TCSS 343 - Week 4 - Wednesday

Jake McKenzie

February 12, 2019

Master Method and some more Divide and Conquer

"What we choose means more than what was handed to us by chance".
...
Ada Palmer

"Great perils have this beauty, that they bring to light the fraternity of strangers".
...
Victor Hugo

"Truth persuades by teaching, but does not teach by persuading".
...
Tertullian

1. In this problem use the Master Theorem to find and prove tight bounds for the following recurrence.

$$T(n) = \begin{cases} \text{c} & \text{if } n \leqslant 1 \\ 9T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + 27n & \text{if } n > 1 \end{cases}$$

2. In this problem use the Master Theorem to find and prove tight bounds for the following recurrence.

$$T(n) = \begin{cases} \text{c} & \text{if } n \leqslant 1 \\ 2T\left(\left\lfloor \frac{2n}{3} \right\rfloor\right) + 3n & \text{if } n > 1 \end{cases}$$

3. In this problem use the Master Theorem to find and prove tight bounds for the following recurrence.

$$T(n) = \begin{cases} \text{c} & \text{if } n \leqslant 1 \\ 70T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + 12n^3 & \text{if } n > 1 \end{cases}$$

4. Design an algorithm that accepts an unsorted array of integers and finds the subarray with the maximum possible sum.

   For example, consider the array $\{2, -4, 1, 9, -6, 7, -3\}$. The maximum subarray would be $\{1, 9, -6, 7, -3\}$, which sums to 11.

   A naive solution that considers every possible subarray would take $O(n^2)$ time. Design a more efficient algorithm that uses divide and conquer and runs in $O(n \log n)$ time.

5. Given an array containing elements of type E design an algorithm that finds the majority element – that is, an element that appears more then $\frac{n}{2}$ times. If no majority element exists, return null.

   Your algorithm should run in $O(n \log n)$ time (and use only $O(1)$ extra memory).

6. Suppose you are trying to write a video game containing thousands of different moving elements and want to check if two elements have collided or overlapped.

   A naive way of implementing this would be to use two nested loops and check every pair of elements. This often ends up being too inefficient for most video games, even for only a few thousand elements (especially games that require a high degree of responsiveness).

   Describe how you would design a data structure to store these points in a way that lets you more efficiently check whether two elements are colliding.

   For the sake of simplicity, you may assume that each element is a circle and has a relatively small radius. You may also assume that the elements are moving on a 2d plane – you don't need to worry about collisions in 3d.

   As a hint: think about recursively subdividing the 2d plane.

7. Give an efficient algorithm to compute the binomial coefficient $\binom{n}{k}$, reminder: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Use the recursive definition below to help you along.

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k = 0 \quad \text{or} \quad k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0 < k < n \end{cases}$$

What is the time complexity of your algorithm? Is it a polynomial time algorithm? Explain.