CS1200: Intro. to Algorithms and their Limitations   Prof. Anurag Anshu

Problem Set 4

Harvard SEAS - Spring 2026   *Due: Thu Mar. 5, 2026 (11:59pm)*

Please see the syllabus for the full collaboration and generative AI policy, as well as information on grading, late days, and revisions.

All sources of ideas, including (but not restricted to) any collaborators, AI tools, people outside of the course, websites, ARC tutors, and textbooks other than Hesterberg–Vadhan must be listed on your submitted homework along with a brief description of how they influenced your work. You need not cite core course resources, which are lectures, the Hesterberg–Vadhan textbook, sections, SREs, problem sets and solutions sets from earlier in the semester. If you use any concepts, terminology, or problem-solving approaches not covered in the course material by that point in the semester, you must describe the source of that idea. If you credit an AI tool for a particular idea, then you should also provide a primary source that corroborates it. Github Copilot and similar tools should be turned off when working on programming assignments.

If you did not have any collaborators or external resources, please write 'none.' Please remember to select pages when you submit on Gradescope. A problem set on the border between two letter grades cannot be rounded up if pages are not selected OR collaborators not noted.

Your name:
Collaborators and External Resources:
No. of late days used on previous psets:
No. of late days used after including this pset:

1. (Randomized vs. Deterministic Hash Tables)

   (a) In the Github repository, we have given you a partial implementation of a `HashTable` data structure for the dynamic DICTIONARY abstract data type. When you initialize an object of the `HashTable` class, you specify the universe $[U]$ for keys, the size m of the hash table, a class of hash functions, and a boolean flag `lazing` to indicate whether to use certain optimization attempt described below. We have given you two classes of hash functions `DeterministicHash` and `RandomHash`; the first implements the common $h(x) = x \bmod m$ and the second implements $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m)$, where $p \geq U$ is a prime number and a and b are numbers in $\{0, 1, \ldots, p-1\}$ chosen uniformly at random at time at initialization.

   When the `lazing` flag is turned on, `HashTable` has our `search` and `delete` implementations always return or delete the key-value pair at the head of the linked list at the $h(K)$'th array entry. This is to save the time of searching the linked list. Unfortunately, as you will see in experiments below, this sometimes leads to erroneous results.

   Implement `HashTable` for the case when `lazing` is turned off, ensuring that it provides an always-correct DICTIONARY data structure.

   (b) We have also given you testing code (experiments.py) that carries out 10,000 operations on a `HashTable` data structure, maintaining a (fake) employee dataset consisting of

pairs $(\texttt{employeeid}, \texttt{salary})$, where both $\texttt{employeeid}$ and $\texttt{salary}$ are given as integers in $[10, 000, 000]$.

The testing code runs experiments with 100 trials for each combination of the following parameters:

- Whether we use $\texttt{employeeid}$ or $\texttt{salary}$ as the key.
- The size m of the hash table, with $m = [1, 10, 100, 1000]$.
- Whether or not $\texttt{lazing}$ is turned on.
- Whether we use $\texttt{DeterministicHash}$ or $\texttt{RandomHash}$.

Note that to run the experiments you simply need to run the experiments.py file and not make any changes to it.

Using the results of these experiments, include your graphs and discuss the following:

i. Were any combinations of parameters particularly slow in an answering individual queries? Why do you think that was the case?

ii. Were any combinations of parameters prone to errors, and how frequently? Why do you think that was the case? For each combination of $\texttt{lazing}$ and choice of the family of hash functions, classify the resulting algorithm as being a correct deterministic algorithm, a correct Las Vegas algorithm, a correct Monte Carlo algorithm, or an incorrect algorithm.

iii. What are benefits or costs you see to using randomization for hash tables?

iv. Are the runtimes you observed consistent with the $O(1 + n/m)$ time bound for hash tables discussed in class?

2. (Weighted Medians) A local jazz club wants to determine the median age of the people that attend its shows. It has a dataset consisting of each customer's age and the number of club visits they have made during the past 5 years. When calculating the median, they want to put more weight on the customers that visit the club more often (since they have a bigger effect on the ages present in the club). This leads to the following computational problem:

Input: An array A of key-value-weight triples $((K_0, V_0, W_0), \ldots, (K_{n-1}, V_{n-1}, W_{n-1}))$, where $K_i, W_i \in \mathbb{N}$, and a threshold $t \in \mathbb{N}$

Output: A key-value-weight triple (K_i, V_i, W_i) such that $\sum_{j:K_j < K_i} W_j \leq t$ and $\sum_{j:K_j > K_i} W_j \leq W - 1 - t$, where $W = \sum_{j=0}^{n-1} W_j$

Computational Problem WEIGHTED SELECTION

For motivating application, consider the keys to be the customer ages, the weights to be the number of times they visited the club, and $t = W/2$. Write out (in pseudocode) a modification of $\texttt{QuickSelect}$ that solves the WEIGHTED SELECTION problem. Note that unlike SELECTION, where there is always a solution when the rank $i \in [n]$, there may be cases where WEIGHTED SELECTION has no solution and your algorithm should indicate so with $\bot$. Prove the correctness of your algorithm, then informally justify why the worse-case expected runtime of $O(n)$ derived from lecture is still valid for this modification.

3. (Choosing Algorithms and Data Structures) Suppose the US Census Bureau was going to develop a new database to keep track of the exact ages of the entire US population, and publish statistics on it. The data it has on each person is an exact birthdate `bday` (year, month, and date) and a unique identifier `id` (e.g. social security number — pretend that these are assigned at birth).

For each of the three (optionally, four) scenarios below,

(i) Select the best algorithm or data structure for the Census Bureau to use from among the following:

- sorting and storing the sorted dataset
- storing in a binary search tree (balanced and possibly augmented)
- storing in a hash table
- running randomized QuickSelect.

(ii) Explain how you would use the algorithm or data structure (including any necessary augmentations) to solve the stated problem, e.g. what would you take as keys and values, what updates and queries (in case you use a dynamic data structure) would you issue, and how you would read off the results to obtain the desired statistics. If you are using an augmented data structure, specify the augmentation you are making and if you are using a hash table, specify its size.

(iii) State what the runtime would be as a function of all of the relevant parameters: the size n of the US population being surveyed, the number u of updates issued at the specified time intervals, and/or the number s of statistics released at the specified time intervals. (These parameters are not all freely varying in the parts below, e.g. s may be a fixed constant or a function of n; state any such constraints in your answers.)

In each scenario, you should assume (unrealistically) that the described queries or statistics are the *only* way in which the data is going to be used, so there is no need to support anything else. If you find that only `id` or `bday` is relevant or necessary for some part, you can feel free to ignore the other component in your data structure.

<div style="border:1px solid">

Example (Decennial Quartiles): Every ten years as part of the Decennial Census, the Bureau obtains a fresh list of $(\texttt{id}, \texttt{bday})$ pairs for the entire US population and wishes to release the 25th, 50th, and 75th percentiles of the population ages (to the day).

Solution:

(i) **Recommendation:** I would recommend that the Census Bureau use `QuickSelect`. I guess that a large enough of a fraction of the US population changes every 10 years that processing the updates individually would be more expensive than doing batch computations.

(ii) **How I would use my data structure/ algorithm:** They should use the `bday`'s as keys; the `id`'s are not needed because we only want to release age percentiles. Each time they should run `QuickSelect` with ranks $i = n/4, n/2, 3n/4$, where n is the size of the population at the time.

(iii) **Runtime:** We are calculating $s = 3$ statistics every 10 years, and each execution of `QuickSelect` takes expected time $O(n)$, for a total runtime of $s \cdot O(n) = O(n)$.

</div>

(a) (Reporting Age Rankings) Every ten years as part of the Decennial Census, the Bureau collects a fresh list of $(\texttt{id}, \texttt{bday})$ pairs from the entire US population. (It does not reuse data from the previous Decennial Census, so everyone is re-surveyed.) In order to incentivize participation, the Bureau promises to tell every respondent their age-ranking in the population after the survey is done (e.g. "you are the 796,421'th oldest person among those who responded to the Census"). If two people have the same birthday, its ok if their age-ranking is arbitrarily ranked with respect to each other.

(b) (Daily Quartiles) After each day, the Bureau obtains a list of data (given as a list of $(\texttt{id}, \texttt{bday})$ pairs) to add to or remove from its database due to births, deaths, and immigration, and publishes an updated 25th, 50th, and 75th percentile of the population ages.

(c) (Age Lookups) For privacy reasons, the Bureau decides to not publish any statistics on the population ages, but just wants to maintain a database where the age of any member of the population can be looked up quickly, and the database can be quickly updated daily according to births, deaths, and immigration.

(d) (Daily Quartiles, optional[1]) The Bureau receives a daily list of updates and publish the 25th, 50th, and 75th percentile of population ages, similar to (b). This time, however, when an update deletes someone's data (i.e. death, emigration), the Bureau only receives their `id`. Incoming data (i.e. birth, immigration) is still in the format $(\texttt{id}, \texttt{bday})$.

4. (Reflection) Skim the course material from the beginning of the course through Lecture 10 (i.e. the scope of the upcoming class midterm). Identify one concept or skill that you would like to study or practice in greater depth, and discuss why. It can be because you feel that

---

[1]This problem won't make a difference between N, L, R-, and R grades.

you haven't fully understood or internalized it, or because you found it interesting and are curious to learn more, or any other motivation you have.

Quick note on grading: Good responses are usually about a paragraph, with something like 7 or 8 sentences. Most importantly, please make sure your answer is specific to this class and your experiences in it. If your answer could have been edited lightly to apply to another class at Harvard, points will be taken off.

Note: As with the previous psets, you may include your answer in your PDF submission, but the answer should ultimately go into a separate Gradescope submission form.

5. Once you're done with this problem set, please fill out this survey so that we can gather students' thoughts on the problem set, and the class in general. It's not required, but we really appreciate all responses!