1 Time Complexity

Master's theorem for $T(n) = aT(\frac{n}{b}) + f(n)$ where $a \geq 1$ and $b > 1$

Let $c_{crit} = log_b(a)$ and if $f(n) = \theta(n^c)$

1. If $c < c_{crit}$ then $T(n) = \theta(n^{c_{crit}})$
2. If $c = c_{crit}$ then $T(n) = \theta(n^c log(n))$
3. If $c > c_{crit}$ then $T(n) = \theta(f(n))$
4. If $f(n) = \theta(n^{c_{crit}} log^k(n))$, then $T(n) = \theta(n^{c_{crit}} log^{k+1}(n))$

2 Sorting

2.1 Bubblesort

Time complexity: $\Omega(n)$ $\theta(n^2)$ $O(n^2)$
Invariant: At the end of iteration `j`, the biggest `j` items are correctly sorted in the final `j` positions of the array.

2.2 SelectionSort

Time complexity: $\Omega(n^2)$ $\theta(n^2)$ $O(n^2)$
Invariant: At the end of iteration `j`: the smallest `j` items are correctly sorted in the first `j` positions of the array.

2.3 InsertionSort

Time complexity: $\Omega(n)$ $\theta(n^2)$ $O(n^2)$
Invariant: At the end of iteration `j`: the first `j` items of the array are sorted in order.

```
for (x = 1 to arr.len):
j = x − 1
store = arr[x]
while(j >= 0 && arr[j] > store):
    arr[j + 1] = arr [j]
    j−−
arr[j+1] = store
```

2.4 QuickSort

Time complexity: $\Omega(nlog(n))$ $\theta(nlog(n))$ $O(n^2)$ $O(nlog(n))$ (for paranoid QuickSort)
Invariant: For every `i < low`: `B[i] < pivot` and for every `j > high`: `B[j] > pivot`. Duplicates: Use three-way partition to store duplicates.

2.5 MergeSort

Time complexity: $\Omega(nlog(n))$ $\theta(nlog(n))$ $O(nlog(n))$
Invariant: At the end of each loop, the subarrays are sorted.

2.6 QuickSelect

Time complexity: $\Omega(n)$ $\theta(n)$ $O(n^2)$
Invariant: For every `i < low`: `B[i] < pivot` and for every `j > high`: `B[j] > pivot`.

2.7 TopoSort

Time complexity: $O(V + E)$

3 Trees

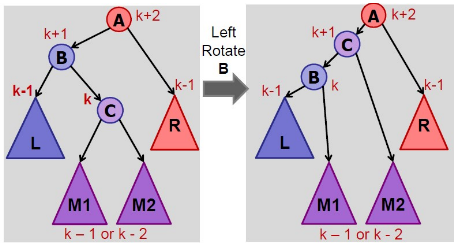
3.1 Binary Search Tree (BST)

Only has 2 children per node. Left child is smaller than parent. Right child is larger than parent.
Operations: $O(log(n))$ / $O(n)$ (balanced)

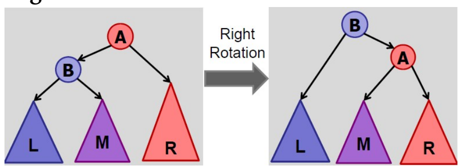
3.2 AVL Tree

Maximum height between two children tree is 1.
Operations: $O(log(n))$

Left Rotation:



Right Rotation:



Insertion Max Rotations: 2
Deletion Max Rotations: $log(n)$
Invariant: `|height(u) - height(v)| < 2` if v and u are sibling nodes. `|height(u) - height(v)| > 0` if u is the parent node of v.

3.3 Order Statistics (Rank finding)

AVL Tree augmented with a `weight` property in each node.

Updating weights on rotate: $O(1)$

```
rank = weight(node.right) + 1
while(node != null && node.parent != null):
    if (node.parent.left == node):
        rank += weight(node.parent.right) + 1
        node = node.parent

FindRanks(currentRank, node, targetRank):
    check = currentRank + weight(node.left) + 1
    if (check == targetRank):
        return node
    elif (check < targetRank):
        return FindRanks(currentRank, node.right,
            targetRank)
    else:
        return FindRanks(currentRank, node.left,
            targetRank)
```

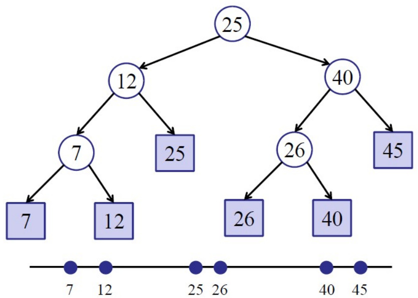
3.4 Interval Trees

Store the maximum value of entire subtree under a node as a parameter. Update `max` during rotations by taking `Math.max` of all children under the two nodes being swapped.
Time complexity: $O(log(n))$

```
interval−search(x)
    c = root;
    while (c != null and x is not in c.interval) do
        if (c.left == null) then
            c = c.right;
        else if (x > c.left.max) then
            c = c.right;
        else c = c.left;
            return c.interval;
```

3.5 1D Range Finding

All leaves hold the value, each internal node v stores the `MAX` of any leaf in the left sub-tree.



Operations:

`findSplit`	Finds the node to start searching.
`traverseLeft`	Traverse the left subtree after findSplit.
`traverseRight`	Traverse the right subtree after findSplit.

Complexity:

`Query`	$O(k + log(n))$
`Build`	$O(nlog(n))$
`Space`	$O(n)$

3.6 (a, b) trees

An (a, b) tree has a bound of $2 \leq a \leq \frac{b+1}{2}$.

Rules:

Node Type	#Keys		#Children	
	Min	Max	Min	Max
Root	1	b - 1	2	b
Internal	a - 1	b - 1	a	b
Leaf	a - 1	b - 1	0	0

4 Hashing

5 Heaps

6 Graphs

7 Dynamic Programming