Problem 1
Solve the following recurrences:

1. $T(n) = 4T(n/2) + n^2$
2. $T(n) = T(n/2) + 2^n$
3. $T(n) = 16T(n/4) + n$
4. $T(n) = 2T(n/2) + n \log n$
5. $T(n) = \sqrt{2}T(n/2) + logn$
6. $T(n) = 3T(n/2) + n$
7. $T(n) = 3T(n/3) + \sqrt{n}$

Problem 2
Consider the following algorithm.

Algorithm 1 Reverse

function reverse(A)
    if $|A| = 1$ **then**
        return A
    else
        Let B and C be the first and second half of A
        return concatenate reverse(C) and reverse(B)
    end if
end function

Let $T(n)$ be the running time of the algorithm on a instance of size n. Write down the recurrence relation for $T(n)$ and solve it by unrolling it.

Problem 3
The product of two $n \times n$ matrices X and Y is a third $n \times n$ matrix $Z = XY$, where the (i, j) entry of Z is $Z_{ij} = \sum_{k=1}^{n} X_{ik}Y_{kj}$. Suppose that X and Y are divided into four $n/2 \times n/2$ blocks each:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Using this block notation we can express the product of X and Y as follows

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}.$$

In this way, one multiplication of $n \times n$ matrices can be expressed in terms of 8 multiplications and 4 additions that involve $n/2 \times n/2$ matrices. Let $T(n)$ be the time complexity of multiplying two $n \times n$ matrices using this recursive algorithm.

1. Derive the recurrence for $T(n)$. (Assume adding two $k \times k$ matrices takes $O(k^2)$ time.)
2. Solve the recurrence by unrolling it.

Problem 4
Your friend Alex is very excited because he has discovered a novel algorithm for sorting an array of n numbers. The algorithm makes three recursive calls on arrays of size $\frac{2n}{3}$ and spends only $O(1)$ time per call.

Algorithm 2 NEW-SORT

```
procedure NEW-SORT(A)
    if |A| < 3 then
        sort A directly
    else
        NEW-SORT(A[0], ..., A[2n/3])
        NEW-SORT(A[n/3], ..., A[n])
        NEW-SORT(A[0], ..., A[2n/3])
    end if
end procedure
```

Alex thinks his breakthrough sorting algorithm is very fast but has no idea how to analyze its complexity or prove its correctness. Your task is to help Alex:

1. Find the time complexity of NEW-SORT
2. Prove that the algorithm actually sorts the input array

Problem 5
Given an array A holding n objects, we want to test whether there is a *majority* element; that is, we want to know whether there is an object that appears in more than $n/2$ positions of A.

Assume we can test equality of two objects in $O(1)$ time, but we cannot use a dictionary indexed by the objects. Your task is to design an $O(n \log n)$ time algorithm for solving the majority problem.

1. Show that if x is a majority element in the array then x is a majority element in the first half of the array or the second half of the array
2. Show how to check in $O(n)$ time if a candidate element x is indeed a majority element.
3. Put these observation together to design a divide an conquer algorithm whose running time obeys the recurrence $T(n) = 2T(n/2) + O(n)$
4. Solve the recurrence by unrolling it.

Problem 6
Suppose we are given an array A with n distinct numbers. We say an index i is locally optimal if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ for $0 < i < n-1$, or $A[i] < A[i+1]$ for if $i = 0$, or $A[i] < A[i-1]$ for $i = n-1$.

Design an algorithm for finding a locally optimal index using divide an conquer. Your algorithm should run in $O(\log n)$ time.