(1) ADTs

List. `List()`, `Retrieve`(L, i), `Insert`(L, b, i), `Delete`(L, i), `Length`(L) *optional:* `Search`(L, b), `Concat`(L_1, L_2), `Plant`(L_1, i, L_2), `Split`(L, i) *special cases:* `Retrieve/Insert/Delete-First/Last`.

Dictionary. `Dictionary()`, `Insert`(D, X), `Delete`(D, x), `Search`(D, k), `Min`(D)), `Max`(D), `Successor`(D, x), `Predecessor`(D, x) *(for rank trees:)*`Select`(D, k) [the k$^{\text{th}}$ smallest element], `Rank`(D, x) [the position in sorted order].

Stack. (LIFO) `Push`(L, b) [=ins.-last], `Top`(L) [=ret.-last], `Pop`(L) [=del.-last]. (all $\mathcal{O}(1)$ using arrays)

Queue. (FIFO) `Enqueue`(L, b) [=ins-last], `Head`(L) [=ret.-first], `Dequeue`(L) [=del-first]. (all $\mathcal{O}(1)$ using circular arrays)

Deque. Queue + Stack

Vector. `Vector`(m), `Get`(V, i), `Set`(V, i, val). (All $\mathcal{O}(1)$ using *legals* and *positions* arrays that reference each other) d

function isGarbage(i) **is**
  if $0 \leq$ positions$[i] <$ legals.*size* **and** legals[positions$[i]$] $= i$ **then**
    return false
  end
  return true
end

Graph. `Edge`(i, j), `Add-Edge`(i, j), `Remove-Edge`(i, j), `InDeg`(i), `OutDeg`(i) etc.

(2) Graphs

definition 1 (Topological sorting algo.). Input: directed graph / Output: numbering $(n_i)_{i=1}^N$ of the graph nodes where $\forall (i, j) \in E \colon n_i < n_j$.

theorm 1. *Topological Sorting exists iff the graph doesn't contain cycles*

k $\leftarrow 0$;
while *there are sources* **do**
  find source v;
  $n_i \leftarrow k$;
  $k \leftarrow k + 1$;
  remove v from the graph
end
if k $= n$ numbering completed, otherwise isn't possible.

building "source queue" takes $\mathcal{O}(n)$, dequeuing source $\mathcal{O}(1)$, and enqueuing new sources to sources-queue $\mathcal{O}(d_{\text{out}}(i))$. Total $\mathcal{O}(n + m)$ for topological ordering.

definition 2. A *source* is a node that has no incoming edges.

remark 1. any DAG has at least one source

(3) Complexity

definition 3. Suppose there's a data structure with k types of operations $(T_i)_{i=1}^k$, then for sequence of operations $(op)_{i=1}^n$, then:

$$\text{time}(op_1 \ldots op_n) \leq \sum_{i=0}^n \text{bound}(\text{type}(op_i))$$

Where (W.C. bound) worst(T_i) is the maximal time for a single operation typed T_i, and (amortized bound) amort(T_i) is the bound for the cost of every valid sequence $(op_i)_{i=1}^n$.

Amortization methods. aggregation (regular average), accounting (bank method), and potential function (defined to be the balance of the bank) that satisfies amort$(op_i) = \text{time}(op_i) + \Phi_i - \Phi_{i-1}$.

Potential for doubling by $1 + \alpha$. $\Phi := \begin{cases} \frac{1+\alpha}{\alpha} n - \frac{M}{\alpha} & n > \frac{M}{\alpha+1} \\ 0 & \text{else} \end{cases}$ yields to un amortized bound of $\mathcal{O}\left(\frac{1+\alpha}{\alpha} + 1\right)$

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1} = \Theta(x^n) (x \neq 1)$$

$$\sum_{i=1}^n \frac{1}{i} = H_n = \Theta(\log n)$$

(3.1) Master Theorem

let $f \colon \mathbb{R} \to \mathbb{R}$ be an function, and let $a \leq 1, b > 1$ be constants, assuming $T \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$, $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, then:

1. $\exists \varepsilon > 0. f(n) = O(n^{\log_b a - \varepsilon})$ $\implies T(n) = \Theta(n^{log_b a})$
2. $f(n) = \Theta(n^{\log_b a})$ $\implies T(n) = \Theta(n^{log_b a} \cdot \log n)$
3. $\exists \varepsilon > 0. f(n) = \Omega(n^{\log_b a + \varepsilon}) \wedge$ $\exists c > 1, n_0 \geq 0. \forall n \geq n_0. a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ $\implies T(n) = \Theta(f(n))$

Note that $\frac{n}{b}$ *could be* $\left\lfloor \frac{n}{b} \right\rfloor$ *nor* $\left\lceil \frac{n}{b} \right\rceil$

(4) Dictionaries

(4.1) General Trees

definition 4. a in *full* tree all internal nodes have exactly i children.

definition 5. a *BST* satisfies: $\forall x \forall y$ if y is in the left subtree of x, then $y.key < x.key$, and vise-versa.

definition 6. *height* of a node = maximal length of downward path between that node and a leaf.

definition 7. *depth* of a node is the length of the path up the tree to the root.

theorm 2. *minimal height of a tree is* $\lfloor \log n \rfloor$

definition 8. a BST is balanced if $h = \mathcal{O}(\log n)$

theorm 3. *for a given set of n distinct keys, there are* $\frac{1}{n+1}\binom{2n}{n}$ *(catalan number) BSTs.*

theorm 4. *the expected search complexity in a random BST is* $\leq (1 + 4 \log n)$.

lemma 1. the heights of a binary tree containing ℓ leaves $\geq \log \ell$.

Tree walks. pre: SLR, in: LSR, post: LRS

Postfix syntax algo. (...)

(4.2) AVL trees

definition 9. an AVL tree is a BST where $\forall v \in V \colon |\text{BF}(v)| \leq 1$

theorm 5. *and AVL tree is balanced. Furthermore:* $n \leq \log_\Phi n \approx 1.44 \log n$.

definition 10. Fibonacci tree F_i is:

$$\begin{array}{ccc} & F_i & \\ / & & \backslash \\ F_{i-1} & & F_{i-2} \end{array}$$

theorm 6. *an AVL tree with minimum edges is a fibonacci tree, sized* $f_n = \frac{\Phi^n - \bar{\Phi}^n}{\sqrt{5}}$, $\Phi = \frac{1+\sqrt{5}}{2}$.

definition 11. a *rank tree*, is a tree that maintains the size of each subtree, hence supports the rank & select operations in $\mathcal{O}(\log n)$.

theorm 7. *if the information that a given attribute f defined for each node, can be computed merely from its direct children (local attribute), then we can maintain f in an AVL tree.*

remark 2. The theorem above is sufficient condition but not necessary.

definition 12. a *Finger Tree* is a tree that has a pointer to a specific node.

theorm 8. *in a finger tree* `Select`(T, k) *can be implemented in* $\mathcal{O}(\log k)$.

(4.3) B-trees

definition 13. a *B-tree* $(d, 2d)$ satisfies:

1. each non-leaf expect for the root has $d \leq r \leq 2d$ children (hence $d - 1$ to $2d - 1$ keys);
2. all leaves are at the same depth;
3. the root has between 2 and $2d$ children (hence 1 to $2d - 1$ keys).

definition 14. a B$^+$-tree is a B-tree with keys only on leafs.

definition 15. a B*-tree is a B-tree with nodes $\frac{2}{3}$ full (instead of $\frac{d}{2d} = \frac{1}{2}$ full).

definition 16. a red-black tree is a $(1, 2)$ B-tree (BST).

theorm 9. *at depth h there are at least* $2d^{h-1}$ *nodes.*

theorm 10. *a B-tree $(d, 2d)$ with n edges and h height fulfills* $n \geq d^h$, $h \leq \log_d n$

theorm 11. *search in a b-tree requires* $\mathcal{O}(\log_d n)$ *I/Os, and* $\mathcal{O}(\log_2 d \cdot \log_d n) = \mathcal{O}(\log n)$ *operations in total.*

theorm 12. *Ins./Del. rebalancing cost is W.C.* $\mathcal{O}(\log n)$*, and using button-up amort. (ins.+del.)* $\mathcal{O}(1)$*, using top-down* $\Omega(\log_d n)$

Insertions

Button-Up. Find and insert in the appropriate leaf. If the current node is overflowing: split. If the parent is overflowing: split (etc., recursively). Requires a total of $\mathcal{O}(d \log_d n)$ operations.

Top-Down. if a node is full, we will split it on the way down while searching.

Button-Up non-leaf deletion. replace the item by its predecessor and delete the predecessor (must be a leaf).

Deletions

Button-Up leaf deletion. if the current node is underflowing, borrow and terminate and if not possible fuse and recursively check the if parent if underflowing.

Top-Down leaf deletion. while searching, checking if the items along the way contains d keys, otherwise borrow or fuse.

Top-Down non-leaf deletion. replace the node with its predecessor, while making sure that nodes along the way contains at least d keys.

(5) Priority Queue

(6) Sorting

(6.1) Comparison-based sorting

theorm 13. *insertion sort with $I > n$ inversions* $(I \leq \binom{2n}{n})$ *takes* $\mathcal{O}\left(n \log \frac{I}{n}\right)$.

definition 17. *stable sort* is a sorting algo. the preserves order of items with the same key.

definition 18. a comparison-based algo. uses only two-key comparisons to decide on key position.

assumption. two keys can be compared in $\mathcal{O}(1)$, and an item can be moved in $\mathcal{O}(1)$.

theorm 14. *the W.C. and average case of any comparison-base sorting algo. runs in* $\Omega(n \log n)$

lemma 2. comparison trees are a full binary tree, and has $\geq n!$ leafs.

theorm 15. *the worst/best/average case in the comparison-based model is the max/min/average depths of the leafs.*

(6.2) Other sorting algos.

Count sort. For dataset A, assumes $\exists R \forall a \in A \leq R$ constant. Counts each element $a \in A$, takes a cumulative sum (a_i), then for all $a \in A$ puts a in a_i and decreases $a_i \leftarrow a_i - 1$. Takes $\mathcal{O}(n+R)$. Stable sort.

Count sort. similar to count sort, takes R bins and throws A into them, then collects them.

Radix sort. For a dataset A sized n, assumes $a \in A$ contains exactly d digit and each digit is bounded by b. Preforms count sort on the LSD $\rightarrow$ MSD. [note: relies on count sort being stable]. Takes $\mathcal{O}(d(n+b))$.

theorm 16. *Radix sort is enough to make IBM.*

(6.3) QuickSort

Lomuto's Partition.

Hoare's Partition.

theorm 17. *W.C. of quicksort if* $\binom{n}{2} = \mathcal{O}(n^2)$.

theorm 18. *Average case of quicksort is* $2(n+1)H_n - 4n \approx 1.39 n \log n$.

(7) Probability

definition 19. an *Experiment* is a case where we the result is uncertain.

definition 20. the *Sample Space* is the set of all the expected outcomes of a given experiment.

definition 21. an *Event* is a subset of the sample space. A singleton subset called a *simple event.*

definition 22. *Disjoint Events* are events A, B that fulfills $A \cap B = \varnothing$.

definition 23. a *Probability Function* is a function $P \colon S \to [0,1]$ for S sample space, so that $\forall E, F$ disjoint: $P(E \cup F) = P(E) + P(F)$ and $P(S) = 1$.

definition 24. the conditional probability of event E given the event F is $P(E \mid F) := \frac{P(E \cap F)}{P(F)}$.

theorm 19. *for disjoint events* $(F_i)_{i=1}^{n}$*, if* $\bigcup F_i = E$ *then* $\forall E \colon P(E) = \sum_{i=0}^{n} P(E \mid F_i) \cdot P(F_i)$.

definition 25. events E, F are independent if $P(E \cap F) = P(E) \cdot P(F)$ (iff $P(E \mid F) = P(E)$).

definition 26. a *Random Variable* if a function $X \colon S \to \mathbb{R}$.

definition 27. $X = x$ is the event on which $X(E) = x$, and its probability noted as $P(X = x)$.

definition 28. the *Expectation* of a random variable X is $\mathbb{E}[x] = \sum_x x \cdot P(X = x)$.

theorm 20. *the expectation is linear for all constants, and additive for all random variables.*

definition 29. a random variable I is called an *Indicator of an event* A if $I = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A^c \text{ occurs} \end{cases}$

lemma 3. $\mathbb{E}[I] = P(A)$.

definition 30. *Uniform Distribution* of a random variable X occurs when $\exists c \colon \forall x \in \mathbb{R} \colon P(X = x) = c$.

definition 31. *Geometric Distribution* satisfies $P(x = k) = (1-p)^{k-1} p$, hence $\mathbb{E}[X] = \sum_{k=1}^{\infty} k(1-p)^{k-1} p = \frac{1}{p}$.

note: geometric dist. is equal to having the probability of succession p and for failure $p - 1$, and P is a rand. var. that is equal to the number of required experiments to get to an solution.

(8) Selection

(9) Hashing

(10) Other

Reduction. reduction (in our case) is the process of showing the a problem is at least as hard as another problem.

(11) Complexity Tables

Lists

ℓet mid $:= \min\{i, n-i\} + 1$

	Arrays	**Circular Arr.**	**D-Linked**	**AVL List**
`Ins/Del-Last`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
`Ins/Del-First`	$\mathcal{O}(n+i)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
`Ins`(i)	$\mathcal{O}(n-i+1)$	$\mathcal{O}(\text{mid})$	$\mathcal{O}(\text{mid})$	$\mathcal{O}(\log n)$
`Retrieve`(i)	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\text{mid})$	$\mathcal{O}(\log(i)+1)$
`Concat`$(\lvert n_1 \rvert, \lvert n_2 \rvert)$	$\mathcal{O}(n_2+1)$	$\mathcal{O}(\min\{n_1, n_2\}+1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log(n_1+n_2))$
`Split`(i)	$\mathcal{O}(n-i+1)$	$\mathcal{O}(\text{mid})$	$\mathcal{O}(\text{mid})$	$\mathcal{O}(\log n)$

(in a lazy doubly-linked list, amortized del./ins. $\mathcal{O}(1)$ and ret. $\mathcal{O}(i+1)$)

Priority Queues

	`Insert`	`Minimum`	`Delete-Min`	`Dec.-Key`	`Delete`	`Meld`	`Init`
AVL tree	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$		$\mathcal{O}(n \log n)$
Binary Stack	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
W.C Binomial Stack	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	
Amort. Binomial Stack*	$\mathcal{O}(1)$	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	
Lazy Amort. Binomial Stack	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	
Amort. Fib. Stack:	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	

*amortized for a sequence of operations from the same type.