Big O notation

Orde	n= 100	n= 10000	n= 1000000	n= 100000000
O(1)	1 ms	1 ms	1 ms	1 ms
O(logn)	1 ms	2 ms	3 ms	4 ms
O(n)	1 ms	0,1 s	10 s	17 min
O(nlogn)	1 ms	0,2 s	30 s	67 min
$O(n^2)$	1 ms	10s	28 uur	761 jaar
$O(n^3)$	1 ms	17 min	32 jaar	31710 eeuw
$O(10^n)$	1 ms	oneindig	oneindig	oneindig

Je ziet dat een $O(n^2)$ algoritme niet bruikbaar is voor grote hoeveelheden data en dat een $O(n^3)$ en $O(10^n)$ algoritme zo wie zo niet bruikbaar zijn.

naam	insert	remove	find	applicaties	implementaties
stack	push O(1)	pull O(1) LIFO	top O(1) LIFO	dingen omdraaien is ... gebalanceerd? evaluatie van expressies	array, statisch + snel linked list, dynamisch + meer overhead in space en time
queue	enqueue O(1)	dequeue O(1) FIFO	front O(1) FIFO	printer queue wachtrij	array, statisch + snel linked list, dynamisch + meer overhead in space en time
vector	O(1)	O(n) schuiven	O(n) op inhoud O(1) op index	vaste lijst code conversie	array, static random access via operator[]
sorted vector	O(n) zoeken + schuiven	O(n)	O(log n) op inhoud O(1) op index	lijst waarin je veel zoekt en weinig muteert	array, static + binary search algorithm
linked list	O(1)	O(n)	O(n)	dynamische lijst waarin je weinig zoekt en verwijdert	linked list, dynamic + more space overhead sequential access via iterator
sorted list	O(n)	O(n)	O(n)	dynamische lijst die je vaak gesorteerd afdrukt	
tree	O(log n)	O(n)	O(n)	meerdimensionale lijst file systeem expressie boom	more space overhead + minimal n+1 pointers with value 0
search tree	O(log n)	O(log n)	O(log n)	dynamische lijst waarin je veel muteert en zoekt	sorted binary tree, more space overhead than list
hash table	O(1)	O(1)	O(1)	symbol table (compiler) dictionary	semi-static, reduced performance if overfilled
priority queue	O(log n)	O(log n)	O(1)	event driven simulation	binary heap - array, static + little space overhead - binary tree, dynamic + space overh.