1. Assume that you have the C array from counting sort (that is, you’ve set \( C[i] = 0 \) for all \( i \), then you’ve incremented \( C[A[i]] \) for all values of \( i \) from 0…\( n - 1 \), then you’ve set \( C[i] = C[i] + C[i - 1] \) for all values of \( i \) from 1…\( m \))

Write a function `numBetween` that takes as input this \( C \) array, the number of elements \( m \) in the \( C \) array, and two values low and high, and returns the number of elements in the original list whose values are between low and high, inclusive. You need to be able to handle duplicate entries. Your function should take time \( \Theta(1) \). You do not need to turn this in electronically.

```c
int numBetween(int C[], int m, int low, int high)
```

2. The following array is to be sorted using radix sort (base-10). Show the result after sorting each digit.

<table>
<thead>
<tr>
<th>321</th>
<th>142</th>
<th>225</th>
<th>451</th>
<th>673</th>
<th>009</th>
<th>329</th>
<th>025</th>
<th>320</th>
<th>765</th>
<th>143</th>
<th>663</th>
<th>233</th>
<th>774</th>
<th>129</th>
</tr>
</thead>
</table>

3. The following elements are inserted into a hash table of size 11 (in this order): 1, 12, 23, 2, 3

(a) Show the resulting hash table, assuming open hashing (separate chaining)

(b) Show the resulting hash table, assuming closed hashing and linear probing

(c) Show the resulting hash table, assuming closed hashing and quadratic probing

(d) Show the resulting hash table, assuming closed hashing and double hashing (second hash function: \( \text{hash2}(x) = 5 - x \% 5 \))

4. Assuming that disjoint sets uses union by rank, but not path compression, show the resulting Parent array after the following operations:

- MakeSets(12)
- Union(3,5)
- Union(6,9)
- Union(2,5)
- Union(6,2)
- Union(10,1)

5. Assuming that disjoint sets use union by rank, but not path compression, give a sequence of Union calls that result in the highest possible tree for a set of 16 elements, and show the resulting Parent array.