[Big-O Cheat Sheet]

- Searching
- Sorting
- Data Structures
- Heaps
- Graphs
- Chart
- Comments

Know Thy Complexities!

Hi there! This webpage covers the space and time Big-O complexities of common algorithms used in Computer Science. When preparing for technical interviews in the past, I found myself spending hours crawling the internet putting together the best, average, and worst case complexities for search and sorting algorithms so that I wouldn't be stumped when asked about them. Over the last few years, I've interviewed at several Silicon Valley startups, and also some bigger companies, like Yahoo, eBay, LinkedIn, and Google, and each time that I prepared for an interview, I thought to myself "Why oh why hasn't someone created a nice Big-O cheat sheet?". So, to save all of you fine folks a ton of time, I went ahead and created one. Enjoy!

Good Fair Poor

Searching

Algorithm	Data Structure	Time Complexity	Space Complexity

		Average	Worst	Worst
Depth First Search (DFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Breadth First Search (BFS)	Graph of |V| vertices and |E| edges	-	O(|E| + |V|)	O(|V|)
Binary search	Sorted array of n elements	O(log(n))	O(log(n))	O(1)
Linear (Brute Force)	Array	O(n)	O(n)	O(1)
Shortest path by Dijkstra, using a Min-heap as priority queue	Graph with |V| vertices and |E| edges	O((|V| + |E|) log |V|)	O((|V| + |E|) log |V|)	O(|V|)
Shortest path by Dijkstra, using an unsorted array as priority queue	Graph with |V| vertices and |E| edges	O(|V|^2)	O(|V|^2)	O(|V|)
Shortest path by Bellman-Ford	Graph with |V| vertices and |E| edges	O(|V||E|)	O(|V||E|)	O(|V|)

Sorting

Algorithm	Data Structure	Time Complexity			Worst Case Auxiliary Space Complexity
		Best	Average	Worst	Worst
Quicksort	Array	O(n log(n))	O(n log(n))	O(n^2)	O(n)
Mergesort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Heapsort	Array	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)	O(1)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)	O(1)
Bucket Sort	Array	O(n+k)	O(n+k)	O(n^2)	O(nk)
Radix Sort	Array	O(nk)	O(nk)	O(nk)	O(n+k)

Data Structures

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Indexing	Search	Insertion	Deletion	Indexing	Search	Insertion	Deletion	
Basic Array	O(1)	O(n)	-	-	O(1)	O(n)	-	-	O(n)
Dynamic Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartresian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

Heaps

Heaps	Time Complexity						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List (sorted)	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Binomial Heap	-	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))

Fibonacci Heap	-	O(1)	O(log(n))*	O(1)*	O(1)	O(log(n))*	O(1)

Graphs

Node / Edge Management	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	O(|V|+|E|)	O(1)	O(1)	O(|V| + |E|)	O(|E|)	O(|V|)
Incidence list	O(|V|+|E|)	O(1)	O(1)	O(|E|)	O(|E|)	O(|E|)
Adjacency matrix	O(|V|^2)	O(|V|^2)	O(1)	O(|V|^2)	O(1)	O(1)
Incidence matrix	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|V| · |E|)	O(|E|)

Notation for asymptotic growth

letter	bound	growth
(theta) Θ	upper and lower, tight[1]	equal[2]
(big-oh) O	upper, tightness unknown	less than or equal[3]
(small-oh) o	upper, not tight	less than
(big omega) Ω	lower, tightness unknown	greater than or equal
(small omega) ω	lower, not tight	greater than

[1] Big O is the upper bound, while Omega is the lower bound. Theta requires both Big O and Omega, so that's why it's referred to as a tight bound (it must be both the upper and lower bound). For example, an algorithm taking Omega(n log n) takes at least n log n time but has no upper limit. An algorithm taking Theta(n log n) is far preferential since it takes AT LEAST n log n (Omega n log n) and NO MORE THAN n log n (Big O n log n).SO

[2] f(x)=Θ(g(n)) means f (the running time of the algorithm) grows exactly like g when n (input size) gets larger. In other words, the growth rate of f(x) is asymptotically proportional to g(n).

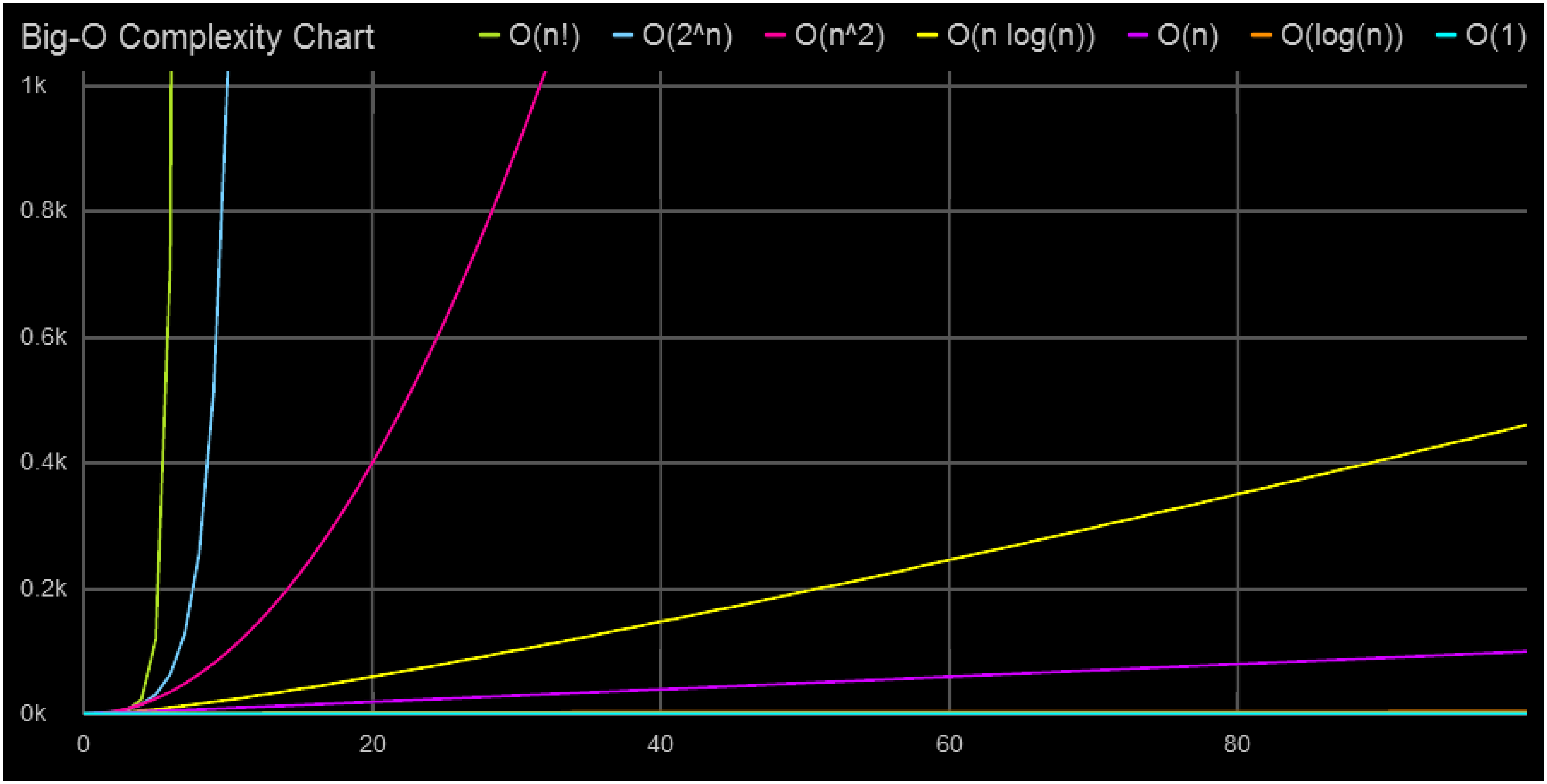
[3] Same thing. Here the growth rate is no faster than g(n). big-oh is the most useful because represents the worst-case behavior.

In short, if algorithm is __ then its performance is __

algorithm	performance
o(n)	< n
O(n)	≤ n
Θ(n)	= n
Ω(n)	≥ n
ω(n)	> n

Big-O Complexity Chart

This interactive chart, created by our friends over at MeteorCharts, shows the number of operations (y axis) required to obtain a result as the number of elements (x axis) increase. O(n!) is the worst complexity which requires 720 operations for just 6 elements, while O(1) is the best complexity, which only requires a constant number of operations for any number of elements.



Contributors

Edit these tables!

1. Eric Rowell
2. Quentin Pleple
3. Nick Dizazzo
4. Michael Abed