Cheatography

Java Data Structures Cheat Sheet

by Ieternalleo via cheatography.com/45716/cs/13401/

Array

Definition	- Stores data elements based on an sequential, most commonly 0 based, index. - Based on [tuples](http://en.wikipedia.org/wiki/Tuple) from set theory. - They are one of the oldest, most commonly used data structures.
Details	- Optimal for indexing; bad at searching, inserting, and deleting (except at the end). - **Linear arrays**, or one dimensional arrays, are the most basic. - Are static in size, meaning that they are declared with a fixed size. - **Dynamic arrays** are like one dimensional arrays, but have reserved space for additional elements. - If a dynamic array is full, it copies it's contents to a larger array. - **Two dimensional arrays** have x and y indices like a grid or nested arrays.
Big-O efficiency	- Indexing: Linear array: O(1), Dynamic array: O(1) - Search: Linear array: O(n), Dynamic array: O(n) - Optimized Search: Linear array: O(log n), Dynamic array: O(log n) - Insertion: Linear array: n/a Dynamic array: O(n)

Linked List

Definition	- Stores data with **nodes** that point to other nodes. - Nodes, at its most basic it has one datum and one reference (another node). - A linked list _chains_ nodes together by pointing one node's reference towards another node.
Details	- Designed to optimize insertion and deletion, slow at indexing and searching. - **Doubly linked list** has nodes that reference the previous node. - **Circularly linked list** is simple linked list whose **tail**, the last node, references the **head**, the first node. - **Stack**, commonly implemented with linked lists but can be made from arrays too. - Stacks are **last in, first out** (LIFO) data structures. - Made with a linked list by having the head be the only place for insertion and removal. - **Queues**, too can be implemented with a linked list or an array. - Queues are a **first in, first out**(FIFO) data structure. - Made with a doubly linked list that only removes from head and adds to tail.
Big-O efficiency	- Indexing: Linked Lists: O(n) - Search: Linked Lists: O(n) - Optimized Search: Linked Lists: O(n) - Insertion: Linked Lists: O(1)

Hash Map

Definition	- Stores data with key value pairs. - **Hash functions** accept a key and return an output unique only to that specific key. - This is known as **hashing**, which is the concept that an input and an output have a one-to-one correspondence to map information. - Hash functions return a unique address in memory for that data.
Details	- Designed to optimize searching, insertion, and deletion. - **Hash collisions** are when a hash function returns the same output for two distinct inputs. - All hash functions have this problem. - This is often accommodated for by having the hash tables be very large. - Hashes are important for associative arrays and database indexing.
Big-O efficiency	- Indexing: Hash Tables: O(1) - Search: Hash Tables: O(1) - Insertion: Hash Tables: O(1)

Binary Tree

Definition	- Is a tree like data structure where every node has at most two children. - There is one left and right child node.

Binary Tree (cont)

Details	- Designed to optimize searching and sorting. - A **degenerate tree** is an unbalanced tree, which if entirely one-sided is a essentially a linked list. - They are comparably simple to implement than other data structures. - Used to make **binary search trees**. - A binary tree that uses comparable keys to assign which direction a child is. - Left child has a key smaller than it's parent node. - Right child has a key greater than it's parent node. - There can be no duplicate node. - Because of the above it is more likely to be used as a data structure than a binary tree. An **AVL Tree** is a balanced binary search tree. -The process for inserting or deleting is the same as in a regular(unbalanced) **BST**, except you have to rebalance after each operation. A node in an AVL tree is balanced if its **balance factor** is either **-1,0,** or **1**
Big-O efficiency	- Indexing: Binary Search Tree: O(log n) - Search: Binary Search Tree: O(log n) - Insertion: Binary Search Tree: O(log n)

The **balance factor** of a node is the height of its right subtree minus the height of its left subtree

Search Basics

Breadth First Search

Definition	- An algorithm that searches a tree (or graph) by searching levels of the tree first, starting at the root. - It finds every node on the same level, most often moving left to right. - While doing this it tracks the children nodes of the nodes on the current level. - When finished examining a level it moves to the left most node on the next level. - The bottom-right most node is evaluated last (the node that is deepest and is farthest right of it's level).
Details	- Optimal for searching a tree that is wider than it is deep. - Uses a queue to store information about the tree while it traverses a tree. - Because it uses a queue it is more memory intensive than **depth first search**. - The queue uses more memory because it needs to stores pointers
Big-O efficiency	- Search: Breadth First Search: O(|E| + |V|) - E is number of edges - V is number of vertices

Depth First Search

Definition	- An algorithm that searches a tree (or graph) by searching depth of the tree first, starting at the root. - It traverses left down a tree until it cannot go further. - Once it reaches the end of a branch it traverses back up trying the right child of nodes on that branch, and if possible left from the right children. - When finished examining a branch it moves to the node right of the root then tries to go left on all it's children until it reaches the bottom. - The right most node is evaluated last (the node that is right of all it's ancestors).

Depth First Search (cont)

Details	- Optimal for searching a tree that is deeper than it is wide. - Uses a stack to push nodes onto. - Because a stack is LIFO it does not need to keep track of the nodes pointers and is therefore less memory intensive than breadth first search. - Once it cannot go further left it begins evaluating the stack.
Big-O efficiency	- Search: Depth First Search: O(|E| + |V|) - E is number of edges - V is number of vertices

Breadth First Search Vs. Depth First Search

- The simple answer to this question is that it depends on the size and shape of the tree.
- For wide, shallow trees use Breadth First Search
- For deep, narrow trees use Depth First Search

Nuances:

- Because BFS uses queues to store information about the nodes and its children, it could use more memory than is available on your computer. (But you probably won't have to worry about this.)
- If using a DFS on a tree that is very deep you might go unnecessarily deep in the search. See [xkcd](http://xkcd.com/761/) for more information.
- Breadth First Search tends to be a looping algorithm.
- Depth First Search tends to be a recursive algorithm.

Efficient Sorting Basics

Merge Sort

Definition	- A comparison based sorting algorithm - Divides entire dataset into groups of at most two. - Compares each number one at a time, moving the smallest number to left of the pair. - Once all pairs sorted it then compares left most elements of the two leftmost pairs creating a sorted group of four with the smallest numbers on the left and the largest ones on the right. - This process is repeated until there is only one set.
Details	- This is one of the most basic sorting algorithms. - Know that it divides all the data into as small possible sets then compares them.
Big-O efficiency	- Best Case Sort: Merge Sort: O(n) - Average Case Sort: Merge Sort: O(n log n) - Worst Case Sort: Merge Sort: O(n log n)

Quicksort

Definition	- A comparison based sorting algorithm - Divides entire dataset in half by selecting the average element and putting all smaller elements to the left of the average. - It repeats this process on the left side until it is comparing only two elements at which point the left side is sorted. - When the left side is finished sorting it performs the same operation on the right side. - Computer architecture favors the quicksort process.

Quicksort (cont)

Details	- While it has the same Big O as (or worse in some cases) many other sorting algorithms it is often faster in practice than many other sorting algorithms, such as merge sort. - Know that it halves the data set by the average continuously until all the information is sorted.
Big-O efficiency	- Best Case Sort: Merge Sort: O(n) - Average Case Sort: Merge Sort: O(n log n) - Worst Case Sort: Merge Sort: O(n^2)

Bubble Sort

Definition	- A comparison based sorting algorithm - It iterates left to right comparing every couplet, moving the smaller element to the left. - It repeats this process until it no longer moves and element to the left.
Details	- While it is very simple to implement, it is the least efficient of these three sorting methods. - Know that it moves one space to the right comparing two elements at a time and moving the smaller on to left.
Big-O efficiency	- Best Case Sort: Merge Sort: O(n) - Average Case Sort: Merge Sort: O(n2) - Worst Case Sort: Merge Sort: O(n2)

Merge Sort vs. QuickSort

- Quicksort is likely faster in practice.
- Merge Sort divides the set into the smallest possible groups immediately then reconstructs the incrementally as it sorts the groupings.
- Quicksort continually divides the set by the average, until the set is recursively sorted.

Heap Sort

Definition:	Sorts using a complete binary Tree.
Details:	**ArrayList** can be used to store a Heap For a node of **i**: -Left child: 2**i**+1 -Right child: 2**i**+2 -Parent: (**i** - 1)/2
Big-O:	O(nlogn)