

15-451/651 Algorithm Design & Analysis, Fall 2026

Homework #2

Homework #2, Due: Wednesday, September 9th at 11:59 pm

This homework has several short-answer problems followed by two written problems. The short-answer problems are graded on correctness only (we reserve the right to give a point penalty for providing a long answer to a short answer question). You **should not** provide long-winded justification or proofs unless explicitly asked for. Long-answer problems require you to justify your answers and provide proofs unless stated otherwise. On this homework, there is a SOLO problem and some GROUP problems. You may work in a group of up to 3 people on the GROUP problems. **In all cases the write-up you submit must be written by you alone. You may not share your written solutions with each other. You can discuss the problem as a group, but then must write up your final solutions independently in your own words.**

Submission: Solutions to the problems should be submitted as a PDF file using gradescope, with the answer to each problem starting on a new page. Each problem has a specified page limit that you should stick to. Your solutions **must** be typeset, **not handwritten**. You should use US letter-size paper, a font size no smaller than 10pt, margins no smaller than 1in. 10 points on this homework will be allocated to correctly following the style guidelines (see the Written HWs section of the [course policies](#)).

0 **Collaboration statement** Fill out the [collaboration form](#) to disclose your group members for this homework. If you worked solo, you must still fill out the form. Note that you will have to fill out this form separately for every written homework.

1. (SOLO, 15 pts) **Short-answer problems** (Limit: 1 page)

- (a) You want to sort one million 64-bit integers using Radix Sort, but you're not sure which base b to use. In lecture we said that using $b = n$ was optimal, but this only for big- O , and might not be perfect if we care about constant factors in practice.

In base b , each integer has $\lceil 64 \log_b 2 \rceil$ digits, and each digit triggers one pass of counting sort over the $n = 10^6$ integers plus b buckets. Assuming each of those cost one operation, write down a formula in terms of b approximating the number of operations that Radix Sort will perform on one-million 64-bit integers. You can make any assumptions about the constant factors that are reasonable. Find the value of b that minimizes the number of operations. How many iterations does Radix Sort perform for this b ?

- (b) Describe an explicit universal hash function family from $U = \{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{0, 1\}$. Hint: you can do this with a set of 4 functions.
- (c) Say we have four keys—a, b, c, and d—which we want to hash into two locations, 0 and 1. We have a family of five hash functions— f , g , h , i , and j .

x	$f(x)$	$g(x)$	$h(x)$	$i(x)$	$j(x)$
a	0	0	0	0	0
b	1	1	0	1	0
c	1	0	1	1	1
d	0	1	1	1	0

Is this family universal? Either prove that it is universal, or provide a succinct counterexample if it is not.

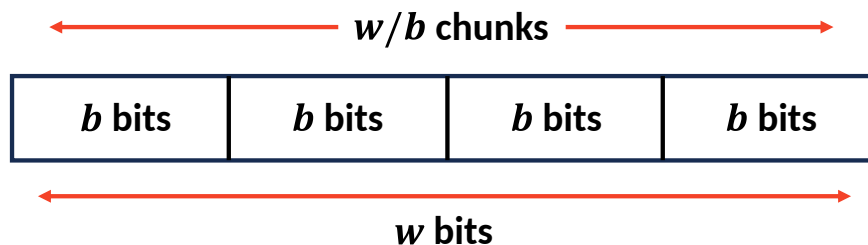
2. (GROUP, 25 pts) **Sacrifice stability, save space** (Limit: 2 pages)

In lecture we presented an implementation of counting sort that sorts n items with integer keys in the range $0 \dots u-1$ in $O(n+u)$ time and $O(n+u)$ space. Our implementation returns a new copy of the sorted array rather than sorting the input array directly. Suppose we wanted to instead sort the elements directly in the input array. One way to do this would be to swap the elements out into a temporary array and then swap them back in sorted order, but this still uses the same $\Theta(n)$ additional words of memory. We want to design a version of the algorithm that uses very little additional space.

- (a) (5 points) Let c_x be the number of elements with key less than x . Prove that an element with key x sitting at index j is in a correct sorted position iff $c_x \leq j < c_{x+1}$
- (b) (20 points) Using the fact you just proved, design an algorithm that performs counting sort while only using $\Theta(u)$ additional words of memory. This algorithm can be unstable. As a hint, note that such an algorithm will essentially need to swap the elements around the input array in place. Your algorithm will need a way to determine the correct locations to swap elements to, and what order to do it in so that the correct output is achieved.

3. (GROUP, 25 pts) **A table-based hashing scheme** (Limit: 2 page)

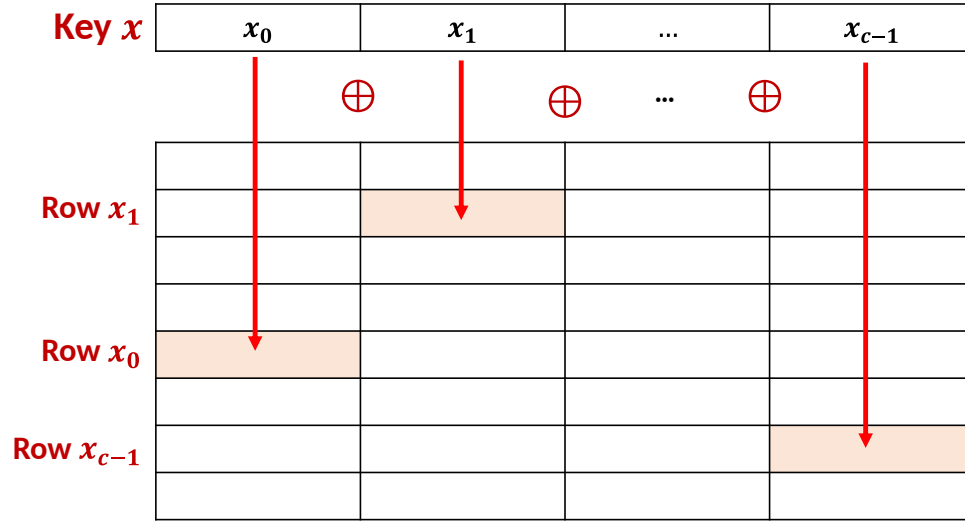
In this problem we will analyze a very nice and powerful hash family. Consider the universe of possible keys $U = \{0, 1, \dots, 2^w - 1\}$ and suppose we want to hash into the range $\{0, \dots, 2^b - 1\}$, i.e., the table size is $m = 2^b$, a power of two. To make things nice, let's assume that b is a divisor of w , so we can represent a key as a sequence of $c = w/b$ chunks of b bits (another way to say this is that you can write the key in base- m). Call the chunks $\langle x_0, x_1, \dots, x_{c-1} \rangle$, where for each i , we have $0 \leq x_i < m$.



We can derive a hash family as follows. We will generate a two-dimensional table R with exactly $c = w/b$ columns and $m = 2^b$ rows. Each cell of the table contains an independent uniformly random b -bit integer. The hash is computed by taking the XOR of the values of $R[x_i][i]$ for each i , i.e.:

$$h(x) = \bigoplus_{i=0}^{c-1} R[x_i][i]$$

In picture form:



In other words, for each chunk x_i , we look at row x_i and column i of the random table, then XOR all of these values together. Call the resulting family of hash functions, i.e., the set of all possible hash functions that could be generated by this construction $\mathcal{H}$.

- (a) (20 pts) In this part we will prove that $\mathcal{H}$ is three-wise independent. Let x, y, z be three arbitrary distinct keys from U , and let $v_1, v_2, v_3 \in \{0, \dots, m-1\}$ be arbitrary but not necessarily distinct values. Our goal is to prove that

$$\Pr_{h \in \mathcal{H}} [h(x) = v_1 \text{ and } h(y) = v_2 \text{ and } h(z) = v_3] = \frac{1}{m^3}$$

Recall that in the proof that the random binary matrix method was universal, we used the trick of isolating a uniformly random piece of the hash that was guaranteed to be different for x and y . Similarly, in Recitation 2 we saw a more general trick for proving pairwise independence by arguing that we can isolate two independent sources of randomness, one of which randomizes $h(x)$, and the other which randomizes $h(y)$ while being independent of $h(x)$. You need to do something similar here, but need to handle more cases since there are three keys x, y, z . Hint: Consider whether or not there is some column i^* that separates all three keys. If not, can you still find a column to separate pairs of the keys?

- (b) (5 pts) Prove that $\mathcal{H}$ is **not** four-wise independent. (Hint: A single counter-example is enough).