Complexities of Data Structures and Algorithms

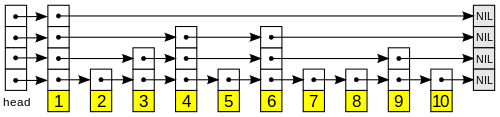
This page covers the Space and Time Big-O complexities of common algorithms widely used in Computer Science and Information Technology.  When preparing for technical interviews in the past, one can find spending hours crawling the internet putting together the best, average, and worst case complexities for search and sorting algorithms.  Over the last few years, IT experts interviewed at several Silicon Valley start-ups, and also some bigger companies, like Yahoo, eBay, LinkedIn, and Google, and each preparation time of interview, one can think surely that "Why someone hasn't created a nice Big-O summery sheet of complexity analysis so that one can save a lot of preparation time.

Data Structure Operations

| **Data Structure** | **Time Complexity** | | | | | | | | **Space Complexity** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Average** | | | | **Worst** | | | | **Worst** |
|  | **Access** | **Search** | **Insertion** | **Deletion** | **Access** | **Search** | **Insertion** | **Deletion** |  |
| [Array](http://en.wikipedia.org/wiki/Adjacency_list) | O(1) | O(n) | O(n) | O(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| [Stack](http://en.wikipedia.org/wiki/Binary_heap) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Singly-Linked List](http://en.wikipedia.org/wiki/Bucket_sort#Singly_linked_lists) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Doubly-Linked List](http://en.wikipedia.org/wiki/Doubly_linked_list) | O(n) | O(n) | O(1) | O(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| [Skip List](http://en.wikipedia.org/wiki/Linked_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](http://en.wikipedia.org/wiki/Array_data_structure) | - | O(1) | O(1) | O(1) | - | O(n) | O(n) | O(n) | O(n) |
| [Binary Search Tree](http://en.wikipedia.org/wiki/Linked_list) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |
| [Cartesian Tree](http://en.wikipedia.org/wiki/Singly_linked_list) | - | O(log(n)) | O(log(n)) | O(log(n)) | - | O(n) | O(n) | O(n) | O(n) |
| [B-Tree](http://en.wikipedia.org/wiki/Skip_list) | 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](http://en.wikipedia.org/wiki/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](http://en.wikipedia.org/wiki/Hash_table) | - | O(log(n)) | O(log(n)) | O(log(n)) | - | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| [AVL Tree](http://en.wikipedia.org/wiki/Stack_(abstract_data_type)) | 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) |

***Skip lists*** *are a probabilistic data structure that seem likely to supplant balanced trees as the implementation method of choice for many applications. Skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.*

***Skip List****is a*[*data structure*](https://en.wikipedia.org/wiki/Data_structure)*that allows fast search within an*[*ordered sequence*](http://en.wikipedia.org/wiki/B_tree)*of elements. Fast search is made possible by maintaining a*[*linked*](https://en.wikipedia.org/wiki/Linked_list)*hierarchy of subsequences, each skipping over fewer elements. Searching starts in the sparsest subsequence until two consecutive elements have been found, one smaller and one larger than the element searched for. Via the linked hierarchy these two elements link to elements of the next sparsest subsequence where searching is continued until finally we are searching in the full sequence. The elements that are skipped over may be chosen probabilistically or deterministically with the former being more common.*



Above schematic picture shows the skip list data structure. Each box with an arrow represents a pointer and a row is a [linked list](http://en.wikipedia.org/wiki/Binomial_heap) giving a sparse subsequence; the numbered boxes at the bottom represent the ordered data sequence. Searching proceeds downwards from the sparsest subsequence at the top until consecutive elements bracketing the search element are found.

**Array Sorting Algorithms**

| **Algorithm** | **Time Complexity** | | | **Space Complexity** |
| --- | --- | --- | --- | --- |
|  | **Best** | **Average** | **Worst** | **Worst** |
| [Quick Sort](http://en.wikipedia.org/wiki/Quicksort) | O(n log(n)) | O(n log(n)) | O(n^2) | O(log(n)) |
| [Merge Sort](https://en.wikipedia.org/wiki/Splay_tree) | O(n log(n)) | O(n log(n)) | O(n log(n)) | O(n) |
| [Tim Sort](http://en.wikipedia.org/wiki/AVL_tree) | O(n) | O(n log(n)) | O(n log(n)) | O(n) |
| [Heap Sort](https://en.wikipedia.org/wiki/Cartesian_tree) | O(n log(n)) | O(n log(n)) | O(n log(n)) | O(1) |
| [Bubble Sort](https://en.wikipedia.org/wiki/Ordered_sequence) | O(n) | O(n^2) | O(n^2) | O(1) |
| [Insertion Sort](http://en.wikipedia.org/wiki/Insertion_sort) | O(n) | O(n^2) | O(n^2) | O(1) |
| [Selection Sort](http://en.wikipedia.org/wiki/Merge_sort) | O(n^2) | O(n^2) | O(n^2) | O(1) |
| [Shell Sort](http://en.wikipedia.org/wiki/Timsort) | O(n) | O((n.log(n))^2) | O((n.log(n))^2) | O(1) |
| [Bucket Sort](https://en.wikipedia.org/wiki/Linked_list) | O(n+k) | O(n+k) | O(n^2) | O(n) |
| [Radix Sort](http://en.wikipedia.org/wiki/Selection_sort) | O(nk) | O(nk) | O(nk) | O(n+k) |

**Graph Operations**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Node / Edge Management** | **Storage** | **Add Vertex** | **Add Edge** | **Remove Vertex** | **Remove Edge** | **Query** |
| [Adjacency List](http://en.wikipedia.org/wiki/Binary_search_tree) | O(|V|+|E|) | O(1) | O(1) | O(|V| + |E|) | O(|E|) | O(|V|) |
| [Incidence List](http://en.wikipedia.org/wiki/Heapsort) | O(|V|+|E|) | O(1) | O(1) | O(|E|) | O(|E|) | O(|E|) |
| [Adjacency Matrix](http://en.wikipedia.org/wiki/Bubble_sort) | O(|V|^2) | O(|V|^2) | O(1) | O(|V|^2) | O(1) | O(1) |
| [Incidence Matrix](http://en.wikipedia.org/wiki/Incidence_matrix) | O(|V|⋅|E|) | O(|V| ⋅ |E|) | O(|V|⋅ |E|) | O(|V|⋅ |E|) | O(|V|⋅ |E|) | O(|E|) |

**Heap Operations**

| **Type** | **Time Complexity** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Heapify** | **Find Max** | **Extract Max** | **Increase Key** | **Insert** | **Delete** | **Merge** |  |
| [Linked List (Sorted)](http://en.wikipedia.org/wiki/Radix_sort) | - | O(1) | O(1) | O(n) | O(n) | O(1) | O(m+n) |  |
| [Linked List (Unsorted)](http://en.wikipedia.org/wiki/Shellsort) | - | O(n) | O(n) | O(1) | O(1) | O(1) | O(1) |  |
| [Binary Heap](http://en.wikipedia.org/wiki/Incidence_list) | O(n) | O(1) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(m+n) |  |
| [Binomial Heap](http://en.wikipedia.org/wiki/Adjacency_matrix) | - | O(1) | O(log(n)) | O(log(n)) | O(1) | O(log(n)) | O(log(n)) |  |
| [Fibonacci Heap](http://en.wikipedia.org/wiki/Fibonacci_heap) | - | O(1) | O(log(n)) | O(1) | O(1) | O(log(n)) | O(1) |  |

**Big-O Complexity Chart**

