ADT	File1.dat				File2.dat				File3.dat				File4.dat			
Trial #:	1	2	3	Avg.	1	2	3	Avg.	1	2	3	Avg.	1	2	3	Avg.
LinkedList	70	19	32	40.3	22	17	14	17.7	47569	55875	55582	53008.7	45410	50340	47314	47688
StackArray	45	25	18	29.3	25	18	14	19	13	14	11	12.7	15	15	26	18.7
StackList	42	28	27	32.3	35	14	16	21.7	28	13	13	18	14	14	8	12
QueueList	42	29	41	37.3	19	15	13	15.7	13	14	13	13.3	15	14	29	19.3
ArrayList	31	17	19	22.3	749	731	716	732	20655	27669	26283	24869	17755	18139	18209	18034.3
Array	33	17	15	21.7	10236	11646	11518	11133.3	11476	11368	11642	11495.3	11085	10752	11255	11030.7

ADT	File1.dat					File2.dat				
	individual insertion	individual deletion	series of inserts	series of deletes	entire file	individual insertion	individual deletion	series of inserts	series of deletes	entire file
LinkedList	O(1)	N/A	O(n)	N/A	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
StackArray	O(n)	N/A	O(n^2)	N/A	O(n^2)	O(n)	O(n)	O(n^2)	O(n^2)	O(n^2)
StackList	O(1)	N/A	O(n)	N/A	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
QueueList	O(1)	N/A	O(n)	N/A	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
ArrayList	O(n)	N/A	O(n^2)	N/A	O(n^2)	O(n)	O(n)	O(n^2)	O(n^2)	O(n^2)
Array	O(1)	N/A	O(n)	N/A	O(n)	O(1)	O(1)?	O(n)	O(n^2)	O(n^2)

ADT	File3.dat					File4.dat				
	individual insertion	individual deletion	series of inserts	series of deletes	entire file	individual insertion	individual deletion	series of inserts	series of deletes	entire file
LinkedList	O(1)	O(n)	O(n)	O(n^2)	O(n^2)	O(1)	O(n)	O(n)	O(n^2)	O(n^2)
StackArray	O(n)	O(n)	O(n^2)	O(n^2)	O(n^2)	O(n)	O(n)	O(n^2)	O(n^2)	O(n^2)
StackList	O(1)	O(1)	O(n)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
QueueList	O(1)	O(1)	O(n)	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)
ArrayList	O(n)	O(n^2)	O(n^2)	O(n^3)	O(n^3)	O(n)	O(n^2)	O(n^2)	O(n^3)	O(n^3)
Array	O(1)	O(n)	O(n)	O(n^2)	O(n^2)	O(1)	O(n)	O(n)	O(n^2)	O(n^2)

Write-Up:

LinkedList:
The LinkedList method has very different run time in File1 and File2 compared to File3 and File4. The reason behind this comes down to the deletions, insertions will always take O(n) run time no matter the order but this is not the case with deletions. Deletions in LinkedList start with the very first Node in the list and then searches for the given node in order to delete it. The reason that File2 is such a short runtime is because it is O(1) because even though there are 125000 deletions, since the deletions start at 0 and go to 125000 each deletion will be the first node that is checked. But for File3 and File4, this is not the case because the nodes are not deleted in that same order so the empirical run time jumps up and theoretical goes up from O(n) to O(n^2).

StackArray:
Stack Array really doesn't have a lot of discrepancy among the four files, this is because the insertions and deletions are always O(n). It is O(n) because the array may need to be copied over to a smaller/bigger array while inserting or deleting. The runtime doesn't change with the order as it is always just creating a new node and reassigning it to the top or removing the top by reassigning the topOfStack variable, there is no searching for the element to delete because it just delete the top element regardless of what it is.

StackList:
Very similar to Stack Array in that the different files don't change the empirical runtime much, but StackList does not end up being O(n^2) for file1, 2 and 3 like StackList does. This is because it doesn't have to do the coping of the array that the StackArray must do. The order of insertion and deletions does not affect StackList again because insertions are always just added to the top and deletions are always just deleted from the top regardless of what it is.

Queue List:
QueueList is almost the same as StackList, empirical runtimes are all very similar and theoretical are the exact same. The QueueList adds all new elements to the front of the queue and then just removes the front element whenever there is a command to remove so the order of deletions changes nothing to the runtime. One small pattern that seems to be present in both the QueueList and StackList is that the all insertions files takes slightly longer, this could be because the deletions actually take less runtime because there's less steps. Both are still constant run time for an individual deletion or insertion.

ArrayList:
The array list is similar to the array except for the fact that it may need to be copied onto another array, which can add runtime. For File1, the whole runtime is O(n^2) because individual insertions are O(n). For File2, the runtime of a series of deletions would be O(n^3) because an individual deletion have to search for the element and then possibly copy the array onto a new array, but it is just O(n^2) because searching for the element is constant time because the element will always be the first on that is checked. This is similar reasoning to why LinkedList runtimes are different from File 2 to File3 and 4.

array:
The array is a set size structure, so it has a set size of 250000. This means it will never have to do the resizing that the ArrayList has to do, this is why the array has a faster empirical runtime for File3 and 4 because it doesn't need to do this copying over that ArrayList must do. It had a long runtime for File2 though because of the fact that the resizing of the array in the ArrayList will keep the elements that need to be deleted much closer to the front so the search does not take as long. Even though the array deletes in the same order, the array does a lazy delete which makes it still need to search through each of the "deleted" indices, the LinkedList and ArrayList doesn't have to so it is much faster for File2.