Data Structures				
	Operation	Time - Average Case	Time - Worst Case	
Array	Search* Insert at first position Insert at the end Remove first position Remove at the end Update**	O(N) O(N) O(1) O(N) O(1) O(1)	O(N) O(N) O(1) O(N) O(1) O(1)	If we are using a resizable array and it is currently full, inserting at the end can have O(N): Duplicating array's size and copying N elements to it.
Linked List	Search* Insert at first position Insert at the end Remove first position Remove at the end Update**	O(N) O(1) O(N) O(1) O(N) O(N)	O(N) O(1) O(N) O(1) O(N) O(N)	
Hash Table	Search* Insert Remove Update**	O(1) O(1) O(1) O(1)	O(N) O(N) O(N) O(N)	Performance depends on the choice of hash function and hash size
Unbalanced Binary Search Tree	Search* Insert Remove Update**	O(log(N)) O(log(N)) O(log(N)) O(log(N))	O(N) O(N) O(N) O(N)	Could also use notation O(h) for Average Case, with h being the tree's height (=log(N))
Balanced Binary Search Tree	Search* Insert Remove Update**	O(log(N)) O(log(N)) O(log(N)) O(log(N))	O(log(N)) O(log(N)) O(log(N)) O(log(N))	Rotations in a AVL tree are O(1), so they don't impact overall complexity
Min/Max Heap	Search* Insert Remove Peak	O(N) O(1) O(log(N)) O(1)	O(N) O(log(N)) O(log(N)) O(1)	Inserting is just putting it in the rightmost position, except if it is the min/max value (worst case)

Stack* **	Push Pop Peak Is Empty	O(1) O(1) O(1) O(1)	O(1) O(1) O(1) O(1)	
Queue ****	Add Remove Peek Is Empty	O(1) O(1) O(1) O(1)	O(1) O(1) O(1) O(1)	

*Search = Suppose we are searching for an element which value is V, we want to find its position. *int search(V)*
**Update = Suppose we want to update an element which is in position P. *void update(P)*
***Stack = Implemented as an array (doesn't shrink or grow well because you have to provide a max size) or linked list
****Queue = Implemented as a linked list and saving two pointers (front and rear). Implementing as an array would provide worse time complexity.

Algorithms			
	Time - Average Case	Time - Worst Case	
Breadth-First Search	O(V+E)	O(V+E)	V = Nb of Vertices E = Nb of Edges
Depth-First Search	O(V+E)	O(V+E)	
Binary Search	O(log(N))	O(log(N))	
Quick Sort	O(N*log(N))	$O(N^2)$	
Merge Sort	O(N*log(N))	O(N*log(N))	

Python Functions				
	Operation	Time - Average Case	Time - Worst Case	
List	Get	O(1)	O(1)	
	Append	O(1)	O(N)	
	Insert	O(N)	O(N)	
	Del	O(N)	O(N)	
	Find	O(N)	O(N)	
	Min/Max	O(N)	O(N)	
Dict	Insert	O(1)	O(N)	Hash table with linear probing
	Get	O(1)	O(N)	
	Pop	O(1)	O(N)	
	Clear			
Set	Checking if item is in set	O(1)	O(N)	Set implements a hash table
	Add	O(1)	O(N)	
	Union	O(len(s1)+len(s2))	O(len(s1)+len(s2))	
	Intersection	O(min(len(s1),len(s2)))	O(min(len(s1),len(s2)))	
Join	-	O(N)	O(N)	
Split	-	O(N)	O(N)	
Strip	-	O(N)	O(N)	
Sort	-	O(N*log(N))	O(N*log(N))	Implements Timsort, hybrid of Mergesort and Insertionsort