

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.

2.4 QuickSort

Time complexity: $\Omega(n \log(n)) \theta(n \log(n)) O(n^2)$
 $O(n \log(n))$ (for paranoid QuickSort)

Invariant: For every $i < \text{low}$: $B[i] < \text{pivot}$ and for every $j > \text{high}$: $B[j] > \text{pivot}$. Duplicates: Use three-way partition to store duplicates.

2.5 MergeSort

Time complexity: $\Omega(n \log(n)) \theta(n \log(n)) O(n \log(n))$

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 $i < \text{low}$: $B[i] < \text{pivot}$ and for every $j > \text{high}$: $B[j] > \text{pivot}$.

3 Trees

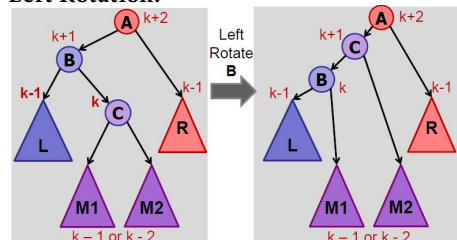
3.1 Binary Search Tree (BST)

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)

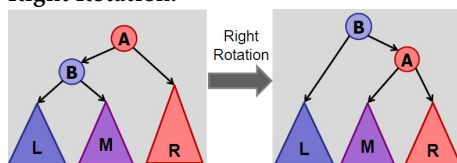
3.2 AVL Tree

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: $|\text{height}(u) - \text{height}(v)| < 2$ if v and u are sibling nodes. $|\text{height}(u) - \text{height}(v)| > 0$ 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: $O(1)$

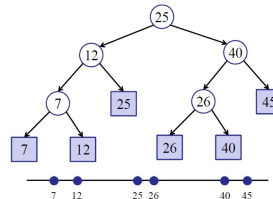
3.4 Interval Trees

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: $O(\log(n))$

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 $O(n \log(n))$
- Space $O(n)$

3.6 (a, b) trees

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

Rule 1:

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

Rule 2: A non-leaf node must have one more child than its number of keys.

Rule 3: All leaf nodes must be at the same depth (from the root).

- search** Traverse from the root node and binary search the node array. ($O(\log(n))$).
- insert** Traverse from the root to find point of insertion. Split if needed ($O(\log(n))$).
- delete** Traverse from the root to find node and delete. Sacrifice/merge/split if needed ($O(\log(n))$).
- split** Find the median element in the node array and sacrifice to the parent. The split halves are now the child of the promoted node.
- merge** Merges two siblings. Delete the parent and add the parent to the keylist of one child. Merge the two children.
- share** Merge and split combined.

4 Hashing

4.1 Hashing with Chaining

If a current bucket is occupied, add to the linked list in the bucket.

- Insert:** $O(1 + \text{cost}(f))$
- Search/Delete:** $O(n + \text{cost}(f))$ where n is the number of nodes.
- Expected Number of items per bucket = $\frac{n}{m}$.
- Under Simple Uniform Hashing Assumption (every key is likely to be mapped to every permutation), $E(\text{search}) = O(1)$.
- Can still add items when $m == n$ and search efficiently.

4.2 Hashing with Open Addressing

If the current bucket is full, find the next available bucket (Linear Probing).

- Insert/Delete/Search:** $O(1)$ if $\alpha < 1$ where $\alpha = \frac{n}{m}$ where n is the #items and m is #buckets.
- When deleting a key, replace the bucket with DELETED. Functions will probe past DELETED.
- Clusters of size $\Theta(\log(n))$ form when table is $\frac{1}{4}$ full. Caching of arrays make this fast.
- Unable to insert and search efficiently when $m == n$.
- Expected cost of operation is $\leq \frac{1}{1-\alpha}$.

4.3 Double Hashing

Two ordinary hash functions $f(k)$ and $g(k)$ with new hash function:

$$h(k, i) = f(k) + i \cdot g(k) \bmod m$$

$g(k)$ must be relatively prime to m for $h(k, i)$ to hit all buckets.

4.4 Hashcode

Integer	The int value itself.
Long	Split the long into 32 bits, XOR.
String	Iterate through each char, sum the value with an offset.

4.5 Resizing

If $(n == m)$, then $m = 2m$ and if $(n < \frac{m}{4})$, then $m = \frac{m}{2}$.

Definition 4.1 Operation has amortized cost $T(n)$ if for every integer k , the cost of k operations is $\leq kT(N)$.

4.6 HashSet

- Stores a bit instead of key.
- $P(\text{false positive}) = 1 - (1 - \frac{1}{n})^n \approx 1 - (\frac{1}{e})^{\frac{n}{m}}$.

4.7 Bloom Filter

- Use 2 hash functions $f(k)$ and $g(k)$. Both buckets must return True for a positive match.
- $P(\text{false positive}) = (1 - \frac{1}{e^{\frac{2n}{m}}})^2$.
- $P(\text{bit is 0}) = (1 - \frac{1}{m})^{kn} \approx e^{-kn/m}$.
- $P(\text{collision at 1 spot}) = 1 - e^{-kn/m}$.
- $P(\text{collision at 1 spot}) = (1 - e^{-kn/m})^k$

5 Graphs

Definition 5.1 A graph consists of at least a node, and can consist of edges that connect nodes in the graph.

Definition 5.2 A connected graph has all nodes connected by a path.

Definition 5.3 The degree of a node is the number of adjacent edges on that node.

Definition 5.4 The degree of a graph is the maximum number of adjacent edges in the graph.

Definition 5.5 Diameter is the maximum distance between two nodes following the shortest path.

Definition 5.6 Star graph has all nodes connected to a central node.

Definition 5.7 Clique is a complete graph with diameter 1 and degree $n - 1$.

Definition 5.8 A cycle is a connected graph with a loop. It has a diameter $\frac{n}{2}$ or $\frac{n}{2} - 1$ and degree 2.

Definition 5.9 A bipartite graph is a graph with nodes divided into two sets. There are no edges between nodes of the same set.

5.1 Representation

- Adjacency List. Each node has a linked list of edges to other nodes. ($O(V + E)$ memory).
- Adjacency Matrix. 2D array that stores path from a node to another. ($O(V^2)$ memory).

5.2 Traversal

- BFS. Use a queue to explore neighbours level by level.
- DFS. Use a stack to explore the maximum depth, then neighbours.

6 Heaps

7 Dynamic Programming