CS 161
Summer 2018
7/12/18

Tutorial 3: Linear-Time Sorting
Randomized Algorithms

Linear-Time Sorting

1. **(Fun with radix sort) (Difficulty: Easy)**

   Suppose you want to alphabetize the following list of four letter words in ascending a-z alphabetical order with radix sort: `[byte, pits, bits, pins]`. Recall that radix sort calls bucket sort as a subroutine.

   [We are expecting: A list consisting of four elements for each of the parts.]

   (a) After the first call to bucket sort, what is the order of the list?

   (b) After the second call to bucket sort, what is the order of the list?

   (c) After the third call to bucket sort, what is the order of the list?

   (d) After the fourth call to bucket sort, what is the order of the list?

2. **(20-questions?) (Difficulty: Hard)**

   Suppose you want to sort an array A of n numbers (not necessarily distinct), and you are guaranteed that all the numbers in the array are in the set $\{1, \ldots, k\}$. A **"20-question sorting algorithm"** is any deterministic algorithm that asks a series of YES/NO questions (not necessarily 20 of them, that's just a name) about A, and then *writes down* the elements of A in sorted order. (Specifically, the algorithm does not need to rearrange the elements of A, it can just write down the sorted numbers in a separate location).

   Note that there are many YES/NO questions beyond just comparison-questions— for example, the following are also valid YES/NO questions: "If I ignored A[3] and A[17] would the array be sorted?" and "Did it rain today?"

   (a) Describe a 20-question sorting algorithm that, for every input, asks only $O(k \log n)$ questions. For now, assume that the algorithm accepts n and k as input.

   [We are expecting: A description of the algorithm and a brief justification for its runtime.]

   (b) Now suppose the algorithm does not accept n or k as input. Describe an algorithm that determines these values and still asks only $O(k \log n)$ questions.

   [We are expecting: A description of the algorithm and a brief justification for its runtime.]

(c) Prove that for *every* 20-question sorting algorithm, there exists some array A consisting of n integers between 1 and k that will require $\Omega(k \log \frac{n}{k})$ questions, provided $k \leq n$.

[We are expecting: A mathematically rigorous proof (which does NOT necessarily mean something long and tedious).]

Randomized Algorithms

1. **(Big Problem) (Difficulty: Easy)**

   Suppose not-so-many years from now, you're a fancy boss-person at a large tech company. Your company relies on solving `BigProblem` quickly—specifically, after rigorous market testing, you determine that if you solve the problem in less than 10 seconds, all is well. If, however, it takes more than 10 seconds, then your users go away. For example, maybe `BigProblem` is the computational problem of matching a user to an Uber driver–if it takes more than 10 seconds, the user will have already closed the app and tried their luck with Lyft. One day, an engineer tells you that they have a randomized algorithm that correctly solves `BigProblem` and has the property that, for every input, the expected runtime is 1 second. Assume throughout this problem that you know nothing else about the engineers implementation other than the expected runtime.

   (a) Prove that the probability that the randomized algorithm takes more than 10 seconds to run is at most $1/10 = 0.1$. Feel free to use Markov's inequality, which states that for any non-negative random variable, X, and any positive constant c, $Pr[X > cE[X]] < 1/c$.

   [We are expecting: A short proof (1 sentence) using Markov's inequality.]

   (b) Let `BigProblem1()` be the engineer's original implementation. You propose an approach to improve the above bound, expressed via the following pseudocode:

```
BigProblem2(input x):
  run BigProblem1(x)
  if the execution has not terminated after 8 seconds
    abort the execution
    run BigProblem1(x) again (from scratch)
  else:
    return the output of the original execution.
```

   Prove that the probability that your improved implementation, `BigProblem2()`, does not terminate within 10 seconds is at most $1/16 < 0.07$. (Progress!!) Assume successive executions of `BigProblem1()` are independent.

   [We are expecting: A short proof (1 sentence) using rules of probability and Markov's inequality.]

2. **(Colinear points) (Difficulty: Medium)**

You are given n distinct ordered pairs of integers $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where for all i, j, $x_i \neq x_j$ and $y_i \neq y_j$. Recall two points uniquely define a line $y = mx + b$, with slope m and intercept b. We say a set of points S is **collinear** if they all fall on the same line; that is, for all $(x_i, y_i) \in S$, $y_i = mx_i + b$ for some fixed m and b. You need to find the maximum cardinality subset of the given points A which are collinear. Assume that, given two points, you can compute the corresponding m and b for the line passing through them in constant time, and you can compare two slopes or two intercepts in constant time. Your algorithms should not use any form of hashing.

(a) Design an algorithm to find a maximum cardinality set of collinear points in $O(n^2 \log n)$-time. If there are several maximal sets, your algorithm can output any such set.

[We are expecting: An English description of your algorithm and a brief justification of its runtime.]

(b) It is not known whether we can solve this collinear points problem in under $O(n^2)$ time. But suppose we know that our maximum cardinality set of collinear points consists of exactly n/k points for some constant k. Design a randomized algorithm that reports the points in some maximum cardinality set in expected $O(n)$-time. (Hint: your running time may also be expressed as $O(k^2 n)$). Briefly justify the correctness and runtime of the algorithm.

[We are expecting: An English description of your algorithm and a brief justification of its expected runtime.]

(c) What is the worst-case runtime of your algorithm from part (b)?

[We are expecting: A runtime expressed in Big-O, with a short explanation.]

Hash Functions

1. **(Fun with hash functions) (Difficulty: Easy)**

   Consider the universe $\mathcal{U}$ of alphabetic strings of exactly length 8 (`teardown`, `meetings`, and `abcdefgh` are part of this universe, but `te@rdown`, `meeting`, and `12345678` are not part of this universe). Let $\mathcal{H}$ be the exhaustive set of hash functions mapping this particular universe to the buckets $\{1, \ldots, n\}$.

   [We are expecting: Short answers for each of the parts.]

   (a) What is the size of the universe $|\mathcal{U}|$?

   (b) How many hash functions are in $\mathcal{H}$?

   (c) How many bits are required to write down the name of one hash function from $\mathcal{H}$?

   (d) Is $\mathcal{H}$ a universal hash family?

   (e) Suppose we draw hash function h uniformly at random from $\mathcal{H}$. What is the probability that $h(\texttt{teardown}) = h(\texttt{teardown})$?

   (f) For the same hash function h, what is the probability that $h(\texttt{meetings}) = h(\texttt{teardown})$?

   (g) Now suppose we draw two hash functions h_1 and h_2 uniformly at random from $\mathcal{H}$. What is the probability that $h_1(\texttt{teardown}) = h_2(\texttt{teardown})$?

2. **(An exhausting set of hash functions) (Difficulty: Medium)**

   In this problem, we'll investigate the definition of universal hash functions. Let $\mathcal{U}$ denote a universe of size M, and let n be the number of buckets in a hash table.

   (a) Let $\mathcal{H}$ be the family of all possible functions mapping from $\mathcal{U}$ to $\{1, ..., n\}$. Prove that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h randomly chosen from $\mathcal{H}$,

   $$\Pr[h(x_i) = h(x_j)] = \frac{1}{n}.$$

   This shows that $\mathcal{H}$ is a universal hash family, and moreover that we have *equality* in the definition of the universal hash family property, not just a $\leq$ relationship.

   [We are expecting: A careful and rigorous proof, though it should not need to be more than one or two paragraphs. You should be especially careful about what is random and what is not.]

(b) There also exist hash families such that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h randomly drawn from the family, $\Pr[h(x_i) = h(x_j)] < \frac{1}{n}$. (Notice that the inequality here is strict!) We will now explore one such family. Consider $\mathcal{H}' = \mathcal{H} \setminus \{h_1\}$, where h_1 is the function defined by

$$h_1(x_i) = 1 \qquad \text{for all} \qquad x_i \in \mathcal{U}.$$

That is, $\mathcal{H}'$ is the family of all functions from $\mathcal{U}$ to $\{1, ..., n\}$ *except* for the function h_1 which sends all elements of $\mathcal{U}$ to 1.

Prove that, for all $x_i \neq x_j$ in $\mathcal{U}$, for h drawn randomly from $\mathcal{H}'$, we have

$$\Pr[h(x_i) = h(x_j)] < \frac{1}{n}.$$

[We are expecting: A clear and rigorous proof. As above, this needn't be more than a paragraph of two, but you should be very careful about what is random and what is not.]