CS21003 Algorithms–I, Spring 2017–2018

Class Test 1

08–February–2018 | CSE 107/119/120, 07:00pm–08:00pm | Maximum marks: 20

Roll no: ____________________ **Name:** ____

Write your answers in the question paper itself. Be brief and precise. Answer all questions.
If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.

1. Catalan numbers $C(n)$ are given by the formula $C(n) = \frac{1}{n+1}\binom{2n}{n}$ for all integers $n \geqslant 1$.

(a) Find a simple closed-form expression for $C(n+1)/C(n)$ for all $n \geqslant 1$. **(3)**

Solution We have

$$\begin{aligned} C(n+1)/C(n) &= \left(\frac{n+1}{n+2}\right)\binom{2n+2}{n+1}\Big/\binom{2n}{n} \\ &= \left(\frac{n+1}{n+2}\right)\left(\frac{(2n+2)(2n+1)\cdots(n+2)}{(n+1)!}\right)\left(\frac{n!}{(2n)(2n-1)\cdots(n+1)}\right) \\ &= \left(\frac{n+1}{n+2}\right)\left(\frac{(2n+2)(2n+1)}{(n+1)^2}\right) \\ &= \frac{4n+2}{n+2}. \end{aligned}$$

(b) Prove that $C(n) = O(4^n)$. **(3)**

Solution For all $n \geqslant 1$, we have $\frac{C(n+1)}{C(n)} \leqslant \frac{4n+8}{n+2} = 4$. It follows that

$$C(n) = \left(\frac{C(n)}{C(n-1)}\right)\left(\frac{C(n-1)}{c(n-2)}\right)\cdots\left(\frac{C(2)}{C(1)}\right)C(1) \leqslant 4^{n-1} = \left(\frac{1}{4}\right)4^n.$$

(c) Prove that $C(n) = \Omega((4-\varepsilon)^n)$ for any constant ε satisfying $0 < \varepsilon < 3$. **(4)**

Solution $\dfrac{C(n+1)}{C(n)} = \dfrac{4n+2}{n+2} \geqslant 4-\varepsilon$ for all $n \geqslant n_0$, where $n_0 = \left\lceil \dfrac{6-2\varepsilon}{\varepsilon} \right\rceil$. For all $n \geqslant n_0$, we then have

$$C(n) = \left(\frac{C(n)}{C(n-1)}\right)\left(\frac{C(n-1)}{c(n-2)}\right)\cdots\left(\frac{C(n_0+1)}{C(n_0)}\right)C(n_0) \geqslant (4-\varepsilon)^{n-n_0}C(n_0) = \left(\frac{C(n_0)}{(4-\varepsilon)^{n_0}}\right)4^n.$$

Since ε and n_0 are constant, the result follows.

2. Let n be a positive integer. The *partition number* $p(n)$ is the count of ways in which n can be expressed as a sum of positive integers. For example, 5 can be written in seven ways as $1+1+1+1+1$, $2+1+1+1$, $2+2+1$, $3+1+1$, $3+2$, $4+1$, and 5. Therefore $p(5) = 7$. Notice that permuting the summands does not give a new way of expressing n. For example, $2+2+1$, $2+1+2$ and $1+2+2$ are considered the same partition of 5. By convention, $p(0) = 1$ (0 is the empty sum of zero number of summands). In this exercise, you use dynamic programming to compute $p(n)$ efficiently.

(a) In order to avoid duplicate permutations of the summands in a given partition, we choose the summands in non-increasing order. For example, if 3 is chosen as a summand, the remaining summands are allowed to be 3, 2, and 1 only. Keeping this in mind, we build a two-dimensional table T such that $T[i][j]$ is meant for storing the count of partitions of i, in which the largest summand allowed is j. Make a recursive formulation of $T[i][j]$. Also supply the initial conditions. **(4)**

Solution If $i = 0$, there are no more explorations of choosing summands. If $1 \leqslant j \leqslant i$, there are two options: we choose all summands $< j$, or we choose j as a summand. Finally, if $i+1 \leqslant j \leqslant n$, the maximum summand allowed is i. These imply the following.

Initial conditions:	$T[0][j] = 1$ for all $j = 0, 1, 2, \ldots, n$,
Recursive relation 1:	$T[i][j] = T[i][j-1] + T[i-j][j]$ for $1 \leqslant j \leqslant i$,
Recursive relation 2:	$T[i][j] = T[i][i]$ $\Big($or $T[i][j-1]\Big)$ for $i+1 \leqslant j \leqslant n$.

(b) Propose a bottom-up approach of filling the table T. Do not use memoization. Mention how you finally obtain $p(n)$ from T. **(4 + 1)**

Solution The following pseudocode describes the algorithm. The recursion formulas in Part (a) indicate that we can fill T in the row-major order.

1. For $j = 0, 1, 2, \ldots, n$, set $T[0][j] = 1$.
2. For $i = 1, 2, 3, \ldots, n$, repeat:
   (a) For $j = 1, 2, 3, \ldots, i$, set $T[i][j] = T[i][j-1] + T[i-j][j]$.
   (b) For $j = i+1, i+2, \ldots, n$, set $T[i][j] = T[i][i]$ $\left(\text{or } T[i][j] = T[i][j-1]\right)$.

In the count $p(n)$, the sum is n, and the maximum allowed summand is n. So $T[n][n]$ is returned as $p(n)$.

(c) What is the running time of your algorithm of Part (b)? **(1)**

Solution $\Theta(n^2)$ (the running time is dominated by the time for building T which has $(n+1)^2$ entries, each of which can be computed in $\Theta(1)$ time).

FOR ROUGH WORK