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: $f(n) \in \Theta(g(n)) \longleftrightarrow f(n) \in O(g(n)) \land f(n) \in \Omega(g(n))$

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

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

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*

Trees

Proper Binary Tree: internal nodes have 2 children (levels $\leq$ full)
Complete Binary Tree: levels $0 \to h-1$ full, level h left-most

Pre-Order	In-Order	Post-Order
self $\to$ left $\to$ right	left $\to$ self $\to$ right	left $\to$ right $\to$ 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, l & d use 0-based index.

Heaps & Priority Queues

Binary Heap: complete binary tree with heap property order

$key(node) \leq key(parent(node))$ OR $key(node) \geq key(parent(node))$

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

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

RemoveMin: swap root $\longleftrightarrow$ last node then **downHeap** -

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

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

Algorithm HeapSortGeneric (A):

```
for a in A do:                      ▷ O(n log n) loop
    heap.add(a)
i ← 1
while !heap.isEmpty() do:           ▷ O(n log n) loop
    A[i] ← heap.removeMin()
    i ← i + 1
```

Algorithm HeapSortBottomUp (A):

!!!

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)`