DSC 40B - Quiz 05

Name: Student ID:

About the quizzes:

- Quizzes in DSC 40B are *optional* and graded pass/fail.
- A score of 70% or higher earns a "pass" and 1.5 credits toward your final grade.
- If you don't pass, no credits are earned, but it doesn't hurt your grade.
- You have 30 minutes to complete the quiz.
- Unfortunately, we can't answer clarifying questions during the quiz. If you think a question has a bug or is unclear, please let us know in a private post on Campuswire after the quiz, and we'll take it into account when grading.

Problem 1.

Suppose a hash table has 50 bins, and 400 elements are inserted into it using a hash function that places each element into a bin uniformly at random. What is the expected number of elements in the first bin?

◯ 1 ◯ 4 ● 8 ◯ 50 ◯ 400

Solution: 8.

When n elements are placed into m bins uniformly at random, the expected number of elements in any one bin is n/m. Here, that is $400/50 = 8$.

This problem was similar to Problem #174 in the Quiz 05 practice problems.

Problem 2.

Suppose a hash table is constructed so that collisions are resolved through chaining. The hash table has 10 bins, and 100 items are inserted into the hash table. True or False: it is possible for one of the bins to contain all 100 items.

● True

◯ False

Solution: True.

Chaining places no limit on how many items a bin can hold. If the hash function sends every item to the same bin (for instance, a bad hash function like $h(x) = 0$, or just unlucky keys that all collide), that bin will contain all 100 items. This is the worst case for a hash table, and it makes membership queries take $\Theta(n)$ time.

This problem was Problem #175 from the Quiz 05 practice problems.

Problem 3.

Assume that a hash table is implemented using chaining to resolve collisions. The hash table stores n numbers. Suppose a membership query is made for 42. True or False: more than one bin in the hash table may be checked with a linear search during the query.

◯ True

● False

Solution: False.

The query computes the hash of 42, which determines exactly one bin, and only that bin's chain is searched. If 42 is in the table at all, it must be in that bin, so there is no reason to look anywhere else.

This problem was Problem #176 from the Quiz 05 practice problems.

Problem 4.

Suppose a hash table stores n elements and resolves collisions by chaining. What is the **worst case** time complexity of a membership query? You may assume that computing the hash of an element takes $\Theta(1)$ time.

◯ $\Theta(1)$ ◯ $\Theta(\log n)$ ● $\Theta(n)$ ◯ $\Theta(n \log n)$

Solution: $\Theta(n)$.

In the worst case, every element hashed to the same bin. Finding the bin takes $\Theta(1)$ time, but then the query does a linear search through all n elements in that bin, which takes $\Theta(n)$ time. This is very unlikely with a good hash function, which is why the *expected* time of a query is $\Theta(1)$.

This problem was similar to Problem #178 in the Quiz 05 practice problems.

Problem 5.

Suppose we are building a hash table with 1000 bins, numbered 0 through 999, to store nonnegative integers. Each time we insert a number, we place it in the bin whose number is the **last digit** of the number. For example, 1234 goes into bin 4, and 57 goes into bin 7.

True or False: this is a good hash function for this hash table.

◯ True

● False

Solution: False.

The last digit of a number is always one of $0, 1, \ldots, 9$, so this hash function only ever places elements into the first 10 of the 1000 bins. The other 990 bins are never used, so there will be many more collisions than necessary. A good hash function uses all of the bins evenly.

This problem was similar to Problem #179 in the Quiz 05 practice problems.

Problem 6.

Suppose you insert 11 elements into a hash table with 20 bins. True or False: there must be **exactly** 9 empty bins, no more, no less, since the number of elements being hashed is less than the number of bins in the hash table by nine.

◯ True

● False

Solution: False.

The key word here is **exactly**. It is tempting to think that the 11 elements are spread out with one per bin, leaving the other 9 bins empty. But this isn't necessary: some elements may collide. For example, all of the elements might hash to the same bin, leaving 19 bins empty.

This problem was Problem #180 from the Quiz 05 practice problems.

Problem 7.

Suppose `arr` is a list of n integers. What is the **expected** time complexity of the code below? Remember that Python's `set` is implemented using a hash table.

```
def find_duplicates(arr):
    seen = set()
    duplicates = set()
    for num in arr:
        if num in seen:
            duplicates.add(num)
        else:
            seen.add(num)
    return duplicates
```

◯ $\Theta(1)$ ● $\Theta(n)$ ◯ $\Theta(n \log n)$ ◯ $\Theta(n^2)$

Solution: $\Theta(n)$.

The loop runs n times. Each iteration does one membership query (`num in seen`) and one insertion into a set, and both take $\Theta(1)$ expected time, since sets are hash tables. So the total expected time is $n \times \Theta(1) = \Theta(n)$.

This problem was similar to Problem #182 in the Quiz 05 practice problems.

Problem 8.

Consider the function below, where `A` and `B` each contain n numbers.

```
def intersection(A, B):
    common = set()
    for x in A:
        if x in B:
            common.add(x)
    return common
```

a) What is the **expected** time complexity of `intersection` if `B` is a Python `set`?

◯ $\Theta(1)$ ● $\Theta(n)$ ◯ $\Theta(n \log n)$ ◯ $\Theta(n^2)$

b) What is the **expected** time complexity of `intersection` if `B` is a Python `list`? Remember that `x in B` does a linear search when `B` is a list.

◯ $\Theta(1)$ ◯ $\Theta(n)$ ◯ $\Theta(n \log n)$ ● $\Theta(n^2)$

Solution: $\Theta(n)$ if `B` is a set, and $\Theta(n^2)$ if it is a list.

The loop runs n times, and each iteration does one membership query on `B`, plus (sometimes) an insertion into the set `common`, which takes $\Theta(1)$ expected time.

If `B` is a set, the membership query takes $\Theta(1)$ expected time, so the total is $\Theta(n)$. If `B` is a list, the membership query is a linear search. When `x` isn't in `B`, it looks at all n elements, and on average it looks at about half of them when `x` is, so each query takes $\Theta(n)$ time and the total is $\Theta(n^2)$.

This problem was similar to Problem #181 in the Quiz 05 practice problems.

Problem 9.

Suppose a hash table stores n elements, and inserting one more element causes it to grow: it allocates a larger table and copies (rehashes) every element into the new table. What is the time complexity of this one insertion?

◯ $\Theta(1)$ ◯ $\Theta(\log n)$ ● $\Theta(n)$ ◯ $\Theta(n^2)$

Solution: $\Theta(n)$.

Growing the table means rehashing all n elements and copying them into the new table, which takes $\Theta(1)$ time per element, or $\Theta(n)$ in total. Most insertions don't trigger a grow and take $\Theta(1)$ time, but the ones that do take $\Theta(n)$.

This problem was similar to Problem #184 in the Quiz 05 practice problems.

Problem 10.

For each situation below, which data structure has the better time complexity? Assume that the hash table uses a good hash function and that the binary search tree is balanced.

a) You will insert n numbers, and make n membership queries.

● Hash table ◯ Balanced binary search tree

b) You will insert n numbers. After each insertion, you will be given a number x (a different one each time), and must find the smallest number inserted so far that is larger than x.

◯ Hash table ● Balanced binary search tree

Solution: (a) A hash table. (b) A balanced binary search tree.

(a) Insertions and membership queries take $\Theta(1)$ expected time in a hash table, but $\Theta(\log n)$ time in a balanced BST. So the hash table takes $\Theta(n)$ expected time in total, and the BST takes $\Theta(n \log n)$.

(b) This query depends on the *order* of the numbers, and a hash table keeps its elements in an arbitrary order. To answer it, a hash table would have to look at every element, taking $\Theta(n)$ time per query, or $\Theta(n^2)$ in total. A BST keeps its elements in sorted order, so the query takes $\Theta(\log n)$ time in a balanced BST: walk down the tree as if querying for x, keeping track of the smallest key larger than x seen along the way. In total, that is $\Theta(n \log n)$. As in lecture, BSTs win when order is important.

(Since x changes each time, we can't just keep the answer up to date as we go, as we could for a fixed x.)

This problem was similar to Problem #188 in the Quiz 05 practice problems.