CS2800 Design and Analysis of Algorithms
Problem Set #3 February 6, 2023

1. Given an $n \times n$ matrix A where each row and column is sorted in the increasing order and a number x, design an efficient algorithm to search if x is present in the matrix A. Prove the correctness and analyze the time complexity of your algorithm.

2. For an array A of length n with distinct elements, an *inversion* is a pair (i, j) such that $i < j$ and $A[i] > A[j]$.
   (a) Give an $O(n^2)$ algorithm to count the number of inversions in a array A of length n.
   (b) Suppose you are given an array $A[1, 2, \ldots, n]$ such that $A[1, 2, \ldots, n/2]$ and $A[n/2, n/2 + 1, \ldots, n]$ are sorted in the increasing order. Design an algorithm to sort the array A as well as count the number of inversions in the original array A.
   (c) Use the part above in a divide-and-conquer algorithm to obtain an $O(n \log n)$ algorithm for counting the number of inversions.

3. For a given set of points on the plane, there are other ways to define a notion of distance. For instance, the L_1-distance between two points (x_1, y_1) and (x_2, y_2) is defined as $|x_1 - x_2| + |y_1 - y_2|$. The L_∞-distance between (x_1, y_1) and (x_2, y_2) is defined as $\max(|x_1 - x_2|, |y_1 - y_2|)$. Describe how you will modify the algorithm for finding the closest pair of points that we saw in class so that it can now find the closest pair according to the L_1-distance and L_∞-distance. If you feel that the algorithms will work without any modifications, then justify that as well.

4. Suppose we try to solve the closest-pair problem in $2D$ in the following way: Sort the points in P according to the x and y coordinates separately and keep the arrays X and Y. Choose the point p that has the smallest x coordinate from the set P, delete it from P and recursively find the closest pair of points among the remaining $n - 1$ points. Once you have this, you try to find if p is involved in a closest pair. If the distance between the closest pair of points in $P \setminus \{p\}$ is δ, then we need to consider only the 2δ-sized square around p, like we saw in class. We now find the candidate points to which p is close to by doing a binary search on the sorted array Y. Once we find the position, we look at the $O(1)$ points before and after the correct location of p and calculate the closest pair of points. This should give a recurrence of the form $T(n) = T(n - 1) + O(\log n)$, thus giving a running time of $O(n \log n)$.

   Is this algorithmic idea correct? Justify your answer. If it is wrong, describe where the flaw is in the reasoning.

5. For any two sets X and Y of n integers, the Minkowski sum $X + Y$ is the set of sums $\{x + y \mid x \in X, y \in Y\}$.
   (a) Describe an algorithm to compute the cardinality of the set $X + Y$ in time $O(n^2 \log n)$
   (b) If M the largest absolute value of any element in $X \cup Y$, give an algorithm to compute the size of $X + Y$ in time $O(M \log M)$.

   Hint: Use convolution

6. For an array $A[0, 2, \ldots, n-1]$ of size $n = 2^k$, the *bit-reversal permutation* of A is defined as follows: Consider the $\log n$-bit representation of i. The bit-reversal permutation $\pi(i)$ is obtained by reversing the bit-representation of i. The bit-reversal permutation of $A[0, 2, \ldots, n-1]$ is $A[\pi(0), \pi(2), \ldots, \pi(n-1)]$. For example, the bit reversal permutation of $A[0], A[1], A[2], A[3]$ is $A[0], A[2], A[1], A[3]$.

   For an array A, design an $O(n)$-time algorithm to compute the bit-reversal permutation of A.