Assignment 2 — *Contemporary Algorithms, T. II/2019–20*
(due: fri feb 14 @ 11:59pm)

Ground Rules: Do all problems below. Solve them either by yourself or in teams. Typeset your answers and hand in a PDF file electronically to Canvas. You can look up things on the Internet; refrain from copying solutions straight-up.

Problem 1. *LSH for Euclidean Distance.* We'll further study an LSH family for Euclidean distance in $\mathbb{R}^d$.

(i) To warm up, consider the following problem: Fix $w > 0$ and draw $s \in [0, w]$ uniformly at random. Prove that if we define $f(x) = \lfloor \frac{1}{w}(x + s) \rfloor$, then

$$\mathbf{Pr}[f(x) = f(y)] = \max\left(0, 1 - \tfrac{1}{w}|x - y|\right).$$

(ii) We define a hash family $\mathcal{H}_r$ for $\mathbb{R}^d$ by describing a process to generate a (random) hash function from this family: To draw a random hash function h from $\mathcal{H}_r$, draw a random number $s \in [0, r]$ uniformly at random and form a vector $\mathbf{v} = \langle v_1, v_2, \ldots, v_d \rangle$ such that $v_i \sim N(0, 1)$. Then, $h \colon \mathbb{R}^d \to \mathbb{Z}$ is given by

$$h(\mathbf{x}) = \lfloor \tfrac{1}{r}(\mathbf{x}^\top \mathbf{v} + s) \rfloor$$

Show that for fixed r and c, the hash family $\mathcal{H}_r$ is (r, cr, p_1, p_2)-locality-sensitive by deriving p_1 and p_2. It is okay to leave the expressions for p_1 and p_2 in terms of definite integrals involving the Gaussian pdf; however, you'll want to show that $p_1 > p_2$.

Problem 2. *Dual Binary Search and Better Merge Sort.* Consider the implementation of `kth(A, B, k)` in the given handout. We will analyze its cost and use it to build merge sort with better span.

(i) Show that `kth(A, B, k)` runs in $O(\log |A| + \log |B|)$ work and span. (*Hint:* Can we guarantee that something goes down in half in each recursive call?)

(ii) We'll start by improving the span bound of `merge`. To merge two sorted sequences, we'll split them into $\sqrt{n}$ equal-sized pieces (previously, we split them into 2 equal-sized pieces). Describe how you would use the `kth` routine to accomplish this. Give a (pseudocode) implementation of the upgraded `merge`.

(iii) What is the work recurrence for our upgraded `merge`? How about `span`? (*Hint:* You should get $W(n) = \sqrt{n}W(\sqrt{n}) + \ldots$ and $S(n) = S(\sqrt{n}) + \ldots$.)

(iv) Let $f(n)$ be $o(n)$—this reads little-O of n. Solve the following recurrence: $W(n) = \sqrt{n}W(\sqrt{n}) + f(n)$. (*Hint:* $\Theta(n)$)

(v) Now that we know how to solve the recurrences, what are the work and span bounds for our upgraded `merge`?

(vi) If merge sort now uses the improved `merge` routine, what is its overall work and span?

Problem 3. *Quick Sort's Span.* In class, we made quick sort parallel by showing that both the partitioning and concatenation steps can be accomplished in $O(n)$ work and $O(\log n)$ span. Show that the span of this version of quick sort is $O(\log^2 n)$ whp. Notice that the work bound follows from the argument we did in the previous assignment.

Problem 4. *String Comparison.* Derive a parallel algorithm `compare(X, Y)` that lexicographically compares two given strings $X = x_1 x_2 \ldots x_n$ and $Y = y_1 y_2 \ldots y_m$ and returns the following result

$$\texttt{compare}(X, Y) = \begin{cases} -1 & \text{if } X < Y \\ 0 & \text{if } X = Y \\ +1 & \text{if } X > Y \end{cases}.$$

Your algorithm should run in $O(\min(n, m))$ work and $O(\log \min(n, m))$ span.

Problem 5. *Parallel Closest Pair.* In the sequential setting, the closest pair of points in a set of n 2-d points can be found in $O(n \log n)$ time using divide and conquer. Make this algorithm parallel so that it runs in $O(n \log n)$ work and at most $O(\log^3 n)$ span. (*Hint:* The points on the boundaries of the two sides of a recursive call can be examined in parallel. Also, we can prove that there is no point in looking at more than a constant number of points.)