Randomization

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, $H_n = \sum_{i=1}^{n} \frac{1}{i} \le \ln n + 1 \in O(\log n)$

① Insertion sort: on iteration i, first (i-1) elements are sorted, insert i-th element by swapping left.

- elements i,j swap ⇔ $A[i] > A[j]$ $(i<j)$
- $E[X] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} E[X_{ij}]$ → indicator, $1 \Leftrightarrow i \Leftrightarrow j$, $Pr[A[i] > A[j]] = \frac{1}{2}$

② bamboo, $X_0 = n$, cut off x meters, stop: $X_m \le 1$

- if current len = i, $E[\text{cut size}] = \sum_{j=1}^{i} j \cdot Pr[\text{cut size} = j]$
  $= \sum_{j=1}^{i} j \cdot \frac{1}{i} = \frac{i+1}{2} \Rightarrow E[\text{rem. size}] = i - \frac{i+1}{2} = \frac{i-1}{2}$
- $E[X_m] = \sum_i E[X_m \mid X_{m-1} = i] \, Pr[X_{m-1} = i]$
  $= \sum_i \frac{i-1}{2} Pr[X_{m-1} = i] \le \frac{1}{2} \sum_i i \cdot p_{X_{m-1}}(i) = \frac{E[X_{m-1}]}{2}$

* Markov: if $x \ge 0$, $Pr[X \ge a] \le \frac{E[X]}{a}$ $\forall a$

& $W(n) \in O(f(n))$ W.H.P if ∃ constants c, n_0 s.t. $W(n) \in O(k \cdot f(n))$ $\forall n \ge n_0$, w/ probability $\ge (1 - \frac{1}{n^k})$ ∀ constants k.

- $E[X_m] \le \frac{n}{2^m} \Rightarrow Pr[X_{k \log_2 n} \ge 1] \le \frac{E[X_{k \log_2 n}]}{1}$
  $\le n(\frac{1}{2})^{k \log_2 n} = n(\frac{1}{2})^{\log_{\frac{1}{2}} n^{-k}} = \frac{1}{n^{k-1}}$

* $(\frac{1}{c})^{k \log_c n} = n^{k \log_c (\frac{1}{c})}$ ↙ union bound

$Pr[\text{any of k calls bad}] \le \sum_{i=1}^{k} Pr[i^{th} \text{ call bad}]$

③ Quicksort: generate a random priority p for each element of S; choose element w/ higher priority as pivot → i,j compared if either has highest priority in range of ranks $i \le r \le j$

$E[X] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{2}{j-i+1} = 2n H_n \in O(n \log n)$

→ pivot tree: depth = # of recursive calls to place node in sorted spot, span = max depth.
= # calls of quickselect $= O(\log n)$ W.H.P.

* rank-k = element k in sorted (ascending) seq.

$Pr[\text{pivot tree of height } n] = \frac{2^{n-1}}{n!}$

E.g. cut in half w.p. p, then
$E[Y_n] = p \cdot \frac{1}{2} \cdot E[Y_{n-1}] + (1-p) \cdot E[Y_{n-1}]$

BST & Treaps

- inorder traversal: left, node, right
- preorder traversal: root, left, right.
- joinMid T_1, T_2 has $O(\log(T_1 + T_2))$ work/span
  ↑ rebalances ⇔ depth = log n
  split, find, insert, delete $T\ k \in O(\log |T|)$
- Treap = BST with priority function $p: U \to \mathbb{Z}$ (↑ set of keys)
  1. BST invariant: ∀ Node(L,k,R), $l < k$, $r > k$, $\forall l \in L, r \in R$
  2. Heap invariant: $p(k) > p(x)$ $\forall x \in (L \cup R)$
- $A_i^j = 1$ if $S[i]$ is an ancestor of $S[j]$ of pivot tree
  $\text{depth}(j) = \sum_{i=0}^{n-1} A_i^j$, $\text{size}(j) = \sum_{i=0}^{n-1} A_i^j$,

SETS & TABLES

- implemented via arrays/hash tables/ balanced trees (↙ used.)
- tables: sets of key-value pairs.
  ↳ restrict T S : restricts domain to ones in S
  subtract TS : $\{k \to v \in T \mid k \notin S\}$
- Ord tables: ordered keys (total ordering)
  Aug tables: values reducible w/ combining function

```
struct
  type t = int
  val f = Int.max
  val I = valOf (Int.minInt)
  val toString = Int.toString
```

structure Table = MkTreapAugTable (Key = IntElt structure Val = Val)

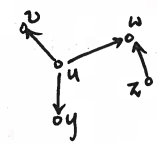
- fun allval (T) = Table.reduce Set.union ∅ T

$\sum_{j=1}^{n} A_i^j = \sum_{j=1}^{n} \frac{1}{|j-i|+1} = H_i + H_{n-i+1} - 1$

1. 1 unique path between 2 vertices in a tree
2. (n-k) edges in forest w/ n nodes & k trees
3. # of SCC in DAG with 10 vertices & 20 edges = 10 (since acyclic, either u ↛ v or v ↛ u).

v_j reachable from $v_i \Rightarrow i < j$ in top ordering

GRAPHS



- v is an out-neighbor of u
  u is an in-neighbor of v
  $N^+(u) = \{v, w, y\}$, $d^+(u) = 3$
  $N^-(w) = \{u, z\}$, $d^-(w) = 2$

path = sequence of vertices, adj. vertices are connected by an edge; if no repeats, simple
cycle = path that start & end on same vertex, no repeat edges (trivial = self-edge)

- undirected G connected if $\forall u,v \in G$, $u \to v$
  directed G strongly connected if $\forall u,v$, $u \to v$

* Directed Acyclic Graph ⇔ no nontrivial connected components

A. adjacency matrix: good on dense $(m \in O(n^2))$ graphs, bad on sparse $(m \in \omega(n^2))$ graphs (↗ neighbor query $\in O(n)$)

B. adjacency sets: (V×V set) table → O(1), → O(log n)

C. adjacency sequence: int seq seq. (tree-based)
↳ check if u,v neighbors in O(1).

Depth-First Search

→ fully explore all nodes reachable from a vertex before moving on to the next vertex

① DFS tree: tree, forward, back, cross edge

② Numbers: increment a counter when visit/finish

→ critical edges: $\Sigma = (1, Tv, C, L, pList)$
↳ update $L[p] \leftarrow \min(Tv[v], L[p])$ when revisiting not-self.
→ finish: if $L[v] \not< Tv[p]$, $C \leftarrow (v,p) :: C$
else, $L[p] \leftarrow \min(L[v], L[p])$

* directed G has back edge ⇔ DFSAll has back edge

→ topological sort: total order/dependencies
$a \le_p b \Rightarrow a \le_t b$, $\le_t$ is total ordering
↳ reverse sort finish times ↑ require DAG!

→ finding SCC — Kosaraju's algo.

DFS $G\ ((\Sigma, X), v) =$
 if $v \in X$ then (revisit $(\Sigma\ v)$, X)
 else let
  Σ' = visit Σ v, $X' = X \cup \{v\}$ (↙ add v to visited set)
  Σ'', X'' = iterate (DFS G) (Σ', X') $N_G^+(v)$
 in (finish Σ'' v, X) (explore everything reachable from v ↗)

DFSAll G Σ =
 iterate (DFS G) $(\Sigma, \{\})$ V (↙ vertex set)

operation	# times computed
$v \in X$	n+m
$X \cup \{v\}$, $N_G^+(v)$, visit, finish	n
revisit	m

→ work
- $\boxed{O(\log n)}$ for adjacency table → $O((n+m)\log n)$
- $\boxed{O(1)}$ for adjacency sequence → $O(n+m)$

Breadth-First Search

→ explore all new neighbors of current frontier at once in parallel (i.e. union)

→ diameter: length of longest shortest path = # layers to search graph from source

BFS G s =
 let explore X F = (↙ visited set of vertices) $\|F\| := \sum_{v \in F} 1 + d_G^+(v)$
  if $|F| = 0$ then X (↗ frontier empty = nth left to explore)
  else let $X' = X \cup F$, $F' = N_G^+(F) \setminus X'$
  in explore X' F' end (↳ new frontier = out-neighbors − visited vertices)
 in explore {} {s} end

→ compute work/per round in terms of $\|F\|$, add up w/r over all d=diameter rounds

G = Adj. Table, X = set, F = set	work $\in O((n+m)\log n)$, span $\in O(d \log^2 n)$
G = Adj. Seq, X = STSeq, F = Seq	work $\in O(n+m)$, span $\in O(d \log n)$

Shortest Paths

- $\forall G,\ \delta_G(s,v) \le \delta_G(s,u) + w_G(u,v)$
- BFS for unweighted graphs: single source
- Dijkstra: let $y \notin X$ be the vertex that minimizes $\delta_G(s,x) + w_G(x,y) = m$, then $\delta_G(s,y) = m$

  * only works when edge weights ≥ 0

  ↳ priority-first search, use for single-source

```
dijkstraPQ (G,s) =
  let dijkstra (X,Q) =            (minimum) PQ (lowest val, highest priority)
PQ empty ⇒ done
    case deleteMin Q of
      (NONE, _) ⇒ X
    | (SOME (d,v), Q') ⇒          if alr visited, ignore & continue
        if v∈X then dijkstra(X,Q')
        else let                   add to visited table.
          X' = X ∪ {v ↦ d}
for each neighbor u of v, add (d+w(u,v),u) to PQ
          relax (Q,(u,w)) =
              insert (Q,(d+w,u))
          Q'' = iterate relax Q' N+_G(v)
        in dijkstra(X',Q'') end
  in dijkstra({S}, singleton (0,s)) end
```

① inserting only if not visited do not change asymptotic costs (worst case e.g.)

② if $\exists$ decreaseKey function to change priority $\in O(1)$,
- if sth alr in PQ, don't need to delete & insert $\Rightarrow$ n inserts & deletes, m deckey
- new cost $O(m + n\log n)$ ← req. special heap (ex. Fibonacci)

③ usual work = span $\in O(m \log n)$

* if unweighted, multi-source (any $u \in U$): run BFS with U as initial frontier

A. shortest paths from U: BFS w/ U as frontier

B. shortest paths to U: reverse edges (via flatten & collect), then do A.

- Bellman Ford: for each vertex v, keep track of the shortest path from s to v that uses $\le k$ edges
  * works unless negative cycle exists → ret. NONE
  ↳ longest cycle has $(n-1)$ edges
    ⇒ if path lengths don't converge after n iterations, negative cycle must exist
  → each round takes $O(m)$ work, $O(\log n)$ span
    ↳ each edge considered as last edge of a k-edge path to v (round k)
  → if implemented w/ non-enumerable G: $W \in mn\log n$
- Johnson's: for all-pairs shortest paths
  - create a dummy node connected to all vertices
  - run Bellman Ford, $w' \leftarrow w - \phi(b) + \phi(a)$
  - run Dijkstra from every vertex, $mn\log n$ work, $m\log n$ span ⇒ run in parallel!

→ meld: joins 2 PQs, size, findMin ≠ deleteMin

→ sets: filterKey (key.t → bool) → set → set
  reduceKey f b x = Seq.reduce f b (toSeq x)
  ↑ similar for iterateKey

→ ordsets: first (least), last, prev/next S k
  split $(S,k) = (l,m,r)$, $m \leftarrow$ true if $k \in S$
  join $(a,b) = (a \cup b)$ if $\forall a < \forall b$, else Order (*)
  getRange $S(x,y) = \{k \in S \mid x \le k \le y\}$ $|S|-i$ largest elt
  rank$(S,k) = |\{k' \in S \mid k' < k\}|$, splitRank$(S,i) = (l,r)$
  select(S,i) = i^{th} smallest element   i smallest elements

→ table: insertWith f $(t,(k,v)) = t \cup \{k \to v\}$ if $k \notin t$, else $t \cup \{k \to f(v',v)\}$, v' existing
  tabulate f set = table $\{(k \to f(k)) : k \in S\}$
  mapKey f t = $\{(k \to f(k,v)) : (k \to v) \in t\}$
  filterKey p t = table of $(k \to v) \in t$ satisfying $p(k,v)$
  reduce, iterate, iteratePrefixes.

→ ordtable: first, last, prev, next ⇒ $k \mapsto v$
  split$(t,k) = (l$, v or NONE, $r)$, etc. (*)

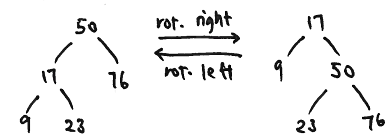
→ AugTable: domain T = set of all keys
  range T = sequence of all values, reduceVal
  collect ⟨(3,7),(3,5),(2,6)⟩ = ⟨(2,⟨6⟩),(3,⟨7,5⟩)⟩
  f must be associative!

SCCs: if we contract each SCC into a single vertex, we get a directed acyclic graph

Lemma: if u is the first vertex to be visited in its SCC, all vertices reachable from u finishes before u

SCC @ v = reachable(G,v) ∩ reachable(G^T,v)
  ↳ G^T = G with every edge reversed (in direction)

```
SCC G = let F = reverseFinish G,          if X[u], (X, [])
  accumSCCs ((X,L), u) =                   else, visit prev. unvisited vertices return in A
    let (X', A) = reach G^T X u
    in if |A|=0 then (X,L)
       else (X', append (L,<A>)) end
  in iterate accumSCCs ({},<>) F
```



- rotate tree T to left to get valid treap T', T may not be valid treap, but T satisfies BST inv. (T doesn't satisfy heap inv.)

WORST CASE COMPLEXITY of find in treaps: $O(n)$!!!

Seq.scan Table.union Table.empty S → $O(nm)$

① contraction: union 2 tables of size m

② expansion: even indices come from recursive step; odd indices come from f(ori[2i], rec[i])
  ↳ (i+1)-th index union: m & 2im
  $\Rightarrow W_{ex} = \sum_{i=0}^{n/2} m\log\left(\frac{2im}{m}\right) \in O(nm\log n)$

then, $W(n) = W(\frac{n}{2}, 2m) + O(nm\log n)$ → balanced.
  $\Rightarrow O(nm\log^2 n)$, here, $m=1 \Rightarrow O(n\log^2 n)$

* reduceFunc = merge, then cost of updating reducedVal of ancestors is $O(n)$ for root, $O(\frac{n}{2})$ for childs, ... $W(n) = W(\frac{n}{2}) + O(n) = O(n)$.

undirected graphs have NO CROSS EDGES.
  ↳ only appear b.c. ∃ directed edge from vertex searched later to vertex searched earlier BUT in undirected, they'd be in same branch.

* $\sum_{v \in V} \deg(v) = 2|E|$,

table/set/via BSTs: filter, map, tabulate:
  $W \in O\left(\sum_{(k,v)\in T} W(f(v))\right)$, $S \in O\left(\lg|T| + \max_{(k,v)\in T} S(f(v))\right)$

$\cap, \cup, \setminus$: $O(m\lg\frac{n+m}{m})$, $O(\lg(n+m))$

BSTs: join & joinMid $\in \log(|A|+|B|)$ work & span
  split$(T,k) \in \log|T|$ (implemented w/ treaps)
  ↳ keys hashed to gen. pseudo-random priorities.