Shahar Perets ∼ DS Shit Cheat Sheet

(1.0) 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^{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

Priority Queue. Insert(x,Q), Min(Q), Delete-Min(Q), (optional:) Decrease-Key(x,Q,Δ), Delete(x,Q)

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 ≤ 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.

(2.0) 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 ← 0;
while there are sources do
    find source v;
    n_i ← k;
    k ← 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

(3.0) 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:

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

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 a series of bounds for 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 $\text{amort}(op_i)=\text{time}(op_i)+\Phi_i-\Phi_{i-1}$.

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 $\mathcal{O}\left(\frac{1+\alpha}{\alpha}+1\right)$

$$\sum_{i=0}^n x^i=\frac{x^{n+1}-1}{x-1}=\Theta(x^n)(x\neq1)$$

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

Theorm 2. $\alpha+\beta=1\wedge T(n)\le cn+T(\alpha)+T(\beta n)\implies T(n)=\mathcal{O}(n)$.

(3.1) Asymptotic Notations

$$f=O(g)\iff\exists n_0,c>0\,\forall n\ge n_0\colon f(n)\le cg(n)$$
$$f=\Omega(g)\iff\exists n_0,c>0\,\forall n\ge n_0\colon f(n)\ge cg(n)$$
$$f=\Theta(g)\iff f=\Omega(g)\wedge f=O(g)$$
$$f=o(g)\iff\forall c\,\exists n_0\,\forall n\ge n_0\colon f(n)\le cg(n)$$
$$f=\omega(g)\iff\forall c\,\exists n_0\,\forall n\ge n_0\colon f(n)\ge cg(n)$$
$$f=\Omega(g)\iff g=O(f)$$
$$f_1=O(g_1)\wedge f_2=O(g_2)$$
$$\implies f_1(n)+f_2(n)=O(\max(g_1(n),g_2(n)))$$
$$f=O(g)\iff\limsup_{n\to\infty}\frac{f(n)}{g(n)}<\infty$$
$$f=\Omega(g)\iff\limsup_{n\to\infty}\frac{f(n)}{g(n)}>0$$
$$f=o(g)\iff\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$
$$f=\omega(g)\iff\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

(3.2) Master Theorem

let $f\colon\mathbb{R}\to\mathbb{R}$ be an function, and let $a\le1,b>1$ be constants, assuming $T\colon\mathbb{R}_{\ge0}\to\mathbb{R}$, $T(n)=a\cdot T\left(\frac{n}{b}\right)+f(n)$, 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. $\exists\varepsilon>0.f(n)=\Omega(n^{\log_b a+\varepsilon})\wedge$ $\exists c>1,n_0\ge0.\forall n\ge n_0.a\cdot f(\frac{n}{b})\le c\cdot f(n)$ $\implies T(n)=\Theta(f(n))$

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

(4.0) 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 3. *minimal height of a tree is $\lfloor\log n\rfloor$*

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

Theorm 4. *for a given set of n distinct keys, there are $\frac{1}{n+1}\binom{2n}{n}$ (catalan number) BSTs.*

Theorm 5. *the expected search complexity in a random BST is $\le(1+4\log n)$.*

Lemma 1. the heights of a binary tree containing ℓ leaves $\ge\log\ell$.

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

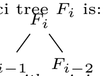
Postfix syntax algo. (...)

(4.2) AVL trees

Definition 9. an AVL tree is a BST where $\forall v\in V\colon|\text{BF}(v)|\le1$

Theorm 6. *and AVL tree is balanced. Further more: $n\le\log_\Phi n\approx1.44\log n$.*

Definition 10. Fibonacci tree F_i is:



Theorm 7. *an AVL tree with minimum edges is a fibonacci tree, sized $f_n=\frac{\Phi^n-\bar\Phi^n}{\sqrt5}$, $\Phi=\frac{1+\sqrt5}{2}$.*

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 8. *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.*

Lemma 2. the sum of the keys lesser than v, and the sum of the keys in the subtrees, can be implemented both without harming time complexity.

Lemma 3. Between(s,t) = Tree-Rank(t) − Tree-Rank(s) + 1

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

Theorm 10. *Given a sorted array, we can create an AVL tree in $\mathcal{O}(n)$ on which $h=\lfloor\log n\rfloor$*

Join(T_1,T_2): **where $T_1<x<T_2$ is done $\mathcal{O}(h_{T_1}+h_{T_2}+1)$ (see image).** Split(T,x): splits T into $T_1<x<T_2$ in $\mathcal{O}(\log n)$ using joins.

(4.3) B-trees

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

1. each non-leaf expect for the root has $d\le r\le2d$ 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).

Theorm 11. *at depth h there are at least $2d^{h-1}$ nodes.*

Theorm 12. *a B-tree $(d,2d)$ with n edges and h height fulfills $n\ge d^h$, $h\le\log_d n$*

Theorm 13. *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.*

Lemma 4. In a B-tree #leaves = #internal nodes + 1

Theorm 14. *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.

(5.0) Priority Queue

(5.1) Binary Heap

Definition 16. a *binary minimum binary heap* is an almost perfect BST (only possibly misses nodes at the last level), and satisfies the heap order: the keys at the children of v are greater than they key in v.

Lemma 5. the height of binary heap is $\lfloor\log n\rfloor$

Theorm 15. *in a d-ary heap representation as an array (in brackets for binary):*

$$\text{Left}(i)=dk-(d-2)\quad(2i)\quad\text{Right}(i)=dk+1\quad(2i+1)$$

$$\text{Parent}(i)=\left\lfloor\frac{k+(d-2)}{d}\right\rfloor\quad\left(\left\lfloor\frac{i}{2}\right\rfloor\right)$$

Heapify-Down(i): if Parent(i) is bigger, then replace i with Parent(i), and recursively continue on Parent(i).

Heapify-Up(i): exchange with the smallest child until fixed.

Insert: insert on the last place in the array, then heapify up.

Delete: delete the required place in the array (the root) the replace it with the last one, then heapify down until fixed (in d-ary $\mathcal{O}(d\log_d n)$).

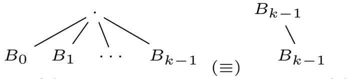
Dec-Key: decrease the key (assumes $\Delta\ge0$) then heapify up.

Init: iterate over internal nodes bottom-up, and heapify-down each one.

HeapSort: create a min-heap from input, the do delete-min and put the deleted element at the last position of the array. Repeat n times. At the we get a reversely-sorted array (using min-heaps).

(5.2) Binomial Trees

Definition 17. B_k is a binomial tree of degree k if



Theorm 16. *(1) The root of B_k has k children (2) B_k contain 2^k nodes (3) its depth is k (4) $\binom{k}{i}$ of the nodes of B_k are at level i.*

Definition 18. A *binomial min-heap* is a list of heap-ordered binomial trees, at most one of each degree, and a pointer to the root with the minimal key.

note: usually the trees are saved using a linked list.

Lemma 6. There are at most $\lfloor\log n\rfloor+1$ trees.

Link: if two binomial trees x,y has the same degree, linking could be preformed in $\mathcal{O}(1)$ by attaching y as a child of x and replacing the roots if needed.

Insert: insertion could be done the same way as binary incrementing, where linking≡carrying.

Dec-Key: just heapify up as before.

Meld: link trees with the same degree, like binary addition.

Del-Min: the children of the deleted root are a binomial heap, Meld them into the main tree.

Lazy Binomial Trees adds just B_0-s (allows melding in $\mathcal{O}(1)$), and consolidates when runs delete-min.

Consolidating (on del-min) is the process of taking the nodes and adding them into respected bins (numbered $0\dots\lfloor\log n\rfloor$), and when two trees are in the same bin – linking them together and moving them into the next bin.

Definition 19. T_0 is #trees before Del-Min, T_1 after Del-Min, and L is the total #Links through consolidating.

Lemma 7. $L\le T_0+\log n$ (we have at most $\lfloor\log n\rfloor$ trees exposed on Del-Min)

Theorm 17. *The cost of consolidating is $T_0-1+\log n+L=\Theta(T_0+\log n)$.*

Theorm 18. *Using Φ = #trees we get $\Delta\Phi=T_1-T_0$ hence amort. cost of consolidating is $\mathcal{O}(\log n)$.*

Lemma 8. incrementing a binary number has an amortized bound of $O(1)$.

(5.3) Fibonazi Heaps

todo

(6.0) Sorting

(6.1) Comparison-based sorting

Theorm 19. *insertion sort with $I>n$ inversions ($I\le\binom{2n}{n}$) takes $\mathcal{O}\left(n\log\frac{I}{n}\right)$.*

Definition 20. *stable sort* is a sorting algo. the preserves order of items with the same key.

Definition 21. 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 20. *the W.C. and average case of any comparison-base sorting algo. runs in* $\Omega(n \log n)$

Lemma 9. comparison trees are a full binary tree, and has $\geq n!$ leafs.

Theorm 21. *the worst/best/average case in the comparison-based model is the max/min/average depths of the leafs.*

(6.2) Other sorting algos.

HeapSort: see 5.1. **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 $\to$ MSD. [note: relies on count sort being stable]. Takes $\mathcal{O}(d(n + b))$.

Theorm 22. *Radix sort is enough to make IBM.*

(6.3) QuickSort

Lomuto's Partition.

Hoare's Partition.

Theorm 23. *W.C. of quicksort if* $\binom{n}{2} = \mathcal{O}(n^2)$.

Theorm 24. *Average case of quicksort is* $2(n+1)H_n - 4n \approx 1.39n \log n$.

(7.0) Probability

Definition 22. an *Experiment* is a case where we the result is uncertain.

Definition 23. the *Sample Space* is the set of all the expected outcomes of a given experiment.

Definition 24. an *Event* is a subset of the sample space. A singleton subset called a *simple event.*

Definition 25. *Disjoint Events* are events A, B that fulfills $A \cap B = \varnothing$.

Definition 26. 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 $P(S) = 1$.

Theorm 25. *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 27. events E, F are independent if $P(E \cap F) = P(E) \cdot P(F)$ (iff $P(E \mid F) = P(E)$).

Definition 28. a *Random Variable* if a function $X\colon S \to \mathbb{R}$.

Definition 29. $X = x$ is the event on which $X(E) = x$, and its probability noted as $P(X = x)$.

Definition 30. the *Expectation* of a random variable X is $\mathbb{E}[x] = \sum_x x \cdot P(X = x)$.

Theorm 26. *the expectation is linear for all constants, and additive for all random variables.*

Definition 31. 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 10. $\mathbb{E}[I] = P(A)$.

Definition 32. *Uniform Distribution* of a random variable X occurs when $\exists c\colon \forall x \in \mathbb{R}\colon P(X = x) = c$.

Definition 33. *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.

Theorm 27 (The Tail Formula)**.** $\sum_{i=0}^{m} i \cdot P[X = i] = \sum_{i=1}^{m} P(x \geq i)$

(8.0) Selection

Definition 34. given n numbers, `Select`(n) is defined to return the k^th^ smallest key.

This equals for the item in position k, assuming the array was sorted.

Definition 35. Dynamic settings assumes one-time building cost (e.g. `Tree-Select`).

Definition 36. Static settings is not a dynamic setting

Theorm 28. *Using min-heap + supporting heap the selection problem is solvable is* $\mathcal{O}(n + k \log k)$.

Theorm 29. *The expected number of items removed during each quickselect run is* $\mathbb{E}[\#\text{removed}] = \frac{k}{2} \cdot \frac{k}{n} + \frac{n-k}{2} \cdot \frac{n-k}{n} \underset{\forall k}{\geq} \frac{n}{4}$.

Theorm 30. *The expected runtime of quickselect is* $\mathcal{O}(n)$.

Theorm 31. *MedofMed cost is W.C.* $\mathcal{O}(n)$.

(9.0) Hashing

Direct Addressing. Create a bit vector with the universe size. e.g. `Insert`(D, x) iff `D`$[x.key] \leftarrow x$ etc.

(9.1) Chaining

Lemma 11. There are $|m|^{|U|}$ hashes in $h \in U \to [m]$, hence takes $|U| \log m$ to store.

Chaining. each cell points to a linked list of items.

Definition 37. $\alpha := \frac{n}{m}$ where n is the universe, and m is the table size, and called the *load factor.*

Lemma 12. the probability of a random two specific insertions colliding is geometric.

Theorm 32. *the expected number of values in each cell is* α.

Theorm 33. *when* $n = \Theta(m)$, *with probability* $\geq 1 - \frac{1}{n}$, *each cell contains at most* $\mathcal{O}\left(\frac{\log n}{\log n \log n}\right)$ *elements.*

Theorm 34. *Assuming the keys are distributed ideally (uniformly and independently), and assuming n keys were previously inserted, the expected complexity during search is* $\alpha + 1$ *for unsuccessful and* $\frac{\alpha}{2} + 1$ *for a successful search.*

(9.2) Open Addressing

Open Addressing. ℓet $h\colon U \times [m] \to [m]$ be a hash function, we'll insert the key k in the first free position in the probing sequence.

Note: make sure to use special marking (not null) for deleted items.

Theorm 35. *Under ideal conditions* $(\forall k \in [n]\colon P\left((h(k,i)_{i=0}^{m-1}) = \frac{1}{m}\right))$, *the expected time for unsuccessful search is* $\frac{1}{1-\alpha}$ *and fo successful search* $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.

Theorm 36. *under linear probing, unsuccessful search takes* $\frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$ *and successful search* $\frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$.

Note: under linear probing, we can delete by recursively checking if item j can me moved to deleted cell i for all $h'(T[j]) \in [j+1, i]$.

Definition 38. *Linear Probing* is a hash func $h(k,i) := (h'(k) + i) \bmod m$ (less cache misses + easy to calculate).

Definition 39. *Quadratic Probing* is a hash func $h(k,i) := (h'(k) + ic_1 + c_2 i^2) \bmod m$.

Definition 40. *Double Probing* is a hash func $h(k,i) := (h'(k) + ih''(k)) \bmod m$.

(9.3) Hash Families

Definition 41. hash family is *Universal* if $\forall k_1 \neq k_2 \in U\colon P_{h \in H}(h(k_1) = h(k_2)) \leq \frac{1}{m}$.

Theorm 37. *For all p prime,* $h_{a,b}\colon [p] \to [m]$ *defined as* $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$, *and* $H_{p,m} := \{h_{a,b} \mid a \in [1,p), b \in [0,p)\}$ *is a universal hash family.*

Theorm 38. *for each p prime,* ℓ*et* $x_1 \neq x_2 \in [p]$. *Then* $\forall y_1 \neq y_2 \in [p]\ \exists! a, b \in [p], a \neq 0\colon y_1 \equiv_p ax_1 + b \wedge y_2 \equiv_p ax_2 + b$.

Theorm 39. *for a table* $m = 2^k$ *so* $h_a\colon U = [2^w] \to [2^k]$ *where w is computer word size,* h_a *defined as* $\left\lfloor \frac{ax \mod 2^w}{2^{w-k}} \right\rfloor$, *and H is almost universal.*

Definition 42. if $\forall k_1 \neq k_2 \in U\colon P_{h \in H}(h(k_1) = h(k_2)) \leq \frac{2}{m}$ then U is called *almost universal.*

Theorm 40. *using universal hash family,* $\mathbb{E}[collisions] \leq \frac{\binom{n}{2}}{m}$.

(9.4) Perfect Hashing

Content...

(10.0) Other

Reduction. reduction (in our case) is the process of showing the a problem is at least as hard as another problem.
Information Bound. a bound derive by an argument that the algo. has to read a specified amount of the input, in order to get a decision. Notice that in comparison, reading isn't counted.

$$\sum_{i=1}^{k} \binom{k}{i} = 2^k$$

$$\binom{k}{i} = \binom{k-1}{i} + \binom{k-1}{i-1}$$

Theorm 41. *merging k sorted arrays with the total of n items can be done in* $\mathcal{O}(n \lfloor k \rfloor)$

(11.0) Complexity Tables

Lists

ℓet mid $:= \min\{i, n-i\} + 1$

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

(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	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$
Binary Heap	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
W.C Binomial Heap	$\mathcal{O}(\log n)^{(*)}$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
Lazy Amort. Binomial Stack	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$
Amort. Fib. Heap:	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(1)_{W.C.}$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	

*amortized $\mathcal{O}(1)$ for a sequence of operations from the same type.