• This quiz is closed book, closed notes and 80 minutes long
• Read the questions carefully
• No electronic equipment allowed (smart phones, tablets, computers, …)
• Write legibly. What can’t be read will not be graded
• When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
• Always remember to analyze the time complexity of your solution
• If you have a question about the meaning of a question, raise your hand
Problem 1. [Analysis] (17 points)

Use the Master method (after an appropriate substitution) to give an asymptotic tight bound for $T(n)$ defined by the following recurrence relation

$$T(n) = \begin{cases} 2 & n = 2 \\ 3T\left(n^{1/3}\right) + (\log n) (\log \log n) & n > 2 \end{cases}$$
Problem 2. [Amortized Analysis] (17 points)

A sequence of Push and Pop operations is performed on a stack whose size never exceeds a given value $k$. After every $k$ operations, an operation COPY is automatically invoked which copies the entire stack for backup purposes (including the empty slots). In other words, COPY always costs $O(k)$ irrespective of the number of items in the stack. Show that the cost of a sequence of $n$ operations Push, Pop, COPY, is $O(n)$ by assigning suitable amortized costs to Push and Pop.
Problem 3. [Divide and Conquer] (17 points)

An array $A$ is said to have a majority element if more than half of the entries in $A$ are exactly the same. Describe an $O(n \log n)$ divide-and-conquer algorithm that determines whether an array $A$ of $n$ items has a majority element, and if so, returns that item. The only comparison operation allowed on the items is equality. That is, your algorithm can determine whether “$A[i] == A[j]$” or not in $O(1)$ time, but it cannot, for example, compare the items to sort them, or hash the items into buckets. Explain why your algorithm takes $O(n \log n)$ time.
Problem 4.

Write the pseudo-code of the direct (recursive) FFT and analyze its time complexity as a function of the input size $n$. You do not need to prove its correctness.
Problem 5. (17 points)

Assume that you are given two unsorted arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of $n$ positive integers. We want to determine an ordering of the elements of $A$ and $B$ such that $W = \prod_{i=1}^{n} a_i^{b_i}$ is maximized. Consider the following greedy algorithm.

**Algorithm** Greedy $(A, B)$; sort $A$ and $B$ in decreasing order; return $(A, B)$

Show that the greedy-choice property holds for the algorithm Greedy (no need to prove that the problem has the optimal substructure property).
**Problem 6.** (15 points)

Consider an implementation of union-find that uses union by rank and path compression. Draw the tree(s) that result at the end of each of the following set of operations, and indicate the rank of each node using subscripts.

\[
\text{Makeset}(a); \text{Makeset}(b); \text{Makeset}(c); \text{Makeset}(d); \text{Makeset}(e); \text{Makeset}(f); \text{Makeset}(g); \text{Makeset}(h);
\]

\[
\text{Union}(a, b); \text{Union}(c, d); \text{Union}(e, f);
\]

\[
\text{Union}(b, g); \text{Union}(d, h);
\]

\[
\text{Union}(g, f);
\]

\[
\text{Union}(f, h);
\]