CS358: Applied Algorithms **Name:** _____

This exam contains 8 pages (including this cover page) and 8 problems. (The final problem is a bonus problem just for practice—I think the actual midterm will be a little shorter than this.) Check to see if the exam you received contains all the pages. **This is a 75 minute exam and the total number of points is 100.**

- Please enter your name at the top.
- Please read through the entire exam before attempting any problem.
- If you need clarification on a question you may ask the instructor.
- You may use any results we've seen in class (you do not need to reprove any results we saw in class unless explicitly asked). You may also refer to algorithms we have seen in class (for example, you may say "run DFS on the graph").
- All scratch work must be done on the scratch paper provided along with this exam—either the sheets at the end, or the blank reverse side of each page. You are not allowed to write on a personal notebook during the exam.
- You are allowed a 1-page 8x11 2-sided "cheat sheet." You can use it for reference, but you should not use it as scratch paper (use the scratch paper provided).
- For each problem, the specifics of what it's asking for (a proof, an algorithm, a running time, etc.) is written in **bold**. Please check the bold instructions on every question to make sure you are answering it correctly!
- If you need additional space for a solution, you may use the scratch paper at the end of the exam, or the back of any sheet. Be sure to note where your solution is if you do this.
- **Remember that this exam has partial credit.** If you are unsure of the full solution, it's still a good idea to write down what you know.

1. (10 points) Which of the following is most expensive on most modern computers? **Circle** the most expensive operation. You do not need to explain your answer.

   a) A cache miss b) A branch misprediction c) Dividing two integers

2. (10 points) **What is the Jaccard similarity** of the following two sets A and B? You do not need to explain your answer.

$$A = \{a, b, c\} \qquad B = \{b, c, d, f\}$$

3. (10 points) Let's say I am using a Locality-Sensitive Hash to find a close pair of items like in Assignment 4, and I want to *decrease* the number of repetitions necessary to find the close pair. Which of the following is the best method to do so? **Please circle one, and explain your answer in one-two sentences.**

   a) Make the number of concatenated hashes smaller

   b) Make the number of concatenated hashes larger

   c) Make the number of buckets in the hash table smaller

   d) Make the number of buckets in the hash table larger

   Explanation:

4. (20 points) Let's say we want to find an item X in a sorted array of length n. For simplicity, assume that X is actually located in the array, at some arbitrary position. Here are two algorithms to do this:

   - **linear search**: iterate through each element of the array; if we ever see an element equal to X we return it. This takes $O(n)$ time.
   - **binary search:** look at the middle element of the array; if it is equal to X return it. If it is $< X$, recurse on the right side of the array; if it is $> X$ recurse on the left side of the array. This takes $O(\log n)$ time.

   For each of the following four questions, please **answer the question using big-O notation**; then **give a 1-sentence explanation** of your answer.

   What is the *cache efficiency* of linear search in the external-memory model? You should assume that n is much larger than M or B.

   What is the *cache efficiency* of binary search in the external-memory model? You should assume that n is much larger than M or B.

   About how many branch mispredictions will linear search incur while searching for X?

   About how many branch mispredictions will binary search incur while searching for X?

5. (20 points) You are given an array A of n integers. Your goal is to find four indices a, b, c, d such that $A[a] + A[b] = A[c] + A[d]$. We can solve this problem in $O(n^4)$ time by trying all n values for a, b, c, and d.

   Give a faster algorithm using the *meet-in-the-middle* strategy we used for the two towers problem. Your algorithm should use $O(n^2)$ space, and should run in either $O(n^2)$ or $O(n^2 \log n)$ time (either time bound is sufficient for full credit).

6. (20 points) Consider a stream of elements, but now each element of the stream is being *inserted* or *deleted*. The total number of times each element occurs is the number of times it is inserted, minus the number of times it is deleted.

   For example, in the following stream:

   Ins. 1, Ins. 2, Del. 1, Ins. 3, Ins. 2, Del. 2,

   1 appears a total of 0 times, 2 and 3 each appear once.

   We can naturally create a count-min-sketch like data structure for this kind of stream. We keep an array of w counters[1] and a hash h. (We'll only do "one row"—so only one hash h.) When an element e is inserted, we increment the counter at $h(e)$; when an element e is deleted, we decrement the counter at $h(e)$. Assume that the stream consists of N_i total insertions and N_d total deletions.

   For some query q, let $\hat{o}_q$ be the value returned by the data structure, and let i_q and d_q be the *true* number of times that q is inserted and deleted (respectively) in the stream. **Below, give the value of** $\mathrm{E}[\hat{o}_q]$ (i.e. the expected value returned by the data structure) in terms of i_q, d_q, w, N_i, and N_d. **Give the steps you use** to obtain your result.

[1] We used $w = e/\varepsilon$ for the CMS in class, but we'll just call it w here to keep things easier to work with.

7. (20 points) Consider a simple cuckoo filter with n items, cn slots (this is a basic cuckoo filter, with 1 "slot per bucket"), fingerprints of length f, and two independent hash functions h_1 and h_2.

   Give an upper bound on the false positive probability of the cuckoo filter in terms of c and f.

8. (10 points) The following is a **bonus problem**—I think it is too similar to other problems on the practice exam so I removed it, but it's not a bad use to use it as practice.

   You are given a *sorted* array A of n integers. The goal is to solve the following problem. We are given a target value T, and we want to find indices i and j such that $A[i]+A[j] = T$.

   Because the array is sorted, we can solve this problem using the following strategy. We keep two pointers i and j; to start, $i = 0$ and $j = n-1$. We compare $A[i]+A[j]$ to T. If $A[i]+A[j] = T$, we are done. If $A[i]+A[j] < T$, we increment i and continue. Otherwise $A[i]+A[j] > T$, so we decrement j and continue. We keep going until $i > n-1$ or $j < 0$, at which point we return that there is no solution.

   To summarize, i starts at 0, j starts at $n-1$, and at each iteration of the algorithm either i is incremented or j is decremented.

   Analyze this algorithm in the external memory model. Give the number of cache misses using big-O notation. You should assume that n is much larger than M or B.

Scratch Paper