progressively halved (discarding any remainder) and the right one doubled, until the left number is 1. Then you add all of the number2s together in rows where the number 1 is odd (see below).

13	238
6 (remainder discarded)	476
3	952
1 (remainder discarded)	1904

13	238
~~6~~	~~476~~
3	952
1	+ 1904
	3094

Exercise 1:

Your first exercise is to use the Russian Peasant's Algorithm to multiply the following two integers **by hand**:

68 x 139

68	139	68	~~139~~
34	278	34	~~278~~
17	556	17	556
8 (remainder discarded)	1112	8	~~1112~~
4	2224	4	~~2224~~
2	4448	2	~~4448~~
1	8896	1	+ 8896
			9452

2. Implement the Russian Peasant's algorithm in Java and verify its correctness.

Take the pseudocode below and translate into a basic java algorithm that implements the Russian Peasant's algorithm:

1. Try your best to write this code yourself (no cutting and pasting or searching the web for solutions) - this is the best way to learn. If you get stuck as a demonstrator for assistance.

Option A:

2. Your algorithm should be able to take input from a file of integers found in your data folder
3. Test your algorithm on several input integer files and verify the correctness of the output using a calculator (or by hand).

Option B:

2. Your algorithm should be able to take two integers as arguments, multiply them together and then provide the product

3. Test your algorithm for correctness and then with very large input integers (you may need to use BigInteger for this).

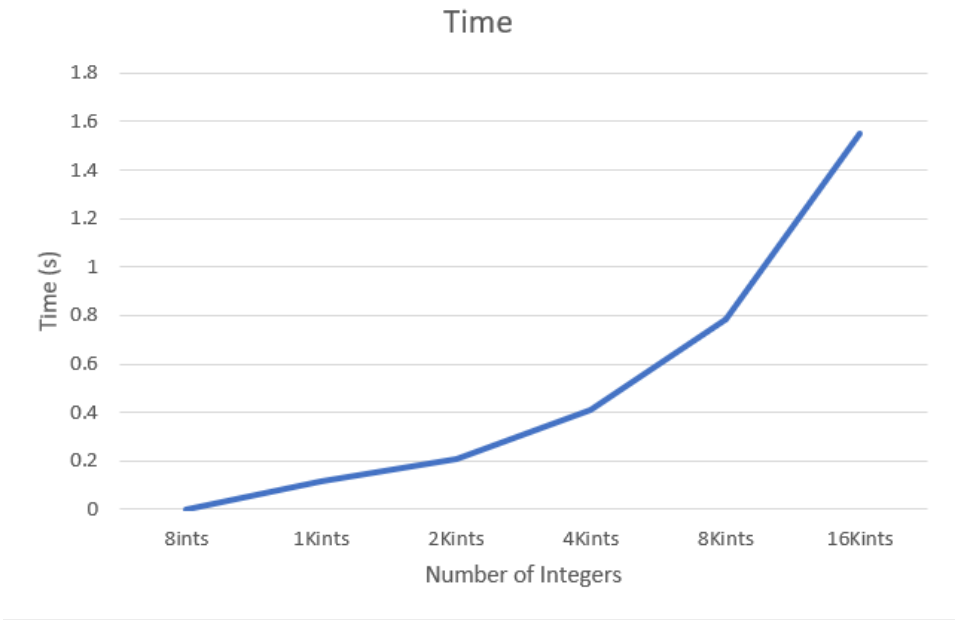
English-style Pseudocode:

Let the two given numbers be 'a' and 'b'.
1) Initialize result 'res' as 0.
2) Do following while 'b' is greater than 0
  a) If 'b' is odd, add 'a' to 'res'
  b) Double 'a' and halve 'b'
3) Return 'res'.

3. Time the performance of the algorithm with various input numbers and plot the results

1. To get a very rough idea of how your algorithm performs, add code to time how it performs multiplying numbers of different size. The Stopwatch() method from last week's practical can be used here or you can rely on Java.lang.System.currentTimeMillis().
2. Graph the results of your experiments
3. What do you think is the complexity of your algorithm and why?

Algorithm **(Option A)**	**Input**	**Time**	**Number of Multiplication Operations**
russianPeasant	8ints.txt	0.003	7
	1Kints.txt	0.118	999
	2Kints.txt	0.211	1999
	4Kints.txt	0.411	3999
	8Kints.txt	0.785	7999
	16Kints.txt	1.553	15999



I believe that the worst case time complexity for the russianPeasants algorithm is O(log N) because the running time grows in proportion to the logarithm of the input file size.