Weekly Assignment 8: Divide-and-Conquer Algorithms

1. Alice and Bob both come up with their own algorithm for a notoriously hard problem. Alice's algorithm has time complexity

$$T_A(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2T_A(n-1) + 37 & \text{if } n > 0 \end{cases}$$

and Bob's algorithm has time complexity

$$T_B(n) = \begin{cases} 1 & \text{if } n = 0 \\ 3T_B(n-1) + 42 & \text{if } n > 0 \end{cases}$$

Alice claims the faster algorithm. But Bob responds: "But the difference is only a few constants, asymptotically these are the same: we have $T_A \in \Theta(T_B)$!". Alice disagrees, and states that only $T_A \in \mathcal{O}(T_B)$ but *not* $T_A \in \Theta(T_B)$.

   (a) Who is right? Explain your answer, so that both Alice and Bob will be convinced. In particular, give the Θ-class to which T_A and T_B belong (no proof needed).

   (b) In a collaborative effort, Alice and Bob create a new algorithm, with the following complexity:

$$T_C(n) = \begin{cases} 37 & \text{if } n = 1 \\ 2T_C(\frac{n}{2}) + 42n & \text{if } n = 2^k,\ k > 0 \end{cases}$$

   Compare this with the previous solutions, by again giving the Θ-class to which it belongs.

2. Given a sorted array of distinct integers $A[1 \cdots n]$, you want to find out whether there is an index i for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $\mathcal{O}(\log n)$. Explain why your algorithm is correct, and analyse the complexity.

3. You are given two sorted arrays of size m and n. Give an $\mathcal{O}(\log m + \log n)$ time algorithm for computing the kth smallest element in the union of the two arrays.

4. Suppose we want to install software packages from a given set P of n packages. At most d packages in P (we don't know which ones) may have a *defect*: these packages can not be part of any successful installation. Assume there is a procedure `install`(Q), which attempts to install all the packages from a subset $Q \subseteq P$. This procedure has two possible outcomes: `success` when all packages of Q have been successfully installed, or `fail` in case the attempt was unsuccessful.

   (a) Assume an installation attempt `install`(Q) is successful if and only if none of the packages in Q has a defect. Give an algorithm that uses at most $\mathcal{O}(d \log n)$ calls to `install` to successfully install a maximal subset of P.

   (b) Now assume that there may also be (known) *dependencies* between packages: we have a directed acyclic graph (DAG) $G = (P, K)$ such that for any pair $(p, q) \in K$ any installation that includes q but excludes p will fail. In this setting, an installation attempt `install`(Q) is successful if and only if none of the packages in Q has a defect and moreover, for each dependency $(p, q) \in K$, $q \in Q$ implies $p \in Q$. Again give an algorithm that uses at most $\mathcal{O}(d \log n)$ calls to procedure `install` to successfully install a maximal subset of P.