HW 10 (due Monday, 6pm, November 30, 2015)

NEW CS 473: Theory II, Fall 2015 Version: **1.2**

Collaboration Policy: This homework can be worked in groups of up to three students.
Submission is online on moodle.

---

1. (50 PTS.) Adding numbers.

In the following we work with multisets. For an element x in a multiset X, we denote by $\#_X(x)$ its ***multiplicity*** in X. For a multiset S, we denote by $\mathrm{set}(S)$ the set of distinct elements appearing in S. The ***size*** of a multiset S is the number of distinct elements in S (i.e., $|\mathrm{set}(S)|$. The ***cardinality*** of S, denoted by $\mathrm{card}(S) = \sum_{s \in S} \#_S(s)$.[1]

We use $[\![x : y]\!] = \{x, x+1, \ldots, y\}$ to denote the set of integers in the interval $[x, y]$. Similarly, $[\![\mathsf{U}]\!] = [\![1 : \mathsf{U}]\!]$.

Abusing notation, we denote by $S \cap [\![x : y]\!]$ the multiset resulting from removing from S all the elements that are outside $[\![x : y]\!]$. We denote that a multiset S has all its elements in the interval $[\![x : y]\!]$ by $S \subseteq [\![x : y]\!]$.

For two multisets X and Y, we denote by $X \oplus Y$ the multiset with the ground set being

$$\{x + y \mid x \in X \text{ and } y \in Y\},$$

where the multiplicity of $x + y$ is the number of different ways to get this sum (in particular, $x \in X$ and $y \in Y$ contribute the element $x + y$ with multiplicity $\#_X(x) \cdot \#_Y(y)$ to the resulting multiset).

For a multiset S of integers, let $\Sigma_S = \sum_{\alpha \in S} \#_S(\alpha) * \alpha$ denote the total ***sum*** of the elements of S. The multiset of all ***subset sums*** is

$$\sum(S) = \mathrm{set}\Big(\{\Sigma_T \mid T \subseteq S\}\Big).$$

Here, we would be interested in the subset sums up to U; that is $\sum_{\leq \mathsf{U}}(S) = \sum(S) \cap [\![0 : \mathsf{U}]\!]$. Formally, we want to compute $\mathrm{set}\Big(\sum_{\leq \mathsf{U}}(S)\Big)$.

(A) (5 PTS.) Let $S \subseteq [\![u]\!]$ be a multiset of cardinality n. Describe an algorithm that computes, in $O(n\mathsf{U})$ time, the set of subset sums up to U; that is, $\mathrm{set}\Big(\sum_{\leq \mathsf{U}}(S)\Big)$.

(B) (5 PTS.) Given two sets $S, T \subseteq [\![\mathsf{U}]\!]$, present an algorithm that computes $S \oplus T$ in $O(\mathsf{U} \log \mathsf{U})$ time.

(C) (5 PTS.) Given a multiset S of non-negative integers of cardinality n, present an algorithm that, in $O(n \log n)$ time, computes a multiset T, such that $\mathrm{set}(\sum(S)) = \mathrm{set}(\sum(T))$, and no number appears in T more than twice.

(D) (5 PTS.) Given a multiset $X \subseteq [\![\Delta]\!]$, of cardinality n, show how to compute the $\sum(X)$ in time $O(n\Delta \log(n\Delta) \log n)$.

(E) (5 PTS.) Given a multiset $S \subseteq [\![x : 2x]\!]$ of cardinality n', which is at most $\lfloor \mathsf{U}/x \rfloor$, show how to compute all possible subset sums of S, in time $O(\mathsf{U} \log \mathsf{U} \log n')$.

(F) (5 PTS.) Given a multiset $S \subseteq [\![x : 2x]\!]$ of cardinality $n \geq \mathsf{U}/x$, show how to compute all possible subset sums of S that are at most U (i.e., $\sum_{\leq \mathsf{U}}(S)$). The running time of your algorithm should be $O(nx \log^2 \mathsf{U})$. (Hint: Use (1E).)

(G) (10 PTS.) [Hard.] Given a set $S \subseteq [\![x : x + \ell]\!]$ of cardinality n, show an algorithm that compute all possible subset sums of multisets of cardinality at most t in S, in time $O(n\ell t \log(n\ell t) \log n)$.

(H) (10 PTS.) [Hard.] Given a multiset $S \subseteq [\![\mathsf{U}]\!]$ of cardinality n, show how to compute $\sum_{\leq \mathsf{U}}(S)$ in $O(\mathsf{U}\sqrt{n} \log^2 \mathsf{U})$ time. For credit, your solution has to be self contained, and use the above.
(Hint: Partition the set S into intervals $I_i = [\![a_i : b_i]\!]$, for $i = 1, \ldots t$ (for some t), and compute the subset sums of the sets $S_i = S \cap [\![a_i : b_i]\!]$ using the above. Then combine them into subsets sums of

[1] For more information about multisets, see wikipedia.

all the numbers of S, again using the above. Observe that for S_i we care only about subsets sums involving at most U/a_i terms (why?).)

In the above, you can assume that $n < \mathsf{U}$.

2. (25 PTS.) Sorting networks stuff

(A) (5 PTS.) Prove that an n-input sorting network must contain at least one comparator between the ith and $(i+1)$st lines for all $i = 1, 2, ..., n-1$.

(B) (10 PTS.) Prove that in a sorting network for n inputs, there must be at least $\Omega(n \log n)$ gates. For full credit, your answer should be short, and self contained (i.e., no reduction please).
[As an exercise, you should think why your proof does not imply that a regular sorting algorithm takes $\Omega(n \log n)$ time in the worst case.]

(C) (5 PTS.)
Suppose that we have $2n$ elements $\langle a_1, a_2, ..., a_{2n}\rangle$ and wish to partition them into the n smallest and the n largest. Prove that we can do this in constant additional depth after separately sorting $\langle a_1, a_2, ..., a_n\rangle$ and $\langle a_{n+1}, a_{n+2}, ..., a_{2n}\rangle$.

(D) (5 PTS.)
Let $S(k)$ be the depth of a sorting network with k inputs, and let $M(k)$ be the depth of a merging network with $2k$ inputs. Suppose that we have a sequence of n numbers to be sorted and we know that every number is within k positions of its correct position in the sorted order, which means that we need to move each number at most $(k-1)$ positions to sort the inputs. For example, in the sequence 3 2 1 4 5 8 7 6 9, every number is within 3 positions of its correct position. But in sequence 3 2 1 4 5 9 8 7 6, the number 9 and 6 are outside 3 positions of its correct position.
Show that we can sort the n numbers in depth $S(k) + 2M(k)$. (You need to prove your answer is correct.)

3. (25 PTS.) Computing Polynomials Quickly

In the following, assume that given two polynomials $p(x), q(x)$ of degree at most n, one can compute the polynomial remainder of $p(x) \bmod q(x)$ in $O(n \log n)$ time. The ***remainder*** of $r(x) = p(x) \bmod q(x)$ is the unique polynomial of degree smaller than this of $q(x)$, such that $p(x) = q(x) * d(x) + r(x)$, where $d(x)$ is a polynomial.

Let $p(x) = \sum_{i=0}^{n-1} a_i x^i$ be a given polynomial.

(A) (6 PTS.) Prove that $p(x) \bmod (x - z) = p(z)$, for all z.

(B) (6 PTS.) We want to evaluate $p(\cdot)$ on the points $x_0, x_1, \ldots, x_{n-1}$. Let

$$P_{ij}(x) = \prod_{k=i}^{j} (x - x_k)$$

and

$$Q_{ij}(x) = p(x) \bmod P_{ij}(x).$$

Observe that the degree of Q_{ij} is at most $j - i$.
Prove that, for all x, $Q_{kk}(x) = p(x_k)$ and $Q_{0,n-1}(x) = p(x)$.

(C) (6 PTS.) Prove that for $i \le k \le j$, we have

$$\forall x \quad Q_{ik}(x) = Q_{ij}(x) \bmod P_{ik}(x)$$

and

$$\forall x \quad Q_{kj}(x) = Q_{ij}(x) \bmod P_{kj}(x).$$

(D) (7 PTS.) Given an $O(n \log^2 n)$ time algorithm to evaluate $p(x_0), \ldots, p(x_{n-1})$. Here $x_0, \ldots, x_{n-1}$ are n given real numbers.