CS2302 Data Structures
Spring 2020
Exercises – Hash Tables with Chaining

Consider a hash table that solves collisions by chaining implemented in the program *hash_table_chain.py* provided in the class web page.

1. Trace the execution of the following program (do not submit an answer, just make sure you understand every line in the code well enough to trace it).

```
import hash_table_chain as htc

if __name__ == "__main__":
    h = htc.HashTableChain(9)

    players = ['Bellinger','Betts', 'Hernandez', 'Pederson', 'Pollock', 'Taylor']
    numbers= [35, 50, 14, 31, 11, 3]

    for i in range(len(players)):
        h.insert(numbers[i],players[i])

    h.print_table()
```

Rename the provided *htc_ex1_start.py* program as *yourlastname_yourfirstname_htc_ex1.py* and complete the implementation of the following functions:

2. The load factor of a hash table is the number of elements in the table divided by the size of the table, or, equivalently, the average length of the lists (or buckets) in the table. Write the function *load_factor(h)* that computes the load factor of hash table h.

3. In the worst case, the access to an element in the table will be proportional to the length of the longest bucket in the table. Write the function *longest_bucket(h)* that returns the length of that bucket in hash table h that contains the most elements.

4. Write the function *check(h)* that verifies that every record has been inserted in the right bucket in hash table h.

5. (Extra credit) Suppose we have a list and wish to determine if it contains duplicates. A naïve solution consists of, for every element L[i] in the list, determining if L[i] is in L[:i] +L[i+1:]; this would take $O(n^2)$. Here's an implementation of that:

```
def has_duplicates_v1(L):
    # O(n^2)
    for i in range(len(L)):
        if L[i] in L[:i] + L[i+1:]:
            return True
    return False
```

We can do a little better if we sort the list, then, if we have duplicates, they will appear in contiguous positions in the sorted list. In this case the running time is O(n log n) . Here's an implementation of that:

```
def has_duplicates_v2(L):
    # O(n log n)
    L.sort()
    for i in range(1,len(L)):
        if L[i] == L[i-1]:
            return True
    return False
```

We can solve the problem in O(n) using a hash table. Write an implementation of that using our implementation of hash tables.