Algorithms and Data Structures
Spring 2018
Exercises for Units 15 and 16

1. Let $U = \{0, 1, \ldots, K-1\}$, let $p \geq K$ be a prime number, and let $0 < t < K$. For $0 \leq a, b < p$ define

$$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod t\,.$$

   Show that the family $\mathcal{H} = \{h_{a,b} \mid 0 < a < p, 0 \leq b < p\}$ is a universal set of hash functions from U to $T = \{0, \ldots, t-1\}$.

2. Let $\mathcal{H}$ be a set of functions from U to $T = \{0, \ldots, t-1\}$. We call $\mathcal{H}$ pairwise independent if for all distinct x and y in U and all $i, j \in T$ we have

$$\Pr\left(h(x) = i \text{ and } h(y) = j\right) \leq 1/t^2\,.$$

   (a) Show that if $\mathcal{H}$ is pairwise independent, then it is also universal.

   (b) What about the converse? Does universality of $\mathcal{H}$ also necessarily imply pairwise independence of $\mathcal{H}$?

   (c) Is the family $\mathcal{H}$ of question 1 also pairwise independent?

3. Consider the following two functions h_1 and h_2 defined in the following table:

	a	b	c	d	e	f	g
h_1	0	2	2	3	3	0	0
h_2	5	5	6	6	7	7	6

   For each of the sets S_i given below determine the number of ways in which S_i could be stored in the process of cuckoo hashing using a tables of size 8, and using the two hash functions h_1 and h_2. Justify your answers.

$$S_1 = \{a, c, e\} \quad S_2 = \{a, b, c, d\} \quad S_3 = \{a, b, c, d, e, f\} \quad S_4 = \{a, b, c, d, e, f, g\}$$

4. In class we discussed straightforward hollow heaps which allow to implement meldable priority queues with decrease-key needing just constant worst case time for all operations except for the operations Delete and MinDelete which need logarithmic amortized time. There is a disadvantage, however, in that the space usage is not linear in n, the number of items in the structure, but linear in M, the number of operations performed.

   Give a simple method that reduces the space usage, so that at all times the space used is proportional to the number of items in the structure, while all the time bounds mentioned above are maintained.

5. Is it conceivable that hollow heaps also allow an operations IncreaseKey that uses constant amortized time, while keeping the running times guarantees of all the other operations?

   If yes, how would you do it, if no, why is it impossible?

6. Hollow heaps need two node-pointers per node, namely one to its next sibbling and another one to the list of its children. Since every node stores at most one item, this means you always need space for at least $2n$ node-pointers, where n is the number of items in the heap.

   Can you develop a way of reducing the number of necessary node-pointers by keeping more than one item in every node, e.g. between k and $2k$ items for every "full" node?

   What invariants would you maintain? How would linking work? How would you implement all other operations? What running times could you achieve? And what would be the node-pointer space requirement?