# PSO 2 — Week 3: Problems

CS381, Fall 2026

---

## Problem 1. Heaps, continued

Let $T$ be a min-heap storing $n$ distinct keys, with the root at depth 0. You may use the following two results from last week without reproving them:

(F1) the $k$-th smallest key of $T$ lies at depth at most $k-1$;

(F2) for any $x$, all $m$ keys of $T$ smaller than $x$ can be reported in $O(1+m)$ time.

(a) Consider the proposed algorithm: run BFS from the root of $T$ and return the $k$-th node dequeued as the $k$-th smallest. Determine *exactly* the set of pairs $(n,k)$ with $1 \le k \le n$ for which this is correct on every min-heap of $n$ distinct keys. Prove correctness for the pairs in your set, and for every pair outside it exhibit a valid min-heap on which the algorithm fails.

(b) Now let $T$ be a $d$-ary min-heap: every node has at most $d$ children, the tree is complete, and $d$ is not assumed constant. Give the cost of Insert and of ExtractMin as functions of $n$ and $d$, and say which of the two pays for $d$ and why. Then consider a workload of $I$ inserts and $E$ ExtractMins with $I \ge E$: which $d$ minimizes the total cost, what does the total become, and can you name a situation where the optimal $d$ is not a constant?

(c) Given a value $x$ and an integer $k$, decide whether $T$ contains at least $k$ keys smaller than $x$, in $O(k)$ time. Note that (F2) alone does not suffice, since $m$ may be far larger than $k$. Prove correctness and the bound, and say which step removes the dependence on $n$ and which removes the dependence on $m$.

(d) Prove that any algorithm that finds the **maximum** key of $T$ must read the key of *every* leaf, and conclude a bound of the form $\Omega(n)$. Is the bound tight?

## Problem 2. Selecting and merging

(a) You are given an unsorted array $A[1..n]$ of distinct numbers and an integer $k \le n$. Output the $k$ smallest elements of $A$ in sorted order. Give one algorithm running in $O(n + k\log n)$ and one running in $O(n\log k)$. Is either bound always at least as good as the other? Identify a resource other than time on which the two differ, and a setting in which only the slower one is usable.

(b) Now the input arrives as a *stream*: you see $A[1], A[2], \ldots$ once each, in order, $n$ is not known in advance, and you may store only $O(k)$ elements at a time. Give an algorithm that outputs the $k$ smallest elements in sorted order in $O(n + k\log k)$ total time, beating both bounds in (a). Prove the invariant your algorithm maintains and prove the time bound. A naive accounting gives something worse than $O(n)$ for the non-sorting part, so say carefully why the true cost is linear. You may assume a linear-time selection routine.

(c) You are given $k$ sorted lists containing $n$ elements in total (the lists need not have equal lengths). Merge them into one sorted list in $O(n\log k)$ time. Prove correctness and analyze the running

time. What does the obvious alternative, merging list 1 with list 2, then that result with list 3, and so on, cost in the worst case?

(d) *(challenge)* Show that $\Omega(n \log k)$ comparisons are necessary in the worst case for part (c), even when all $k$ lists have the same length $n/k$.