This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question 1</td>
<td>/10</td>
</tr>
<tr>
<td>Question 2</td>
<td>/10</td>
</tr>
<tr>
<td>Question 3</td>
<td>/10</td>
</tr>
<tr>
<td>Question 4</td>
<td>/10</td>
</tr>
<tr>
<td>Question 5</td>
<td>/10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>/50</strong></td>
</tr>
</tbody>
</table>

“Facebook is not your friend, it is a surveillance engine”

Happy birthday to Richard M. Stallman
Question 1 (10 marks)

Devise a double hashing function that would be suitable for storing a set of 800 Canadian Postal Codes, which always have the form “LDL DLD” where L represents any letter from A to Z, and D represents any digit from 0 to 9. Assume the hash table has 1024 addresses. Justify your answer.

a) [7 marks]: Assume that the postal codes come from all across the country, so all combinations of letters and digits are equally likely.

First we need to convert the postal code to an integer. This could be done by treating it as a string of 7 characters (or 6, ignoring the space in the middle) and using one of the methods covered in class to convert to an integer. The method should produce a wide range of different values, so simply summing the ordinal values of the characters will not be good enough.

Let k be the integer resulting from this conversion.

For double hashing we need h'(k) and h''(k), and we need to be sure that h''(k) is relatively prime to m. One pair of functions could be

\[ h'(k) = \text{mid\_square}(k), \text{ where mid\_square returns the middle four digits of } k^k \]

\[ h''(k) = \text{odd\_mult\_method}(k), \text{ where odd\_mult\_method implements the multiplication method described in class, modified so that it only returns odd values (for example, by adding 1 to any computed even value before returning it).} \]

Then \( h(k, i) = (h'(k) + i \cdot h''(k)) \mod 1024 \)

Because h''(k) and 1024 are relatively prime, this will probe every address if it needs to.

Marking guide on next page:
Convert the postal code to an integer: 2 marks
Choose a reasonable \( h'(k) \): 2 marks
Choose an \( h''(k) \) that is relatively prime to 1024: 2 marks
Stating \( h(k,i) = (h'(k) + i \cdot h''(k)) \mod 1024 \)

Take 1 mark off if the “convert postal code to an integer” only gives a small range of values.

Take 1 mark off if \( h''(k) \) is not guaranteed to be relatively prime to 1024.
b) [3 marks]: Assume that all the postal codes come from the Kingston area, so they all begin with K7L, K7M or K7P. What, if anything, would you change?

The only thing to be changed is the conversion of the postal code to an integer. We can completely ignore the first two elements ("K7") since they are the same for all keys. This may require some adjustment of the function that converts each postal code to an integer, to make sure we still get a wide range of integer values.

Marking guide:

It’s actually ok if they decide not to change anything – it will just be a tiny bit faster if they ignore the first two elements of the postal code.

Answer: Don’t change anything: 3 marks

Answer: Ignore the first two elements ("K7"), but the function that converts the postal code to an integer gives a small range of values: 2 marks

Answer: Ignore the first two elements, function that converts the postal code to an integer gives a wide range of values: 3 marks
Question 2 (10 marks)

Suppose you have a hash table with over 1000 addresses, and assume the keys are all 5-digit integers. Explain why this hash function is inappropriate, regardless of what type of collision resolution is being used:

\[ h(k): \]
\[ t1 = k^k \]
\[ t2 = t1 \mod 10000 \]
\[ \text{return } t2 / 100 \]

When we take \( t1 \mod 10000 \), we get values in the range \([0 \ldots 9999]\). When we divide these numbers by 100, we are left with values in the range \([0 \ldots 99]\). Thus every probe sequence (or chain) starts in one of the address 0 to 99 – this will inevitably increase the number of collisions.

Marking guide:

If they say anything about how a hash function should distribute the keys evenly across the table, they should get at least 6/10

If they show or state that the given function can only return a small range of values: at least 8/19

If they recognize that the given function only gives answers in the range \([0 \ldots 99]\), they should get 10/10
Question 3 (10 marks)

Suppose you are asked to create a data structure to store a priority queue for an application in which the only possible priorities are 11, 32, 79 and 100000. The necessary operations are:

- add a new item to the priority queue
- locate and extract the item with the highest priority currently in the queue.

If two items are tied for highest priority, the one that was added earlier should be the one chosen.

Your goal is to minimize the time complexity of the operations.

Describe the data structure (or structures) you would use in this situation. Explain your answer.

I would use a hash table with collisions resolved by chaining. The table would have 4 elements, one for each priority. For each chain, I would keep a “last” pointer that points to the last element in the chain – this allows adding new items in constant time, and removing the “oldest” member of the chain in constant time as well.

I would also use a function like this:

```python
def a(p):
    if p > 79:
        return 3
    elif p > 32:
        return 2
    elif p > 11:
        return 1
    else:
        return 0
```
to map the priority values onto the four addresses in the table.

This allows each of the required operations to be done in \( O(1) \) time.

**Marking guide:**

If they suggest using a max-heap, they should get about 6/10. The reason this is not a very good answer is that it gives \( O(\log n) \) time for each of the operations, as opposed to the \( O(1) \) achieved by the answer given above.

If they use a table with lists, but they make the table have 100,000 elements (of which only 4 are used), they should get about 7/10

If they use a table with lists where the table has only 4 elements, but they forget to mention how they map the priority values to the table elements, they should get about 8/10

If they use four completely separate lists, one for each priority, they should get 10/10
Question 4 (10 marks)

Consider a fixed set of n objects that are stored in a priority queue. (By fixed, we mean that no objects are added or removed from the set.) Each item has priority in the range [1..n]. When the object with the highest priority is removed from the head of the priority queue, it is immediately reinserted into the queue with a random priority in the range [1..n].

How would you implement this priority queue? Describe the data structure you would use, and the algorithm to remove and re-insert the highest priority object.

I would use a max-heap, stored in a 1-dimensional array. The algorithm would be a simplified version of the “bubble down” algorithm used to repair a max-heap after the highest priority item is removed.

It is simplified because we don’t need to move the last element up to the root. We simply change the priority of the value at the top of the heap and bubble it down to a valid location. The “physical” structure of the heap does not change at all.

Marking guide:

Storing the heap in an array is not crucial – no penalty if they do not specify how they are storing the heap.

If they show they understand how a heap works, they should get at least 6/10

If they write out the pseudo-code for the algorithm, make sure it is correct in concepts but don’t penalize for minor errors.

If they do too much work to fix the heap (for example, moving the last value up
to the top then bubbling it down, then re-inserting the previous top value at the bottom and bubbling it up) they should get 7 or 8.
Question 5 (10 marks)

Let $G$ be a graph on $n$ vertices. Using either the adjacency matrix representation or the adjacency list representation, give an algorithm (in pseudo-code) that will start at a given vertex $x$ and will return a list containing all the vertices that are neighbours of $x$'s neighbours. You may assume that the vertices are numbered $1, 2, \ldots, n$

**Using Adjacency Lists:** Let $\text{Neighbours}(v)$ be a list containing all the neighbours of vertex $v$

```python
def N_of_N(x):
    result = []
    for y in Neighbours(x):
        for w in Neighbours(y):
            if w not in result:
                append w to result
    return result
```

**Using Adjacency Matrix:** Let $A$ be the adjacency matrix of $G$

```python
def N_of_N(x):
    result = []
    for j = 1 .. n:
        if $A[x,j] == 1$:
            # $j$ is a neighbour of $x$
            for k = 1 .. n:
                if $A[j,k] == 1$ and $k$ not in result:
                    append $k$ to result
    return result
```
Marking guide:

Students were not required to give both answers!

Their pseudo-code may not look like mine, but the central concepts should be similar. If they show that they understand the basic idea of iterating through the neighbours of x, and find the neighbours of each of those, they should get at least 7 on this question. If they forget to remove duplicates, take 1 mark off.