**Hash Table Exercises - Solutions**

**problem 1:** Given the values \{2341, 4234, 2839, 430, 22, 397, 3920\}, a hash table of size 7, and hash function \( h(x) = x \mod 7 \), show the resulting tables after inserting the values in the given order with each of these collision strategies.

\{2341, 4234, 2839, 430, 22, 397, 3920\}

\( h(x) = x \mod 7 \)

2341 % 7 = 3  
4234 % 7 = 6  
2839 % 7 = 4  
430 % 7 = 3  
22 % 7 = 1  
397 % 7 = 5  
3920 % 7 = 0

1. **separate chaining**
   0 [3920]  1 [22]  2 [ ]  3 [2341, 430]  4 [2839]  5 [397]  6 [4234]

2. **linear probing**

3. **quadratic probing**

   430 collides at 3:  
   next - +1 = 4  
   next - +4 = 7 % 7 = 0

   3920 collides at 0:  
   next - +1 = 1  
   next - +4 = 4  
   next - +9 = 9 % 7 = 2

4. **double hashing with second hash function \( h'(x) = (2x - 1) \mod 7 \)**

   \((2*430-1) \mod 7 = 5\)
   \begin{align*}
   430 &= 3 \\
   &= 3+1*5 = 8 \mod 7 = 1
   \end{align*}

   \((2*22-1) \mod 7 = 1\)

   \begin{align*}
   22 &= 1 \\
   &= 1 + 1*1 = 2
   \end{align*}
**Problem 2:** Suppose you need to insert unique 3-character IDs into a hash table, where each ID is made up of some combination of two of the capital letters A-D, followed by one of the lower case letters x-z, such as: ABx, DCy, BBz, etc. Repeat letters are allowed in an ID.

1. There are 48 possible codes: $4 \times 4 \times 3$

2. The table only needs to be size 24 in order to ensure that no bucket exceeds two codes at a given time.

There are many possible ways to write the hashCode function for these IDs. A correct hash function will have each ID colliding with at most one other ID.

*Constructing a correct hash function*

(Interpret str[0] and str[1] as the result of a mapping from A->0, B->1, C->2, D->3 ; str->[2] is interpreted as x->0, y->1, z->2.)

row: str[0]
column: str[1]
using integer arithmetic…

This table: $str[0] + (str[1] / 2) \times 2$

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
```

This table produces exactly one collision for each two-letter pair. Now we need a unique mapping combining the above function from 0-7 and the resulting mapping of str[2]:

row: $str[0] + (str[1] / 2) \times 2$
col: str[2]

This table: $str[0] + (str[1] / 2) \times 2 + str[2] \times 8$

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>
```

We now have a unique mapping for 48 IDs into 24 slots. The function to produce this works as follows:

```java
public int hashCode(String ID) {
    int hashcode = 0;
    Integer A = new Integer('A');
    Integer x = new Integer('x');
    hashcode += ID.charAt(0) - A.intValue();
    hashcode += 2 * ((ID.charAt(1) - A.intValue) / 2);
    hashcode += 8 * (ID.charAt(2) - x.intValue());
    return hashcode;
}
```

Any function which produces at most one collision for any given ID is also acceptable.
problem 3: Suppose a hash table with capacity $M=31$ gets to be over $3/4$ths full. We decide to rehash. What is a good size choice for the new table to reduce the load factor below .5 and also avoid collisions?

$3/4$ths full: more than ceil($3/4 * 31$) = 24 elements

Must be a prime to avoid collisions
Two possible answers to reduce the load factor below .5:
1) $M = \text{next prime higher than } 31 \times 2 = 62$
   $M = 67$
2) $M = \text{next prime higher than } 24 \times 2 = 48$
   $M = 53$

problem 4: Suppose you are running a food service business which has special promotions for frequent customers. Each month you send out discount offers to different groups based on their spending in your establishment. For example, one month those who spent $25-50$ might get a free drink coupon, or another month those who spent $50-100$ might get a 2-for-1 lunch offer. The spending ranges will change each month, and you want to be able to find the right group of people each month in O($K + \log N$) time, where $K$ is the size of the group and $N$ is the total number of customers. What data structure(s) would you choose and how would you implement a solution to this problem? (sketch briefly)

Assumptions:
The algorithm runtime does not include construction of the customer database;
the customer database is created *before* knowing the interval of spending ranges.

Create a hash table using linear chaining to resolve collisions. The key for the hash table will be the amount that the customer spent last month at the food service business. The value will be some representation for the customer. The hash code function will have to preserve the order of the keys.

The O($K+\log N$) algorithm to find the correct group of customers for an interval $X-Y$:

Do a binary search over the keys in the hash table (this is why order must be preserved in the keys of the hash table). Continue until the smallest key $\geq X$ is found.

Start iterating linearly through the hash table in order by key. Examine each key and add the corresponding customers (or list of customers, in the case where two customers spend the same amount.)

When a key larger than $Y$ is found, stop iterating through and return the list of customers encountered.

Other solutions which also produce the result in O($K+\log N$) are acceptable.

problem 5

Many answers are possible. A good justification will include an explanation of why the data structure is particularly efficient to add items each semester, and find out information about groups of courses according to the department that they're in.