

**Objective:** To experiment with searching and get a feel for the performance of hashing.

**To start the lab:** Download and unzip the file lab7.zip

**Part A:**

- a) Open and run the timeLinearSearch.py program that times the LinearSearch algorithm imported from LinearSearch.py. Observe that it creates a list, evenList, that holds 10,000 sorted, even values (e.g., evenList = [0, 2, 4, 6, 8, ..., 19996, 19998]). It then times the searching for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 so half of the searches are successful and half are unsuccessful. How long does it take to linear search for target values from 0, 1, 2, 3, 4, ..., 19998, 19999?
- b) Open and run the timeBinarySearch.py program that times the binarySearch algorithm imported from binarySearch.py. How long does it take to binary search for target values from 0, 1, 2, 3, 4, ..., 19998, 19999?
- c) Open and run the timeListDictSearch.py program that times the ListDict dictionary ADT in list\_dictionary.py. The ListDict implementation uses a single Python list for storing dictionary entries. The timeListDictSearch.py program adds the 10,000 even values (i.e., 0, 2, 4, 6, 8, ..., 19996, 19998) to a ListDict object, and then times the searching for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 so half of the searches are successful and half are unsuccessful. How long does it take to search for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 in the ListDict?
- d) Open and run the timeChainingDictSearch.py program that times the ChainingDict dictionary ADT in chaining\_dictionary.py. The timeChainingDictSearch.py program adds the 10,000 even values (i.e., 0, 2, 4, 6, 8, ..., 19996, 19998) to an ChainingDict with 16,384 slots (i.e., load factor of 0.61), and then times the searching for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 so half of the searches are successful and half are unsuccessful. How long does it take to search for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 in the ChainingDict?
- e) Explain the relative performance results of searching using linear search, binary search, a ListDict, and ChainingDict.

f) The Python `for` loop allows traversal of built-in data structures (strings, lists, tuple, etc) by an *iterator*. To accomplish this with *our* data structures we need to include an `__iter__` method. In general an `__iter__` method, must loop down the data structure and `yield` each item in the data structure. When done, the `__iter__` needs to `raise StopIteration`. See the end of `UnorderedList` and `ListDict` classes for examples of their `__iter__` methods. Complete the `__iter__` code for the `ChainingDict` and `OpenAddrHashDict` classes.

**After you have completed the above timings, questions and code, raise your hand and explain your answers.**

**Part B:**

a) Open and run the `timeOpenAddrHashDictSearch.py` program that times the `OpenAddrHashDict` dictionary ADT in `open_addr_hash_dictionary.py`. The `timeOpenAddrHashDictSearch.py` program adds the 10,000 even values (i.e., 0, 2, 4, 6, 8, ..., 19996, 19998) to an `OpenAddrHashDict` with 16,384 ( $2^{14}$ ) slots (i.e., load factor of 0.61) using linear probing, and then times the searching for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 so half of the searches are successful and half are unsuccessful. How long does it take to search for target values from 0, 1, 2, 3, 4, ..., 19998, 19999 in the `OpenAddrHashDict`?

b) Why is the above timing so long with a load factor of only 0.61? (Hint: Think about where the even values are being stored in the hash table and the searches performed. Value 0 is stored at home address 0, etc.)

	self._table
0	
1	
2	
3	
4	
5	
6	
7	
16,380	
16,381	
16,382	
16,383	

c) Experiment with changing the load factor of the HashTable by increasing the hash table size to 32,768 ( $2^{15}$ ) for a load factor of 0.31, and 65,636 for a load factor of 0.15. Completing the following table:

Linear Probing	Hash Table Size (Load Factor)		
	16,384 (0.61)	32,768 (0.31)	65,636 (0.15)
Execution time with 10,000 items in hash table (seconds)			

d) In `timeOpenAddrHashDictSearch.py` modify the construction of `evenHashTable` so it uses quadratic probing instead of linear probing (i.e., `evenHashTable = OpenAddrHashTable(2**14, hash, False)`). Completing the following table:

Quadratic Probing	Hash Table Size (Load Factor)		
	16,384 (0.61)	32,768 (0.31)	65,636 (0.15)
Execution time with 10,000 items in hash table (seconds)			

e) Explain why quadratic probing performs better than linear probing.

**After you have performed the timings and answered the questions, raise your hand and explain your answers.**