Chapter VI	Big O								
	Asymptotic notations								
	O(Big O)	Describes an upper bound on time		on the runtime; similar to a less-	then we also say that X<=	In industry, O and theta have been put together and we have to give the tightest description of runtime			
	Omega(n)	Describes the lower bound	for example: printing the values in an array is Omega (n) as well as Omega(logn) as well as Omega(1)						
	Theta(n)	Describes the tight bound on runtime	Theta here means both O and Omega; in this example, it would be Theta(n)						
	Best Case, Worst Case and Expected Case								
	Best Case:	For example, in Quick Sort, if all the elements are equal, then quick sort will, on average, just traverse through the array once - O(N) time	elements greater than pivot - this gives partial sort. then it recursively sorts the left and						
	Worst Case:	The pivot could be repeatedly the biggest element in the array. If pivot is the first element in a reversely sorted array. In this cae, our recursion does not divide the array in half and recurse on other half. Instead, it justs shrinks the subarray by 1 element.	Time taken would O(N^2)						
		both the above best and worst conditions would rarely happen; thus we can expect a runtime of O(nlogn)							
		etween Asymptotic notations , Worst Case and Expected							

There is no particular relationship between the two concepts										
	10 15 110									
Best Case, Wor	st Case and Expected Case act	ually describe the big O or big	g Theta time for part	ticular scenarios w	hereas these asy	mptotic notations	describe the upp	er, lower and tigh	t bounds for the ru	ıntime
Space complexity										
Memory or space required by an algorithm	to create an array - if it is unidimensional, O(N) space complexity; for a 2-D array, O (N^2)									
Stack space in recursive calls counts too. Each call adds a level tot he stack and takes up actual memory.	However, just because you have N calls does not mean it will take O(N) time: check the example on Page 41 for more details									
Drop the constants										
O(2N) is actually O(N)										
Drop the non- dominant terms										
O(N^2 + N) becomes O (N^2)										
O(N + logN) becomes O(N)										
O(5*2^N + 1000N^100) becomes O (2^N)										
$O(x!) > O(2^x) > 0$	$O(x^2) > O(x\log x) > O(x)$									
Multi-Parts algorithms: add versus mutiply										
Add:	Non-nested chunk of work A and B	O(A + B)	"DO THIS THEN WHEN YOU ARE ALL DONE, DO THAT"							

				"DO THIS FOR			
				EACH TIME			
	Multiply	Nested A and B	O(AB)	YOU DO THAT"			