Data Structures and Algorithms

Curated computer science study track (not a degree, no licensure claim). Original explanations and practice only, following the reviewed course handout.

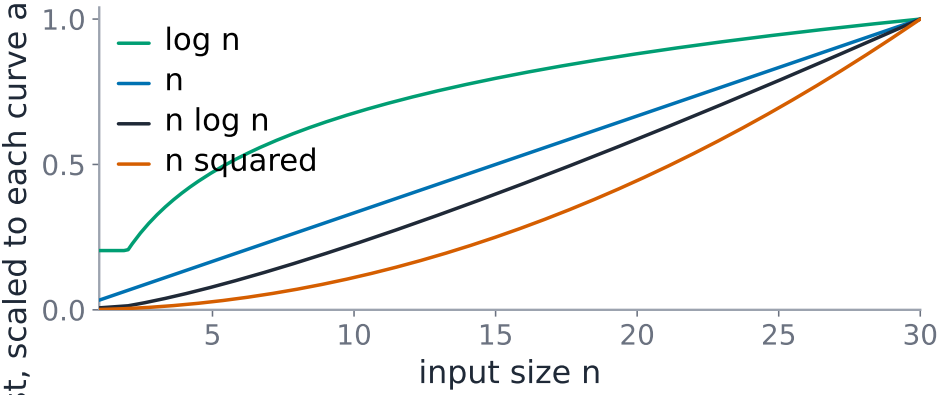
Containers and recipes, chosen by cost

- 5.1 is the yardstick (how cost grows); 5.2-5.5 the containers; 5.6-5.7 the recipes.
- Every cost claim in this deck was counted on the exact datasets printed in the handout.

Asymptotic analysis: read the exponent off the data

1. **Classes** At n = 1000: n log n is about 9 966 and n squared is a million - a factor of 100; log of a million is about 20.
2. **Logs** Halving each step touches log n levels: 16 items resolve in 4 probes because 2 to the 4th is 16.
3. **Doubling** Nested loops at n = 100 run 10 000 bodies; at n = 200 they run 40 000 - the ratio 4 is the exponent.

Complexity curves



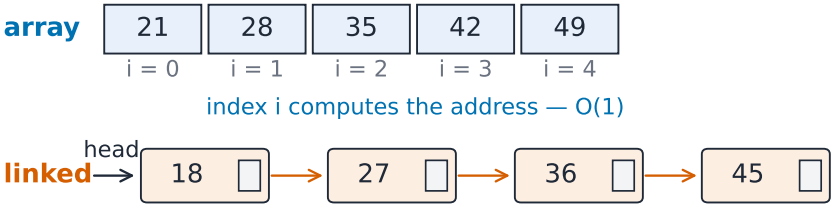
Growth of the common classes over one axis: logarithmic stays flat, quadratic bends the page.

Lists, stacks and queues: which end is cheap

1. **Layout** A Python list is a contiguous **array** - index i computes the address; a **linked list** chains nodes, splicing cheaply but walking to index.
2. **Amortise** Doubling regrows make append amortized O(1); n front inserts move Theta(n squared) - the count quadruples when n doubles.
3. **Trace** push 3, 7, 1 then pop, pop returns 1 then 7 (last in, first out); enqueue a, b, c then two dequeues return a then b.

Array and linked layouts

one block of cells, or nodes chained by pointers



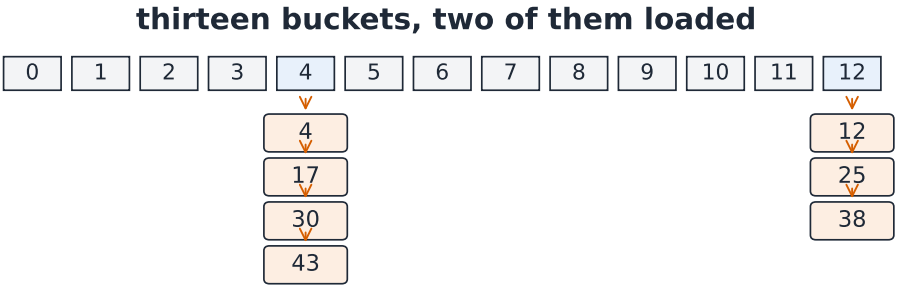
stack: push and pop at ONE end **queue: in one end, out the other**

One contiguous block with computed addresses versus nodes chained by pointers; the **stack** and **queue** differ only in which ends they use.

Hashing: spread, collide, resolve

1. **Buckets** h(k) = k mod 13 sends 4, 17, 30, 43 all to bucket 4 - a chain of four; keys 12, 25, 38 fill bucket 12; alpha = 7/13, about 0.54.
2. **Probing** Linear probing walks the cluster: $1 + 2 + 3 + 4 = 10$ probes to place those four keys - **clustering** grows the runs.
3. **Keep** Real tables hold alpha under about 0.75 and resize - that is why dictionaries stay fast.

Hash buckets



bucket 4: 4 → 17 → 30 → 43 **bucket 12: 12 → 25 → 38**

load factor α = 7 keys / 13 buckets ≈ 0.54

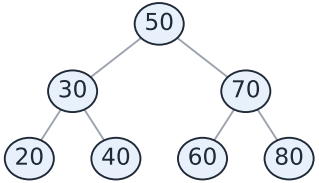
Thirteen buckets under h(k) = k mod 13: keys 4, 17, 30, 43 chain in bucket 4 and 12, 25, 38 chain in bucket 12.

Trees and heaps: height is the cost

1. **BST** Inserting 50, 30, 70, 20, 40, 60, 80 builds height 2; searching 40 costs 3 comparisons - the level count; inorder reads sorted.
2. **Heights** A perfect tree of height h holds 2 to the h+1 minus 1 nodes: 7, 15, 31 - so balanced n needs floor of log n levels.
3. **Heap** Inserting 5, 3, 8, 1 leaves [1, 3, 8, 5]: 1 sifted over 5 then 3; children of i sit at 2i+1 and 2i+2.

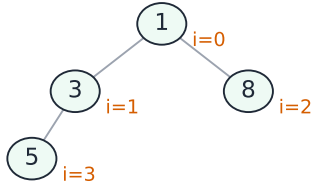
Tree and heap diagrams

BST: height 2



inorder walk: 20 30 40 50 60 70 80

min-heap array [1, 3, 8, 5]



children of i sit at 2i+1 and 2i+2

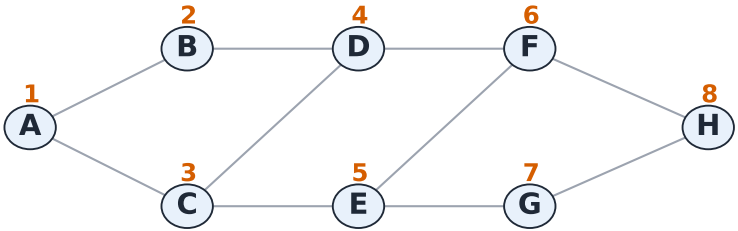
Left: the BST built by inserting 50, 30, 70, 20, 40, 60, 80. Right: the **min-heap array** [1, 3, 8, 5] as its implicit tree with index labels.

Graphs and traversal: two probes, one graph

1. **Storage** 8 nodes, 10 undirected edges: the **adjacency list** holds 20 entries, the matrix 64 cells - the list wins sparse, the matrix dense.
2. **BFS** From A the layers read A, then B C, then D E, then F G, then H - the layer is the shortest hop count.
3. **DFS** From A the deep walk reads A, B, D, C, E, F, H, G - one path with backtracks.

Graph with traversal order

breadth-first from A: the numbers are the arrival order



layers: A | B C | D E | F G | H — the layer is the shortest hop count from A

The course graph: breadth-first from A visits in the numbered order A, B, C, D, E, F, G, H - layer by layer.

exercises.ioi ! exercises.com.cn

Sorting: the inversion bill

1. **Inversions** The dataset [5, 2, 9, 1, 7, 3] holds 8 inversions - every adjacent swap fixes exactly one, so swap-adjacent sorts pay 8.
2. **Insertion** Shifts per insert: 1, 0, 3, 1, 3 - total 8 shifts and 11 comparisons, result [1, 2, 3, 5, 7, 9].
3. **Merge** Halve, sort halves, merge: 10 comparisons on the same data; at $n = 1000$ the families divide by 100.

Insertion-sort snapshots

each row is one insertion; the pale prefix is sorted

start	5	2	9	1	7	3
insert 2 (1 shift)	2	5	9	1	7	3
insert 9 (0 shifts)	2	5	9	1	7	3
insert 1 (3 shifts)	1	2	5	9	7	3
insert 7 (1 shift)	1	2	5	7	9	3
insert 3 (3 shifts)	1	2	3	5	7	9

8 shifts = the inversion count

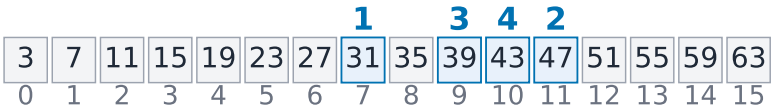
Insertion sort on the course dataset: each row is one insertion; the pale prefix is already sorted.

Searching: halve the candidates

1. **Trail** Searching 43 probes indices 7 (31), 11 (47), 9 (39), 10 (43) - four probes; linear walks 11 values to the same slot.
2. **Miss** Searching 64 probes 7, 11, 13, 14, 15 - five probes, the worst case, then the empty range reports absent.
3. **Trap** Binary on unsorted data returns wrong answers silently; lo = mid (not mid + 1) loops forever.

Binary-search probes

searching 43: numbered probes 1-4, one answer



1: 31 < 43, go right · 2: 47 > 43, go left · 3: 39 < 43, go right · 4: found

each probe halves the range: at most 5 probes for 16 slots, linear up to 16

The 16-slot sorted **array** with the probe trail for 43: mid at 7 (too small), 11 (too big), 9 (too small), 10 - found in 4 probes.

exercises.lol exercises.com.cn

Check yourself

- Doubling n quadruples the count — the class is…? (Quadratic.)
- Keys 4, 17, 30, 43 under mod 13 land in…? (Bucket 4.)
- Children of **heap** index i sit at…? (2i+1 and 2i+2.)
- Sixteen slots, **binary search**, worst case…? (5 probes.)