Algorithmic Analysis

Big O: $f(n) \in O(g(n)) \longleftrightarrow \exists c \in \mathbb{R}, n_0 \in \mathbb{Z}^{\geq 0}$ s.t. $\forall n \geq n_0, f(n) \leq c \cdot g(n)$

Big Ω: $f(n) \in \Omega(g(n)) \longleftrightarrow \exists c \in \mathbb{R}, n_0 \in \mathbb{Z}^{\geq 0}$ s.t. $\forall n \geq n_0, f(n) \geq c \cdot g(n)$

Big Θ: $f(n) \in \Theta(g(n)) \longleftrightarrow f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n))$

$$\longleftrightarrow \exists c_1, c_2 \in \mathbb{R}, n_0 \in \mathbb{Z}^{\geq 0} \text{ s.t. } \forall n \geq n_0, c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Growth Rates: $1 \rightarrow \log(n) \rightarrow n \rightarrow n\log(n) \rightarrow n^2 \rightarrow n^3 \rightarrow c^n \rightarrow n!$

Recursion Analysis

Case 1: work per call follows pattern - $T(n) = T(n-1) + O(1)$

Case 2: work per level is the same - $T(n) = 2T(\frac{n}{2}) + O(n)$

Case 3: work per call is the same - $T(n) = 2T(n-1) + O(1)$

Sorting Algorithms

Sort	Worst	Best	Expected
Bubble $A[i] \overset{swap}{\leftrightarrow} A[i+1]$	$O(n^2)$	$O(n)$ *pre-sorted*	$O(n^2)$
Insertion *by next unsorted*	$O(n^2)$	$O(n)$ *pre-sorted*	$O(n^2)$
Selection *by smallest*	$O(n^2)$	$O(n)$ *pre-sorted*	$O(n^2)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Quick *3-partition*	$O(n^2)$	$O(n)$ *uniform list*	$O(n\log n)$
Radix	$O(d(n+N))$	$O(d(n+N))$	$O(d(n+N))$

Basic Data Structures

Amortisation: $T(n) \div n$, where $n =$ no. of operations

	SPACE	*get*	*add*	*remove*
Dynamic Array	$O(n)$	$O(1)$	$O(n)$*	$O(n)$
Linked List	$O(n)$	$O(1)$	$O(n)$	$O(n)$
Stack - LIFO	*SPACE:* $O(n)$		$O(1)$[1]	$O(1)$
Queue - FIFO	*SPACE:* $O(n)$		$O(1)$	$O(1)$

[1] *amortised if array-based implementation*

Binary Trees

Proper Binary Tree: internal nodes have exactly 2 children

Complete Binary Tree: levels $0 \rightarrow h-1$ full, level h left-most

Pre-Order	In-Order	Post-Order
self → left → right	left → self → right	left → right → self

Properties:

- Full level l has 2^l nodes (note $l \geq 0$)
- Max no. of nodes = $2^{l_{\text{MAX}}} - 1$, max internal nodes = $2^{l_{\text{MAX}}-1} - 1$
- h = no. of edges from lowest leaf = $\lfloor \log_2 n \rfloor$
- ⚠ *care with null leaves in implementation vs. conceptual tree*
- l & d use 0-based index.

Heaps & Priority Queues

Binary Heap: complete binary tree with heap property order

$$key(node) \leq key(parent(node)) \text{ OR } key(node) \geq key(parent(node))$$

Insertion: insert at last node location then **upHeap** -

1. Check $k < parent \wedge k \neq root$
2. Swap $k \longleftrightarrow parent_{\text{greater than current}}$
3. Repeat until $k \geq parent \vee k = root \longrightarrow O(\log n)$

RemoveMin: swap root ⟷ last node then **downHeap** -

1. Check $k > child_{\text{left or right}}$
2. Swap $k \longleftrightarrow child_{\text{less than current}}$
3. Repeat until $k > child_{\text{both}} \vee k = leaf \longrightarrow O(\log n)$

Array-Based: left child @ $2i+1$ & right child @ $2i+2$

Heap Sort: for a in A → add to heap → removeMin back to A

Bottom Up Heap Construction: get $l_{\text{MAX}} = \lfloor \log_2 n \rfloor$ → get no. of leaves on bottom level (fill remaining with nulls) → add nodes to merge heaps

Maps - Hash Tables

General Summary: pre-hash → compress → handle collisions → rehash

Complexities: $O(1)$ expectation, $O(n)$ worst-case

Pre-hash to *Hash Code*:

- Component sum: e.g. sum of all char in string (collision risk)
- Polynomial accumulation: $p(z) = a_0 z^0 + a_1 z^1 + \ldots \; z \in \mathbb{Z}$
- Cyclic shift: replace z with bit-shifted version (e.g. $z \ll 5$)

Compress to *Hash Value*: fow below, N = table size $\wedge N \in \mathbb{Z}^{\text{prime}}$

- Division: $h(k) = k \bmod N$
- MAD: $h(k) = ((ak+b) \bmod p) \bmod N$
  $p > N \wedge p \in \mathbb{Z}^{\text{prime}}, a \in [1, p-1], b \in [0, p-1]$

Collision Handling: separate chaining vs. open addressing

- Probe idx computed as $(h(k) + f(i)) \bmod N$ for $i = 0, 1, 2\ldots$
- 1st probe is just $h(k)$ hence $i = 0$ for open addressing, below:

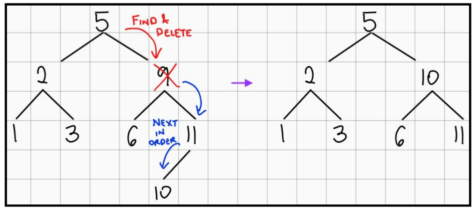
	Linear	Quadratic	Double Hash
$f(i)$	i	i^2	$i \times d(k)$

Load Factor $\alpha = \frac{n}{N} \rightarrow 0.8 \leq \alpha \leq 1$ chaining & $\alpha < \frac{2}{3}$ open addressing

- Expected no. of probes = $\frac{1}{1-\alpha}$

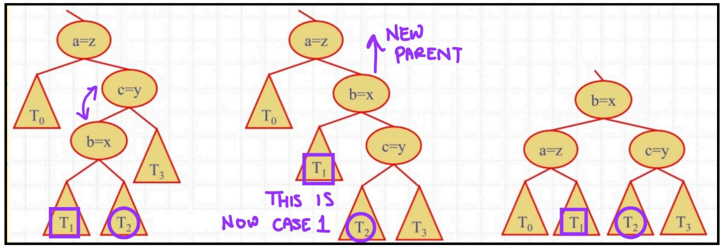
Search Trees

BST: $O(\log n)$ expected get, add, remove (in-order traversal), $O(n)$ worst



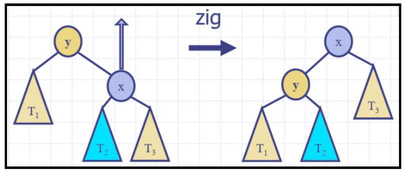
AVL Trees: BST that doesn't degrade to $O(n)$

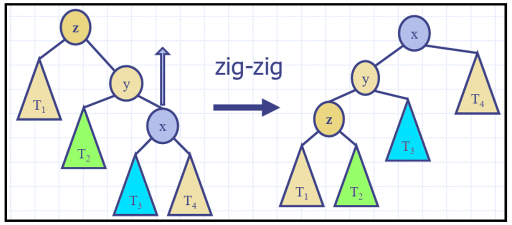
- Insertion: re-balance at **first** unbalanced node from bottom
- Deletion: after BST deletion, re-balance upwards from bottom
- Tri-node inputs: parent, child + grandchild of greater height

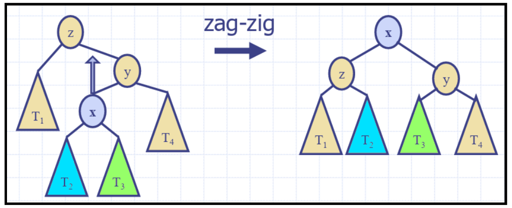


Splay Trees: $O(n)$ worst, $< O(\log n)$ best (popular nodes), $O(\log n)$*

- `get(K)` - splay found node, or last accessed node before null leaf
- `insert(K,V)` - splay inserted/updated node
- `remove(K)` - splay original parent of removal node







Graphs

- **Incident** edge = connected to vertex, **adjacent** vertices = connected by edge
- **Vertex Degree** = no. of incident edges (in vs out degree for directed)
- **Simple Path** = distinct edges AND vertices
- **Simple Cycle** = simple path with same start/end points
- **Subgraph** = subset of vertices/edges
- **Spanning Subgraph** of G = subgraph that contains *all* vertices of G
- **Connected Graph** = $\exists$ simple path between any 2 vertices
- **Unrooted Tree** = connected graph with no simple cycles (can be spanning)
- **Forest** = unconnected graph with no simple cycles (can be spanning)
- **Fully Dense Graph:** $\sum deg(v) = 2 \times |E|$ in undirected graph → $|E| \leq n(n-1) \div 2$
- **Graph Density:**
  - $\frac{2 \times |E|}{n(n-1)}$ undirected or $\frac{|E|}{n(n-1)}$ directed
  - $|E| \sim O(n)$ sparse or $|E| \sim O(n^2)$ dense
- **Graph Representations:**
  - Edge list: simple list of edge pointers only $O(m)$
  - Adjacency List: for each vertex, store all adjacent vertices $O(n+m)$
  - Adjacency Matrix: $V \times V$ grid where $(u, v) = 1 \leftrightarrow$ edge exists

Feature	Edge List	Adjacency List	Adjacency Matrix
Space	$n+m$	$n+m$	n^2
outgoingEdges(v)	m	deg(v)	n
incomingEdges(v)	m	deg(v)	n
getEdge(v, w)	m	min(deg(v),deg(w))	1
insertVertex(o)	1	1	n^2
insertEdge(v, w, o)	1	1	1
removeVertex(v)	m	deg(v)	n^2
removeEdge(v)	1	1	1
Good for	small graphs	sparse graphs	dense graphs

ADT Methods

Stack	`push(V)`	`pop()`	`top()` or `peek()`
Queue	`enqueue(V)`	`dequeue()`	`front()` or `peek()`
Priority Q	`insert(K, V)`	`removeMin()`	`min()`
Entry	`getKey()`	`getValue()`	`compareTo(Entry)`
Map	`get(K)`	`put(K, V)`	`remove(K)`
	`entrySet()`	`keySet()`	`values()`
Graph	`numVertices()`	`vertices()`	
	`numEdges()`	`edges()`	
	`outDegree(V)`	`inDegree(V)`	
	`outgoingEdges(V)`	`incomingEdges(V)`	
	`insertVertex(x)`	`insertEdge(V1, V2, x)`	
	`removeVertex(V)`	`removeEdge(E)`	
	`getEdge(V1, V2)`	`endVertices(E)`	
	`opposite(V, E)`		