Asymptotic Analysis

- Big O notation → ∃ constant $c > 0$
  → ∃ " $n_0 > 0$
  $T(n) = O(f(n))$ s.t. $T(n) \le c f(n)$ $\forall n > n_0$

- Methods to solve recurrence
  1) ~~Continuous~~ Repeated Substitution
  2) Draw Tree
  3) Master Thm / Akra Bazzi
  4) Guess & induction proof

Simple Recurrences
- $T(n) = 2T(\frac{n}{2}) + n \Rightarrow O(n \log n)$
- $T(n) = 2T(\frac{n}{2}) + 1 \Rightarrow O(n)$
- $T(n) = T(\frac{n}{2}) + n \Rightarrow O(n)$
- $T(n) = T(\frac{n}{2}) + 1 \Rightarrow O(\log n)$
- $T(n) = 1 + T(n-1) + T(n-2) \Rightarrow O(2^n)$

Master Thm: $T(n) = aT(\frac{n}{b}) + f(n)$ where $a \ge 1$; $b > 1$
- $f(n) = \Theta(n^d)$, where $d \ge 0$
  1. $a < b^d \longrightarrow T(n) = \Theta(n^d)$
  2. $a = b^d \longrightarrow T(n) = \Theta(n^d \log n)$
  3. $a > b^d \longrightarrow T(n) = \Theta(n^{\log_b a})$

Sterling's approximation: $\log(n!) \approx O(n \log n)$

Order of size

$f(n)$	
5	(constant)
$\log\log n$	(Double Log)
$\log n$	
$\log^2 n$	(polylogarithmic)
n ← $\sqrt{n}$	
$n \log n$	(log-linear)
n^3	(polynomial)
$n^3 \log n$	
n^4	
2^n	(exponential)
2^{2n}	
$n!$	

Hashing

Simple Uniform Hashing assumption
- Every key equally likely to map to every bucket
- Keys mapped independently

Assume: n items, m buckets

① Chaining
- insert: $O(1 + cost(h))$
- search: $O(L + cost(h)) = O(\frac{n}{m} + cost(h))$

w resize
- insert: amortized $O(1)$
- search: expected $O(1)$

cost of resize: $O(m1 + m2 + n)$

② Open Addressing
- operation $\le \frac{1}{1-\alpha}$ & $E(\text{search}) = O(1)$

Implement Set
- Fingerprint — $P(\text{false pos}) \le 1 - (\frac{1}{2})^{\frac{m}{n}}$ → at most p → $\frac{m}{n} \le \ln(\frac{1}{1-p})$
- Bloom Filter — optimal $k = \frac{m}{n} \ln(2)$

Graph

No negative W cycles	Bellman-Ford	$O(VE)$
No negative E	Dijkstra (PQ)	$O(E \log V)$
No directed cycles (acyclic) ~~tree~~	One pass Bellman Ford	$O(V+E) = O(E)$
	Topo Sort + Relax	
	① DFS Post-order	
	② Kahn Algo	
No cycles (undirected) tree	BFS/DFS + relax	$O(V+E) = O(V)$
unweighted	BFS	$O(V+E)$

SSSP
- early-termination after sequence of $|E|$ relax
- Uses AVL & Hashtable
  - $|V|$ deletion/Min
  - $|E|$ relaxations decreasekey

relax(u,v)
if (d[v] > d[u] + w(u,v))
d[v] = d[u] + w(u,v)

Bonwika $O(V+E) \log V$
1 Bonwika step:
- For each connected component, search for min weight outgoing edge (DFS/BFS)
- add selected edges & merge connected comp.

At every step, at most $\frac{n}{2}$ connected components. → At most $\log V$ bonwika steps.

Kaushal $O(E \log E)$ sorting
// union-Find DS
1) Sort edges by weight
2) consider edges in ascending order
   - if both endpts in same blue cycle free, color edge red
   - otherwise, color blue

// $O(E)$ if weight $\{1, \dots 10\}$
// $O(\alpha E)$ " " "

MST Properties
① No cycles
② If cut, both pieces are MST
③ Cycle Property
④ Cut property

PRIM
1) Maintain set of visited nodes
2) Greedily grow set by connecting node by lightest edge
3) Use PQ to order nodes by edge weight

Analysis $(O(E \log V))$
- each V added & removed once from PQ $(V \log V)$
- each edge E uncovered a decreasekey $(E \log V)$

Max Spanning Tree
- Scn 1) Reverse sign of Edge Weight, run MST
- Scn 2) Run Kaushal in reverse

Steiner Tree Prob
- MST of graph w required nodes & steiner nodes
1) Get APSP
2) Construct new graph on required nodes
   V = required nodes
   E = shortest path distances
3) Run MST on new graph → edges on new graph
4) Map NEW edges back to original graph, following SP found in step 1

APSP
① Run SP on every node
- Store & respond to query
  - $O(V^3 E \log V)$ Dijkstra
  - $O(V^2 \log V)$ sparse
  - $O(V(E+V))$ unweighted
  - $O(V^3)$ dense
  - $O(V^2)$ sparse
② Floyd Warshall — $S[v,w,p]$ from v to w passing p
   → $S[v,w,\phi] = E[v,w]$
   $S[v,w,P_8] = \min(S[v,w,P_7],$ // don't use 8
   $S[v,8,P_7] + S[8,w,P_7])$ // use 8

	find	union
Quickfind	$O(1)$	$O(n)$
Quick union	$O(n)$	
weighted union	$O(\log n)$	
path compression	$O(\log n)$	
both	$\alpha(m,n)$	

DP
- optimal sub struct
- overlapping sub prob

LIS
- $S[i] = LIS(A[i \dots n])$
- $S[n] = 0$
- $S[i] = (\max_{(i,j) \in E} S[j] + 1)$

$O(n^2)$

$P_0 = \phi$
$P_1 = \{1\}$
$P_2 = \{1,2\}$
...

Bounded Prob
- $P[v,k]$ start at v, take k steps
  → $P[v,0] = 0$
  $P[v,k] = \max\{P[w_1, k-1] + w(v,w_1), P[w_2, k-1] + w(v,w_2), \dots\}$

$O(KE)$ use table view analysis

Vertex Cover
- $S[v,0]$ & $S[v,1]$ subtree rooted at v if v covered or not
  → $S[leaf,0] = 0$ & $S[leaf,1] = 1$
  $S[v,0] = S[w_1,1] + S[w_2,1] + \dots + S[w_k,1]$
  $S[v,1] = w_1 + w_2 + w_3$
  where $w_1 = \min(S[w_1,0], S[w_1,1])$
  ...

- 2V sub prob
- $O(E) = O(V)$ to solve all sub prob

Algos

- Binary search
- FindPeak // local non-steep peak
- BubbleSort // biggest j items final
- SelectionSort // small j items final
- InsertionSort // small j items sorted
- MergeSort
- QuickSort → QuickSelect $O(n)$ // 3-way allows duplicates
- Order statistics (store weight) $O(\log n)$
  → Select(k), rank(node)
- Intervals (sorted by left end pt) $O(\log n)$
  → IntervalSearch (stores max)
- Orthogonal Range Search (BST)
  (store info in leaves) (max val. in left subtree)
  Build $O(n \log n)$, Space $O(n)$, → Query $(O(\log n))$

Trees Traversal: $O(n)$
- In order (LNR), Preorder (NLR), post order (LRN)

Successor $O(\log n)$
1) Search for key in tree
2) if (res ≥ key) return res
3) if (res ≤ key) search for succ of res

// if no succ will return null

Delete $O(\log n)$
- case 1) No children
- " 2) 1 child
- " 3) 2 children → replace v w its successor

AVL (v.left.ht − v.r.ht ≤ 1)

Binary tree $\boxed{n < 2^{h+1}}$

AVL $\boxed{n > 2^{\frac{h}{2}}}$ → $\boxed{h < 2\log n}$

insert (at most 2 rotations)
del (~~at most~~ $O(\log n)$ rotations)

ab-trees

node	# keys min	# keys max	# children min	# children max
root	1	b-1	2	b
internal	a-1	b-1	a	b
leaf	a-1	b-1	–	–

- $2 \le a \le \frac{b+1}{2}$
- insertion (if keys == b, bring median up) ← normal root
- deletion – y, z siblings
  - case 1: y & z ≤ (b-1) keys → merge
  - case 2: y & z = (b-1, 2b) keys → share (merge + split)

1. (a,b) child policy
2. key-ordering (array of key, subtree pairs)
3. leaf depth

Heap
- insert — add leaf — bubbleUp
- increase/decrease key — update prio — b up/down
- delete — swap(x, last()) — remove(last()) — bubbleDown

left(x) = 2x+1 ; right(x) = 2x+2
parent(x) = floor$(\frac{x-1}{2})$

// Faster than AVL although no search()

- HeapSort unsorted list ① → heap ② → sorted list

```
// int[] A = arr of unsorted int
for (int i = n-1; i>=0; i--) {
    bubbleDown(i, A)
}
O(n)
```

```
// int[] A = arr stored as heap
for (int i = (n-1); i>=0; i--) {
    int val = extractMax(A);
    A[i] = val;
}
O(n log n)
```

// in place
// Deterministic
// Not stable