CS 473, Fall 2017
Homework 5 (due Oct 18 Wednesday at 8pm)

You may work in a group of at most 3 students. Carefully read `http://engr.course.illinois.edu/cs473/policies.html` and `http://engr.course.illinois.edu/cs473/integrity.html`. One member of each group should submit via Gradescope.

1. [*17 pts*] In this problem, we will investigate a simpler family of hash functions that satisfies a weaker version of universality (with some extra logarithmic factors), but has other nicer properties useful for certain applications.

   Let m be a given integer. Let $p_1, \ldots, p_k$ be the list of all prime numbers at most m. You may assume that this list has been precomputed and you may use the known fact that $k = \Theta(m/\log m)$ (obtaining really tight bounds for k is the subject of the well-known "Prime Number Theorem").

   Pick a random index $j \in \{1, \ldots, k\}$ and define $h_j : \{0, 1, \ldots, U-1\} \to \{0, 1, \ldots, m-1\}$ by

   $$h_j(x) = x \bmod p_j.$$

   (a) [*7 pts*] For any fixed $x, y \in \{0, 1, \ldots, U-1\}$ with $x \neq y$, prove that $\Pr_j[h_j(x) = h_j(y)] \leq O(\frac{\log m \log U}{m})$.
   (Hint: can you upper-bound the number of distinct prime divisors that a number may have?)

   (b) [*10 pts*] Recall the following problem from Homework 1: Given three sets of integers A, B, and C with $|A|+|B|+|C| = n$, we want to decide whether there exist elements $a \in A$, $b \in B$, and $c \in C$ such that $c = a + b$. Prof. X claims to have discovered an $O(n^{1.99})$-time algorithm to solve the special case of the problem when $A, B, C \subseteq \{0, 1, \ldots, n^4\}$. Show how to use Prof. X's algorithm to solve the more general case of the problem when $A, B, C \subseteq \{0, 1, \ldots, n^{100}\}$ by a Monte Carlo $O(n^{1.99})$-time algorithm with error probability at most 1/4.
   (Hint: use (a). The property that $h_j(a) + h_j(b)$ is equal to $h_j(a+b)$ or $h_j(a+b) + p_j$ may be helpful. . .)

2. [*23 pts*] Consider the following geometric problem: given a set P of n points in 2D, with integer coordinates from $\{0, 1, \ldots, U-1\}$, find a *closest pair*—i.e., 2 points $p, q \in P$ $(p \neq q)$ such that the (Euclidean) distance between p and q is the smallest. We denote the distance of the closest pair by $\delta(P)$.

   An $O(n^2)$-time algorithm for this problem is trivial, and you can find an $O(n \log n)$-time divide-and-conquer algorithm in 2D in some textbooks. In this question, we give a different, faster randomized algorithm (which has the added advantage that it can be extended to higher dimensions and to other problems).

(a) [*10 pts*] First give an $O(n)$-expected-time (Las Vegas) algorithm for the easier *decision problem*: given a value r, decide whether $\delta(P) < r$.

(Hints: Build a uniform grid where each cell is an $(r/2) \times (r/2)$ square. Use hashing. How many points can a grid cell have? For each grid cell, how many grid cells are of distance at most r?)

(b) [*13 pts*] Now, consider the following recursive Las Vegas algorithm to compute $\delta(P)$:

Closest-Pair(P):

1. if $|P| \leq 100$ then return answer by brute force
2. partition P into subsets $P_1, \ldots, P_{20}$ each with at most $\lceil n/20 \rceil$ points
3. let $S = \{(i,j) \mid 1 \leq i < j \leq 20\}$
4. $r = \infty$
5. for each $(i,j) \in S$ *in random order* do
6.   if $\delta(P_i \cup P_j) < r$ then
7. $r =$ Closest-Pair($P_i \cup P_j$)
8. return r

Explain why the algorithm is always correct, and analyze its expected running time by solving a recurrence.

(Hints: Where is (a) used? What is the size of S? According to a result from class, how many times (in expectation) is line 7 performed?)