Algorithm Design and Analysis

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

Proof sketches with numbers

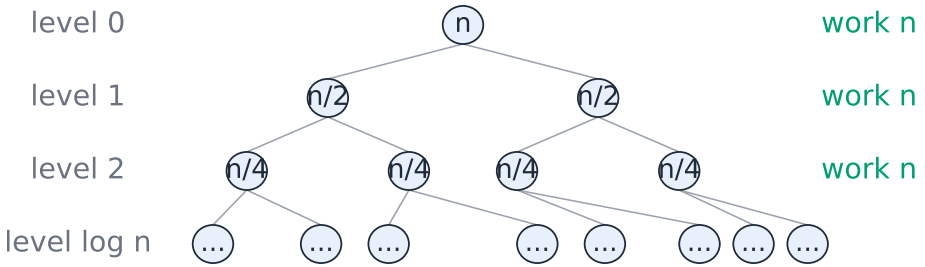
- 10.1-10.2 analysis and the first pattern; 10.3-10.4 the choice patterns.
- 10.5-10.6 the frontier; every claim here is verified on concrete inputs.

Recurrences: the tree is the analysis

1. **Tree** T(n) = 2T(n/2) + n: level i has 2^i nodes of n/2^i - every level costs n, depth log n, product n log n.
2. **Master** Leaves win: n^(log_b a); balanced: f(n) log n; root wins: f(n) - 4T(n/2) + n is n^2, T(n/2) + 1 is log n.
3. **Measure Merge sort** on the 64-element course array: 310 comparisons, under the 321 ceiling n log n - n + 1.

Recursion tree

every level costs n, there are log n levels



T(n) = 2T(n/2) + Theta(n) = Theta(n log n): level cost x depth

> Every level costs the same n; the depth is log n.

Divide and conquer: price the combine step

1. **Merge** $2T(n/2) + n = n \log n$; binary search is the degenerate case $T(n/2) + 1 = \log n$ - one branch still counts.
2. **Quick** Balanced: n log n; sorted input with first-element pivots: T(n-1) + n = n^2; measured 310 vs 356 vs worst 2080 comparisons.
3. **Subarray** Left best, right best, or the crossing best: the crossing scan is a full pass - Theta(n log n); course answer 6.

Recursion structure

the same skeleton, three combine costs

merge sort	quicksort (balanced)	binary search
split in half	split in half	split in half
recurse on the parts	recurse on the parts	recurse on the parts
2T(n/2) + n	**2T(n/2) + n**	**T(n/2) + 1**
Theta(n log n)	**Theta(n log n)**	**Theta(log n)**

quicksort unbalanced: T(n) = T(n-1) + n = Theta(n^2) on sorted input — the pivot decides

course measurements: merge 310 comparisons, quicksort 356, worst case 2080 (all on the 64-element array)

one branch is still the pattern: binary search splits, recurses once, combines in O(1)

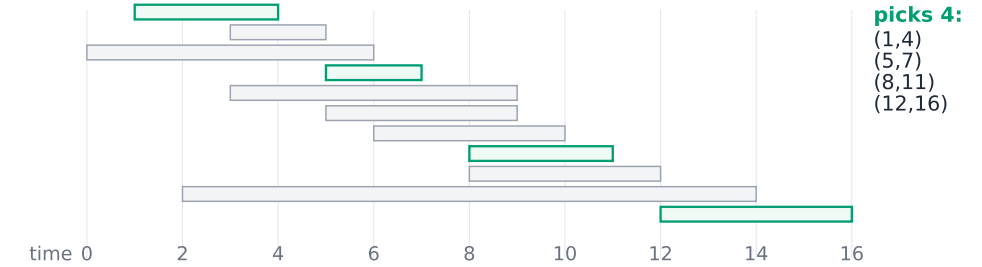
> The same skeleton with three combine costs - three different bills.

Greedy: prove optimal can afford you

1. **Activity** Earliest-finish picks (1,4) (5,7) (8,11) (12,16) - four of eleven course activities, provably optimal.
2. **Exchange** finish(G1) <= finish(O1), so swapping optimal first pick for greedy keeps feasibility - repeat and optimal becomes greedy.
3. **Coins** Coins 1, 3, 4 for 6: greedy pays three (4+1+1), the **DP table** pays two (3+3) - no **exchange argument** there.

Greedy decision diagram

earliest finish first, and why optimal can afford it



exchange: swap any optimal first pick for the earliest-finishing one - still feasible, same size

Earliest finish first; the **exchange argument** shows optimal can start the same way.

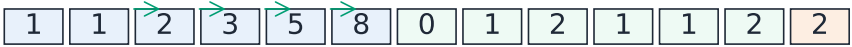
Dynamic programming: remember, do not recompute

1. **Fib** Naive fib(30): 2,692,537 calls - the call count is itself exponential; memoised: 29 subproblems, Theta(n).
2. **Coins** dp[i] = 1 + min(dp[i - c]) over usable coins: dp[6] = 2 (3+3) where greedy pays 3 - the table beats the heuristic.
3. **Knapsack** Capacity 7, items (1,1)(3,4)(4,5)(5,7): dp[7] = 9 - items 2 and 3, weight 7, value 9.

DP table

each cell reads cells to its left or below

Fibonacci table (left to right)



cell i reads i-1 and i-2

Coin table (coins 1, 3, 4)

0	1	2	1	1	2	2

dp[6] = 2 (3 + 3), not greedy 3 (4+1+1)

dependency order is part of the design: iterate the direction the arrows point

memoised fib(30): 29 subproblems - naive: 2,692,537 calls

knapsack capacity 7, items (1,1)(3,4)(4,5)(5,7): dp[7] = 9 (items 2 and 3)

optimal substructure + overlapping subproblems = tabulate

the table IS the explanation

Each cell reads cells to its left or below - the **dependency order**.

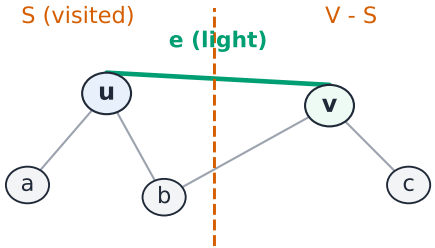
exercises.lol exercises.com.cn

Graph optimisation: light edges across cuts

1. **Kruskal** Sorted course edges: accept CB 1, AC 2, DE 2, EF 3, BD 5; reject the cyclers - MST weight 13.
2. **Proof** Any tree + light crossing edge - the cycle other crossing edge: weight kept or lowered - greedy never traps itself.
3. **Negative** A-B 5, A-C 1, C-B 1: settling B at 5 misses the path of 2 - Bellman-Ford (Theta(VE)) is the honest tool.

Graph cuts

the light edge crossing a cut is safe



Kruskal: accept across cuts

sort by weight: CB 1, AC 2, DE 2, EF 3, AB 4, BD 5, DF 6, CD 8, CE 10

accept CB, AC, DE, EF, BD

reject CD, CE, AB, DF (cycle)

MST weight: 1+2+2+3+5 = 13

negative edges break the greedy: A-B 5, A-C 1, C-B 1 - settling B at 5 misses the path of 2

light-edge proof: any tree + e - cycle through e keeps or lowers the weight

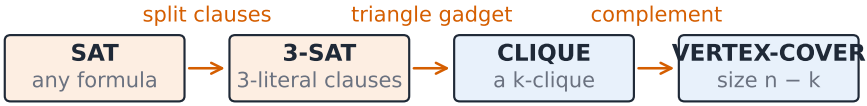
A cut separates visited from unvisited; the light crossing edge is safe.

Reductions: the arrow carries the difficulty

1. **Chain** SAT to 3-SAT (clause splitting) to CLIQUE (the triangle gadget) to VERTEX-COVER (complement the graph) - solve one, solve the chain.
2. **Course** Triangle ABC plus isolated D: complement edges are A-D, B-D, C-D; cover $\{D\}$ of size n - k = 1 covers them all.
3. **Direction** Reducing YOUR problem to SAT proves nothing - the arrow must carry difficulty INTO your problem, from a known hard one.

Reduction map

difficulty flows along the arrows



each arrow is a poly-time translation: a solver for the right solves the left

course case: G = triangle ABC + isolated D, so a 3-clique

complement edges A-D, B-D, C-D: cover {D}, size n − k = 1

NP-complete = in NP + every NP problem reduces to it

the arrow carries the difficulty: reduce the KNOWN hard problem TO yours

P sits inside NP; nobody knows if they are equal

SAT to 3-SAT to CLIQUE to VERTEX-COVER - solve one, solve the chain.

Check yourself

- One level of 2T(n/2) + n costs...? (n.)
- 4T(n/2) + n solves to...? (n^2.)
- Naive fib(30) makes... calls? (2,692,537.)
- To prove B NP-hard, reduce...? (A known hard problem TO B.)