Exercises

Exercises should be completed **on your own**.

1. **(5 pt.)** Recall from lecture that `radix_sort` runs in $O(d(n+k))$-time if the stable sort it uses runs in $O(n+k)$-time. For this problem, assume that `radix_sort` uses `bucket_sort`, which does in fact run in $O(n+k)$-time.

   (a) (1 pt.) Suppose we want to use `radix_sort` to sort a list of lowercase words W in alphabetical order. All words are padded with *space* characters (assume *space* comes before 'a' alphabetically) to make them the same length. For the following W, describe what the three variables d, n, and k refer to (your explanation should include some reference to W or its elements) and what their values would be for this problem. (For example, if the runtime complexity formula included a variable x that referred to the length of the first word in the list of items being sorted, you might say "x is the length of the first word in the list, and is equal to 3 for list W.")

   $$W = \texttt{[the, quick, brown, fox, jumps, over, the, lazy, dog]}$$

   [We are expecting: Values and descriptions for d, n, and k.]

   (b) (1 pt.) Now suppose we convert each of the strings in W to their ASCII representations (8-bit binary strings, with *space* mapping to "00100000" as normal, so the word "the" becomes 40 digits long—8 digits per character plus 8 digits per padding space to make it as long as the longest word in W). We still want to use `radix_sort`, and we want to treat these bit strings as literal strings (i.e., do not try to interpret the 8-bit strings as decimal numbers). What are the values for d, n, and k?

   [We are expecting: Values and descriptions for d, n, and k.]

   (c) (1 pt.) Now we're back to using the character strings from part (a), but you happen to have the date (day, month, year) that each word was first published in an English dictionary. You want to sort first by date, then use alphabetical ordering to break ties. You will do this by converting each of the original words in W into words with date information (digits) prepended, appended, or inserted somewhere in the string. (Assume the digits 0-9 come before the *space* character and a-z). Write the string you would use to represent the word "jumps" (first published November 19, 1562) so that it will be correctly sorted by `radix_sort` for the given objective.

   [We are expecting: A string.]

   (d) (1 pt.) You decide that because you only ever use the words in a certain list V in everyday speech, you would like to save space and simply represent the first word in V with the binary value '0', the second word with '1', the third with '10', etc., continuing to increment by one in binary (and no longer including date information). All subsequent occurrences of a particular word w receive the same binary assignment as the first occurrence of w, and all strings are padded with '0's to make them equal length. V has n words in it, where $n > 2$. Give the time complexity of `radix_sort` on the list V with all words converted to their 0-padded binary strings and explain (informally) why that is correct. Simplify your answer as much as possible where values of d, n, or k are known.

   [We are expecting: A runtime, and a brief justification.]

   (e) (1 pt.) Not wanting to mess with binary conversions, you decide instead to represent the words in your vocabulary V with "one-hot" vectors (vectors of length n with all '0's except for a single '1' in a position corresponding to a particular word. For example, in W, the word "the" would be

represented as '10000000', since there are eight unique words in the list). Give the new worst-case time complexity of `radix_sort` on the list V, again simplifying as much as possible and explaining (informally) why that is the correct complexity.

[We are expecting: A runtime, and a brief justification.]

2. **(2 pt.)** The aptly-named Average Hotel has 100 rooms, each belonging to one of 100 guests. After an evening soiree, all of the guests (not thinking straight) randomly select a room to sleep in for that night. Multiple guests might end up in the same room.

   (a) (1 pt.) What is the expected number of guests that end up returning to their own hotel room?

   [We are expecting: A number, and please show your work.]

   (b) (1 pt.) What is the expected number of guests that end up in a room with exactly one other person? (Hint: you may find it easier to count by rooms instead of guests.)

   [We are expecting a mathematical expression like $(6)(8)$ **or a number like** 48**, and please show your work.]**

3. **(3 pt.)** Let U_k be the universe of all strings consisting of k numeric digits. (`0000`, `0123`, and `9898` are part of universe U_4 but `b000`, `012`, `9!9!` are not.) Let u_i denote the i^{th} digit of a string $u \in U_k$ where $0 \leq i < k$, so u_0 is `0` and u_1 is `2` for the string `0246`.

   Let H_k be a family of k hash functions mapping universe U_k to values $\{0, 1, 2, \ldots, 9\}$ where $h_0 \in H_k$ hashes all strings according to their first digit. (For all strings u where $u_0 = 0$, $h_0(u) = 0$; for all strings u where $u_0 = 1$, $h_0(u) = 1$; for all strings u where $u_0 = 9$, $h_0(u) = 9$.) Likewise, $h_1 \in H_k$ hashes all strings according to their second digit. Generally, for all strings $u \in U_k$ where $u_i = x$, $h_i(u) = x$ for $0 \leq i < k$.

   (a) (1 pt.) What is an example of a maximally-sized subset of U_3 such that H_3 is universal for the subset?

   [We are expecting: An example subset.]

   (b) (2 pt.) Would H_k be a good family of hash functions (where "good" is defined as universal) to use for U_k for $k \geq 3$?

   [We are expecting: A short explanation (2-3 sentences) that answers the question.]

Problems

You can collaborate with your classmates about the problems. However:

- Try the problems on your own *before* collaborating.
- Write up your solutions yourself, in your own words. You should never share your typed-up solutions with your collaborators.
- If you collaborated, list the names of the students you collaborated with at the beginning of each problem.

---

1. **(Quick password recovery) (8 pt.)** Secret Santa has stored n password-protected files on his computer, each with a unique password. He's written down all of these n passwords, but he doesn't know which password unlocks which file. He's put these files into an array F and their passwords into an array P in an arbitrary order (so $P[i]$ does not necessarily unlock $F[i]$). If he tests password $P[i]$ on file $F[j]$, one of three things will happen:

   (a) $P[i]$ unlocks $F[j]$

   (b) The computer tells him that $P[i]$ is lexicographically smaller than $F[j]$'s true password

   (c) The computer tells him that $P[i]$ is lexicographically greater than $F[j]$'s true password

   Secret Santa **cannot** tests whether a password is lexicographically smaller or greater than another password, and he **cannot** test whether a file's password is lexicographically smaller or greater than another file's password.

   (a) (3 pt.) Design a randomized algorithm to match each file to its password, which runs in expected $O(n \log(n))$-time.

   [We are expecting: Either an English description or pseudocode of the algorithm, and an English justification of why it takes expected $O(n \log(n))$-time.]

   (b) (3 pt.) Prove formally, using induction, that your answer to part (a) is correct.

   [[We are expecting: A formal argument by induction. Make sure you explicitly state the inductive hypothesis, base case, inductive step, and conclusion.]]

   (c) (2 pt.) Prove formally that your answer to part (a) runs in expected $O(n \log(n))$-time.

   [We are expecting: A formal analysis of the runtime.]

2. **(Plagiarism detection) (5 pt.)** Hash functions are extremely good at what they do. Unsurprisingly, there are many fancier data structures that can be built on top of them. In this problem we will motivate and explore the idea of a "Bloom Filter," which is one example of a fancier structure built on top of hash functions.

   Suppose you are hired by someone to make a plagiarism detection software for internal use so as to avoid any potentially embarrassing allegations of plagiarism. Specifically, your goal is to make a lightweight (i.e. fast, and relatively low-memory) piece of software that will take a sentence and output one of the following messages: 1) "potentially problematic, please rewrite", or 2) "fresh like an ocean breeze." Suppose your goal is the following: if the input sentence is something that you have already seen, you output "potentially problematic" (with probability 1), and if the input is something new, you want to output "fresh" with probability at least 0.99 (its alright if you have a few false-alarms).

(a) (1 pt.) First, you decide to use a hash table. You will make a has table that maps a piece of text to a bucket, then scrape the web for all English sentences, and hash each one to your table. Given a new sentence, you will check to see if it hashes to an empty bucket—if so, you will output option "fresh" otherwise you will output "potential plagiarism." Suppose there are 1 billion unique sentences online. How many buckets will your hash table need to have to have the desired functionality?

[We are expecting: A number (to the nearest order of magnitude) and one to two sentences of justification.]

(b) (2 pt.) You decide that is a little too much space usage, and consider the following approach: you choose 10 hash functions, $h_1, \ldots, h_{10}$ that each map sentences to the numbers 1 though 10 billion. You initialize an array A of 10 billion bits, initially set to 0. For each sentence s that you encounter, you compute $h_1(s), h_2(s), \ldots, h_{10}(s)$, and set the corresponding indices of A to be 1 (namely you set $A[h_1(s)] \leftarrow 1, A[h_2(s)] \leftarrow 1, \ldots$). Argue that after processing the 1 billion unique sentences, you expect a $(1 - 1/(10 \text{ billion}))^{10 \text{ billion}} \approx 0.37$ fraction of the elements to be 0.

For this part, feel free to assume that the h_i are "idealized" hash functions that map each key s to a uniformly random bucket.

[We are expecting: A paragraph with your argument.]

(c) (2 pt.) Now, given a sentence s, to check if it might be plagiarized, you compute the 10 hashes of s, and check if $A[h_1(s)] = A[h_2(s)] = \ldots = A[h_{10}(s)] = 1$. If so, you output "potential problem," otherwise you output "fresh." Prove that if s is actually in your set of 1 Billion sentences, that you will output "potential problem" with probability 1, and that if s is *not* in your set of 1 Billion sentences, you will output "fresh" with probability $\approx 1 - (1 - 0.37)^{10} \approx 0.99$.

Again, feel free to assume that the hash functions are "idealized," and that the claim of the previous part holds, namely that after processing the 1 Billion sentences, there are 3.7 billion indices in the array A with value 0.

[We are expecting: Informal mathematical justifications for each of the bounds.]