CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

Index

abstract data type (ADT), 1, 47
  functions vs., 47
  merging with queue, 61
  specifications/implementations, 47
    AVL tree, 53
    binary search tree, 53
    dictionary, 51
    graphs, 51
    link list implementation, 55
    list, 47
    orders, 53
    parsing with stack, 62
    priority queue, 50
    queue, 49
    recursive definition of tree, 169
    set, 50
    set system, 50
    simple types, 48
    stack, 48
    trees, 52
    union-find set system, 53
Ackermann's function, 162
  algorithm, 162
  crashing, 163
  recurrence relation, 162
  running time, 163
  solving, 162
adding-made-easy approximations, 537
  examples, 537
  exercise solutions, 586
  proofs, 538
    analytical functions, simple, 538
    arithmetic sums, 536
    close to harmonic, 537
    functions, without basic form, 537
    geometric sums, simple, 534
    harmonic sum, 537
    ratio between terms, 534
  solution classes, 530
  technique, 530
ADT, *See* abstract data type
algorithms, *See also* recursive algorithms
Ackermann's function, 162
best AVL tree bird-and-friend algorithm, 396
bipartite matching using network flow, 419
brute force algorithm, 69, 294
defined, 1
Dijkstra's shortest-weighted path algorithm, 253
dynamic programming algorithms and examples, 338
  all pairs, matrix multiplication, 397
  best AVL tree, 397
  chains of matrix multiplications, 390
  context-free grammar parsing, 398
  dynamic programming algorithms. *via* reductions, 338
  longest increasing contiguous subsequence example, 369
  longest increasing subsequence example, 370
  longest-common-sequence problem, 364
  shortest weight path, directed leveled graph example, 337
  weighted job/event scheduling problem, 371
Euclid's greatest common divisor (GCD) algorithm, 80
graph algorithms
  expander graphs, 428
  max cut problem, 428
  minimum spanning tree, 314
  network flows, 268
  shortest weight path, directed leveled graph example, 337
  3-Colouring, 408, 419
graph search algorithms, 243
  breadth-first search, shortest path, 248, 250
  depth-first search, 259
  generic search algorithm, 243
  partial order linear ordering, 265
    depth-first search algorithm, 265
    easy but slow algorithm, 265
  recursive depth-first search, 263
greedy algorithms, 294, 330, 367, 395
hill-climbing algorithm, 290
iterative algorithms, 9, 13, 21

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

- iterative sorting algorithms, 74, 75, 78
- looking forward vs. backward, recursive algorithms, 134
- meta-algorithms, xiii, 2
- parsing algorithm specifications, 199
- randomized algorithms, 428
- sorting and selection algorithms, 423
- steepest-ascent hill-climbing algorithm, 149
- Strassen's matrix multiplication, 161, 284
- time/space complexity, 83, 508, 518, 579, 582

analytical functions, simple, 538
arithmetic sums, 536 *See also* adding-made-easy approximations,
asymptotic growth
- asymptotic growth rate, 518
- asymptotic notation, 88, 518, 519
  - BigOh definition, 88, 525
  - Little Oh definition, 525
  - Little Omega definition, 525
  - loop invariant for lower bounds, 80
  - Theta definition, 88, 518
- exercise solutions, 586
- function classification, 518
- growth rates classes, 518
- purpose, 518

AVL tree, 53
- best AVL tree, 395
  - best AVL tree problem, 396

binary search, 24
- for cost in optimization problems, 407
- narrowing the search space example, 22
- returning index example, 91
- returning yes/no, 93
- trees, 53, 64
  - ADT, 50
  - balanced, 53
  - basic steps, 64
  - best binary search tree problem, 395
  - coding/implementation details, 65
  - ending, 65
  - establishing loop invariant, 64
  - exit condition, 65
  - is tree a binary tree example, 177
  - loop invariant, 64
  - main steps, 65
  - maintain loop invariant, 65
  - make progress, 65
  - measure of progress, 64
  - nodes in binary search tree example, 174
  - running time, 65
  - specifications, 64
- typical errors, 31

binary trees, recursion, 209
BigOh definition, 526
bipartite edges, 261
bipartite matching using network flow, 419
breadth-first search, 248
- shortest path, 248, 249
  - code, 251
  - exiting loop, 252
  - initial code, 252
  - loop body, 244
  - loop invariant, 245
  - maintaining loop invariant, 245
  - optimization problem, 248
  - shortest path problem, 248
  - shortest path proof, 251

bucket sort by hand, 74
- basic steps, 74
- exit condition, 75
- loop invariant, 75
- maintain loop invariant, 75
- specifications, 74

chains of matrix multiplications, 390
- failed dynamic programming algorithm, 391
- failed greedy algorithm, 391
- little bird question, 391
- optimal solution construction, 394
- optimal solution cost, cost for subinstances, 392
- recursive structure, 392
- reduced to subinstance, 392
- set of subinstances called, 392
- table fill order, 392
- table indexed by subinstances construction, 392
- time/space requirements, 394

coloring the plane, 33
- basic steps, 33
- coding/implementation details, 34
- ending, 34
- establishing loop invariant, 34
- exit condition, 34
- loop invariant, 34
- main steps, 34
- maintain loop invariant, 34
- measure of progress, 34
- running time, 34
- special cases, 34
- specifications, 33

computational complexity, 1
- asymptotic notations, 88, 518, 525
- formal proof of correctness, 549
- nondeterministic polynomial-time decision problems (NP), 408
- time/space complexity, 83, 425, 508, 519, 548, 584

context-free grammar parsing, 198
- abstract data type parsing, 199
- correctness proof, 205
- dynamic programming example
  - abstract data type parsing, 399
  - base cases, 400
  - help from friend, 400

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

little bird question, 399
not look ahead one, 399
number of parsings, 399
optimal solution construction, 400
parsing problem, 399
set of subinstances, 400
table fill order, 400
table indexed by subinstances construction, 400
exercise solutions, 578
expression parsing, 199
GetExp code, 201
GetExp expressions, 201
GetExp, Get Term, GetFact examples, 201
GetFact code, 203
GetTerm code, 203
grammar, 198
look ahead one, 198, 205
nonterminals, 199
parsing algorithm specifications, 199
rules, 199
running time, 204
semantics and, 198
string derivation, 198
syntax and, 198
terminals, 199
tree of stack frames, 204
correctness, 2
formal proof of correctness, 549
for context- free grammar parsing, 199
for depth-first search, 265
for dynamic programming, 353
for greedy algorithms, 321
for recursive algorithms, by strong induction, 147
counting sort, 75
basic steps, 76
code, 77
establishing loop invariant, 77
exit condition, 77
loop invariant, 77
main step, 77
maintain loop invariant, 77
running time, 77
specifications, 75
cyclic edges, 261

Davis–Putnam, 331
depth-first search, 247, 259
code, 259
edges classification, 261
establish/maintain loop invariant, 260
generic search algorithm changes, 259
loop invariants, 259
recursive, 263
time stamping, 262
deterministic finite automation (DFA), 35
addition example, 39
applications, 35
calculator example, 40
compiling iterative program into DFA, 37
division example, 40
dynamic programming, 41
longest block of ones example, 41
longest increasing contiguous subsequence, 42
longest increasing subsequence, 42
dictionary ADT, trees, data structure, 52–53
Dijkstra’s shortest-weighted-path algorithm, 253
code, 255
exiting loop, 258
initial code, 258
loop body, 255
loop invariant, 255
maintaining LI1, 256
maintaining LI2, 256
problem specifications, 253
shortest distance approximation, 253
shortest path proof, 251
dynamic programming, 336, 444–453
algorithm using trusted bird and friend, 445
can’t reduce, 453
cutting task into two independent tasks, 451–452
exercise solutions, 585, 586
G as the graph of life, 450
nodes of *G* are subinstances, 449–450
recursive backtracking, 336, 449
running time, 340
recursive backtracking algorithm, 445
reduction of any dynamic programming algorithm to *LeveledGraph*, 448
running time, 453
specifications, 445
steps in developing, 336
base cases, 344
code, 345
count subinstances, 343
final solution, 344
redundancy, 341
running time, 346
set of subinstances, 341
solution from subsolutions, 343
table fill order, 344
table indexed by subinstances construction, 343
subtle points, 346
dynamic programming algorithms *via* reductions, 375
best path similarity, 375
bigger-is-smarter elephant problem, 379
event scheduling problem, 375
graph instance formation, 376
mapping back algorithm, 377
dynamic programming algorithms and examples

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

all pairs, matrix multiplication, 397
best AVL tree, 395
best AVL tree problem, 396
best binary search tree, 395
chains of matrix multiplications, 390
context-free grammar parsing, 398
dynamic programming algorithms. *via* reductions, 375
longest increasing contiguous subsequence example, 369
longest increasing subsequence example, 370
time/space requirements, 368
longest-common-sequence problem, 364
base cases, 365
code, 366
greedy algorithm, 364
information about subinstance, 367
little bird possible answers, 364
little bird question, 365
longest common sequence, 364
optimal solution construction, 367
set of subinstances, 365
table fill order, 366
table indexed by subindexes construction, 365
time/space requirements, 368
shortest weight path, directed leveled graph example, 337
weighted job/event scheduling problem, 371
failed algorithms, 371
greedy dynamic programming, 372

edges classification, 261
back edges, 261
bipartite edges, 261
cross edges, 261
cyclic edges, 261
forward edges, 261
tree edges, 261
Euclid's GCD algorithm, *See* greatest common divisor algorithm
bug, 86
code, 86
elementary school algorithm, 84
ending with postcondition, 85
Ethiopians, 87
have $code_{loop}$, 85
have (*loop-invariant*), 85
head in the right direction & return to path, 85
high school algorithm, 86, 87
LHS vs. RHS, 85, 86
loop invariant, 84
maintaining the loop invariant, 85, 86
make progress, 86
preconditions, 85
running time, 85, 86
Euler cycle, 44
event scheduling, 307
weighted job/event scheduling problem, 371
existential/universal quantifiers, 500, 513
bound variables definition, 500
combining quantifiers, 500
exercise solutions, 585
expressions building, 501
free variables definition, 500
Loves example, 499
negation, 502
quantifiers order, 502
relation definition, 501
representations, 499
variable domain, 502
exponentials, 515
base, 517
exercise solutions, 585
ratio, 517
rules, 515
uses, 515

fast Fourier transformation, 161
Fast Fourier Transformations (FFT), 163–168
code, 165
$FFT(FFT(F(t)) = n \cdot F(t)$, 167
interpolations, 164
inverse sin-cos-FFT, 167
multiplying, 167
my input, 164
my output, 164
n values of x, 164
nth roots of unity, 167
nth roots of unity are special, 165
radians, 167
recursion, 164
rotating the plane, 166
special values, 164
splitting the input, 164
t^{th} value of x and f^{th} term/frequency, 167
FFT, *see* Fast Fourier Transformations
find-max two-finger algorithm example, 11
forward/cross edges, 261
friends level of abstraction, recursive algorithms, 135
base cases, 137
general input, 135
generalizing problem, 136
link to techniques, iterative algorithms, 137
minimizing number of cases, 136
running time, 137
size, 135
specifications, 135
functions, *See also* Ackermann's function
abstract data types vs., 47
analytical functions, simple, 538
linear function, 523
quadratic function, 523
time/space complexity as, 508

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

game of life, 380–389, *see also* graph of life
  graph G from computation, 380
  sequence of actions vs. a sequence of assertions, 5
GCD, *see* Euclid's GCD algorithm
GCD algorithm, *See* greatest common divisor (GCD) algorithm
geometric sums, simple, 534
GetExp code, 201
GetExp expressions, 199
GetExp, Get Term, GetFact examples, 201
GetExp reasoning, 201
GetFact code, 203
GetFact reasoning, 203
GetTerm code, 203
global vs. local considerations, 270, 274, 291, 295, 322, 347
graph algorithms
  expander graphs, 428
  max cut problem, 428
  minimum spanning tree, 314
  network flows, 268
  shortest weight path, directed leveled graph example, 337
  3-Colouring, 408, 415
graph of life, 382–389, *see also* game of life
  actions vs. landmarks, 382
  algorithm 1 – maximize cash on day t, 387
  algorithm 2 – maximize stock s at time t, 388
  algorithm 3 – two types of states and atomic actions, 388
  assertions, 382
  buy stock, 389
  cutting a task into two independent subinstances, 383
  edge cost, 384
  edge length, 384
  examples, 385–389
  finding optimal path, 384
  forgetting, 383
  G as the graph of life, 450
  goal to find optimal path, 384
  knapsack, 385
  little bird, 384
  longest common subsequence, 386
  path of life, 382
  printing neatly, 387
  produce the solution *Sol*, 383
  pseudocode execution, 382
  reading the input I, 383
  resource, 383
  running time, 384
  sell stock, 389
  sitting on cash, 388
  sitting on stock, 389
  states and possible actions, 383
  stock problem, 387
  TM & DFA, 382
graph search algorithms, 243
  breadth-first search, shortest path, 247, 248
  depth-first search, 259
  Dijkstra's shortest-weighted-path algorithm, 253
  exercise solutions, 575
  generic search algorithm, 243
    basic steps, 243
    code, 245
    exit condition, 246
    exiting loop, 246
    handling nodes order, 247
    initial code, 246
    loop body, 244
    loop invariant, 244
    maintaining loop invariant, 245
    measure of progress, 245
    reachability problem, 243
    running time, 247
  partial order linear ordering, 265
  recursive depth-first search, 263
graph theory problems, 243
greatest common divisor (GCD) algorithm, 80
  code, 82
  ending, 80, 82
  establishing loop invariant, 80
  example, 82
  exercise solutions, 570
  exit condition, 82
  iteration on general instance, 80, 81
  loop invariant, 80
  lower bound, 83
  making progress, 81
  recursive, 160
  running time, 80
  special cases, 81
  specifications, 80
  termination, 82
greatest common divisor algorithm (GCD), *see* Euclid's GCD algorithm
greedy algorithms, 294, 439–444
  alternative proof, 303
  brute force algorithm, 294
  correctness proof, 299
    using loop invariants, 296
  definition of correct, 303
  does not go down, 303
  does not go up, 303
  examples
    game show, 295
    interval cover problem, 310
    job/event scheduling problem, 307
    minimum spanning tree problem, 314
  exercise solutions, 575
  exiting loop, 440, 444

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

first order logic, 304
fix problems, 301
fixity vs. adaptive priority, 304
game, 304
game as proof, 304
globally optimal step, 303
greedy choice, 295, 439
greedy game, 304
head in the right direction & return to path, 301
initially ($(pre \Rightarrow LI)$), 300
instructions for modifying, 300
locally optimal step, 303
loop invariant, 297, 441
loop invariants, types, 296
specifications, 294
make S_t consistent, 301
making change example, 300
many cases, 301
postcondition, 294
$potentialVal(A_t)$, 303
proof, 303
proof globally optimal, 304
proving, 304
proving that she has a witness, 302
specifications, 439
taking a step, 300
theorem, 303
three players, 300
tree of options, 301
using, 304
what we know, 300

harmonic sum, 537
close to harmonic, 537
heap sort/priority queues, 180
array implementation, balanced binary tree, 180
common mistakes, 186
completely balanced binary tree, 180
heap definition, 180
heapify problem, 181
code, 181
iterative algorithm, 182
recursive algorithm, 181
running time, 182, 183
specifications, 181
heapsort problem, 184
algorithm, 184
array implementation, 184
code, 185
specification, 184
makeheap problem, 183
iterative algorithm, 183
recursive algorithms, 183
running time, 184
specifications, 183
priority queues, 186
hill-climbing algorithm, *See also* primal-dual hill climbing method; steepest-ascent hill-climbing algorithm, 290
small local maximum, 270
algorithm, faulty, 272
algorithm fixing, 274
augmentation graph, faulty, 272
basic ideas, 270
counterexample, 274
local maximum, 274

image drawing, *See* recursive image drawing
information hiding, 47
information theoretic lower bounds, 92, 93
iterative algorithms, 9, 13
basic steps, 14
code structure, 9
coding/implementation details, 20
conservation of energy, 26
correctness proof, 10
differential equations, 27
ending, 19
establishing loop invariant, 18
exit condition, 19
find-max two fingeralgorithm example, 11
formal proof, 20
loop invariants, 28
loop invariants for, 9, 14, 16
main steps, 17
maintain loop invariant, 17
make progress, 17
measure of progress, 14
running time, 10, 20
search space narrowing, 22
binary search narrowing example, 24
special cases, 20
specification, 13
types of iterative algorithms, 21
case analysis, 23
insertion sort example, 24
more of input, 22
more of output, 22
selection sort example, 23
work done, 23
bubble sort example, 26
iterative sorting algorithms
bucket sort by hand, 74
counting sort, 75
radix sort, 78

job/event scheduling, *See* event scheduling

kth smallest element example, 152

Las Vegas model, 424
linear function, 531
linear programming, *See also* network flow, 288
Euclidean space interpretation, 289

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

example, 289
formal specification, 289
hill-climbing algorithm, 290
matrix representation, 289
network flows, 289
running time, 292
small local maximum, 291
link list implementation, 55
adding node to end, 57
adding node to front, 56
deleting node, 60
hidden invariants, 56
initialize walk, 58
notation, 56
removing node from end, 57
removing node from front, 57
testing whether empty, 57
walking down linked list, 57
Little Oh definition, 525, 527
Little Omega definition, 525, 527
logarithms, 515
base, 517
exercise solutions, 584
ratio, 517
rules, 515
uses, 515
longest block of ones example, 41
longest increasing contiguous subsequence example, 369
longest increasing subsequence example, 370
longest-common-sequence problem, 241, 364
base cases, 365
code, 366
greedy algorithm, 364
little bird possible answers, 364
little bird question, 365
longest common sequence, 364
optimal solution construction, 367
set of subinstances, 365
table fill order, 366
table indexed by subindexes construction, 365
time/space requirements, 368
looking forward vs. backward, recursive algorithms, 134
algorithm, 134
loop invariant for lower bounds, 88
asymptotic notation, 88
binary search returning index example, 91
binary search returning yes/no, 93
dynamic algorithms, 92
exercise solutions, 565
flipping a bit, 92
loop invariant argument, 89
lower bounds proof, state of art, 94
multiplexer example, 93
parity example, 92, 93
sorting example, 90
time complexity, 88
upper bound, algorithm, 88
loop invariants, 97–101
code from loop invariants, 28
defining, 97
exit condition, 99
greedy algorithms, 297
initializing the system, 101
iterative algorithms, 28
maintaining the loop invariant, 98
maintaining the system invariants, 100
obtaining the postcondition, 99, 101
operation specification, 101
private invariants, 100
public invariants, 100
relationship to system invariants, 100
running time, 99
specification, 97
step, 97
loop invariants for iterative algorithms, 9, 14, 16
lower bounds, *see also* loop invariant for lower bounds, 83, 88, 94, 334, 402, 565, 577
for GCD, 83
loop invariant for lower bounds, 88
lower bounds proof, state of art, 94
reductions, 402

machine learning, 431–438
abstract thinking, 432
algorithm, 434
blind, 433
cat, 432
coding, 431
compression, 434
convolution and recurrent layer, 438
correlation of vectors, 436
error, 433
error surface, 433
evolution, 431
generalizing, 434
gradient descent, 433
hopeful applications, 431
linear and non-linear regression, 432, 433
linear layer = matrix multiplication, 436
machine, 432
neural networks, 433
non-linear layer, 437
overfitting, 434
regularization, 435
scary applications, 431
singularity, 438
smooth/differentiable, 434
supervised training data, 432
theory: Learnable Probably Approximately Correct (PAC), 435
underfitting, 434

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

vectors, 436
magic sevens, 66
basic steps, 67
establishing loop invariant, 67
exit condition, 67
loop invariant, 67
maintain loop invariant, 67
running time, 67
specifications, 66
matrix multiplication
all pairs, 397
chains of, solution as tree, 390
Strassen's matrix multiplication example, 161
measures of progress, loop invariants, 5
merge sort example, 149
merging with queue, 61
meta-algorithms, xiii, 2
min cut specification, 269
Monte Carlo model, 424
more of input, 22
more-of the-output loop invariant algorithms, 22
Euler cycle example, 44
selection sort example, 23
more-of-the-input iterative loop invariant algorithms, 22, 34, 70, 136, 296, 368, 562
coloring the plane example, 33
deterministic finite automation, 35
exercise solutions, 562
in dynamic programming
longest increasing contiguous subsequence example, 369
longest increasing subsequence example, 370
in greedy algorithms, 296
in VLSI chip testing example, 68
more-of-the-input vs. more-of-the-output, 42
tournament example, 42
recursive algorithms, link to, 137
multiplexer example, 92

narrowing the search space, 22, 24, 64, 89, 137, 297, 565
binary search example, 24
binary search trees, 64
exercise solutions, 565
magic sevens, 66
VLSI chip testing, 68
network flows/linear programming, 268
bipartite matching using network flow, 419
exercise solutions, 573
hill-climbing algorithm, small local maximum, 270
linear programming, 288
min cut specification, 269
primal-dual hill-climbing method, 276
specification, 268
steepest-ascent hill-climbing algorithm, 284
nondeterministic polynomial-time decision problems (NP) completeness, 407
bipartite matching, network flow algorithm, 402, 419, 420
classifying problems, 403
exercise solutions, 582
lower bounds, 402
NP completeness proof steps, 407, 409
nondeterministic polynomial-time decision problems (NP), 407
reduction P_{1poly} P2, 402
reverse reductions, 403
satisfiability vs. optimization
Alg for optimization problem, 406
CIR-SAT, 404
optimization problems, 406
3 coloring example, 403
upper bounds, 402
why reduce, 402

operations on integers, 157
b^N example, 157
Strassen's matrix multiplication example, 161
xyz example, 159
optimization problems, 241
examples, 241
airplane, 242
course scheduling, 242
longest common sequence, 241
network flow, 268
problem specification, 241

parity example, 92, 93
parsing with stack, *see also* context-free parsing, 62
code, 62
ending, 62
example, 62
initial conditions, 62
loop invariant, 62
maintaining loop invariant, 62
parsing only, 63
parsing with context-free grammar, 63
specifications, 62
partial order linear ordering, 264
depth-first search algorithm, 265
easy but slow algorithm, 265
partial order definition, 265
shortest weight path, DAG, 267
topological sort problem specifications, 265
total order definition, 264
postconditions, 1
preconditions, 1
primal-dual hill-climbing method, 276
algorithm, 276
ending, 278
max-flow-min-cut duality principle, 282
running time, 283

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

primality testing, randomized, 425
printing neatly example, 346
private invariants, 100
pseudocode
  graph of life, 382
  iterative algorithms, 18
public invariants, 100

quadratic function, 523
quantifiers, *See* existential/universal quantifiers
quick sort
  example, 150
  randomized, 425

radix sort, 78
  basic steps, 78
  ending, 79
  establishing loop invariant, 79
  loop invariant, 79
  maintain loop invariant, 79
  specification, 78
randomized algorithms, 423
  deterministic worst case model, 424, 425
  game show problem, 425
  hiding worst cases from adversary, 423
  Las Vegas model, 424
  Monte Carlo model, 424
  optimization problems with random structure, 427
    expander graphs, 428
    max cut problem, 428
  quick sort, 425
  randomized counting, 426
  randomized primality testing, 425
  VLSI chip testing, 72
randomly generating maze, 195
  initial conditions, 195
  postcondition, 195
  precondition, 195
  running time, 196
  searching maze, 197
  subinstances, 196
recurrence relations, 540
  exercise solutions, 586
  in recursive backtracking algorithms, 328
  proofs, 543
  recursive programs timing, 540
  solving recursive relations, 541
recursion, 133, 208–210
  binary trees, 209
  Fast Fourier Transformations (FFT), 164
  formal proof, 209
  friends level of abstraction, 135
  on trees
    exercise solutions, 570
    generalizing problem solved, 179
    heap sort/priority queues, 180
    recursive definition of tree, 169
    representing expressions with trees, 187
    simple examples, 177
  running time, 209
  steps, 208
recursion on trees
  data structure, 171
  many children, 172
  running time, 171
recursive algorithms, *See also* recursive backtracking
  checklist for, 139
    code structure, 139
    specifications, 140
    tasks to complete, 142
    variables, 140
  correctness proof, with strong induction, 146
  examples
    Ackermann's function, 162
    exercise solutions, 567
    operations on integers, 157
    b^N example, 157
    Strassen's matrix multiplication example, 161
    xyz example, 159
  exercise solutions, 567
    base cases, 137
    general input, 135
    generalizing the problem, 136
    link to techniques, iterative algorithms, 137
    minimizing number of cases, 136
    running time, 137
    size, 135
    specifications, 135
  looking forward vs. backward, 134
  solving, 162
  sorting/selecting algorithms, 149
    choosing the pivot, 152
    finding the *k*th smallest example, 152
    general recursive sorting algorithm, 149
    merge sort example, 149
    partitioning according to pivot element, 155
    quick sort example, 150
  stack frame, 144
  strong induction, 146
  tower of Hanoi, 138
recursive backtracking algorithms, 321, 576
  as sequence of decisions, 321
    best animal searching example, 323
    maze searching example, 322
  developing steps, 325
  exercise solutions, 576
  pruning branches, 329
    greedy algorithms, 330
  queens problem example, 325
  satisfiability, 331
    code, 333
    Davis-Putnam, 332

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

instances/subinstances, 332
pruning, 332
running time, 334
satisfiability problem, 331
solutions iterating, 331
recursive backtracking, dynamic programming, 445
recursive depth-first search, 263
achieving postcondition, 264
code, 263
example, 264
running time, 264
recursive image specification, 192
recursive images, 192
exercise solutions, 571
fixed recursive/base case image, 192
base case, 192
birthday cake, 193
examples, 193
fractal, 194
image drawing, 192
man recursively framed, 193
recursing, 192
rotating square, 193
randomly generating maze, 195
reduction, 402
dynamic programming algorithms via, 375
best path similarity, 375
bigger-is-smarter elephant problem, 379
event scheduling problem, 375
graph instance formation, 376
mapping back algorithm, 377
lower bound, 402
optimization problems reduction, 406
reverse reduction, 403
upper bound, 402
use in classifying problems, 403
running time, *See* time/space complexity, 2
Ackermann's function, 163
best binary search tree, 396
binary search, 24
binary search trees, 65
coloring the plane, 34
context-free grammar parsing, 204
counting sort, 77
dynamic programming, 346
GCD algorithm, 82
generic search algorithm, 247
heapify, 182
heapsort, 186
interval cover, 314
iterative algorithms, 10, 20
job/event scheduling, 310
linear programming, 292
magic sevens, 67
makeheap, 183
merge sort, 150
minimum spanning tree, 316
nodes in binary tree, 171
primal-dual hill-climbing method, 283
queens problem, 329
quick sort, 151
randomly generating maze, 196
recursive algorithms, 137
recursive backtracking, 340
satisfiability, 334
steepest-ascent hill-climbing algorithm, 285
towers of Hanoi, 139
union find set system, 54
VLSI chip testing, 72

sequence of actions vs. a sequence of assertions
check each computation path, 6
code vs. math assertions, 7
correctness, 9
differentiating between values, 7
game of life, 9
loops, 8
one step at a time, 7
proof by case, 8
proof by transitivity, 8
proof of correctness of each step, 8
proof of correctness of the algorithm, 6
shortest weight path, directed leveled graph example, 337
solution is a tree, all pairs using matrix multiplication, 397
sorting and selection algorithms, 149
bubble sort example, 26
finding the *k*th smallest example, 152
choosing the pivot, 152
iterative sorting algorithms
bucket sort by hand, 74
counting sort, 75
radix sort, 78
merge sort, 149
quick sort, 150
randomized quick sort, 425
recursive sorting algorithm, general, 149
stack frame, 144
memory, 145
stack of stack frames, 145
tree of stack frames, 133, 144, 171, 204
using, 146
steepest-ascent hill-climbing algorithm, 284
augmentation path, 284
running time, 285
Strassen's matrix multiplication example, 161
system invariants, 99–101
goal, 99
private invariants, 100
public invariants, 100
relationship to loop invariants, 99

© in this web service Cambridge University Press & Assessment www.cambridge.org

CAMBRIDGE

Cambridge University Press & Assessment
978-1-009-30214-2 — How to Think about Algorithms
Jeff Edmonds
Index
More Information

time/space complexity, 83, 84, 403, 425, 513, 518, 548, 584
  examples, 510
  exercise solutions, 584
  formal definition, 513
  as functions, 508
  operation definition, 510
  purpose, 508
  size definition, 509
  time complexity of problem, 510
tournament example, 42
towers of Hanoi, 138
  code, 139
  divide and conquer, 138
  running time, 139
  specification, 138, 139
  subinstance, 139
tree edges, 261
trees, *See also* binary search trees; recursion on trees, 52, *see also* recursion on trees
  AVL tree, 53
  best AVL tree problem, 396
  best binary search tree, 395
  binary search tree, 53
  chains of matrix multiplications, solution as tree example, 390
  dictionary ADT, data structure, 52–53
  is a tree a binary search tree example, 174
  nodes in binary tree example, 170
  recursive definition of, 169
  representing expressions with
    differentiate expression example, 189
    evaluate expression example, 188
    recursive definition of expression, 188
    simplify expression example, 189
    tree data structure, 188
  of stack frames, 133, 144, 171, 204, 294
  traversals, 174

universal quantifiers, *See* existential/universal quantifiers
upper bound, 88
  cuts as upper bound, 279
  for hill-climbing, 292

VLSI chip testing, 68
  brute force algorithm, 69
  data structure, 69
  exiting loop, 72
  extending the algorithm, 72
  faster algorithm, 70
  initial code, 72
  loop invariant design, 70
  maintaining loop invariant, 71
  measure of progress, 72
  randomized algorithm, 72
  running time, 72
  specification, 69

weighted job/event scheduling problem, 371
  failed algorithms, 371
  greedy dynamic programming, 372
  weighted event scheduling problem, 371
work done-bubble sort example, 26

© in this web service Cambridge University Press & Assessment www.cambridge.org