

ALGORITHMIC ANALYSIS

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

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

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

$\leftrightarrow \exists c_1, c_2 \in \mathbb{R}, n_0 \in \mathbb{Z}^{\geq 0}$ 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 $A[i] \xrightarrow{\text{swap}} A[i + 1]$	$O(n^2)$	$O(n)$ 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 \rightarrow left \rightarrow right	left \rightarrow self \rightarrow right	left \rightarrow right \rightarrow self

Properties:

- Full level l has 2^l nodes (note $l \geq 0$)
- Max no. of nodes = $2^{l_{\max}} - 1$, max internal nodes = $2^{l_{\max}-1} - 1$
- 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

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

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

- Check $k < \text{parent} \wedge k \neq \text{root}$
- Swap $k \leftrightarrow \text{parent}_{\text{greater than current}}$
- Repeat until $k \geq \text{parent} \vee k = \text{root} \rightarrow O(\log n)$

RemoveMin: swap root \leftrightarrow last node then **downHeap** -

- Check $k > \text{child}_{\text{left or right}}$
- Swap $k \leftrightarrow \text{child}_{\text{less than current}}$
- Repeat until $k > \text{child}_{\text{both}} \vee k = \text{leaf} \rightarrow O(\log n)$

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

Heap Sort: for a in $A \rightarrow$ add to heap \rightarrow removeMin back to A

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

MAPS - HASH TABLES

General Summary: pre-hash \rightarrow compress \rightarrow handle collisions \rightarrow 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: $p(z) = a_0 z^0 + a_1 z^1 + \dots z \in \mathbb{Z}$
- Cyclic shift: replace z with bit-shifted version (e.g. $z \ll 5$)

Compress to Hash Value: few 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 $i = 0, 1, 2, \dots$
- 1st probe is just $h(k)$ hence $i = 0$ for open addressing, below:

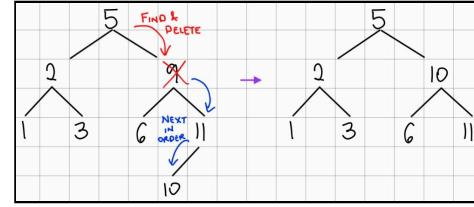
	Linear	Quadratic	Double Hash
$f(i)$	i	i^2	$i \times d(k)$

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

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

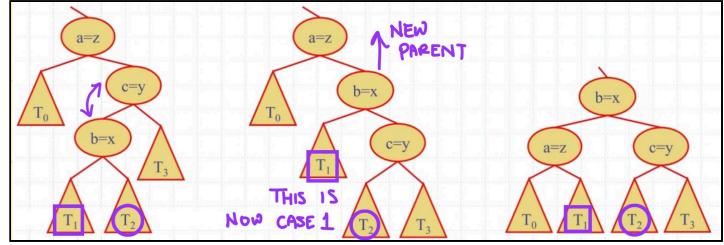
SEARCH TREES

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



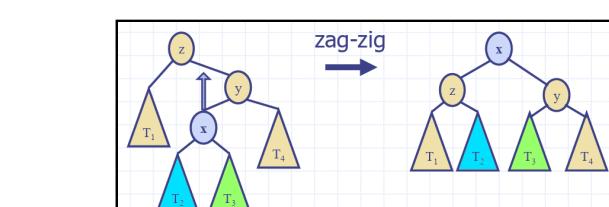
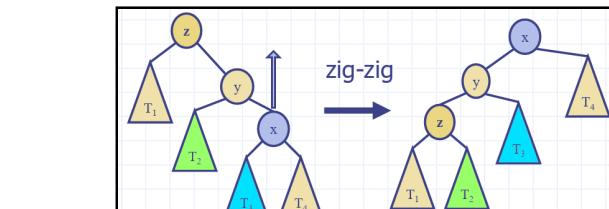
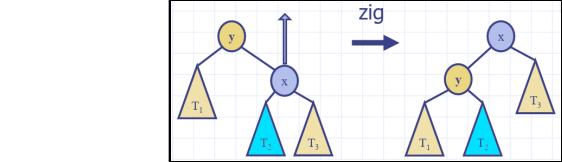
AVL Trees: BST that doesn't degrade to $O(n)$

- 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



ADT METHODS

Stack	<code>push(V)</code>	<code>pop()</code>	<code>top()</code> or <code>peek()</code>
Queue	<code>enqueue(V)</code>	<code>dequeue()</code>	<code>front()</code> or <code>peek()</code>
Priority Q	<code>insert(K, V)</code>	<code>removeMin()</code>	<code>min()</code>
Entry	<code>getKey()</code>	<code>getValue()</code>	<code>compareTo(Entry)</code>
Map	<code>get(K)</code>	<code>put(K, V)</code>	<code>remove(K)</code>
	<code>entrySet()</code>	<code>keySet()</code>	<code>values()</code>