CIS 121 — Data Structures and Algorithms
Homework Assignment 9

Assigned: April 20, 2020 **Due:** April 27, 2020

Note: The homework is due **electronically on Gradescope** on April 27, 2020 by 11:59 pm ET. For late submissions, please refer to the Late Submission Policy on the course webpage. You may submit this assignment up to 2 days late.

A. **Gradescope**: You must select the appropriate pages on Gradescope. Gradescope makes this easy for you: before you submit, it asks you to associate pages with the homework questions. Forgetting to do so will incur a 5% penalty, which cannot be argued against after the fact.

B. **LaTeX**: You must use the LaTex template provided on the course website, or a 5% penalty will be incurred. Handwritten solutions or solutions not typeset in LaTex will not be accepted.

C. **Solutions**: Please write concise and clear solutions; you will get only a partial credit for correct solutions that are either unnecessarily long or not clear. Please refer to the Written Homework Guidelines for all the requirements.

D. **Algorithms**: Whenever you present an algorithm, your answer must include 3 separate sections.

1. A precise description of your algorithm in English. No pseudocode, no code.
2. Proof of correctness of your algorithm
3. Analysis of the running time complexity of your algorithm

E. **Collaboration**: You are allowed to discuss **ideas** for solving homework problems in groups of up to 3 people but *you must write your solutions independently*. Also, you must write on your homework the names of the people with whom you discussed. For more on the collaboration policy, please see the course webpage.

F. **Outside Resources**: Finally, you are not allowed to use *any* material outside of the class notes and the textbook. Any violation of this policy may seriously affect your grade in the class. If you're unsure if something violates our policy, please ask.

NOTE: All final answers must be simplified as much as possible. Each answer should be a single fraction, and you must not involve any combinatorial expressions (e.g., factorial, "choose" notation, etc.). Unsimplified final answers, i.e., final answers involving summations or that are written as sums of multiple terms that could be combined, etc. will lose points. You may use tools like Wolfram Alpha to help you simplify complicated expressions if you wish, but this is the **only** online source you should be using for this homework outside of the resources posted on the course website.

0. [0 pts] Practice Question - Generalized Two-Sum

***Important Note**: This is a practice warm-up question you should be able to solve. You should work out the solution on your own, but please **do not submit it** with your assignment.*

You are given an unsorted array $A[1..n]$ of integers and an integer k. Your goal is to see if there are indices i and j (it is possible that $i = j$) such that when $x = A[i]$ and $y = A[j]$, the equation $4x^2 + 3y^3 = k$ is satisfied. Provide an algorithm that determines whether or not any such indices i and j exist (you are not required to return the actual indices). Your algorithm should run in **expected** $O(n)$ time. No proof of correctness is necessary, but please justify the runtime of your algorithm.

1. [10 pts] Hashing with Chaining

Suppose we have an initially empty hash table $A[1..n]$ with n slots that resolves collisions through chaining. Moreover, suppose we insert exactly n elements into this table such that each element inserted has a probability $\frac{1}{n}$ to be put into slot $A[i]$ *independently of all other elements*. Let R_i be the random variable that counts the number of elements that get hashed to slot $A[i]$.

(a) Give an expression in terms of n and k for $Pr(R_i = k)$ (i.e., the probability that k elements get hashed to slot $A[i]$), where $0 \leq k \leq n$.

(b) Prove that

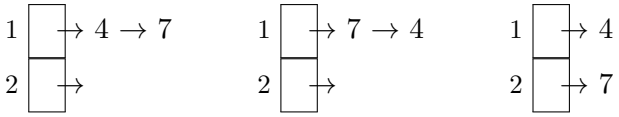
$$Pr(R_i = k) \leq \left(\frac{3}{k}\right)^k, \quad 0 \leq k \leq n$$

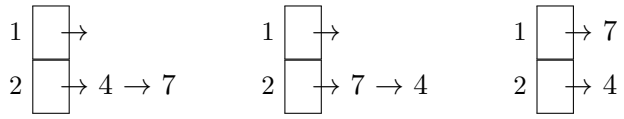
Note, you may find the inequality $\binom{n}{k} \leq \left(\frac{n \cdot e}{k}\right)^k$ useful. Here $e \approx 2.718$ is Euler's number, and you can assume the convention that $\left(\frac{3}{0}\right)^0 = 1$.

2. [10 pts]

Suppose we have an initially empty hash table $A[1..m]$ with m slots that resolves collisions through chaining. You may assume that that each element is equally likely to be inserted into each of these m slots independently of all other elements (simple uniform hashing assumption). Given that we insert n distinct elements into the hash table described above, what is the total number of distinct hash tables that could result? *For this problem, your final answer may involve any combinatorial expressions (e.g., factorial, "choose" notation, etc.).*

For example, if $m = 2$ and $n = 2$ and the elements inserted are 4 and 7, then there are six possible hash-tables:





3. [20 pts] Hashing with Open Addressing

Assume the underlying hash table $A[1..m]$ has m slots and resolves collisions using open addressing. Furthermore, assume the probe sequence used to insert or search for each element x is equally likely to be any of the $m!$ permutations of the numbers $\{1, 2, ..., m\}$ (uniform hashing assumption). We say a probe of slot i is any time you read and/or update the value $A[i]$. **Justify your answers. Correct answers without justification will receive no credit.**

(a) Suppose you insert n distinct elements, $0 < n < m$, into an initially empty hash table $A[1..m]$. What is the expected number of elements that require *at least* 2 probes?

(b) Suppose you insert n distinct elements, $0 < n < m$, into an initially empty hash table $A[1..m]$. What is the expected number of elements that require *exactly* 2 probes?

(c) Suppose you insert exactly three distinct elements into an initially empty hash table $A[1..m]$ where $m \geq 3$. In expectation, what is the total number of probes you make during this process?

4. [15 pts] Hash it out

Scott made a hash table $A[1..m]$ with $m > 1$ slots which resolves collisions using open addressing. He has already inserted n distinct elements, where $n < m$. Furthermore, assume the probe sequence that was used to insert each element x was equally likely to be any of the $m!$ permutations of the numbers $\{1, 2, ..., m\}$ (uniform hashing assumption). We say a probe of slot i is any time you read or update the value $A[i]$.

Unfortunately, he forgot what the hash function is! He wants to find a particular element x that's been inserted. Christina proposes the following algorithm.

```
SEARCH(A[1..m], x):
  for i = 1 to m:
      if A[i] = x then:
          return TRUE
  return FALSE
```

(a) What is the expected number of probes this algorithm makes assuming x is in the table? That is, what is the expected number of probes made by this algorithm in a successful search of a particular element x?

(b) Suppose you decided to search for every element in the table using the same SEARCH algorithm. That is, suppose you inserted the elements $x_1, x_2, ..., x_n$ into an initially empty table $A[1..m]$ and then made a sequence of calls SEARCH(A, x_1), SEARCH(A, x_2), ..., SEARCH(A, x_n). What is the total number of probes you will make in expectation over the n calls to SEARCH?

(c) Notice that since $n < m$, there are some empty slots in the hash table $A[1..m]$ that don't contain any elements. What is the probability that in a successful search for a particular element x, you

probe an empty slot? Depending on how you solve this problem, you may find the following combinatorial identity helpful:

$$\sum_{i=1}^{n} \binom{m-i}{n-i} = \frac{m\binom{m-1}{n-1}}{m-n+1}$$

5. [20 pts] Doggie Daycare

Suppose an even number of n TAs are interested in buying dogs only if they can spend **all** their TA money on dogs, without any money leftover. Each TA has some positive integer amount of money. Note that these amounts are not necessarily distinct. Moreover, all the TAs go to a special shelter where each dog costs exactly $\$k$.

They decide that its best if they all pair up in teams of two to buy the dogs, such that each pair of TAs can pool their money together and purchase dogs with no money left over. To clarify, each pair only has access to their pooled money and can only purchase a whole number of dogs. They cannot have money left over and cannot overpay for any dog.

Create an algorithm that runs in expected $O(n)$ time to determine if it is possible for the TAs to all pair up in this manner. Please do not forget to include a proof of correctness and run time analysis.

For example, if each dog costs \$9, and there are TA's with corresponding money:

TA:	A	B	C	D	E	F
Money:	\$2	\$7	\$5	\$13	\$9	\$27

Here, TAs (A, B) should pair up and purchase exactly 1 dog, TA's (C, D) should pair up and purchase exactly two dogs, and TA's (E, F) should pair up and purchase exactly four dogs so your algorithm should return true.

6. [25 pts] AVL Trees

(a) Extend the AVL tree data structure to implement the following method for an ordered dictionary D in $O(\log n)$ time.

> CountGEQTo(k_1): compute and return the number of items in D with key k such that $k \geq k_1$.

Note that this method returns a single integer, and that the value k_1 may or may not be in D. You may use at most $O(1)$ extra storage per node in the tree. If you decide to store any new values/-fields, you must describe (briefly) how you maintain those new values/fields upon insertion and deletion. Moreover, any modifications you make cannot change the runtimes of Insert, Search, or Delete. As always, prove the runtime and correctness of your modifications/algorithm.

(b) Extend the AVL tree data structure to implement the following method for an ordered dictionary D in $O(\log n)$ time.

> CountInRange(k_1,k_2): compute and return the number of items in D with key k such that $k_1 \leq k \leq k_2$.

Note that this method returns a single integer, and that the values k_1 and k_2 may or may not be in D. You may assume $k_1 \leq k_2$. You may use at most $O(1)$ extra storage per node in the tree. If

you decide to store any new values/fields, you must describe (briefly) how you maintain those new values/fields upon insertion and deletion. Moreover, any modifications you make cannot change the runtimes of Insert, Search, or Delete. As always, prove the runtime and correctness of your modifications/algorithm.

[0 pts] Feedback: No feedback form for this assignment. Feel free to reach out to the Head TAs if you have any comments/concerns.