

INDEX

Numbers

2-3 trees, 304, 311–312. *See also* B-trees

A

abstract data types (ADTs), 37–47

abstraction, 39

creators, 40

implementing, 40–46

with classes, 41

with functions, 43, 45

mutations, 39–40

mutators, 40, 44–45

observers, 40

operations, 39–40

producers, 40

abstraction, 38, 39

adaptive sorting, 92

addressable heap, 346

Adelson-Velsky, Georgy, 249

ADTs (abstract data types),
37–47

algorithms

analysis of, 47, 50, 52, 55

backtracking, 69–71, 89

brute-force, 82

complexity, 50, 52

design, 63

divide-and-conquer, 65–68, 72

performance, 50–58

amortized time performance, 54

archery puzzle, 89

assured balance binary search
trees, 249

asymptotic notations, 51

average-case performance of
algorithms, 54

AVL trees, 235, 249–254, 261

adding to, 251

creating, 250

number of nodes, 280

performance, 255

removing from, 251

rotation, 252

B

Babel, 13, 19

backtracking, 69, 70, 71, 89

bags, 40–47, 203–205, 219–220,
232–233

binary search trees, implementing
with, 239

lists, implementing with, 207

operations on, 204

Bayer, Rudolf, 291

BB[α] trees. *See* weight-balanced
bounded trees

Bellman-Ford shortest path
algorithm, 466

best-case performance of
algorithms, 54

bidirectional bubble sort, 99

Bierce, Ambrose, 218

big Omega notation, 51

big O notation, 51–59

big Theta notation, 51

binary heap, 318, 340, 341

binary search, 56–57, 59, 166–168, 172

binary search trees, 239–282, 485

adding values to, 241–242

assured balance, 249–261

AVL trees, 249–254, 261

bag, implementing, 239

balanced, 249–250

deleting from, 485

finding the maximum, 244–245

inorder traversal, 246–248, 280

operations on sets, 239

performance, 248–249

- binary search trees (*continued*)
 - postorder traversal, 246, 280
 - preorder traversal, 246, 279–280
 - probabilistically balanced, 249, 261–278, 281–282
 - randomized binary search trees, 262–270, 281
 - rebalancing, 282
 - red-black trees, 304–314
 - removing values from, 242–244
 - searching in, 239–241
 - self-adjusting trees, 249, 261–278, 281–282
 - set, implementing, 239
 - play trees, 270–278, 281–282
 - traversing, 246–248, 280
 - weight-bounded balanced trees, 255–261, 281
 - binary trees, 237–282, 287, 288, 304
 - 2-3 trees, 304, 311–312
 - complete, 238, 244, 247, 279
 - copying, 279
 - full, 237–238, 279, 280
 - heap-ordered, 347
 - height, 237, 238, 249–257, 279, 280
 - perfect, 238, 248, 256, 279, 282
 - size, 279
 - binomial heaps, 351–367, 385
 - adding values to, 354–356
 - changing values in, 360–362
 - implementing, 353–354
 - merging, 356–358
 - performance, 362–363
 - removing values from, 358–360
 - binomial trees, 351–352
 - bitmaps, 206
 - bitmap selection, 122–123
 - bitmap sort, 112–113
 - bogosort, 117
 - brute-force algorithms, 82–86, 87
 - B-trees, 291–303, 315. *See also* red-black trees
 - adding keys to, 295–298
 - implementing, 292–293
 - inorder traversal, 294–295
 - optimization, 315
 - performance, 303
 - removing keys from, 298–303
 - searching in, 293–294
 - traversing, 294–295
 - bubble sort, 97–98, 100, 103–104, 118
 - Burton, F. Warren, 470
- ## C
- chaining, hashing with, 219–221
 - change, making, 87
 - circular lists, 195–200
 - adding elements to, 197
 - implementing, 198–199
 - joining, 200
 - operations on, 196
 - performance, 199–200
 - removing elements from, 197–199
 - cocktail shaker sort (shuttle sort), 99–100
 - coin tossing shuffling, 140–141
 - comb sort, 103–104
 - complexity of algorithms, 50–58
 - connectivity detection, 428, 453–458
 - search-based algorithm, 456–458
 - sets-based algorithm, 454–456
 - counting selection, 123–124
 - counting sort, 114–115
 - cryptarithmic puzzles (cryptarithms), 83–86, 90
 - cycle detection, 427, 452–453
 - Tarjan’s algorithm, 448, 452–453
- ## D
- d-ary heaps, 340–341
 - data structures
 - binary search trees, 239–282, 485
 - dequeues, 191–195, 201
 - hash tables. *See* hashing
 - heaps. *See* heaps
 - lists, 177–184, 195–200, 201
 - queues, 188–191, 201, 326–327, 342, 346–347, 531
 - stacks, 184–188, 200, 201
 - treaps, 332–340, 343
 - trees. *See* trees
 - tries. *See* tries
 - declarative-style programming, 26–30
 - dequeues (double-ended queues), 191–195, 201
 - adding elements to, 192–194
 - implementing, 194–195

- operations on, 192
- performance, 195
- removing elements from, 193
- Dijkstra's algorithm, 438
 - simplifying, 466
- divide-and-conquer, 65–68, 72
- double hashing, 226–229
- double hashing with prime lengths, 229–232
- doubling search. *See* exponential search
- doubly linked lists. *See* circular lists
- Dutch National Flag Problem, 119
- dynamic programming (DP), 63, 72–82, 89–90
 - bottom-up, 72, 79–82
 - calculating Fibonacci series, 72–74, 79–80
 - line breaking, 74–79, 89
 - memoization, 72–74, 78–81
 - summing ranges, 80–82, 90
 - tabulation, 72
 - top-down, 72–79

E

- ECMAScript, 6, 8, 10–11, 13
- ESLint, 18, 19
- exponential search, 168–169, 173
- extended heaps
 - binomial heaps, 351–363, 385
 - Fibonacci heaps, 367–376, 386
 - lazy binomial heaps, 363–367
 - pairing heaps, 376–384
 - skew heaps, 347–351, 385
- external sorting, 92

F

- factorial, 52, 53, 65–66, 82, 88
- Fibonacci heaps, 367–376, 386
 - adding values to, 371
 - changing values in, 372–375
 - implementing, 368–369
 - merging, 370–371
 - performance, 375–376
 - removing values from, 371–372
- Fibonacci series, 66–67, 72–74, 79–80, 89
 - calculating with bottom-up DP, 79–80

- calculating with top-down DP, 72–74
- Fira Code Font, 15–16
- Fisher-Yates sampling, 151–152
- Fisher-Yates shuffling, 145–146, 151–152, 156
- Flow type checker, 18
- Floyd, Robert, 142, 329
- Floyd's sampling, 148–150
- Floyd's shuffling, 142–143
- Floyd-Warshall algorithm, 430–434
- forests, implementing, 288
- FORTH, 185
- FORTRAN, 24
- functional data structures, 470–484
 - arrays, 470
 - binary search trees, 478–481
 - common lists, 470–473
 - Fibonacci heap, 481
 - hash tables, 470
 - queues, 474–478
 - stacks, 473–474
- functional programming (FP)
 - declarative style, 26–30
 - higher-order functions, 30–32
 - impure functions, 33–34
 - reasons for using, 24
 - side effects, 32–33

G

- galloping search. *See* exponential search
- general trees, 237
- graphs
 - adjacency lists representation, 429–430
 - adjacency matrix representation, 428–429
 - adjacency set representation, 430
 - arcs, 425
 - arrows, 425
 - Bellman-Ford algorithm, 434–438
 - connectivity detection, 428, 453–458
 - cycle detection, 427, 452–453
 - defined, 425–427
 - degree of, 425
 - Dijkstra's algorithm, 438–444
 - edges, 425
 - Floyd-Warshall's algorithm, 430–434

- graphs (*continued*)
 - Hamiltonian cycle, 88
 - Kahn's algorithm, 445–448
 - Kruskal's algorithm, 88, 462–465
 - links, 425
 - minimum spanning trees, 88, 427, 458–465
 - neighbor, 425
 - nodes, 425
 - points, 425
 - Prim's algorithm, 459–462
 - representing, 428–430
 - shortest path problem, 427, 430–443, 466
 - sorting, 444–452
 - Tarjan's algorithm, 448, 452–453
 - topological sort, 427, 445–448, 451–452, 466
 - vertices, 425
 - greedy algorithms, 63, 87–88
- H**
- hashing, 218–232, 233
 - adding values to, 220, 224, 228, 231
 - chaining, with, 219–221
 - collision, 219, 222
 - creating, 220, 223–224, 227, 230–231
 - double hashing, 226–229
 - double hashing with prime lengths, 229–232
 - hash remainder function, 218
 - load factor, 222, 223, 225, 226, 228
 - open addressing, with, 221–226
 - performance, 221, 225
 - removing values from, 221, 225, 229, 232
 - resizing, 233
 - searching in, 220, 224, 228, 231
 - heaps
 - adding to, 321–323
 - addressable heaps, 346–347
 - binary heaps, 318–325
 - binomial heaps, 351–363, 385
 - d-ary heaps, 340–341
 - Fibonacci heaps, 367–376, 386
 - Fredman, Michael, 368
 - heap-ordered binary trees. *See* skew heaps
 - heap property, 318, 319–320, 330, 332–337
 - implementing, 320–325
 - lazy binomial heaps, 363–367
 - leftist heaps, 347
 - max heap, 319
 - min heap, 319
 - operations, 320
 - pairing heaps, 376–384
 - performance, 325
 - priority queues, 326–327, 342
 - quaternary heaps, 340
 - removing from, 323–325, 342
 - searching, 342, 386
 - skew heaps, 347–350, 385
 - structure property, 318–319, 332
 - treaps, 332–340, 343
 - trinary heaps, 340
 - heapsort, 320, 327–331, 342, 343
 - analysis, 329
 - Floyd's heap-building enhancement, 329
 - Williams' original heapsort, 327
 - higher-order functions, 30–32
 - Hindley-Milner type system, 41
- I**
- immutability. *See* functional data structures
 - impure functions
 - avoiding state, 33–34
 - using injection, 34
 - inefficient sorting, 116–117
 - infinite loop. *See* infinite loop
 - inorder traversal of binary search trees, 246–248, 280
 - in-place sorting, 93
 - insertion sort, 101–103, 104, 105, 108, 111, 118
 - internal sorting, 92
 - interpolation search, 169–171
- J**
- JavaScript, 3–21, 25–34. *See also*
 - functional programming (FP)
 - arrow functions, 4–5
 - classes, 5–6

- closures, 11–13
- CommonJS modules, 9
- destructuring, 7–8, 9–10
- development tools, 13–20
- ECMAScript modules, 10–11
- filtering arrays, 27
- as a functional language, 25–34
- .indexOf method, 160
- looping through arrays, 30
- modules, 8–11, 13
- .pop method, 185, 200
- .push method, 185, 200
- .random method, 138
- reducing arrays to a value, 29–30
- searching arrays, 27–28
- sleep sort, 117
- .sort method, 95–96
- spread operator, 6–7
- testing arrays, 28, 36
- transforming arrays, 28–29, 35
- JSDoc, 16–18
- jump search, 163–166, 172

K

- Kahn’s algorithm, 445–448
- Knuth, Donald, 52, 152, 157
- Knuth’s sampling, 152
- Kruskal’s algorithm, 462–465

L

- Landis, Evgenii, 249
- lazy binomial heaps, 363–367
 - adding values to, 364–365
 - changing values in, 366–367
 - implementing, 363–364
 - performance, 367
 - removing values from, 365–366
- lazy selecting, 132–134
- leftist heap, 347
- Lehmer code, 144
- linear search, 160–162
- LISP, 24, 177
- lists
 - adding values to, 182
 - appending to, 200
 - circular lists, 195–200
 - creating, 182
 - deques, 191–195, 201

- getting values at a position, 183
- implementing with arrays, 179–180
- implementing with dynamic
 - memory, 180–183
- operations, 178
- ordered lists, 207–210
- performance, 184
- queues, 188–191, 201
- removing values from, 182–183
- searching in, 183–184
- self-organizing, 215–218, 232
- skip lists, 210–215, 232
- stacks, 184–188, 200, 201
- lottery drawing sampling, 150–151
- Loyd, Sam, 70, 89
- Lucas, Édouard, 67

M

- maps, 203, 205, 220, 221
 - defined, 203
 - operations, 205
- max heap, 319
- maze, solving, 69–70
- McCreight, Edward, 291
- median of medians selection,
 - 127–130
- meldable priority queues (MPQs),
 - 346–347
 - operations, 347
- merge sort, 93, 110–112, 119, 133
- min heap, 319
- minimum spanning trees, 88, 427,
 - 458–465
 - Kruskal’s algorithm, 462–465
 - Prim’s algorithm, 459–462
- Mozilla Developer Network
 - (MDN), 13
- multiset, 40–47
- mutual recursion. *See* recursion,
 - mutual

O

- object-based tries, 401–405
 - adding keys to, 404–405
 - implementing, 402
 - performance, 406
 - removing keys from, 405–406
 - searching, 402–404

- object-oriented programming (OOP),
 - 23, 32
- offline sorting, 93
- O’Neill, Melissa, 470
- online sorting, 93
- only one value sampling, 146–147
- open addressing, hashing with,
 - 221–226
- orchards, 284
- ordered lists, 207–210
 - adding values to, 208–209
 - performance, 210
 - removing values from, 209–210
 - searching in, 207–208
 - sentinels and, 211, 212, 232
- out-of-place sorting, 93

P

- pairing heaps, 376–384
 - adding values to, 378
 - changing values in, 382–384
 - implementing, 377
 - melding (merging), 377–378
 - performance, 384
 - removing values from, 378–381
- Pandita, Narayana, 84
- partition-exchange sort. *See* quicksort
- Pascal’s triangle, 352
- performance, algorithm, 50–59
 - amortized time, 54
 - average case, 54
 - best case, 52, 54, 59
 - big Omega notation, 51–52
 - big O notation, 51–52, 57, 59
 - big Theta notation, 51, 59
 - classes, complexity, 52–54
 - constant order, 52, 53, 54
 - exponential order, 52, 53
 - factorial order, 52, 53
 - linear order, 52, 53, 55
 - logarithmic order, 52, 53
 - log-linear order, 52
 - measurements for, 54–55
 - quadratic order, 52, 53
 - small omega notation, 51, 52, 57
 - small o notation, 51, 52, 59
 - trade-offs, 57–58
 - worst case, 52, 54, 55, 57, 59

- performance measurements for
 - algorithms, 54–55
- performance trade-offs, 57–58
- permutation, generating next, 84–86
- permutation sort, 117
- persistent data structures. *See*
 - functional data structures
- PostScript, 185
- Prettier formatting, 16
- Prim’s algorithm, 459–462
- priority queues, 326–327, 342,
 - 346–347, 531
 - addressable, 346–347
 - addressable heaps, 346
 - leftist heap, 347
 - meldable, 346–347
 - operations, 326, 347
- probabilistic balance binary search
 - trees, 249, 261–278, 281–282

Q

- quaternary heaps, 340
- queues, 188–191, 201, 326–327, 342,
 - 346–347, 531
 - adding elements to (queueing), 189
 - implementing, 189–191
 - operations, 188
 - performance, 191
 - priority, 326–327, 342, 346–347, 531
 - removing elements from
 - (dequeueing), 189–190
- quickselect, 125–132, 135
- quicksort, 105–110, 119, 125–126, 127,
 - 247, 264
 - dual-pivot selection, 108–110
 - hybrid method, 107–108
 - “median of three” pivot
 - selection, 107
 - pivot selection techniques, 106–107
 - standard version, 105–106

R

- radix sort, 112, 115–116, 119
- radix trees, 406–413
 - adding keys to, 410–411, 424
 - creating, 407
 - performance, 414

- removing keys from, 412–414
- searching, 407–409
- randomized binary search trees,
 - 262–270, 281
 - adding keys to, 263–264
 - creating, 262–263
 - joining, 268–269
 - performance, 269–270
 - removing keys from, 267–268
 - splitting, 264–267
- random number generation, 138
- React Redux, 40
- recursion, 64–72, 82, 89, 208
 - divide-and-conquer, 65–68, 72
 - memoization, 73–74, 78–81
 - mutual. *See* mutual recursion
- red-black trees, 304–314
 - adding keys to, 306–307
 - implementing, 305–306
 - performance, 313–314
 - removing keys from, 309–313
 - restoring structure of, 307–309
- repeated step pivot selection,
 - 130–132
- reservoir sampling, 153–154
- reverse Polish notation (RPN), 185
- Robson, J. M., 143
- Robson’s algorithm, 143–145, 156

S

- sampling, 146–153, 156, 157
 - Fisher-Yates algorithm, 151–152
 - Floyd’s algorithm, 148–150
 - Knuth’s algorithm, 152–153
 - lottery drawing, 150–151
 - only one value, 146–147
 - with repetition, 146–147, 156
 - without repetition, 147–154
 - reservoir sampling, 153–154
 - several values, 147
 - by sorting or shuffling, 148
- search
 - binary search, 56–57, 59,
 - 166–168, 172
 - definition, 159–160
 - exponential search, 168–169, 173
 - interpolation search, 169–171
 - jump search, 163–166, 172

- linear search, 160–161
- linear search with sentinels,
 - 162–163
- ordered arrays, 163–171
- unsorted arrays, 160–162
- Sedgewick, Robert, 304, 312
- selecting
 - bitmap selection, 122–123
 - counting selection, 123–124
 - lazy select, 132–134
 - median of medians pivot selection,
 - 127–130
 - merge sort, 133
 - quickselect, 125–132, 135
 - quicksort, 105–110, 119, 125–126,
 - 127, 247, 264
 - repeated step pivot selection, 130,
 - 131, 135
 - selecting with comparisons,
 - 124–125, 130
 - selecting without comparisons,
 - 122–124
 - selection sort, 100–101,
 - 124–125, 135
- self-adjusting trees, 249, 261–278,
 - 281–282
- self-organizing lists, 215–218, 232
 - adding values to, 217
 - count ordering strategy, 218
 - move to front (MTF) strategy, 218
 - performance, 217–218
 - removing values from, 217
 - searching in, 215–217
 - swap with previous strategy, 218
 - variants, 217–218
- sets
 - binary search trees,
 - implementation with,
 - 239–282
 - bitmaps, implementation
 - with, 206
 - hashed, 233
 - JavaScript objects, with,
 - 205–206
 - lists, implementation with,
 - 207–218
 - operations, 204
- Shell sort, 104–105

- shortest path problem, 427, 430–443, 466
 - Bellman-Ford algorithm, 434–438
 - Dijkstra’s algorithm, 438–444
 - Floyd-Warshall’s algorithm, 430–434
- shuffling, 139–146, 148, 151, 155–157
 - by coin tossing, 140–141
 - Fisher-Yates algorithm, 145–146, 151–152, 156
 - Floyd’s algorithm, 142–143
 - in linear time, 142–146
 - permutation, 139, 142–146, 155
 - Robson’s algorithm, 143–145, 156
 - by sorting, 139–140
- shuttle sort, 99–100
- side effects
 - keeping inner state, 32
 - mutating arguments, 33
 - returning impure functions, 33
 - using global state, 32
- sinking sort, 99, 118
- skew heaps, 347–350, 385
 - adding keys to, 350
 - implementing, 348
 - merging, 348–350
 - performance, 350–351
 - removing keys from, 350
- skip lists, 210–215, 232
 - adding values to, 213–214
 - creating, 211–212
 - performance, 215
 - removing values from, 214–215
 - restructuring, 232
 - searching in, 212
- sleep sort, 117
- slow sort, 116
- small omega notation, 51, 52, 57
- small *o* notation, 51, 52, 59
- sorting
 - adaptive, 92–93
 - bidirectional bubble sort, 99
 - bitmap sort, 112–113, 115
 - bogosort, 117
 - bubble sort, 97–98, 100, 103–104, 118
 - cocktail shaker sort, 99
 - comb sort, 103–104
 - counting sort, 114–115
 - Dutch National Flag Problem, 119
 - external, 92
 - heapsort, 320, 327–331, 342, 343
 - inefficient, 116–117
 - in-place, 93
 - insertion, 108
 - insertion sort, 101–103, 104, 105, 108, 111, 118
 - internal, 92
 - merge sort, 93, 110–112, 119, 133
 - offline, 93
 - online, 93, 105
 - out-of-place, 93
 - partition-exchange sort, 105
 - performance, 93, 96–98, 101, 105–108, 110–112, 117–119
 - permutation sort, 117
 - preorder traversal of binary search trees, 280
 - quicksort, 105–110, 119, 125–126, 127, 247, 264
 - radix sort, 112, 115–116, 119
 - selection sort, 100–101, 124–125, 135
 - Shell sort, 104–105
 - shuttle sort, 99
 - sinking sort, 99, 118
 - sleep sort, 117
 - slow sort, 116
 - .sort method, 95–96
 - stability, 93–94, 96, 111, 116, 118
 - stooge sort, 116
 - Tim sort, 111
- splay trees, 270–278, 281–282
 - adding keys to, 274–275
 - performance, 278
 - removing keys from, 275–278
 - searching in, 274
 - splaying a tree, 271–274
- Squarest Game on the Beach, 70, 89
- stacks, 184–188, 200, 201
 - adding to (pushing), 185–187
 - implementing with arrays, 185
 - operations, 185
 - performance, 188
 - popping (removing from), 185–187

Stirling's approximation, 97
stooge sort, 116

T

Tarjan, Robert, 368, 448, 452–453

Tarjan's algorithm, 448, 452–453

tautologies, detecting, 82–83

ternary heaps, 340–341

ternary search tries, 414–424

adding keys to, 417–419

creating, 415–416

performance, 423

removing keys from, 419–423

searching in, 416–417

storing extra data in, 416

Tim sort, 111

topological sort, 427, 445–448,
451–452, 466

Kahn's algorithm, 445

Tarjan's algorithm, 448, 452–453

Towers of Hanoi, 67, 89

traveling salesman problem, 84, 87–88

treaps, 332–340, 343

adding keys to, 333–336

creating, 332–333

merging, 343

performance, 339–340, 344

removing keys from, 336–339, 343

searching, 332–333

splitting, 343

trees

2-3 trees, 304, 312

adding nodes to, 286

ancestors, defined, 237

AVL trees, 235, 249–254, 261

binary search trees, 239–282, 485

binary trees, 237–282, 287, 288, 304

binomial trees, 351–352

breadth-first (level-order)

traversal, 290–291

B-trees, 291–303, 315

children, defined, 237

defining, 283–290

degree, 237, 284

depth-first traversal, 289–290

descendants, defined, 237

equality, 314–315

forests, 284, 288

general trees, 237

groves, defined, 284

height, 237, 272

implementing with arrays, 284–287

implementing with binary trees, 287

inorder traversal, 289, 294

level, defined, 237

minimum spanning trees, 88, 427,
458–465

multiary (multiway), defined, 284

orchards, defined, 284

ordered forests, 284

organigram (organizational
chart), 236

parent, defined, 237

postorder traversal, 289, 314

preorder traversal, 289, 314

probabilistic balance binary
search trees, 249,
261–278, 281–282

radix trees, 406–413

red-black trees, 304–314

removing nodes from, 286–287

representing, 237, 287–291

roots, defined, 237

size, defined, 237

splay trees, 270–278, 281–282

traversing, 288–291

weight-bounded balanced trees,
255–261, 281

tries

adding keys to, 394–397, 404–405,
410–411, 417–419

classic, 388–401, 424

implementing, 388–390, 402, 407,
415–416

object-based, 401–406

performance, 401, 406, 414, 423

radix trees, 406–413

removing keys from, 397–401,
405–406, 412–413,
419–423

searching in, 390, 402–403,
407–409, 416–417

storing extra data in, 390–406

ternary search tries, 414–423

trinary heaps, 340
TypeScript, 18–19

V

Visual Studio Code (VSC), 4,
13–15, 18

W

Waterman, Alan, 153
WebAssembly (WASM), 185

weight-bounded balance (BB[α]) trees,
255–261, 281

adding keys to, 257

creating, 256–257

finding by rank, 260–261

fixing balance, 257–259

performance, 261

removing keys from, 257

Williams, John W. J., 327–329

worst-case performance of algorithms, 54