CS 574, Spring 2021
Timothy Chan

Homework 2 (due March 10 Wednesday 5pm (CT))

Instructions: You may work in groups of at most 3; submit one set of solutions per group. Always acknowledge any discussions you have with other people and any sources you have used (although most homework problems should be doable without using outside sources). In any case, *solutions must be written entirely in your own words.*

1. [*23 pts*] Let X be a set of n elements. Let $\mathcal{S}$ be a collection of dn subsets of X, each subset containing exactly c elements. Here, c is a constant (but d is not).

   We want to construct a large subset $A \subseteq X$ such that no set $S \in \mathcal{S}$ is completely contained in A.

   (a) [*7 pts*] Consider the following greedy strategy: pick an unmarked element x that is in the smallest number of sets of $\mathcal{S}$, add x to A, mark all elements y such that x and y lie in a common set of $\mathcal{S}$, and repeat until all elements are marked.

   Prove that this yields a subset $A \subseteq X$ of size $\Omega(n/d)$ such that no subset $S \in \mathcal{S}$ is completely contained in A. (Hidden constant factors in the Ω notation may depend on c.)

   (b) [*16 pts*] Give a randomized (Las Vegas) algorithm to construct a subset $A \subseteq X$ of size $\Omega(n/d^{1/(c-1)})$ such that no subset $S \in \mathcal{S}$ is completely contained in A.

   For example, for $c = 4$, the bound is $\Omega(n/d^{1/3})$, which is better than in part (a).

   Hint: start with a random sample...

   [*Note*: this result was recently applied to obtain an improved algorithm for a well known problem—I'll reveal the source afterwards...]

2. [*24 pts*] We are given an $n \times n$ matrix A where every row is in increasing order and every column is in increasing order (i.e., $A[i,j] < A[i,j+1]$ and $A[i,j] < A[i+1,j]$ for every i, j).

   (a) [*7 pts*] Given a value t, describe a deterministic algorithm for listing all elements in A that are at most t. The output does *not* have to be in sorted order. The running time should be $O(n + K)$, where K denotes the output size, i.e., the number of elements at most t.

   (b) [*17 pts*] Given k, describe a randomized Las Vegas algorithm for finding the first k smallest elements in A. The output does *not* have to be in sorted order. The expected running time should be $O(n + k)$. The number of random bits used should be $O(\log n)$.

   Hint: use part (a) and random sampling.

3. [*20 pts*] Given a set S of n points in 2D and a number k, we want to find the smallest circle C^* that encloses at least $n-k$ points of S (i.e., there are at most k points outside the circle).

   Describe a randomized algorithm (Monte Carlo is fine) for this problem that has expected running time of the form $O(nk^c)$ for some constant c.

   Hint: if we pick a random sample $R \subseteq S$ of size r, what is the probability that the minimum enclosing circle of R is equal to C^*?

   Note: constant error probability (say, 1/3) suffices. (For high probability bounds, an extra $\log n$ factor in the time bound would be acceptable...)

4. [*33 pts*] Given a set S of n disjoint line segments in 2D, consider the problem of computing the vertical decomposition $\mathrm{VD}(S)$ (also called trapezoidal decomposition), as studied in class. In this question, you will explore a different approach to solve the problem, based on random sampling instead of randomized incremental construction.

   (a) [*5 pts*] Let $R \subseteq S$ be a random subset of size r, chosen by sampling with replacement. For each trapezoid Δ, let S_Δ denote the subset of all line segments of S that intersects Δ. Prove that $\max_{\Delta \in \mathrm{VD}(R)} |S_\Delta| = O((n/r)\log n)$ with high probability (e.g., with probability at least $1 - 1/n$).

   (b) [*14 pts*] Consider the following divide-and-conquer approach to compute $\mathrm{VD}(S)$: pick a random sample $R \subseteq S$ of size r, naively compute $\mathrm{VD}(R)$ (e.g., by a quadratic-time algorithm), and recursively compute $\mathrm{VD}(S_\Delta)$ for each $\Delta \in \mathrm{VD}(R)$, and combine these vertical decompositions. Suppose we set $r = f(n)$ *for some suitably chosen, slow-growing function* $f(n)$.

   Provide the details and analyze the expected running time (which should be $o(n^{1+\varepsilon})$ for an arbitrarily constant $\varepsilon > 0$).

   (c) [*14 pts*] Next, consider a different approach where we pick a *large* value for r. More precisely, consider a generalized problem: the input is a set S of n line segments and a subset $T \subseteq S$ of size m, the goal is to compute $\mathrm{VD}(T)$ along with the sets S_Δ for each $\Delta \in \mathrm{VD}(T)$. The approach is: pick a random sample $R \subseteq T$ of size $r = m/2$, recursively compute $\mathrm{VD}(R)$ and the sets S_Δ for each $\Delta \in \mathrm{VD}(R)$, then naively compute $\mathrm{VD}(T_\Delta)$ for each $\Delta \in \mathrm{VD}(R)$, and combine answers.

   Provide the details and analyze the expected running time (which should be better than in part (b)).