Average seconds per operation:

n	**1**	**10**	**100**	**1k**	**10k**	**100k**	**1M**	**10M**	**100M**
add to dictionary	1.9 e-06	1.4 e-06	1.6 e-06	2.1 e-06	1.9 e-06	1.7 e-06	1.6 e-06	1.5 e-06	1.3 e-06
del from dictionary	8.6 e-07	7.3 e-07	1.2 e-06	1.1 e-06	2.9 e-06	1.1 e-06	2.6 e-06	1.1 e-06	7.4 -e07
add to front of list	1.8 e-06	5.4 e-07	5.5 e-07	1.0 e-06	6.8 e-06	too big	too big	too big	too big
add to middle of list	2.0 e-06	3.2 e-06	7.0 e-07	1.3 e-06	1.3 e-05	too big	too big	too big	too big
add to end of list	1.2 e-06	4.8 e-07	3.5 e-07	3.8 e-07	9.3 e-07	too big	too big	too big	too big
del from front of list	1.3 e-06	7.2 e-07	3.3 e-07	4.7 e-07	3.8 e-06	too big	too big	too big	too big
del from middle of list	1.0 e-06	8.7 e-07	4.7 e-07	6.3 e-07	2.4 e-06	too big	too big	too big	too big
del from end of list	1.0 e-06	6.3 e-07	4.6 e-07	5.4 e-07	2.1 e-06	too big	too big	too big	too big

The table was consistent with my expectations. The times for adding and deleting from the end of a list remained pretty constant as the list either grew or shrink, as the times should since adding and removing from the end of a list are O(1) operations. The times for adding and deleting (key, value) pairs in the dictionary remained pretty constant as the dictionary grew or shrink, as they should since adding and removing from a dictionary are O(1) operations as well. The times for adding and deleting from the front of the list changed in a way that was not constant as the list grew from 0 to size n or shrank from size n to 0 because when an element is added at index 0, the other elements in the list are shifted over an index; therefore, adding and deleting from the front of the list are O(n) operations. The times for adding and deleting from the middle of the list changed as the list grew from 0 to size n or shrank from size n to 0 since an element is being inserted or removed in the middle of the list and the item at the middle index gets shifted over; therefore, adding and deleting from the middle of the list are O(n) operations. Inserting or removing from the front of a list will take longer than inserting or removing from the middle of the list because when you insert or remove in the middle you only shift over half the elements in the list, instead of all elements in the list. Eventually the size of the input, n, becomes too big and because of memory and CPU constraints the table was not able to be completed for the list operations. The table was completed for the dictionary. If one were to input beyond 100 million key, value pairs in the dictionary, they would eventually run out of memory.