Introduction to Algorithms	**Problem Set 5 (3 problems)**
CS 4820 Fall 2018	**Solutions**

(1) Randomized Median Finding. In class, we saw the randomized median finding algorithm (section 13.5), and showed that its expected running time is $O(n)$ with n elements. In this problem, we will only focus on the comparisons the algorithm has to make.

Randomized median finding starts by picking a random element x and identifying its position i in the sorted list using with $n-1$ comparisons. If $n/4 \le i \le 3n/4$, we discard the part of the array that we know doesn't contain the median, which is at least $n/4$ elements, and recurse. If i is not in this middle range, we try again with a new random element. As shown in class and in the textbook, in expectation it takes two tries to find an element in the desired range (using $2(n-1)$ comparisons). Solving the recurrence $T(n) \le 2(n-1) + T(3n/4)$, we find that the expected number of comparisons is at most $2n\frac{1}{1-3/4} = 8n$.

In this problem we will consider a small variation on this algorithm. Assume you proceed as follows: instead of picking 1 element at random, pick 3 different element of the array (with replacement). Compare the three (all three comparisons), and pick the median x **of the three** selected elements. Now determine the position i of x in the array by comparing to the remaining elements not selected in the initial sampling step. As before if $n/4 \le i \le 3n/4$, we discard the part of the array that we know doesn't contain the median, which is at least $n/4$ elements, and recurse. If i is not in this middle range, we try again with picking new random elements.

(a) What is the probability that the index of the median of the three points, i, is in the middle range $n/4 \le i \le 3n/4$?

Solution Consider the complement of this problem: what would have to happen for the median of the three elements *not* to be in the middle interval? In this case, this would require one of the following scenarios to happen: either

(i) all three elements fall outside of the middle interval, or

(ii) one element falls inside the middle interval, and exactly two of the three elements fall outside of the middle interval on the same side.

The probability of drawing from either side of the middle interval is at most 1/2 (less after picking 1 or 2 elements as there are 1 or 2 fewer elements on the side by then), so the probability of (i) is the joint probability of doing that three times: $P_i = 1/2 \cdot 1/2 \cdot 1/2 = 1/8$.

The probability of drawing from the middle interval is 1/2. We have 3 options of which of our draws is from the middle, for the other two draws the probability that the first fall outside the middle is 1/2, while the probability that the other one falls to the same side is 1/4. This is a total of $3 \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{4} = \frac{3}{16}$.

If you do the selection of the three elements without replacement the same probability bound works out, but its a bit more complicated to see it: the probability of (i) for this case is at most $1/2 \cdot 1/2 \cdot 1/2 = 1/8$, as the probability of the second and 3rd item from the two sides is less than 1/2. For the probability of (ii) the first choice in the middle interval is 1/2 as before, then the second choice outside the middle interval is $\frac{n/2}{n-1}$ as one in the middle is already selected. So this is a bit above 1/2. However, the probability that the last selection falls on the same side is $\frac{n/4-1}{n-2}$, which is less than 1/4. Combining the last two, we get

$$\frac{n/2}{n-1} \cdot \frac{n/4-1}{n-2},$$

which is less than 1/8, so the above analysis still applies. (To see why this is true, we need to have that $8 \cdot n/2 \cdot (n/4-1) \le (n-1)(n-2)$, i.e., that $-4n \le -3n+2$, so the same bound applies in this case also.

These two are disjoint probabilities, so the combined probability of not having the median in the correct interval is $1/8 + 3/16 = 5/16$. This means the probability that the index is in the middle range is $1 - 5/16 = 11/16$.

Expected incorrect solution A common mistake is to use cases that over-count some probabilities; for example, finding the probability that the median of the three elements is *not* in the middle interval by noting that this holds when two of the selections fall on the same side (no matter where the third falls). Then, saying that there are 2 sides, 3 different pairs of selections that could fall on a fixed side, and a third selection that may fall in any section, and concluding that the probability is $2 \cdot 3 \cdot 1/4 \cdot 1/4 \cdot 1 = 3/8$. The trouble with this argument is that you are *triple* counting the case when all three fall on the same side.
Note that this bad answer will result in an expected running time of $8/5 \cdot 4 \cdot n = 6.4n$ so the result is above $6n$ (!)
Another common way to accidentally over-count probabilities is to break your complex event (e.g., the median of the 3 elements not hitting the middle portion) into simpler cases that are *not disjoint*. Even if you calculate the probabilities for each simpler case correctly, you will end up counting the same event more than one time.

(b) State the recurrence for this algorithm, and show that the expected number of comparisons made by this method is upper-bounded by cn for a $c < 6$.

Solution Because we expect to find a valid median point 11/16 of the time, we expect on average that it will take 16/11 tries to find a median. In our previous median finding method, we required $n - 1$ comparisons; in this case, we require 3 comparisons to choose the median of the three candidates, followed by $n - 3$ comparisons to order the remaining elements, for a total of n. If we ended up with less than 3 elements (as we are picking with replacement), then we are only doing at most $n - 1$ comparisons, so less than n.
This gives us the following recurrence for the expected number of comparisons:

$$T(n) = 16/11 \cdot n + T(3n/4).$$

To solve this recurrence:

$$\begin{aligned} E[\hat{X}] &\le \sum_j (16/11)n(3/4)^j \\ &\le (16/11)n \sum_j (3/4)^j \\ &\le \frac{(16/11)n}{1 - 3/4} \\ &\le 64/11n. \end{aligned}$$

So cn comparisons with $c = 64/11 < 6n$ is a tight upper bound.
Alternately, one can use the Master Theorem for recurrences. We have $q = 1$ and $p = 4/3$ and the top level number of comparisons is $16/11 \cdot n$ in expectation, which gets is the same total of $E[\hat{X}] = \frac{p}{1-1/q}\frac{16}{11}n = \frac{64}{11}n < 6n$.

Expected incorrect solution Forgetting to count comparisons, and writing a recurrence of the form $T(n) = 16/11 + T(3n/4)$.

(2) Finding plurality in a list. Suppose you are supervising voting for County Person of the Year at the county fair. At the fair, n voters each write on a handwritten slip of paper the name of their favorite candidate and put it in the ballot box. You will have to process these ballots manually; furthermore, because any resident of the county is a valid candidate, the total number of possible candidates m is actually almost as large as the total number of voters. Counting the votes received by every candidate will be resource-intensive and hard to do accurately.

However, you suspect that tracking all possible candidates won't be necessary. You think it is likely that either (a) one of the candidates were selected by a large plurality of the voters, in which you can declare a winner, or (b) two candidates obtained significant shares of the vote, in which case you can immediately perform a runoff election. The list of votes is written out on a "read-only" array for allowing audits later.

Setting the definition of "large plurality of voters" to 1/3 of the votes, find a deterministic algorithm running in $O(n \log n)$ time and using at most $O(\log n)$ additional space (in addition to the read-only array) that returns all candidates that were selected by more than $n/3$ of the voters (there can be at most two of these). You may assume that n is a power of 2. (As always, include a proof of correctness and runtime complexity analysis.)

Solution: Use Divide-and-Conquer. Divide n into two sets of size $n/2$. We recursively solve the problem on both halves. This can return at most two candidates on both sides, for a total of 4 candidates. We test all four against the other half of the array, and return any that is selected by at least a third of the voters.

As a base case, if $n \leq 2$, we just return the two elements, even if different, both are at least a third of the votes.

The main observation is that if any a is selected by at most the third of the voters in both halves, than it is also selected by a 3rd of the voters overall. So if a is an element voted for by more than a third of the voters, than it must be returned by at least one of the two recursive calls. This implies that the algorithm correctly solves the problem.

To think about the running time, let $T(n)$ be the maximum number of comparisons used for inputs of size n. By increasing the voters by at most a factor of 2 we can assume n is a power of 2. Now we get the recurrence that $T(n) \leq 4n + 2T(n/2)$. So the algorithm spends $O(n)$ comparisons at each level of the recurrence, and there are $\log_2 n$ levels, for a total of $O(n \log n)$ comparisons.

For the space used: notice that to run a reclusive algorithm with $\log_2 n$ levels of recurrence, we need $O(\log n)$ space just to keep track of what point in the computation we are at. Beyond this, the algorithm only need the two to write down candidates returned at each level.

Expected Mistakes:

- One common mistake would be to only return candidates that were returned on both sides: if a candidate has more than 2/3 on just one side, but is not returned from the other half, it should be returned since it has more than 1/3 total.

(3) Hashing. Hashing functions are defined as maps from a universe U to some target set $T = [0, \ldots, n-1]$. We define a *set* of hashing functions $\mathcal{H}$ as *universal* if for any two elements $u, v \in U$, if $u \neq v$, then for a random $h \in \mathcal{H}$, the probability that $h(u) = h(v)$ is at most $1/|T|$ (where T is the target set).

For each of the following we define some set of hashing functions $\mathcal{H}$ based on a protocol to choose a random function h. For each option, either prove that it is universal, or give a counter example that shows it is not (that is, give u and v with higher collision probability than $1/n$ when uniformly sampling $h \in \mathcal{H}$).

(a) Suppose we want the target set to be $[0, 1, \ldots, p-1]$ for a prime p, and suppose the universe is $U = [0, 1, \ldots, M]$ for $M >> p$. We propose the following random selection of the function: select an $a \in [0, \ldots, p-1]$ uniformly random, and let the function be $h_a(x) = (ax \mod p)$.

Solution Not universal. If $u = 0$ and $v = p$, than $h_a(u) = h_a(v)$ for all a, that is the probability of collision is 100% for these two values

(b) Suppose we want the target set to be $[0, 1, \ldots, n-1]$, where n is not a prime, and suppose the universe is k-dimensional vectors $(x_1, x_2, \ldots, x_k)$, where each x_i is an integer in $[0, n-1]$. We propose the following random selection of the function: let p be a prime that is $p > n$, and select an $a_i \in [0, \ldots, p-1]$ uniformly random, let $a = (a_1, a_2, \ldots, a_k)$ and let the function be $h_a(x) = ((\sum_i a_i x_i \mod p) \mod n)$

Solution Not universal. Without the last $\mod n$, this is exactly the function we defined in class, and indeed it's universal. However, the last mod n makes it fail. Assume $k = 1$, and consider $u = 0$ and $v = 1$, assuming $n = p - 1$. Now a is just a scalar (as $k = 1$). From our function, $h_a(u) = h_a(0) = 0$ and $h(v) = h(1) = (a \mod n)$, so collision occurs if $a = 0$ or $a = n = p - 1$. The probability of collision is therefore $2/p = \frac{2}{n+1} > \frac{1}{n}$ for $n > 1$.
We can see that this is not universal for *any* $p > n$. Let $k = 1$, $u = 0$, $v = 1$. There will be a collision if $a \mod p$ is a multiple of n. The probability of this happening is m/p where m is the number of multiples of n less than p, including 0. Since, $mn > p > (m-1)n$, we have $m/p > 1/n$.

(c) Suppose we want the target set to be $[0, 1, \ldots, p-1]$, where p is a prime, and suppose the universe is vectors $(x_1, x_2, \ldots, x_k)$ where each x_i is an integer in $[0, n-1]$. We propose the following random selection of the function: let p be a prime that is $p \geq n$, and select an $a_i \in [0, \ldots, p-1]$ uniformly random, let $a = (a_0, a_1, a_2, \ldots, a_k)$ and let the function be $h_a(x) = (a_0 + \sum_i a_i x_i \mod p)$

Solution This is universal. Without adding a_0, it is exactly the function we defined in class, and indeed it is universal. Adding a_0 only permutes the target values, and doesn't change the probability of collision. Consider $a = (a_0, a_1, \ldots, a_k)$. Let $\bar{a}$ be the same as vector a except with $\bar{a}_0 = 0$. Notice that $h_a(u) = h_a(v)$ if and only if $h_{\bar{a}}(u) = h_{\bar{a}}(v)$. As $h_{\bar{a}}$ is exactly the universal hash function from class, h_a must have the same probabilities of collision and thus must also be universal.