CS 217 – Algorithm Design and Analysis

Shanghai Jiaotong University, Fall 2018

Handed out on 2018-09-13
First submission and questions due on 2018-09-17
Final submission due on 2018-09-24

1 Bit Complexity, Recursion, and Dynamic Programming

1.1 Bit Complexity of Euclid's Algorithm

We have proved that Euclid's algorithm for computing $\gcd(a, b)$ makes at most $O(\log a)$ iterations. What is the overall running time? Each iteration computes $u \mod v$ for some integers. This can be done by integer division. What is its running time? There are very sophisticated algorithms, but python probably does not come with them. Recall the "school method" for dividing integers. Have a look at the pdf slides on the webpage for an illustration of the school method. It is especially simple if we are dealing with binary numbers. If a and b have at most n bits, then the school method has complexity $O(n^2)$.

Exercise 1. Show the following, more precise bound of the school method for integer division: If a has n bits and b has k bits, then the school method can be implemented to run in $O(k(n-k))$ operations.

Exercise 2. Show that the bit complexity of Euclid's algorithm, using the school method to compute $a \mod b$, is $O(n^2)$. That is, if a and b have at most n bits, then $\gcd(a, b)$ makes $O(n^2)$ bit operations.

In order to do so, recall Jin's python code for computing the gcd of two integers:

```
def jin_gcd(a,b):
  while (b > 0):
      r = a % b # so a = bu+r
      if (r == 0):
          return b
      s = b % r # so b = rv + s
      a = r
      b = s
  return a
```

Don't be afraid to introduce notation! I recommend to let n denote the number of bits of a. Take some other letters for the number of bits in b and so on.

1.2 Computing the Binomial Coefficient

Next, we will investigate the binomial coefficient $\binom{n}{k}$, which you might also know by the notation C_n^k. The number $\binom{n}{k}$ is defined as the number of subsets of $\{1, \ldots, n\}$ which have size exactly k. This immediately shows that $\binom{n}{k}$ is 0 if k is negative or larger than n. You might have seen the following recurrence:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \text{ if } n, k \geq 0 \ .$$

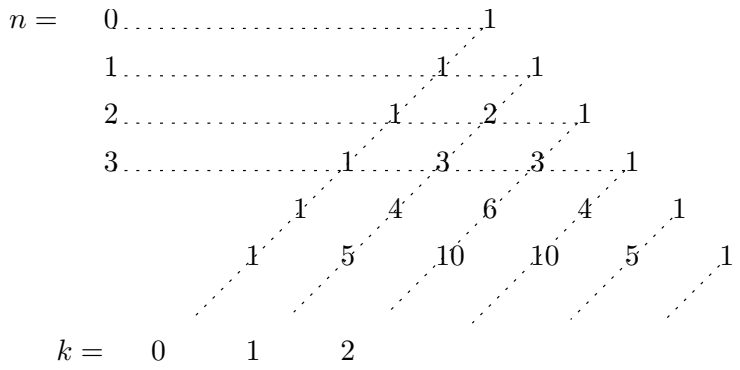
Exercise 3. [A Recursive Algorithm for the Binomial Coefficient] Using pseudocode, write a recursive algorithm computing $\binom{n}{k}$. Implement it in python! What is the running time of your algorithm, in terms of n and k? Would you say it is an efficient algorithm? Why or why not?

Exercise 4. [A Dynamic Programming Algorithm for the Binomial Coefficient] Using pseudocode, write a dynamic programming algorithm computing $\binom{n}{k}$. Implement it in python! What is it running time in terms of n and k? Would you say your algorithm is efficient? Why or why not?

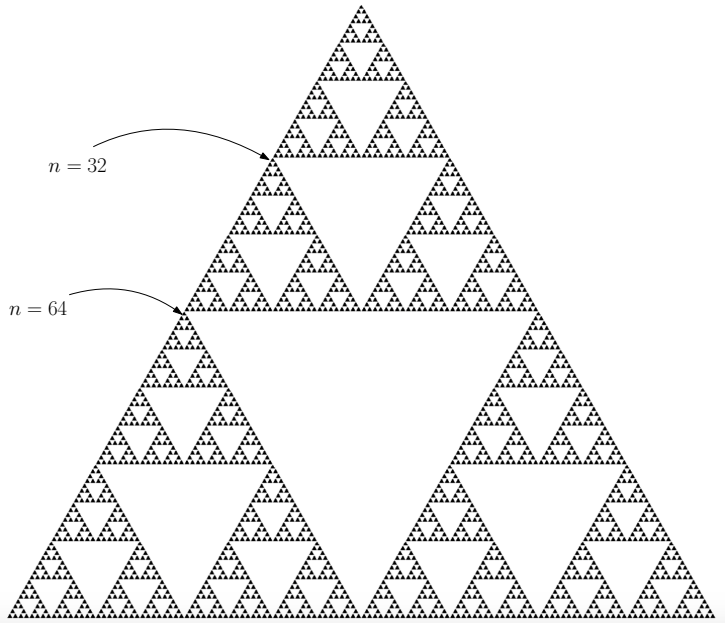
Exercise 5. [Binomial Coefficient modulo 2] Suppose we are only interested in whether $\binom{n}{k}$ is even or odd, i.e., we want to compute $\binom{n}{k} \mod 2$. You could

do this by computing $\binom{n}{k}$ using dynamic programming and then taking the result modulo 2. What is the running time? Would you say this algorithm is efficient? Why or why not?

You might have heard that the numbers $\binom{n}{k}$ can be arranged to form Pascal's triangle[1]. Here is its top part:



If we draw a black dot for every odd number in this triangle (and nothing for an even number), we get the *Sierpinski triangle*:



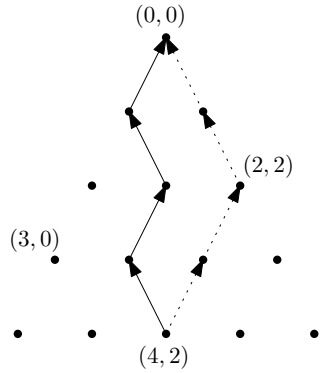
[1]Also known as Yanghui triangle or Khayyam triangle, after the Chinese (respectively Iranian) mathematician who described it. Read the Wikipedia page!

Notice its strong recursive structure. In particular, if $n = 2^d$ is a power of 2, then the n^{th} row of the triangle is completely white, except for the two endpoints. More formally, we have the following lemma:

Lemma 6. *If $n = 2^d$ is a power of 2, then $\binom{n}{k} = 0$ for all $1 \leq k \leq n-1$.*

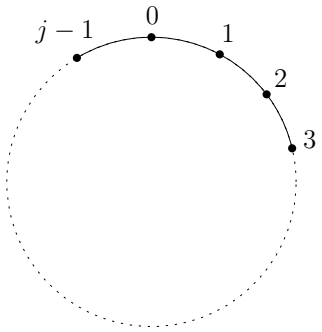
Exercise 7. [Binomial Coefficient modulo 2, Recursive] Give a recursive algorithm for computing $\binom{n}{k} \mod 2$ using the above lemma. Your algorithm should not have problems handling numbers n and k with tens of thousands of digits. Implement your algorithm in python!

Hint. Look closely at the structure in the picture above. If you want to reason more formally, here is a good way to think of $\binom{n}{k}$: it is the number of paths leading from the location of $\binom{n}{k}$ to the top of Pascal's triangle, if you are only allowed left-up and right-up moves, as shown in this picture:

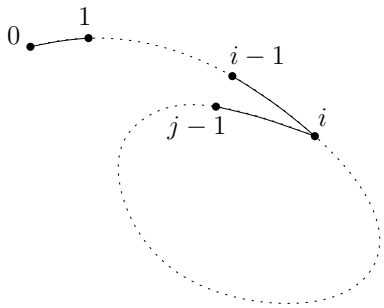


two of the $\binom{4}{2} = 6$ paths leading from $(4, 2)$ to $(0, 0)$.

Exercise 8. Remember the "period" algorithm for computing $F'_n := (F_n \mod k)$ discussed in class: (1) find some i, j between 0 and k^2 for which $F'_i = F'_j$ and $F'_{i+1} = F'_{j+1}k$. Then for $d := j - i$ the sequence F'_n will repeat every d steps, as there will be a cycle. This cycle can either be a "true cycle" or a "lasso":



a cycle



a lasso

Show that a lasso cannot happen. That is, show that the smallest i for which this happens is 0, i.e, for some j we have $F'_0 = F'_j$ and $F'_1 = F'_{j+1}$ and thus $F'_n = F'_{n \bmod j}$.

Python. Please write your code in python. It is a very simple programming language. If you do not know any python, I can put same example code online and you can learn by example. Also, for this homework you definitely need a Big Integer class since numbers with ten thousand digits do not fit into any `long long int` or similar. Python automatically supports Big Integer, so there is no problem here.