Pre-tutorial questions

Do you know the basic concepts of this week's lecture content? These questions are only to test yourself. They will not be explicitly discussed in the tutorial, and no solutions will be given to them.

1. Divide and Conquer
   (a) What is the general idea of a Divide-and-Conquer algorithm? Can you break up the general structure into three steps?
   (b) Why are recursions usually needed to analyse the running time of a Divide-and-Conquer algorithm?
2. Algorithms
   (a) Describe the general idea of merge sort.
   (b) What are the three main steps in the algorithm for counting inversions?
3. Recurrences
   (a) State the master method.
   (b) Try to explain the master method using a recursion tree.

Tutorial

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

```
1: function REVERSE(A)
2:     if |A| = 1 then
3:         return A
4:     else
5:         Let B and C be the first and second half of A
6:         return concatenate REVERSE(C) and REVERSE(B)
7:     end if
8: 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

```
1: procedure NEW-SORT(A)
2:     if |A| < 3 then
3:         sort A directly
4:     else
5:         NEW-SORT(A[0], ..., A[2n/3])
6:         NEW-SORT(A[n/3], ..., A[n])
7:         NEW-SORT(A[0], ..., A[2n/3])
8:     end if
9: 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.

Problem 7

[Hard] Given two sorted lists of size m and n. Give an $O(\log(m+n))$ time algorithm for finding the kth smallest element in the union of the two lists.

Problem 8

[Advanced] Can you prove a lower bound on the approximation ratio of the greedy algorithm for Set Cover?

Problem 9

[Advanced] Can you prove a lower bound on the approximation ratio of the "MST-approach" (doubling the MST) for the Travelling Salesman Problem?