There are a few important ones in my opinion:

1) Segment trees — those are very important. There's a lot of confusion with the name, so I do not know how to call them exactly, but I do [**NOT MEAN THIS**](http://en.wikipedia.org/wiki/Segment_tree) but rather what is explained [here](https://codeforces.com/blog/entry/3327) (I couldn't find a better link, this is not the best explanation/implementation so you might want to refer to the comments of that blog). Basically segment trees allow you to work fast when you have queries on segments of an array. There is lots to learn about it, ranging from lazy propagation technique, to complicated things you must keep in each node in order to merge them successfully.

2) Binary search tree — those are obviously great. Map and set are two STL structures (in C++) that can often save you the time of coding BST yourself, but they often don't use the tree's full potential. It often is required to create one yourself. The most famous BST I know of are AVL, Red-Black, Treap and Splay. In almost every case, they are mutually replaceable!

[AVL Tree](http://en.wikipedia.org/wiki/AVL_tree) — Most basic binary search tree that uses rotations to balance itself. The complexity is worst-case O(log) per query.

[Red-black Tree](http://en.wikipedia.org/wiki/Red%E2%80%93black_tree) — This tree is similar to AVL, but it uses colouring of the vertices to achieve better worktime. It uses rotations to balance itself once again and is O(log) in worst-case per query.

[Treap](http://en.wikipedia.org/wiki/Treap) — This is actually a very clever binary search tree and my personal favourite. The structure uses randomization of the tree to build it and there are a few techniques to keep it balanced — usually involving randomization. It is theoretical worst-case O(n), but if you use randomization it is O(log) on average, and the worst-case is practically never achieved.

[Splay tree](http://en.wikipedia.org/wiki/Splay_tree) — This one is interesting and it has complexity of amortized O(log) per query. I do not use it so I don't know much about it, but in some specific cases it actually performs really well.

3) Heaps — heaps are lovely structures to get min/max and there are actually quite a few of them. Knowing the regular heap is usually enough, but out of curiosity or desire for crazy optimizations, you might want to read about more of them:

[Binary heap](http://en.wikipedia.org/wiki/Binary_heap) (regular heap) ; [Binomial heap](http://en.wikipedia.org/wiki/Binomial_heap) ; (Strict and not) [Fibonacci heap](http://en.wikipedia.org/wiki/Fibonacci_heap) ; [Pairing heap](http://en.wikipedia.org/wiki/Pairing_heap) ; [Brodal queue](http://en.wikipedia.org/wiki/Brodal_queue) (theoretically optimal, though its author claims it's too complicated for practical use)

[And in general wikipedia has a good article on heaps.](http://en.wikipedia.org/wiki/Heap_%28data_structure%29)

This is all I can think of now, those are the tree-structures I use mostly. Each of them has tons of applications and details, and lots of them are mutually replaceable. Some of them like segment trees can be static or dynamic, and most of them have a persistent variation.

Hope that helps :)