

Index

COPYRIGHTED MATERIAL

Index

NUMBERS

0(1) data lookup, achieving with hashing, 265
4-sort example, 146–147

A

abstract test class

creating for generic suite of set tests,
 297–299
 for FIFO queue, 78–80

`AbstractFifoQueueTestCase` **class**,
explanation of, 80

`AbstractHashtableTestCase` **class**
 effect of, 274
 using with hash tables and bucketing, 283
 using with hash tables and linear probing, 277

`AbstractListSearcherTestCase`
 using with list searcher, 204
 using with recursive binary searcher, 206

`AbstractListSorter`, **testing, 125–126**

`AbstractListSorterTest`, **extending for
 shellsort, 148–149**

`AbstractListTestCase` **class**
 extending, 47

extending for array lists, 59–60
 extending for linked lists, 66–67
 extending for undo/redo, 109

`AbstractMapTest` **class, extending for list
 maps, 331**

`AbstractMapTestCase` **class**

creating, 322–325
 extending for B-Trees, 382
 extending for hash maps, 336
 using with tree maps, 342

`AbstractPriorityQueue` **test case, creating,
 179–182**

`AbstractPriorityQueueTestCase`
 extending for sorted list priority queue, 184
 extending in unsorted list priority queue, 182

`AbstractSetTestCase` **class**

using with hash sets, 308
 using with list sets, 304
 using with set tests, 300
 using with tree sets, 314

`AbstractStringSearcherTestCase` **class**,
extending for brute-force algorithm, 401
**accessors for coordinates, providing for Point
 class, 445**

`add()` **method**

testing for lists, 49
 using with array lists, 62
 using with hash sets, 308
 using with hash tables, 288–290
 using with hash tables and bucketing, 285
 using with hash tables and linear probing, 279
 using with list sets, 305
 using with lists, 44
 using with set tests, 301
 using with ternary search trees, 367, 368

add() method (continued)

add() method (continued)

- using with tree sets, 314, 315
- using with `UndoableList` class, 113
- using with unsorted list priority queue, 184

add set operation, description of, 294

addAll() method, using with hash maps, 336

addComparator() method, using with `CompoundComparator`, 159–160

algorithms. *See also* deterministic algorithms

- complexity of, 3–4
- definition and example of, 1–2
- describing in pseudo-code, 2
- expected runtime performance of, 5
- representing computational effort required by, 5
- running in quadratic time, 8

answer() method, using in call center simulator, 88

arguments

- for compare operation, 117
- for reverse comparators, 120

array iterators

- implementing, 24–25
- testing, 22–25

array lists. *See also* linked lists; lists

- completing interface for, 65–66
- creating test class for, 59–60
- deleting values from, 64–65
- implementing, 59
- methods for finding values in, 63–64
- methods for inserting and adding values in, 61–62
- methods for storing and retrieving values by position in, 62–63

array-based problems, overcoming with iterators, 19

`ArrayIndexOutOfBoundsException` class, avoiding in array lists, 65

`ArrayList` class

- creating, 60–61
- implementing, 480–483
- using with unsorted list priority queue, 183

arrays

- processing in reverse, 18
- using with hash tables and liner probing, 278–280
- using with lists, 46

`assertAllEquals()` method, using with **Soundex encoder**, 421, 422

`assertDistance()` method, using with distance calculator, 430

`assertEquals()` method

- using in JUnit, 13
- using with ternary search trees, 360–361

`assertPatternEquals()` method, using with ternary search trees, 361

`assertPrefixEquals()` method, using with ternary search trees, 361

`assertTrue()` method, using in JUnit, 13

associative arrays. *See* maps

`attachBefore()` method, using with linked lists, 69

AVL binary search trees, determining rotation numbers for, 237–238

axes, relationship to computational geometry, 437–438

B

balancing binary search trees, 236–238

base case, relationship to recursive algorithms, 40

base number

- definition of, 16
- raising to powers, 16

`baseexp`, significance of, 16

big-O notation, overview of, 4–5. *See also* performance

binary insertion. *See also* list inserter

- comparing algorithms for, 220–224
- comparing to other sorting algorithms, 223
- overview of, 216–217

binary search trees

- assessing performance of, 261–263
- balancing, 236–238
- versus B-Trees, 376
- deletion in, 232–234
- implementing, 256–260
- in-order traversal of, 235
- insertion in, 230–231
- minimum and maximum of, 227
- overview of, 226
- post-order traversal of, 235
- predecessors of nodes in, 227

- pre-order traversal of, 235
 - searching for values in, 228–230
 - successors of nodes in, 227
 - testing, 251–256
 - testing node classes for, 239–244
 - variations of, 231
 - binary searching**
 - with iterative binary searcher, 208–210
 - versus linear searching, 215
 - with list searcher, 202–205
 - overview of, 199–202
 - performing in ternary search trees, 346–349
 - with recursive binary searcher, 205–208
 - `BinarySearchCallCountingTest` **class, using with list searcher, 212–213**
 - `BinarySearchTree` **class**
 - creating, 256–258
 - versus `TreeMap`, 342
 - using, 253–254
 - black-box testing, explanation of, 10**
 - blocking queues, overview of, 82–86**
 - `BlockingQueue`, **code for, 83–84**
 - bookshelf examples of selection sorts, 129–131**
 - bounded queue, definition of, 76**
 - Boyer-Moore algorithm**
 - comparing to brute-force, 413
 - creating tests for, 404–407
 - implementing, 404–407
 - overview of, 402–404
 - brute-force algorithm**
 - comparing to Boyer-Moore, 413
 - creating and implementing, 400–402
 - `BruteForceStringSearcher` **class, effect of, 402**
 - B-Tree map, implementing, 382–392**
 - `BTreeMap` **class, using comparator with, 386**
 - `BTreeMapTest` **class, creating, 381–382**
 - B-Trees. See also indexes**
 - deletion from, 380
 - inserting keys in, 378
 - overview of, 375–381
 - redistributing keys in, 381
 - testing, 381–382
 - bubble sorts. See also sorting**
 - comparing to other sorting algorithms, 170–171
 - overview of, 121–122
 - performing, 122–124
 - `BubblesortListSorter`
 - implementing, 127–128
 - testing, 126–127
 - `bucketFor()` **method, effect of, 284–285**
 - `bucketIndexFor()` **method, effect of, 284**
 - bucketing**
 - versus linear probing, 291
 - using with hash tables, 281–285, 289–290
 - `BucketingHashtable` **class versus** `HashMap`, **336**
- ## C
- `calculate()` **method, using with distance calculator, 434**
 - calculations, performing and testing, 16–18**
 - calculator, implementing, 17–18**
 - call center simulator. See also queues**
 - creating call class for, 88
 - creating `CustomerServiceAgent` class for, 89
 - features of, 86–88
 - running, 95–96
 - `CallCenter` **class, creating, 90–92**
 - `CallCenterSimulator` **class, creating, 93–94**
 - `CallCountingCharSequence` **class, effect of, 411**
 - `CallCountingComparator` **sorting algorithm**
 - using with list searcher, 210–215
 - using with priority queues, 194–195
 - `CallCountingListComparator` **sorting algorithm, using, 139–140**
 - `CallGenerator` **class, creating, 92–93**
 - cards example of insertion sorts, 134–136**
 - cell values, calculating with Levenshtein word distance, 426–427**
 - character lookups, counting, 410–411**
 - CHARACTER_MAP array, using with Soundex encoder, 424, 425**
 - characters. See letters**
 - `charAt()` **method, counting calls to, 410, 411**
 - `CharSequence` **class**
 - using in counting character lookups, 410
 - using with `StringSearcher` class, 397
 - `checkIterator()` **method**
 - using with generic suite of map tests, 328
 - using with set tests, 302–303
 - child nodes, comparing, 251**

ClassCastException, throwing with Comparator interface

`ClassCastException`, **throwing with Comparator interface, 119**

classes, naming in unit testing, 10

clear map operation, description of, 319

`clear()` **method**

implementing, 85

using with `AbstractPriorityQueue`, 182

using with B-Trees, 392

using with generic suite of map tests, 328

using with linked lists, 68, 73–74

using with lists, 45, 58

using with `peek()` and stacks, 102

using with set tests, 302

using with stacks, 98

using with `UndoableList` class, 112

using with undo/redo, 108

clear queue operation

description of, 76

example of, 80

clear set operation, description of, 294

`close()` **method, using with call center simulator, 91–92**

`ClosestPairFinder` **interface, creating, 464**

collisions

relationship to hash functions, 267

resolving, 271

`Comparable` **interface, overview of, 118**

comparators. See also xy point comparator

and bubble sorts, 121–124

compound comparator, 157–160

natural comparators, 117–119

overview of, 116–117

reverse comparators, 119–121

using with `BTreeMap` class, 386

using with `FileSortingHelper` program, 472–473

using with heap-ordered priority queues, 191

using with recursive binary searcher, 207

using with unsorted list priority queues, 183

`compare()` **method**

arguments for, 117

and `CallCountingListComparator`, 139–140

using with `CompoundComparator`, 160

using with natural comparators, 119

using with reverse comparators, 119–120

using with `XYPointComparator`, 461

`compareTo()` **method, calling, 119**

comparison, linear searching for, 210–212

complexity

measures of, 5

versus performance, 3–4

composition, implementing stacks with, 103

`CompoundComparator`

implementing, 159–160

testing, 158–159

compressed words, relationship to ternary search trees, 349

computational geometry

coordinates and points in, 437–438

finding intersection of two lines with, 440–441

finding intersection points with, 443–444

lines in, 438–439

overview of, 437

and slope, 441–442

triangles in, 439–440

`computeLastOccurrences()` **method, using with Boyer-Moore algorithm, 405–406**

constant time (O(1)), overview of, 6

constants, absence in orders, 5

constructors. See also private constructor

for `BlockingQueue`, 84

for `CallGenerator` class, 93

for node class of binary search tree, 248

consumers, relationship to queues, 76

contains map operation, description of, 319

`contains()` **method**

implementing in array lists, 63–64

using with B-Trees, 391

using with generic suite of map tests, 326, 327–328

using with hash sets, 308

using with hash tables, 288–290

using with `Line` class, 454–455

using with list maps, 332

using with lists, 44, 57–58

using with set tests, 300, 301

using with ternary search trees, 360, 366

using with tree maps, 343

contains set operation, description of, 294

coordinates and points, role in computational geometry, 437–438

costs of operations, significance in Levenshtein word distance, 426–427

CPU usage, role in profiling, 471

- `createFifoQueue()` **abstract method, explanation of, 80**
- `createList()` **method**
 implementing, 47
 implementing for array lists, 60
- `createListSorter()` **method**
 implementing in `BubblesortListSorter`, 126–127
 using with shellsort, 149
- `createMap()` **method**
 using with generic suite of map tests, 325–326
 using with hash maps, 336
 using with list maps, 331
- `createQueue()` **method, using with AbstractPriorityQueue, 180**
- `createSearcher()` **method**
 using with iterative binary searcher, 208
 using with linear searcher, 212
 using with list searcher, 204
 using with `StringSearcher` interface, 399
- `createSet()` **method**
 using with hash sets, 308
 using with list sets, 304
- `createTable()` **abstract method**
 using with hash tables, 274
 using with hash tables and bucketing, 283
 using with hash tables and liner probing, 277
- crossword helper application, creating, 370–374**
See also ternary search trees
- current iterator operation, description of, 19**
- `current()` **method**
 using with lists, 57
 using with `StringMatchIterator` class, 409
- `CustomerServiceAgent` **class, creating for call center simulator, 89–90**
- D**
- `DefaultEntry` **class**
 creating for maps, 320–322
 using with B-Trees, 387
 using with generic suite of map tests, 328
- delete elements, creating for linked lists, 69**
- delete map operation, description of, 319**
- `delete()` **method**
 using with array lists, 65
 using with binary search tree, 255, 259
 using with B-Trees, 391
 using with generic suite of map tests, 327, 328
 using with hash sets, 308
 using with linked lists, 72
 using with list maps, 333
 using with lists, 44
 using with set tests, 301–302
 using with stacks, 104–105
 using with `UndoableList` class, 110–111, 113
 using with undo/redo, 107, 109
- delete set operation, description of, 294**
- `deleteCost()` **method, using with distance calculator, 434**
- deleting values from lists, 52–55**
- deletion in binary search trees, process of, 232–234**
- `dequeue()` **method**
 and blocking queues, 82
 calling on FIFO queue, 77
 description of, 76–77
 implementing, 85
 implementing for FIFO queues, 81–82
 using with `AbstractPriorityQueue`, 181
 using with heap-ordered priority queues, 192, 194
 using with priority queues, 196
 using with sorted list priority queue, 185
 using with stacks, 105
 using with unsorted list priority queues, 183
- `detach()` **method, using with linked lists, 69**
- deterministic algorithms, definition of, 2. See also algorithms**
- dictionaries. See maps**
- directory tree**
 printing contents of, 37–40
 relationship to binary search trees, 226
 relationship to recursion, 35
- disjoint line, definition of, 453**
- disk I/O, expense of, 375**
- distance calculator**
 implementing, 431–435
 testing, 429–431
- `distance()` **method, using with Point class, 445–446**
- double value**
 calculating nonvertical slope as, 447
 calculating ration of slope's rise as, 449

doubly linked list, definition of, 66
drag net, relationship to plane sweep algorithm, 458

DummyPredicate **inner class, effect of, 32**

E

element class, creating for linked lists, 68–69

EmptyQueueException

relationship to blocking queues, 82–83
significance of, 77

EmptyStackException, **throwing, 99**

encode() **method, using with Soundex encoder, 425**

enqueue() **method**

calling on FIFO queue, 77
description of, 76–77
implementing for BlockingQueue, 84
using with AbstractPriorityQueue, 181
using with FIFO queues, 81
using with heap-ordered priority queues, 192, 193
using with priority queues, 196
using with sorted list priority queue, 185, 186
using with unsorted list priority queues, 183

ensureCapacity() **method**

implementing inverse of, 65
using with array lists, 62

ensureCapacityForOneMore() **method, using with hash tables and linear probing, 278**

entries in maps

removing with delete() method, 333
significance of, 320

entryFor() **method, using with list maps, 332**

enumerators. See iterators

equals() **method**

implementing with Slope class, 448–449
using to test slope of line, 447
using with generic suite of map tests, 328–329
using with hash tables, 287–288
using with maps, 321
using with Node class for binary search tree, 244
using with Node class of binary search tree, 250
using with Point class, 446

equalsLarger() **method, using with node class of binary search tree, 251**

equalsSmaller() **method, using with node class of binary search tree, 251**

evaluate() **method, using with predicate class, 29, 30**

execute() **method, using with UndoableList class, 113**

exponent, definition of, 16

F

factorial time (O(N!)), overview of, 8–9

FIFO (first-in-first-out) queues

example of, 77–81
implementing, 81–82
significance of, 75–76

Figures

adding strings to hash tables, 270
binary insertion, 216–217
binary search tree, 226
binary search tree search, 228–234
binary search trees, 236–238
binary searching, 200–201
bookshelf examples of selection sorts, 129–131
B-Trees, 376–381
call center simulator, 86
cards example of insertion sorts, 134–136
coordinates and lines in computational geometry, 439
directory tree structure, 36
disjoint lines, 453
family-member examples of bubble sorts, 122–124
group members greeting each other, 7
heap-ordered priority queues, 186–191
index mapping names to record numbers, 318
intersecting lines, 441, 443
keys and values in maps, 322
last occurrence table, 406
Levenshtein word distance, 426–428
linear probing, 269
mergesort algorithm, 160–166
orders of complexity, 5
parallel lines, 442
plane sweep algorithm, 458, 459
point in x-y coordinate system, 438

point with x and y coordinates within span of line, 456

priority queues, 174–179

producers and consumers interacting with queues, 76

pushing and popping values on stacks, 98

quadratic time ($O(N^2)$), 7

quicksort lists, 151–154

right-angled triangle, 440

scattered points, 457

of sets, 294–296

shellsort data, 146–148

slope of line, 441, 442

Soundex encoding, 417–418

stack pictured vertically, 97

ternary search trees, 346–357

tree branches are trees, 37

x-y coordinate system with axes, 438

files, searching, 411–413

`FileSortingHelper`. **See also optimization**

implementing, 473–475

optimizing, 483–487

overview of, 471–479

`filterForwards`, **using with predicate class, 35**

filtering iterators, using, 28–29

`FilterIterator`, **using with predicate class, 34**

`first()` **method**

calling on reverse iterators, 27

implementing in array iterators, 24–25

relationship to iterator idioms, 21

using with lists, 57

using with predicate class, 33, 34

for loops, using with iterators, 21

formulas

for crossing y axis, 442–443

for intersection of two lines, 440

for intersection points, 443–444

of lines, 448

Pythagoras' theorem, 439

functional testing, explanation of, 10

G

#GC column in JMP Objects window, examining, 482, 486

general case, relationship to recursive algorithms, 41

`generateCalls()` **method, using with call center simulator, 93**

get map operation, description of, 319

`get()` **method**

using with array lists, 62–63

using with B-Trees, 391

using with generic suite of map tests, 326, 327

using with linked lists, 71

using with list maps, 332

using with lists, 44

`getCallCount()` **method, effect of, 411**

`getElement()` **method, using with linked lists, 70–71, 72**

`getIndexOfLargestElement()` **method, using with unsorted list priority queue, 183**

`getKey()` **method, using with maps, 321**

`getLarger()` **method, using with node class of binary search tree, 249**

`getName()` **method, using with `ListSorterCallCountingTest`, 142–143**

`getRoot()` **method, using with binary search tree, 254**

`getSmaller()` **method, using with node class of binary search tree, 249**

`getValue()` **method, using with maps, 321**

H

handshakes, using in quadratic-time example, 6–7

hash functions

and collisions, 267

definition of, 265

modifying, 266

hash maps, testing and implementing, 333–337

hash sets, testing and implementing, 305–309

hash tables. See also linear probing

adding values to, 268–269

assessing performance of, 285–291

creating interface for, 272–273

creating tests for, 273–275

definition of, 265

resizing, 272

using bucketing with, 281–285

using linear probing with, 275–280

hash value, definition of, 265

hashCode() method

- code for, 274
- definition of, 278–279
- implementing, 272
- implementing with Slope class, 448–449
- using with Point class, 446

hashing

- example of, 267
- overview of, 265–272

HashtableCallCountingTest, **using, 287–288**

heap-ordered priority queues

- example of, 197
- overview of, 186–191
- testing and implementing, 191–194

heuristics, significance of, 2–3

horizontal lines, slope of, 442

hprof command, profiling with, 474–477

hSort() method, calling in shellsort, 149–150

H-sorting, relationship to shellsort, 146, 149

hypotenuse, definition of, 439

I

id, assigning to calls in call center simulator, 88
illustrations. *See* Figures

indentation, increasing with SPACES constant,
38–39

indexes. *See also* B-Trees

- implementing quicksort with, 156
- mapping names to record numbers with, 318
- role in mergesort algorithm, 160
- using with quicksort, 151, 154

indexOf() method, using with hash tables and
linear probing, 279

indexOf() method

- using with array lists, 63–64
- using with B-Trees, 388
- using with hash tables and linear probing, 279
- using with lists, 44, 57
- using with UndoableList class, 111

IndexOutOfBoundsException

- throwing with lists, 48, 50–51, 52, 53–54, 82
- throwing with stacks, 105

inner loop, using with

InsertionSortListSorter, 137

in-order traversal

- of binary search trees, 235
- of ternary search trees, 352

inOrderTraversal() method, using with
ternary search trees, 369

insert() method

- using with array lists, 62
- using with binary search tree, 259
- using with linked lists, 70
- using with list inserter, 219, 220
- using with lists, 44
- using with ternary search trees, 367, 368
- using with tree maps, 343
- using with tree sets, 314
- using with UndoableList class, 110, 113

insertion in binary search trees, process of,
230–231

insertion sorts. *See also* sorting

- comparing to other sorting algorithms, 170–171
- overview of, 133–138

insertions, performance in binary search trees,
263

InsertionSortListSorter, testing and
implementing, 136–138

INSTANCE constant, using with Soundex encoder,
424

instance member field, setting to null in JUnit, 12

Integer objects, using with

ListSorterCallCountingTest sorting
algorithm, 141

integers, using comparators with, 116–117

integration testing, explanation of, 11

intersecting lines. *See also* lines

- determining x coordinate of, 455–456
- testing, 451

intersection points

- determining, 455, 457
- finding, 443–444

intersectionPoint() method, using with Line
class, 451

isDone() method

- description of, 19
- implementing in StringMatchIterator class,
409

- using with array iterators, 25
- using with lists, 57

isEmpty() method

- implementing, 86
- using with AbstractPriorityQueue, 181
- using with array lists, 66
- using with B-Trees, 392

- using with FIFO queues, 79
 - using with linked lists, 73–74
 - using with lists, 45
 - isEmpty operation**
 - performing on maps, 319
 - performing on queues, 79
 - performing on sets, 294
 - performing on stacks, 98
 - `isEndOfWord()` **convenience method, using with ternary search trees, 365–366**
 - `isFull()` **method, using with B-Trees, 387**
 - `isLarger()` **method, using with node class of binary search tree, 249**
 - `isLeaf()` **method, using with B-Trees, 387**
 - `isParallel()` **method, using with Line class, 451, 454**
 - `isSmaller()` **method, using with Node class of binary search tree, 249**
 - `isValid()` **helper method, using with Soundex encoder, 425**
 - `isVertical()` **method, implementing with Slope class, 448**
 - Iterable interface**
 - explanation of, 20
 - extending in Map interface, 320
 - iteration**
 - definition of, 2, 15
 - performing calculations with, 16–18
 - processing arrays with, 18
 - testing in lists, 55–57
 - iterative binary searcher, testing and implementing, 208–210**
 - iterator idioms, using, 21**
 - Iterator interface**
 - creating in Java, 20
 - using with predicate class, 34
 - using with `StringMatchIterator` class, 408–409
 - iterator map operation, description of, 319**
 - `iterator()` **method**
 - overview of, 19
 - testing in generic suite of map tests, 328
 - using with array lists, 66
 - using with B-Trees, 392
 - using with hash sets, 309
 - using with lists, 45
 - `iterator` **set operation, description of, 294**
 - `IteratorOutOfBoundsException`, **throwing with lists, 57**
 - iterators**
 - advantages of, 22
 - array iterators, 22–25
 - creating for linked lists, 72–73
 - filtering iterators, 28–29
 - reverse iterators, 26–28
 - solving array-based problems with, 19
- ## J
- Java, creating iterator interface in, 20**
 - Java Memory Profiler (JMP), profiling with, 477–479**
 - JDK String class, hashing algorithm used in, 267**
 - JMP (Java Memory Profiler), profiling with, 477–479**
 - JMP Methods window**
 - displaying `ArrayList` implementation in, 482
 - displaying `OptimizedFileSortingHelper` in, 486
 - examining for `LinkedList`, 481
 - JMP Objects window, displaying `OptimizedFileSortingHelper` in, 486**
 - JMP output, displaying for `OptimizedFileSortingHelper`, 485**
 - JUnit**
 - performing unit tests with, 11–14
 - testing array iterators with, 22–23
- ## K
- keys**
 - deleting from B-Trees, 380
 - inserting in B-Trees, 378
 - managing in `BTreeMap` class, 386–387
 - redistributing in B-Trees, 381
 - key/value pairs in maps**
 - removing with `delete()` method, 333
 - significance of, 320
- ## L
- `last()` **method**
 - calling on reverse iterators, 27
 - implementing in array iterators, 24–25

last() method (continued)

`last()` method (continued)

relationship to iterator idioms, 21
using with predicate class, 33–34

last occurrence table, computing for Boyer-Moore algorithm, 405–406

latency, definition of, 376

leaf nodes

and B-Trees, 378
creating for binary search tree, 248
deleting from binary search trees, 232–233
relationship to binary search trees, 226
role in insertion, 230

letters. *See also* words

adding in JDK String class algorithm, 267–268
adding in string hashing, 266
in binary insertion, 216
in binary search trees, 228–234, 236–238
in binary searching, 200–201
in B-Trees, 376–381
in heap-ordered priority queues, 186–191
manipulating in priority queues, 174–179
searching in ternary search trees, 347–348
in sets, 294–296
sorting with quicksort, 151–154
sorting with shellsort, 146–148
in Soundex encoding, 416–418
in ternary search tree, 346–357

Levenshtein word distance, overview of, 426–429

`LevenshteinWordDistanceCalculator` class, description of, 433

`LevenshteinWordDistanceCalculatorTest` class, contents of, 430

LIFO (last-in-first-out) queues, stacks as, 98

Line class

implementing, 454–457
instance members of, 456
testing, 449–454

line intersections, finding with computational geometry, 440–441

linear probing. *See also* hash tables

versus buckets, 291
relationship to hashing, 269, 271
using with hash tables, 275–280

linear searcher

versus binary searching, 215
testing and implementing, 211–212

linear time ($O(N)$), overview of, 6

`LinearprobingHashtable` class, example of, 277–278

lines. *See also* intersecting lines

determining points on, 456
formula of, 448
role in computational geometry, 438–439
slopes of, 441–442
testing slopes of, 446–448

linked lists. *See also* array lists; lists

completing interface for, 73–74
creating element class for, 68–69
creating iterators for, 72–73
creating test class for, 66–67
definition of, 58
versus linear probing, 291
methods for deleting values from, 72
methods for finding values in, 71–72
methods for inserting and adding values in, 69–70
methods for storing and retrieving values in, 70–71
overview of, 66
using with FIFO queues, 81

`LinkedList` class

versus `ArrayList`, 480–483
creating, 67–68
using with `InsertionSortListSorter`, 137
using with unsorted list priority queue, 182

list inserter. *See also* binary insertion

assessing performance of, 220–224
creating tests for, 217–219
implementing, 219–220

List interface

completing for array lists, 65–66
creating, 45–46
implementing for `UndoableList` class, 109–110
using with array lists, 64
using with undo/redo, 106

list maps, testing and implementing, 330–333

list searcher

assessing performance of, 210–215
creating interface for, 202–203
creating test class for, 212–213
creating tests for, 204–205
implementing tests for, 214–215

using with iterative binary searcher, 209

writing tests for, 203–204

list sets, testing and implementing, 303–304

ListFifoQueue implementation class

coding, 81

problems with, 82

lists. See also array lists; linked lists

creating generic test class for, 46–47

implementing, 58–59

overview of, 43–46

sorting, 138–139

versus stacks, 98

testing iteration in, 55–57

testing methods for deleting values from,
52–55

testing methods for finding values in, 57–58

testing methods for inserting and adding values
in, 47–50

testing methods for retrieving and storing values
in, 50–52

testing results of clearing of, 58

using with heap-ordered priority queues, 186–187

ListSorter interface

implementing in shellsort, 149

sorting with, 124

using with `ListSorterCallCountingTest`,
141

using with mergesort, 167

**ListSorterCallCountingTest sorting
algorithm**

extending, 169

using, 140–143

ListStack class, implementing, 103–104

load factor, monitoring for hash tables, 272

loadWords() method

using with crossword helper application, 372

using with `FileSortingHelper`, 484

using with `FileSortingHelper` class,
473, 474

lock objects, using with blocking queues, 83

**logarithmic time ($O(\log N)$ and $O(N \log N)$),
overview of, 8**

lookup tables. See maps

looping, definition of, 2

lowerIndex variable

using with iterative binary searcher, 210

using with recursive binary searcher, 206, 208

M

main() method

of `CallCenterSimulator` class, 94

role in searching files, 413

using with crossword helper application, 372

using with `FileSortingHelper`, 473, 483–484

**maintainLoad() method, using with hash tables
and bucketing, 283–284**

**map() helper method, using with Soundex
encoder, 425**

map implementations, performance of, 343

Map interface, creating, 320

**map tests, creating generic suite of, 322–329
maps**

creating `DefaultEntry` class for, 320–322

hash maps, 333–337

list maps, 329–333

overview of, 317–320

tree maps, 337–343

math in algorithms, resource for, 169

**maximum() method, using with node class for
binary search tree, 242, 249**

**maximum of binary search tree, explanation of,
227**

merge() method, example of, 168

mergesort algorithm. See also sorting algorithms

comparing to other sorting algorithms,
170–171, 223

implementing, 167–168

overview of, 160–162

versus quicksort, 162

testing, 166–167

using, 162–166

methods

for deleting values from array lists, 64–65

for deleting values from linked lists, 72

distinguishing in JUnit, 13

for finding values in array lists, 63–64

for finding values in linked lists, 71–72

for inserting and adding values in array lists,
61–62

for inserting and adding values in linked lists,
69–70

for storing and retrieving values by position in
array lists, 62–63

for storing and retrieving values in linked lists,
70–71

`minimum()` **method, using with node class for binary search tree, 242, 249**
minimum of binary search tree, explanation of, 227
`minimumCost()` **method, using with distance calculator, 434**
multiplication-using-addition algorithm, example of, 2
`Multiply` **function, demonstrating pseudo-code with, 2**
multi-threaded code, resource for, 83
mutexes, using with blocking queues, 83
mutual exclusion semaphore. See lock objects

N

N expression
examples of, 5
meaning in big-O notation, 4–5
subtracting from N^2 , 7
 N^2 , comparing to $N!$ for small integers, 8–9
“Name Search Techniques” (Robert L. Taft), 415
naming convention, applying in unit testing, 10
natural comparators
implementing, 119–120
testing, 118–119
using, 117–119
using with `ListSorterCallCountingTest`, 141
`next()` **method**
calling on reverse iterators, 27
using with lists, 57
using with predicate class, 32, 33, 34, 35
`Node` **class**
implementing for binary search tree, 244–251
testing for binary search trees, 239–244
using with B-Trees, 387
using with ternary search trees, 365
nodes
and B-Trees, 379
marking in ternary search trees, 351
successors of, 227
null
returning in plane sweep algorithm, 462
setting instance member field to (`JUnit`), 12
null objects, using with linked lists, 67
null values, using with maps, 321

O

O^* orders, examples of, 4
 $O(1)$ (constant time), overview of, 6
 $O(\log N)$ and $O(N \log N)$ (logarithmic time), overview of, 8
 $O(\log N)$, relationship to heap-ordered priority queues, 194
 $O(N!)$ (factorial time), overview of, 8–9
 $O(N)$ (linear time), overview of, 6
 $O(N^2)$ (quadratic time), overview of, 6–8
`open()` **method, using with call center simulator, 91–92**
operations
order of magnitude of, 4
using with lists, 44
optimization. See also FileSortingHelper
approach toward, 479
examples of, 480–487
overview of, 469–470
and profiling, 470–471
`OptimizedFileSortingHelper` **class, code for, 483**
order of magnitude of operations, significance of, 4
orders, iterating over array of, 18
orders of complexity, absence of constants in, 5
outer loop
using with `BubblesortListSorter`, 127–128
using with `SelectionSortListSorter`, 132

P

parallel lines
indicating, 452
proving, 452
significance of, 442
parents, manipulating in heap-oriented priority queues, 188–189
`partition()` **method, using with quicksort, 156**
pattern matching, performing in ternary search trees, 353–357
pattern searching, comparing performance of, 413
`patternMatch()` **method**
using with crossword helper application, 373
using with ternary search trees, 361, 369

patterns, searching in text, 398**peek() method**

description of, 98
effect on stacks, 99, 104–105
testing on stacks, 101–102

performance. See also big-O notation

of algorithms, 5
of binary search trees, 236, 261–263
of brute-force algorithm, 402
versus complexity, 3–4
of hash tables, 285–291
of Levenshtein word distance, 429
of list inserter, 220–224
of list searcher, 210–215
of map implementations, 343
of set implementations, 315
of string searching algorithms, 409–413
of ternary search trees, 350

performOrderSearch() method, using with list searcher, 215**performRandomSearch() method, using with list searcher, 215****phonetic encoding algorithm, Soundex encoding as, 415****PhoneticEncoder interface, defining for Soundex encoder, 424****plane sweep algorithm**

implementing, 464–467
overview of, 457–467
testing, 461–464

Point class

versus Line class, 449–450
testing and implementing, 444–446

points

and coordinates in computational geometry, 437–438
determining position on lines, 456
finding closest pair of, 457–459

pop() method

description of, 98
effect on stacks, 99
using with stacks, 100–101, 104–105

populateAndSortList() method, using with binary inserter, 222**post-order traversal of binary search trees, overview of, 235****PowerCalculator class, implementing with calculate method, 16–18****powers, raising base numbers to, 16, 18****predecessor() method, using with node class for binary search tree, 243****predecessors of nodes, role in binary search trees, 227****Predicate class**

implementing, 33–35
testing, 29–32

prefix searching

performing in ternary search trees, 351–353
testing in ternary search trees, 361

prefixSearch() method, using with ternary search trees, 369**premature optimization, definition of, 469–470****pre-order traversal**

of binary search trees, 235
of B-Trees, 390–391

previous iterator operation, description of, 19**previous() method**

calling on reverse iterators, 27
using with lists, 57
using with predicate class, 33–34

print() method, using with recursive directory tree, 38, 41**printAll() method, using with FileSortingHelper, 484–485****printing directory tree contents, 37–40****priority queues. See also queues**

comparing implementations of, 194–197
example of, 174–179
heap-ordered priority queues, 186–194
overview of, 174
sorted list priority queue, 184–186
unsorted list priority queue, 182–184

private constructor, using with calculator example, 18. See also constructors**problems, indicating sizes of, 5****producers, relationship to queues, 76****profiling**

with hprof command, 474–477
with JMP, 477–479
relationship to optimization, 470–471

programming by coincidence, significance of, 19**pseudo-code, describing algorithms in, 2****push() method, using with stacks, 100–101**

push operation on stacks, description of, 98
Pythagoras' theorem, applying, 439, 445–446

Q

quadratic time ($O(N^2)$), overview of, 6–8

queue, testing behavior with JUnit, 12–13

Queue interface

implementing, 81

using with sorted list priority queue, 185

using with unsorted list priority queue, 184

queue operations, overview of, 76–77

queues. See also call center simulator; priority queues

blocking queues, 82–86

creating for call center simulator, 92

interaction of producers and consumers with, 76

interface for, 77

overview of, 75–76

versus stacks, 98

quicksort algorithm. See also sorting algorithms

comparing to other sorting algorithms,

170–171, 223

implementing, 155–157

versus mergesort, 162

overview of, 151–154

testing, 155

R

recursion

definition of, 15

example of, 37–40

overview of, 35–37

recursive algorithms, anatomy of, 40–41

recursive binary searcher, testing and creating, 205–208

redo/undo. See also stacks

implementing with stacks, 105–106

testing, 106–113

reportCalls() method

using with binary search tree, 262–263

using with hash tables, 288

using with list searcher, 213

using with `ListSorterCallCountingTest`, 140–142

using with priority queues, 196–197

resize() method

using with hash tables and bucketing, 283

using with hash tables and linear probing, 278

resources

for algorithms, 263

for math in algorithms, 169

for multi-threaded code, 83

reverse comparators

implementing, 121

testing, 120–121

reverse iterators

implementing, 28

testing, 26–28

testing in lists, 56

reverse() method

absence in JMP Methods window for

`OptimizedFileSortingHelper`, 486

using with `ArrayList`, 482

using with `FileSortingHelper`, 484

reverseAll() method, using with

`FileSortingHelper`, 484

ReverseIterator, using with Predicate class, 32

ring pattern, searching, 400

rise, relationship to slope, 441

root nodes, splitting in B-Trees, 379

rotations, determining for AVL binary search trees, 237–238

RPN (Reverse-Polish-Notation) calculator, using with stacks, 105

rules for testing calculations, explanations of, 16–17

run() method, role in searching files, 412

runAdd() method, using with hash tables, 288

runAll() method, using with hash tables, 289

runContains() method, using with hash tables, 288

Runnable interface, implementing in call center simulator, 89

runScenario() method, using with priority queues, 195, 196

Russell, R.C. (Soundex), 415

S

search keys, using with recursive binary searcher, 207

- `search()` **method**
 using with binary search tree, 254–255, 258, 259
 using with brute-force algorithm, 402
 using with B-Trees, 388–389, 391
 using with files, 412, 413
 using with linear searcher, 212
 using with list searcher, 202–203
 using with recursive binary searcher, 208
 using with `StringSearcher` interface, 396, 397, 399
 using with ternary search trees, 366–367
 using with tree maps, 343
 using with tree sets, 315
- searches, performing with Boyer-Moore algorithm, 406–407**
- `searchForPattern()` **method, using with crossword helper application, 372, 373**
- `searchRecursively()` **method, using with recursive binary searcher, 206, 208**
- seek time, definition of, 376**
- selection sorts. See also sorting**
 comparing to other sorting algorithms, 170–171
 overview of, 128–133
- `SelectionSortListSorter`, **testing and implementing, 132–133**
- sentinels, using with linked lists, 67**
- separation of concerns, relationship to sorting, 117**
- set implementations**
 performance of, 315
 testing, 297–303
- `Set` **interface**
 creating, 297
 implementing in tree sets, 314
 using with plane sweep algorithm, 466
- set map operation, description of, 319**
- `set()` **method**
 descriptions of, 294
 testing lists with, 51–52
 using with array lists, 62–63
 using with B-Trees, 390, 391
 using with generic suite of map tests, 327
 using with linked lists, 71
 using with list maps, 332
 using with lists, 44, 45
 using with tree maps, 343
 using with `UndoableList` class, 111
 using with undo/redo, 108
- sets**
 combining, 295
 hash sets, 305–309
 intersection of, 296
 list sets, 303–305
 overview of, 293–297
 tree sets, 309–315
- `setUp()` **method**
 overriding in JUnit, 12
 using with `AbstractPriorityQueue`, 180
 using with `AbstractSorterTest`, 125
 using with FIFO queues, 80
 using with generic suite of map tests, 325, 326
 using with hash tables, 274
 using with list searcher, 204, 213
 using with `ListSorterCallCountingTest`, 141
 using with priority queues, 195
 using with set tests, 300
 using with Soundex encoder, 420–421
 using with ternary search trees, 359, 360
- `setValue()` **method**
 using with maps, 321
 using with tree maps, 343
- shellsort algorithm. See also sorting algorithms**
 comparing to other sorting algorithms, 170–171, 223
 implementing, 149–150
 testing, 148–149
- simulator. See call center simulator**
- `sink()` **method, using with heap-ordered priority queues, 193, 194**
- sinking, relationship to heap-oriented priority queues, 190**
- size map operation, description of, 319**
- `size()` **method**
 implementing, 86
 using with `AbstractPriorityQueue`, 181
 using with array lists, 66
 using with B-Trees, 392
 using with hash tables and linear probing, 280
 using with linked lists, 73–74
- size operation on stacks, description of, 98**
- `size()` **operations, using with lists, 44**

size queue operation, description of, 76

size set operation, description of, 294

slope

implementing, 448–449

role in computational geometry, 441–442

testing, 446–448

Slope object

instantiation of, 447

using with `Line` class, 456

sort() method

implementing in `InsertionSortListSorter`, 137

using with `FileSortingHelper` class, 473, 474

using with `mergesort`, 167

using with `quicksort`, 155

using with `shellsort`, 149, 150

Sort operation, using with `ListSorter` interface, 124

sort order, reversing with reverse iterators, 27

sorted list, depicting as balanced binary search tree, 230

sorted list priority queue, testing and implementing, 184–186

sorting. See also bubble sorts; insertion sorts; selection sorts

and compound comparator, 157–160

importance of, 115–116

with `ListSorter` interface, 124

and stability, 138–139

testing `AbstractListSorter`, 124–126

sorting algorithms. See also mergesort

algorithm; quicksort algorithm; shellsort algorithm

comparing, 139–143, 169–171, 220–224

limitations of, 145

sortSubList() method, using with shellsort, 150

Soundex encoder

implementing, 423–425

overview of, 415–419

testing, 419–423

SoundexPhoneticEncoder class, implementing, 424

SPACES constant, increasing indentation with, 38–39

split() method, using with B-Trees, 389

stability

relationship to sorting algorithms, 138–139

relationship to sorting and compound

comparator, 157–160

stable algorithm, definition of, 138

stack traces, generating with hprof, 476

StackOverflowException, throwing from recursive algorithms, 40

stacks. See also undo/redo

implementing, 102–105

as LIFO (last-in-first-out) queues, 98

operations on, 98

overview of, 97–99

popping values from, 104–105

test cases for, 99–102

startIndexFor() method, using with hash tables and linear probing, 278, 280

String class of JDK

hashing algorithm used in, 267

using with hash tables, 288

string hashing technique, example of, 266

StringMatchIterator class, creating, 408–409

StringSearcher interface

creating, 396

creating test class for, 397–399

implementing with

`BruteForceStringSearcher` class, 402

sublists, merging with mergesort, 166

substitutionCost() method, using with distance calculator, 433–434

successor() method, using with node class of binary search tree, 243, 250

successors of nodes, role in binary search trees, 227, 233

swap() method

using with `BubblesortListSorter`, 128

using with heap-ordered priority queues, 192

using with `quicksort`, 157

swim() method, using with heap-ordered priority queues, 192, 193

swimming, relationship to heap-ordered priority queues, 188–191

synchronization, implementing for blocking queues, 83

T

Taft, Robert L. (“Name Search Techniques”), 415

TDD (test-driven development), overview of, 14

tearDown() method

implementing with `AbstractSorterTest`, 126

using with `AbstractPriorityQueue`, 180

templates, using with iterators, 21

ternary search trees. See also crossword helper application

implementing, 362–370

inserting words in, 350–351

overview of, 345–346

pattern matching in, 353–357

performance of, 350

performing binary searches in, 346–349

prefix searching in, 351–353

testing, 358–362

test cases

for `AbstractPriorityQueue`, 179–182

for stacks, 99–102

test classes

for binary search trees, 261–262

creating and running for undo/redo, 106–109

creating for array lists, 59–60

creating for linked lists, 66–67

creating for list searcher, 212–213

creating for lists, 46–47

creating for string searcher interface, 397–399

test prefix, using with methods in JUnit, 13

TEST_SIZE constant, using with list searcher, 213

testAdd() method, using with lists, 49

testAddExistingValueHasNoEffect() method, using with set tests, 301

testAddNewKey() method, using with set tests, 301

testBinaryInsert() method, using with binary inserter, 222

TestCase

extending for calculations, 16

extending with `AbstractMapTestCase` class, 325

testClear() method

using with generic suite of map tests, 328

using with lists, 58

using with set tests, 302

testContains() method

using with lists, 58

using with ternary search trees, 360

testContainsNonExisting() method

using with generic suite of map tests, 326

using with set tests, 300–301

testDeleteByValue() method, using with lists, 55

testDeleteExisting() method

using with generic suite of map tests, 327

using with set tests, 301–302

testDeleteFromLastElementInArray() method, using with array lists, 60

testDeleteNodeWithOneChild() method, using with binary search tree, 255

testDeleteNonExisting() method, using with generic suite of map tests, 328

testDeleteOutOfBounds() method, using with lists, 55

testDuplicateCodesAreDropped() method, using with Soundex encoder, 422

testEmptyToEmpty() method, using with distance calculator, 430

testEmptyToNonEmpty() method, using with distance calculator, 431

testEquals() method, using with node class for binary search tree, 244

testFirstLetterIsAlwaysUsed() method, using with Soundex encoder, 421

testForwardIteration() method, using with lists, 57

testGetExisting() method, using with generic suite of maps, 326

testGetNonExisting() method, using with generic suite of map tests, 327

testGetOutOfBounds() method, using with lists, 52

testIndexOf() method, using with lists, 58

testing

`AbstractSorterTest`, 125–126

binary search trees, 251–256

B-Trees, 381–382

`BubblesortListSorter`, 126–127

`CompoundComparator`, 158–159

distance calculator, 429–431

hash maps, 333–337

hash sets, 305–309

testing (continued)

- hash tables using bucketing, 281–285
- hash tables with linear probing, 275–280
- heap-ordered priority queues, 191–194
- and implementing tree maps, 337–343
- `InsertionSortListSorter`, 136
- intersecting lines, 451
- iterative binary searcher, 208–210
- `Line` class, 449–454
- linear searcher, 211–212
- list maps, 330–333
- list sets, 303–304
- mergesort algorithm, 166–167
- natural comparators, 118
- `Node` class for binary search trees, 239–244
- plane sweep algorithm, 461–464
- `Point` class, 444–446
- quicksort algorithm, 155
- recursive binary searcher, 205–208
- results of clearing lists, 58
- reverse comparators, 120–121
- `SelectionSortListSorter`, 132
- set implementations, 297–303
- shellsort algorithm, 148–149
- slope of line, 446–448
- sorted list priority queue, 184–186
- Soundex encoder, testing, 419–423
- ternary search trees, 358–362
- undo/redo, 106–113
- unsorted list priority queue, 182–184
- xy point comparator, 459–460

testing iteration in lists, 55–57

testing methods

- for deleting values from lists, 52–55
- for finding values in lists, 57–58
- for inserting and adding values in, 47–50
- for inserting and adding values in lists, 47–49
- for retrieving and storing values in lists, 50–52

`testInOrderInsertion()` **method, using with binary search trees, 263**

`testInsert()` **method, using with binary search tree, 254**

`testInsertBeforeFirstElement()` **method, using with lists, 49**

`testInsertBetweenElements()` **method, using with lists, 49**

`testInsertIntoAnEmptyList()` **method, using with lists, 49**

`testInsertOutOfBounds()` **method, using with lists, 49**

`testIteratorBackwards()` **method**
using with generic suite of map tests, 329
using with set tests, 303

`testIteratorForwards()` **method**
using with generic suite of map tests, 329
using with set tests, 303

`testPatternMatch()` **method, using with ternary search trees, 361**

`testPrefixSearch()` **method, using with ternary search trees, 361**

`testRandomInsert()` **method, using with binary search trees, 263**

`testResizeBeyondInitialCapacity()` **method, using with array lists, 60**

tests

- creating for Boyer-Moore algorithm, 404–407
- creating for hash tables, 273–275
- creating for list inserter, 217–219
- creating for list searcher, 204–205
- for hash tables, 286–291
- implementing for list searcher, 214–215
- writing for list searcher, 203–204

`testSamePrefix()` **method, using with distance calculator, 431**

`testSearch()` **method, using with binary search tree, 255**

`testSetExistingKey()` **method, using with generic suite of map tests, 327**

`testSetNewKey()` **method, using with generic suite of map tests, 327**

`testSetOutOfBunds()` **method, using with lists, 52**

`testSomeRealStrings()` **method, using with Soundex encoder, 422–423**

`testSuccessor()` **method, using with node class for binary search tree, 243**

`testVowelsAreIgnored()` **method, using with Soundex encoder, 421–422**

text, searching for patterns in, 398

threads

- getting information about, 476
- using with `BlockingQueue`, 84–85

`Thread.sleep()` **method, using with FileSortingHelper class, 473**

Towers of Hanoi puzzle, solving with stacks, 105

transfer time, definition of, 376

travel, relationship to slope, 441

`traverse()` method, using with B-Trees, 390, 392

tree maps, testing and implementing, 309–315, 337–343

trees. See directory tree

`TreeSetTest` class

creating, 309

description of, 314

triangles, role in computational geometry, 439–440

Try It Out

A Class for Counting Character Lookups, 410–411

A Class That Searches a File, 411–413

Comparing the Binary Inserter with Other Sorting Algorithms, 220–224

Completing the Interface (for Array Lists), 65–66

Completing the Interface (for Linked Lists), 73–74

Computing the Last Occurrence Table (for Boyer-Moore Algorithm), 405–406

Creating a Default Entry Implementation (for Maps), 320–322

Creating a Generic Hash Table Interface, 272–273

Creating a Generic `Map` interface, 320

Creating a Generic `Set` Interface, 297

Creating a Generic Suite of Map Tests, 322–329

Creating a Generic Suite of Set Tests, 297–303

Creating a Generic Test Class (for Lists), 46–47

Creating a Generic Test Class (for Stacks), 100

Creating a `StringMatchIterator` Class, 408–409

Creating an `AbstractPriorityQueue` Test Case, 179–182

Creating an Element Class (for Linked Lists), 68–69

Creating an Iterator (for Linked Lists), 72–73

Creating and Running the Test Class (for Undo/Redo), 106–109

Creating the `ArrayList` Class, 60–61

Creating the `BoyerMooreStringSearcher` Class, 405

Creating the `Call` Class (for Simulator), 88

Creating the `CallCenter` Class, 90–92

Creating the `CallCenterSimulator` Class, 93–94

Creating the `CallGenerator` Class, 92–93

Creating the `ClosestpairFinder` Interface, 464

Creating the Crossword Helper Application, 370–374

Creating the `CustomerServiceAgent` Class, 89–90

Creating the Interface (for String Searcher), 396–397

Creating the `LinkedList` Class, 67–68

Creating the `List` Interface, 45–46

Creating the List Searcher Interface, 202–203

Creating the Test Class (for Array Lists), 59–60

Creating the Test Class (for Boyer-Moore Algorithm), 404

Creating the Test Class (for Brute-force Algorithm), 401

Creating the Test Class (for Linked Lists), 66–67

Creating the Test Class (for String Searcher Interface), 397–399

Creating the Tests (for Hash Table), 273–275, 286–291

Creating the Tests (for List Inserter), 217–219

Creating the Tests (for List Searcher), 204–205

Implementing a Binary Search Tree, 256–260

Implementing a B-Tree map, 382–392

Implementing a Node Class, 244–251

Implementing a Ternary Search Tree, 362–370

Implementing an `ArrayList`, 480–483

Implementing and Running Performance Tests (for Binary Search Trees), 261–263

Implementing `BubblesortListSorter`, 127–128

Implementing `dequeue()`, 85

Implementing `InsertionSortListSorter`, 136–138

implementing mergesort, 167–168

Implementing quicksort, 155–157

Implementing Slope, 448–449

Implementing the Algorithm (Brute-force), 401–402

Implementing the Array Iterator, 24–25

Implementing the Calculator, 17–18

Implementing the `clear()` Method, 85

Try It Out (continued)

- Implementing the Distance Calculator, 431–435
- Implementing the `FileSortingHelper` Class, 473–475
- Implementing the Inserter, 219–220
- Implementing the Line Class, 454–457
- Implementing the `ListStack` Class, 103–104
- implementing the Natural Comparator, 119
- Implementing the Plane Sweep Algorithm, 464–467
- Implementing the `Predicate` Class, 33–35
- Implementing the Reverse Comparator, 121
- Implementing the Reverse Iterator, 28
- Implementing the `size()` and `isEmpty()` Methods, 86
- Implementing the Soundex Encoder, 423–425
- Implementing the Tests (for List Searcher), 214–215
- Implementing the Undo Action with the `UndoableList` Class, 109–113
- Implementing the `XYPointComparator`, 460–461
- Methods for Deleting Values (from Array Lists), 64–65
- Methods for Deleting Values (from Linked Lists), 72
- Methods for Finding Values (in Array Lists), 63–64
- Methods for Finding Values (in Linked Lists), 71–72
- Methods for Inserting and Adding Values (in Array Lists), 61–62, 69–70
- Methods for Storing and Retrieving Values by Position, 62–63
- Methods for Storing and Retrieving Values (in Linked Lists), 70–71
- Optimizing the `FileSortingHelper`, 483–487
- Performing a Bubble Sort, 122–124
- Performing the Search (with Boyer-Moore Algorithm), 406–407
- Popping a Value from the Stack, 104–105
- Testing a Binary Search Tree, 251–256
- Testing a `Node` Class (for a Binary Search Tree), 239–244
- Testing a Shellsort, 148–149
- Testing a Ternary Search Tree, 358–362
- Testing `AbstractSorterTest`, 125–126
- Testing and Implementing a Hash Map, 333–337
- Testing and Implementing a Hash Table That Uses Bucketing, 281–285
- Testing and Implementing a Hash Table That Uses Linear Probing, 275–280
- Testing and Implementing a Heap-ordered Priority Queue, 191–194
- Testing and Implementing a List Map, 330–333
- Testing and Implementing the Linear Searcher, 211–212
- Testing and Implementing the `Point` Class, 444–446
- Testing B-Trees, 381–382
- Testing `BubbleSortListSorter`, 126–127
- Testing Calculations, 16–18
- Testing `CompoundComparator`, 158–159
- Testing `InsertionSortListSorter`, 136
- Testing Iteration (in Lists), 55–57
- Testing Methods for Deleting Values (in Lists), 52–55
- Testing Methods for Finding Values (in Lists), 57–58
- Testing Methods for Inserting and Adding Values (in Lists), 47–50
- Testing Methods for Retrieving and Storing Values (in Lists), 50–52
- Testing `SelectionSortListSorter`, 132
- Testing the Array Iterator, 22–23
- Testing the Distance Calculator, 429–431
- Testing the FIFO Queue, 78–80
- Testing the Line Class, 449–454
- Testing the mergesort Algorithm, 166–168
- Testing the Natural Comparator, 118–119
- Testing the `peek()` Method (on Stacks), 101–102
- Testing the Plane Sweep Algorithm, 461–464
- Testing the `Predicate` Class, 29–32
- Testing the quicksort Algorithm, 155
- Testing the Reverse Comparator, 120–121
- Testing the Reverse Iterator, 26–27
- Testing the Slope of a Line, 446–448
- Testing the Soundex Encoder, 419–423
- Testing the xy point comparator, 459–460
- Testing What Happens When a List Is Cleared, 58
- Using the `BlockingQueue`, 83–84

Using the `push()` and `pop()` Methods (with Stacks), 100–101
 Writing the Tests (for List Searcher), 203–204
try/catch block, using in JUnit, 13

U

unbounded queue, definition of, 76
UndoableList class, implementing undo action with, 109–113
UndoAction interface, implementing, 113
UndoDeleteAction class, using with UndoableList class, 113
undo/redo. See also stacks
 implementing with stacks, 105–106
 testing, 106–113
unit tests
 importance of, 11–14
 overview of, 9–11
 running, 13–14
unordered lists, searching, 200
unsorted list priority queue, testing and implementing, 182–184
unsorted lists, determining positions of values in, 63–64
UnsupportedOperationException, throwing, 19, 409
upperIndex variable
 using with iterative binary searcher, 210
 using with recursive binary searcher, 206, 208

V

Value inner class, using with hash tables, 288
ValueIterator class, using with linked lists, 73
values
 adding to hash tables, 268–269
 calculating in Levenshtein word distance, 426–427
 deleting from array lists, 64–65
 deleting from linked lists, 72
 deleting from lists, 52–55
 enqueueing and dequeuing, 79–80
 finding in array lists, 63–64
 finding in linked lists, 71–72
 finding in lists, 57–58
 inserting and adding in array lists, 61–62
 inserting and adding in linked lists, 69–70

popping from stacks, 104–105
 retrieving and storing in lists, 50–52
 searching in binary search trees, 228–230
 storing and retrieving by position in array lists, 62–63
 storing and retrieving in linked lists, 70–71
 using enqueue and dequeue operations on, 78
values, inserting and adding in lists, 47–49
verify() method, using with list inserter, 219
vertical lines
 slope of, 441
 testing, 452–453
 testing functionality of, 450

W

waitForNotification() method, using with BlockingQueue, 84–85
waitForTermination() method, using with call center simulator, 92
websites
 JMP (Java Memory Profiler), 477
 JUnit, 11
while loops
 using with binary search tree, 258, 259
 using with InsertionSortListSorter, 137
 using with iterative binary searcher, 209
 using with iterators, 21
 using with tree maps, 343
 using with tree sets, 314
white-box testing, explanation of, 10
Widget class, naming convention for unit testing, 10
WILDCARD pattern character, using in ternary search trees, 370
wildcards, using for pattern matching in ternary search trees, 354, 355, 356
word distance, overview of, 426–429
_word variable, using with ternary search trees, 365
words, inserting in ternary search trees, 350–351. See also letters

X

x coordinate
 determining for intersecting lines, 455–456
 determining for intersection point, 457

x-y coordinate system, relationship to computational geometry, 437–438
xy point comparator, testing, 459–460. See also comparators
XYPointComparator, implementing, 460–461

Y

y axis
crossing, 442–443
relationship to computational geometry, 438

y coordinate
calculating for Line class, 455
determining for intersection point, 457

