1. Solve the following recurrence relations; i.e. express each one as $T(n) = O(f(n))$ for the tightest possible function $f(n)$, and give a short justification (some are longer than others, but full inductions are not required). Unless otherwise stated, assume $T(1) = 1$.

1. $T(n) = 4T(n/2) + n^2$
2. $T(n) = 3T(n/4) + \sqrt{n}$
3. $T(n) = 7T(n/3) + n^3$
4. $T(n) = 2T(n/2) + n\log n$
5. $T(n) = 2T(\sqrt{n}) + 1$, where $T(2) = 1$

3. Full write-up. Let us consider a problem that arises in a number of financial and data analysis applications. Suppose you are buying a new car. You would like a sporty car with fast acceleration, but you are frugal and also want a car that has good gas mileage. You research various models of cars and for each you record the acceleration and mileage as points on an $(x, y)$ plot. It is not surprising that some cars have excellent mileage but poor acceleration and vice versa. One thing you are sure is that you don’t want a car that is low in both.

Formally, let $S$ be a set of $n$ points in the plane. A point $p \in S$ is dominant if no point in $S$ is both above and to the right of $p$. Describe and analyze a divide-and-conquer algorithm to find all the dominant points in a given $n$-point set in $O(n\log n)$ time. You may assume all the input points have distinct $x$-coordinates and distinct $y$-coordinates.

![Figure 1: Filled points are dominant](image-url)
3. Full write-up. A typical representation of a grayscale image is an $n \times n$ 2D array, where each $[i, j]$ index stores an integer between 0 and 255, with 0 representing a black pixel and 255 white. In image processing, it is often the case that we would like to find those pixels that are brighter than all the surrounding pixels. This has applications in feature detection and many other filtering techniques. Design a divide-and-conquer algorithm that finds any one such pixel in $O(n\log n)$ time. You may limit the neighbors to the 4 edge neighbors. It is acceptable to interpret brighter than as “brighter than or equal to”, or simply assume that all neighboring pixel values are distinct. Extra credit: find an $O(n)$ solution.

Please hand in your assignment electronically on Moodle.