Algorithms, Recursion and Induction: Euclid and Fibonacci

Desh Ranjan
Department of Computer Science
Old Dominion University, Norfolk, VA 23529

1 Introduction

The central task in computing is designing of efficient computational methods, or algorithms, for solving problems. An algorithm is a precise description of steps to be performed to accomplish a task or solve a problem.

Recursion or recursive thinking is a key concept in solving problems and designing efficient algorithms to do so.Often when trying to solve a problem, one encounters a situation where the solution to a larger instance of the problem can be obtained if an appropriate smaller instance (or several smaller instances) of the same problem could be solved. Similarly, often one can define functions where values at "larger" arguments of the function are based on its own value at "smaller" argument values. In the same vein, sometimes it is possible to describe how to construct larger objects from smaller objects of similar type.

However, the obvious question/difficulty here is how does one obtain solution to smaller instances. If one were to think as before (recursively!) this would require solving further smaller instances (sub-instances) of the same problem. Solving these sub-instances could lead to requiring solution of sub-sub-instances, etc. This could all become very confusing and the insight that the larger problem could have been solved using the solutions to the smaller problems would be useless if indeed one could not in some systematic way take advantage of it. Use of proper recursive definitions and notation allows us to avoid this confusion and harness the power of recursive thinking.

As an example of a recursive method of computation consider the well-known factorial function ($N!$). $N!$ is defined to be $N \times (N-1) \times (N-2) \times \ldots \times 2 \times 1$. How can one compute the factorial function? One can do that by multiplying the numbers 1 through N. But an alternative way to think about solving the problem is the following: If one could compute $(N-1)!$, one could compute $N!$ by simply multiplying $(N-1)!$ by N. In other words, $N! = N \times (N-1)!$. Interestingly enough, today we can program this very simply as a computational method.Most high-level programming languages

used today allow for use of recursion directly by allowing recursive functions.

Recursion is a very powerful concept in defining objects or designing methods to solve problems. However, one has to be careful when using recursion as there is clearly scope for self-reference, cyclic or incomplete definitions, and consequently ill-defined objects or methods (which can lead to infinite loops!).

TASKS:

- The definition of factorial function above is not complete. It does not tell us when the recursion should "stop". All proper definitions should tell us when to stop or what are the base cases. Complete the above definition and write pseudocode for a recursive function that computes the factorial function.

- Consider the sum $1 + 2 + 3 + 4 + \ldots + N$. Let $S(N)$ denote this sum.

  - Write an iterative program that that given N as input computes $S(N)$.
  - Give a recursive definition for S and pseudocode for a recursive program that given N as input computes $S(N)$.

- Explain how one can think of a complete binary tree as a recursive structure. (Think about how one can construct a complete binary tree with $n + 1$ levels using complete binary trees with n levels).

- One can sort a list of N numbers by first sorting the first $N-1$ numbers (ignoring the last number) and then inserting the last number in the appropriate place. Write pseudocode for a recursive algorithm to sort an array of N numbers that uses the above idea.

There is evidence that recursive thinking has been used by human beings for problem solving for many centuries. Some philosophers even contend that ability to think recursively is what separates humans from all other species! A clear example of recursive definition arises when one tries to formalize what constitutes a valid sentence in a natural language (say English). This is because a valid sentence can include within itself another shorter valid sentence which can contain within itself another shorter valid sentence and this "nesting" can continue *ad infinitum*. The grammar for the language

specifies rules that allow for determination of validity of a sentence. The best way to capture these rules require recursive thinking and notation. In the 1960s Noam Chomsky developed what are now called Chomsky Grammars for defining languages that use recursive rules explicitly. Interestingly enough, Indian linguist Panini provided the rules of Sanskrit grammar in his treatise *Ashtadhyayi*, in a similar way, more than two thousand years before Chomsky and Chomsky in his work credits him with that. Many mathematicians have used recursive thinking in problem solving over the centuries explicitly or implicitly. A function that is often referred to as an example of a recursively defined function is the one that computes the n^{th} number of a Fibonacci sequence.

2 Fibonacci and Fibonacci Numbers

Fibonacci (c. 1170 – c. 1250), an Italian mathematician, was born as Leonardo Pisano. His father Gugliemo was nikcnamed Bonaccio (which means "good natured" or simple in Italian). Leonardo aquired his nickname Fibonacci after his death. Leonardo was an enormously talented mathematician. He is best known for the adoption of the Indian numeral system (the numeration system we use today) in Europe. As a young boy Leonardo traveled to help his father who directed a trading post in Bugia, a port east of Algiers. This is where he learned about the Hindu numeral system and realized that it was a much more efficient system than Roman numerals to represent numbers and do calculations with them. He also traveled throughout the Mediterranean world to study under Arab mathematicians of the time, returning to Italy in about 1200. In 1202 he published the book *Liber Abaci* (Book of Calculations) which introduced the Hindu numeral system to Europe. In Liber Abaci (1202), Fibonacci introduces the so-called modus Indorum (method of the Indians), today known as Hindu numerals ([4, 1]). The book advocated numeration with the digits 0–9 and place value. The book showed the practical importance of the new numeral system, using lattice multiplication and Egyptian fractions, by applying it to commercial bookkeeping, conversion of weights and measures, the calculation of interest, money-changing, and other applications. The book was well received throughout educated Europe and had a profound impact on European thought. Fibonacci also authored other books of which *Liber quadratorum* ("The Book of Squares") is also very well known to mathematicians. He also

authored a compendium on geometry and trigonometry called *Practica Geometriae*. Some of the works of Leonardo (including a commentary on Book X of Euclid's Elements) are assumed to be lost. Later in his life Leonardo became a friend of Emperor Frederick II of Hohenstaufen who enjoyed mathematics and science. In 1240, the Republic of Pisa honored Leonardo by granting him a salary.

Liber Abaci also posed, and solved, a problem involving the growth of a hypothetical population of rabbits based on idealized assumptions. The solution was a sequence of numbers that later came to be known as known as Fibonacci numbers. The number sequence was known to Indian mathematicians as early as the 6th century [5, 6], but it was Fibonacci's *Liber Abaci* that introduced it to the West. Below, we produce the part of *Liber Abaci* (translated by Sigler) [4] that talks about this problem and the solution.

$\infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty$

How many pairs of rabbits are created by one pair in one year

A certain man had one pair of rabbits together in a certain enclosed place, and one wishes to know how many are created from one pair in one year when it is the nature of them in a single month to bear another pair, and in the second month those born to bear also. Because the abovewritten in the first month bore, you will double it; there will be two pairs in one month. One of these, namely the first, bears in second month, and thus there are in the second month 3 pairs; of these in one month two are pregnant, and in the third month 2 pairs of rabbits are born, and thus there are 5 pairs in the month; in this month 3 pairs are pregnant, and in the fourth month there are 8 pairs, of which 5 pairs bear another 5 pairs; these are added to the 8 pairs making 13 pairs in the fifth month; these 5 pairs that are born in this month do not mate in this month, but another 8 pairs are pregnant, and thus there are in the sixth month 21 pairs;[p284] to these are added the 13 pairs that are born in the seventh month; there will be 34 pairs in this month; to this are added 21 pairs that are born in the eigth month; there will be 55 pairs in this month; to these are added 34 pairs that are born in the ninth month; there will be 89 pairs this month; to these are added 55 pairs that are born in the tenth month; there will be 144 pairs in this month; to these are added again the 89 pairs that are born in the eleventh month; there will be 233 pairs in this month. To these are still added the 144 pairs that are born in the last month; there will be 377 pairs, and these many pairs are produced from the abovewritten pair in the mentioned place at the end of one year.

first	2
second	3
third	5
fourth	8
fifth	13
sixth	21
seventh	34
eighth	55
ninth	89
tenth	144
eleventh	233
twelfth	377

You can indeed see in the margin how we operated, namely that we added the first number to the second, namely the 1 to the 2, and the second to the third, and the third to the fourth, and the fourth to the fifth, and thus one after another until we added tenth to the eleventh, namely 144 to the 233, and we had the abovewritten sum of rabbits, namely 377, and thus you can in order find it for unending number of months.

$\infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty \quad \infty$

TASKS:

- Compute the number of rabbit pairs at the the end of the 13^{th} and 14^{th} months. How many rabbit pairs are there after 2 years?
- Fibonacci explains in the second paragraph how "we operated". Then at the end of the paragraph, he concludes "..and thus you can in order find it for any unending number of months". How do you think Fibonacci would have computed the number of pairs of rabbits after 2 years ? Carry out the computation as you think he would have. Would you call this a recursive method?
- In his explanation of "how we operated", Fibonacci talks about adding the first number to the second, second to the third, third to the fourth and so on. One can then notice that the twelfth (Fibonacci) number is obtained by adding the eleventh (Fibonacci number) to the tenth (Fibonacci number), etc. Let $F(n)$ denote the number of rabbit pairs after

n months. So, for example, $F(1) = 2, F(2) = 3, F(4) = 5, F(12) = 377$ etc. Fibonacci's method also demonstrates how to obtain the value of $F(n)$ from "previous" values of F. Use this to provide a complete recursive definition of F.

- [**Programming Exercises**]
  - Develop a computer program based on Fibonacci original calculation method to compute the n^{th} Fibonacci number. Analyze the runtime and space requiremrnts for this program.
  - Develop a purely recursive computer program $RECURSIVE_FIB(n)$ that given an input n computes the nth Fibonacci number. How many recursive calls does $RECURSIVE_FIB(n)$ make to compute $F(n)$?
  - The runtime and the number of recursive calls of $RECURSIVE_FIB$ can be reduced substantially by systematically storing the previously computed values and using them. Explain how one can write a *recursive* program to compute n^{th} Fibonacci number that uses auxillary storage to avoid re-computation. Analyze the runtime and space requirements of this program. How do these compare to that of your other two programs.
  - Explain, how you can further reduce the space requirement of your first program (without increasing the runtime) and implement a fourth program to compute n^{th} Fibonacci number that makes this improvement.

- One advantage of defining functions recursively is that it allows easy use of the method of induction to prove some properties of the function. Use induction to show that for all $n \geq 1$, $F(n) \leq 2^n$.

- Use induction to show that for all $n \geq 1$, $F(n) \geq (3/2)^n$. What does this tell you about the runtime of $RECURSIVE_FIB(n)$?

- [**Advanced Work**] Find an exact "formula" for $F(n)$.

3 Euclid's Algorithm

As stated in the introduction, the notion of algorithms is central to computing. One of the very first algorithms introduced in a course on algorithms is Euclid's algorithm to compute the greatest common divisor (GCD) of two numbers. The algorithm was described by Euclid more than two thousand years ago as Proposition 2 in Book VII of the compendum *Elements*. While this algorithm is presented in numerous different ways, Euclid's original description can be paraphrased as follows ([2, 3])

Let a and b ($a < b$) be two numbers that are not prime to each other. If a divides b then a is the gcd of a and b. Otherwise repeatedly subtract the lesser number continually from the greater. In this process some number will be left that will divide the one before it. This will be the GCD of a and b.

There is some ambiguity as to what Euclid meant by "the number left" and the "one before it" ([3]). Today, we understand the "number left" as the number left after repeatedly subtracting the smaller number from the larger number until what remains is less than the smaller number. At this point the "larger" and the "smaller" number are switched. If the smaller now divides the larger then the smaller is the GCD, otherwise the process of subtracting smaller from the larger is repeated.

4 Runtime Analysis of Euclid's Algorithm

In this section, we analyse the runtime of Euclid's Algorithm. It turns out that the the analysis is closely related to the Fibonacci Numbers. It is interesting that the algorithm that is often the first algorithm presented in an algorithms course and a sequence that is one of the earliest presented when recursion is typically introduced happen to be so closely related.

TASKS:

- In modern terminology "the number left" is referred to as the remainder and if we are dividing b by a ($a \leq b$) the remainder is denoted by $b \bmod a$. Use this notation to provide an iterative version of Euclid's algorithm.

- Although Euclid does not state it this way, one can notice that while computing the GCD of two numbers a and b with $a \leq b$, one first checks if $a|b$ (if so, a is the GCD) and if not the same process is applied to the numbers a and b mod a. Use this idea to provide a recursive version of Euclid's algorithm.

- Compute the GCD of 34 and 21 using Euclid's (either) method. Compute the GCD of 377 and 233 using Euclid's method.

- Guess how many mod operations it takes to compute the GCD of F_n and F_{n-1}. Prove this using induction.

- Let a and b denote the two numbers ($a \leq b$) in Euclid's algorithm (either version) at some point in execution. Let n be the smallest number such that $F_n \geq b$. Let a' and b' denote the two numbers ($a' < b'$) after Euclid's algorithm has performed two more mod operations. Show that $b' \leq F_{n-1}$.

  Hint: Consider the two cases, $a \leq F_{n-1}$ and $a > F_{n-1}$ separately.

- Use induction to show that to compute GCD of two numbers a, b with $a < b$, Euclid's algorithm performs no more than $2n$ mod operations where n is the smallest number such that $F_n \geq n$.

- Show that a pair of consecutive Fibonacci numbers is the worst family of inputs for Euclid's algorithm. More precisely, show that among all inputs a, b where $b \leq F_n$, the pair (F_{n-1}, F_n) requires the maximum number of mod operations.

- The number of bits required to represent a number, called $size(b)$, is $\lceil \log_2(b+1) \rceil$. Combine this fact, with the task on growth rate of Fibonacci numbers to show that Euclid's algorithm performs no more than $A \times size(b)$ mod operations for some constant A when computing the GCD of two numbers, the larger one of which is b.

5 Comments for Instructor

The project provides an opportunity to introduce students to the notion of algorithms, recursion and induction. It allows them to explore the interrelationship between recursion and iteration in a mathematical sense. Simul-

taneously, it introduces them to the idea of computational efficieny and rate of growth of functions. It provides a setting where students can perform basic efficiency analysis of an algorithm using induction which reinforces the usefulness of learning the technique. By judiciously selecting the tasks from the list of all tasks, the project can be used in a sophomore or junior level discrete mathematics or algorithms course. The instructor should consider spending more time on tasks in section 1 to ensure that the students feel comfortable with simple recursive definitions if they are not familiar with them already.

References

[1] Grimm, R.E. The Autobiography of Leonardo Pisano, Fibonacci Quarterly, Vol 11(1), pp. 99–104, 1973.

[2] Heath, T.L. *Euclid The Thirteen Books of the Elements*, Volume 2, Second Edition, Dover Publications, New York, 1956.

[3] Ranjan, D. Euclid's Algorithm for the Greatest Common Divisor, `www.cs.nmsu.edu/historical-projects/Projects/EuclidGCD.pdf`, 2008.

[4] Sigler, L.E. English translation of Fibonacci's Liber Abaci, Springer-Verlag, 2002.

[5] Singh, P. Acharya Hemachandra and the (so called) Fibonacci Numbers. Math Ed. Siwan, 20(1):28–30, 1986.

[6] Singh, P. The so-called Fibonacci numbers in ancient and medieval India,, *Historia Mathematica* 12(3), 229–244, 1985.