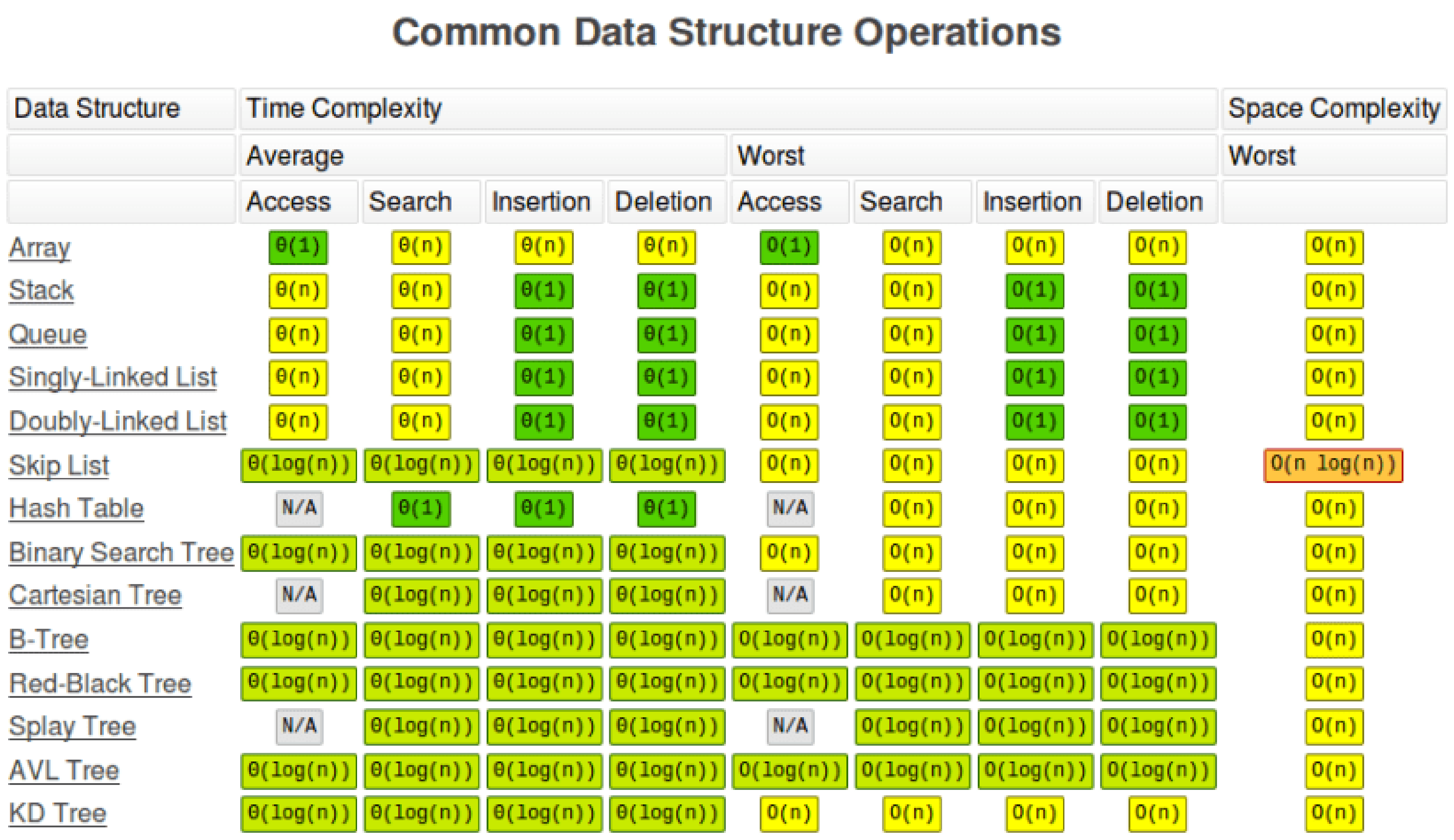
# Language Map for C#

| | |
|---|---|
| **Variable Declaration**<br>*Is this language strongly typed or dynamically typed? Provide an example of how variables are declared in this language.* | Strongly Typed<br>type name = value;<br>string s1 = “hello”;<br>int i = 0;<br>double dubs = 2.0; |
| **Data Types**<br>*List all of the data types (and ranges) supported by this language.* | Int - 4bytes \| bool 1 bit<br>Long - 8 bytes \| char 2 bytes<br>Float – 4 bytes \| String 2 per char<br>Double – 8 bytes |
| **Selection Structures**<br>*Provide examples of all selection structures supported by this language (if, if else, etc.)* | If/end<br>If/else<br>If/else if |
| **Repetition Structures**<br>*Provide examples of all repetition structures supported by this language (loops, etc.)* | While<br>Do while<br>For<br>Foreach |
| **Arrays**<br>*If this language supports arrays, provide an example of creating an array with a primitive data type (e.g. float, int, etc.)* | Type[] name = new type[n];<br>Int[][] matrix1 = new int[2][2];<br>Int[] a2 = {1,2,4,5,65,65,}<br>Found a thing called a jagged array. Basically just a matrix, but in the second column, you put references to other array obj. int [][] jaggedArray = new int[3][]; → jaggedArray[0] = new int[5]; etc |
| **Data Structures**<br>*If this language provides a standard set of data structures, provide a list of the data structures and their Big-Oh complexity.* | *Big O in search request form.<br>Array – O(n)<br>Stack - O(n)<br>Queue - O(n)<br>Linked List - O(n)<br>Hashtable - O(n)<br>Binary Search Tree - O(n)<br>- Seems like there’s some more structures, but I haven’t ever used these so far, but I’ll list them anyway. I’ll write about them below the graph.<br>Cartesian Tree – O(n)<br>Red-Black Tree – O(log n) |

| | |
|---|---|
| | Splay Tree – O(log n)<br>AVL tree – O(log n)<br>KD Tree – O(n) |
| **Objects**<br>*If this language support object-orientation, provide an example of how to create a simple object with a default constructor.* | Public Sword(int metalType, int hp) {<br>Durability = metalType;<br>Attack = hp;<br>}//end sword |
| **Runtime Environment**<br>*What runtime environment does this language compile to? For example, Java compiles to the Java Virtual Machine.*<br>*Do other languages also compile to this runtime?* | Common Language Runtime<br>There's some others that compile to it, examples: Managed C++, C#, Visual Basic, and J# |
| **Libraries/Frameworks**<br>*What are the popular libraries or frameworks used by programmers for this language? List at least three (3).* | Xamarin – app platform<br>Data Structures – Array. List, Dictionary<br>Mono |
| **Domains**<br>*What industries or domains use this programming language? Provide specific examples of companies that use this language and what they use it for.* | Stock Analysis - https://github.com/soross/stockanalyzer<br>Unity Game Engine uses C#, so games like Cuphead, Ori and the Blind Forest, Hollow Knight, Hearthstone<br>Microsoft of course LOL |

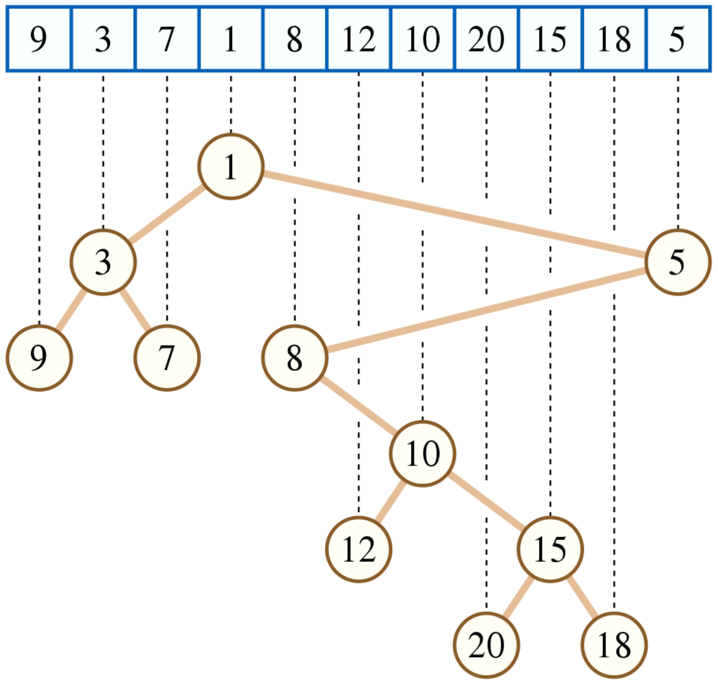


Common Data Structure Operations

| Data Structure | Time Complexity | | | | | | | | Space Complexity |
|---|---|---|---|---|---|---|---|---|---|
| | Average | | | | Worst | | | | Worst |
| | Access | Search | Insertion | Deletion | Access | Search | Insertion | Deletion | |
| Array | Θ(1) | Θ(n) | Θ(n) | Θ(n) | O(1) | O(n) | O(n) | O(n) | O(n) |
| Stack | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Queue | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Singly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Doubly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | O(n) | O(n) | O(1) | O(1) | O(n) |
| Skip List | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n log(n)) |
| Hash Table | N/A | Θ(1) | Θ(1) | Θ(1) | N/A | O(n) | O(n) | O(n) | O(n) |
| Binary Search Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |
| Cartesian Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(n) | O(n) | O(n) | O(n) |
| B-Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Red-Black Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| Splay Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| AVL Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(log(n)) | O(n) |
| KD Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) |

Cool chart. Link: https://dev.to/adavidoaiei/fundamental-data-structures-and-algorithms-in-c-4ocf

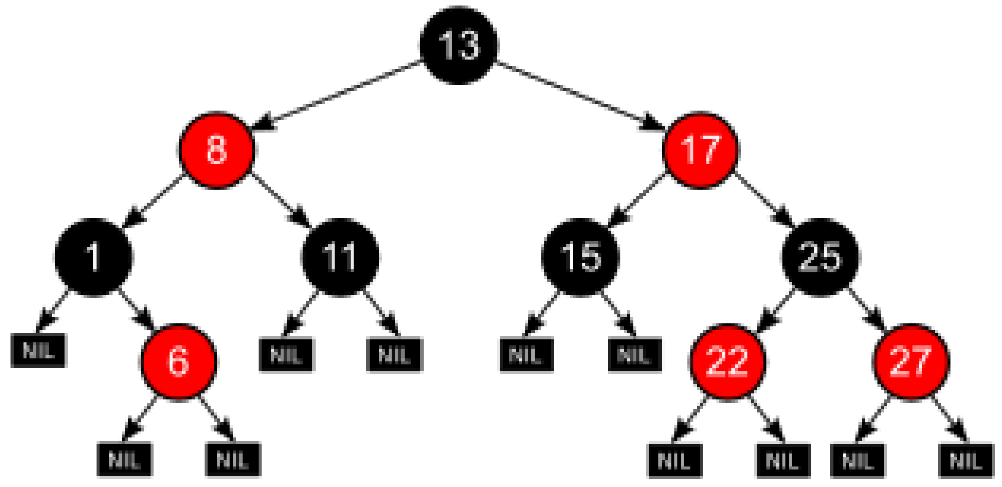
Apparently this is a Cartesian Tree



Not sure what you'd use this for to be honest. You can make subgraphs out of larger linked lists maybe, but that feels a little redundant.

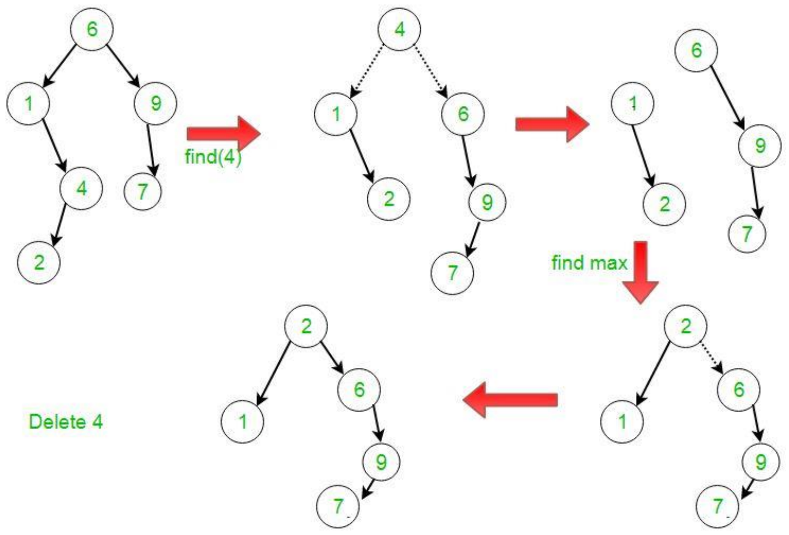
B-tree is just a self balancing tree. m/2 for the leaves. Cool algorithm.

Here’s a Red-Black Tree:



This is pretty cool. It reminds me of combinatorics last semester in the chromatic graphs. It’s another form of self-balancing using combinatorics which is pretty cool cause that class killed me

Splay trees prioritize recently accessed items, which is cool. It moves them to the top periodically, reshuffling the whole tree. Sounds inefficient, but Wikipedia says nah it’s fine because it’s constantly calculating Big O through amortized time. Learned a word!

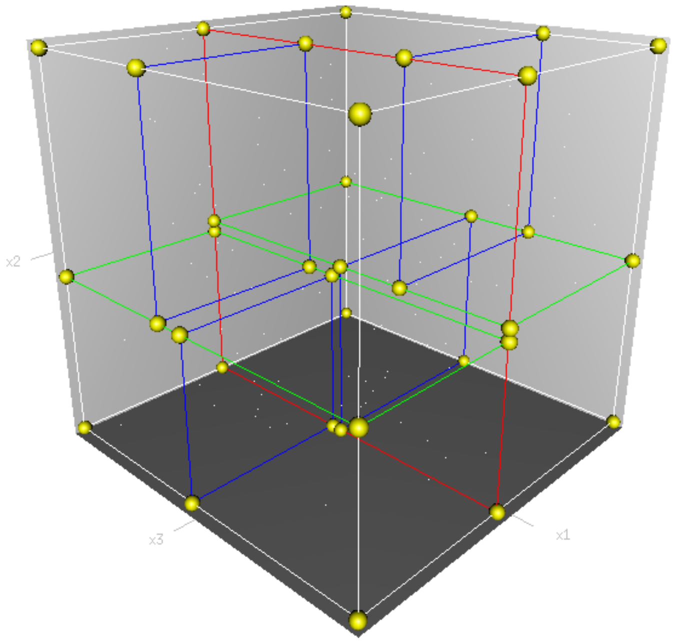


Here’s an AVL tree. We went over these I think, but it’s too vague in my brain so I’m gonna recap it. ’62, better than Red-Blac, soviet made algorithm that are height based, self balancing trees. They reorganize themselves through this formula

BF(X) := Height(RightSubtree(X))- Height(LeftSubtree(X)), which looks complex but it’s literally just a subtraction. Cool!

KD Tree

This is messed up, someone made a 3d object out of an array get me outta here. It technically can exceed the 3rd dimension, but dear lord what do you need this for, NASA get me outta here. I'm just gonna copy paste the Wikipedia entry.



The *k*-d tree is a binary tree in which *every* node is a *k*-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree. The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the *k* dimensions, with the hyperplane perpendicular to that dimension's axis. So, for example, if for a particular split the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with a larger "x" value will be in the right subtree. In such a case, the hyperplane would be set by the x value of the point, and its normal would be the unit x-axis.[1]

Make box.