Teknikker for algoritmedesign:

Brute Force ("Rå kraft" / "Rett-fram-etter-nesa")

F.eks: Selection sort, Bubble sort, sekvensielt søk

- Exhaustive search (Genererer alle kombinasjoner, siler ut løsning(ene))

F.eks: Traveling Salesman, Knapsack Problem, tildeling(er)

Divide-and-conquer (Splitt-og-hersk)

Rekursive algoritmer som generelt er meget effektive. Rekursjonen er splitt-delen, kombineringen av de rekursive løsningene er hersk-delen.

- Divide before processing

F.eks: Mergesort

- Process before dividing

F.eks: Quicksort, binært søk, pre-/in-/post-ordertraversering av trær

Decrease-and-conquer (Minsk-og-hersk)

Reduser problemet, løs dette, for å finne totalløsningen. Ofte rekursivt.

- Decrease by a constant (som oftest med 1-en)

F.eks: Insertion sort, Shellsort, DFS, BFS, topologisk sortering, permutasjoner, utplukk/subsett

- Decrease by a constant factor (som oftest med 2-to)

F.eks: Binært søk, finne-*en*-falsk-mynt, Josephus Problem, beregne eksponent vha. kvadrering

- Variable size decrease

F.eks: Interpolasjonssøk, Euclid's algoritme, søk/insert i binært søketre, n'te minst element i mengde, Nim's-spill

Transform-and-conquer (Omgjør-og-hersk)

- Instance simplification

Omgjør til et enklere/mer "behagelig" problem å løse.
F.eks: For-sortering, Gauss-eliminiasjon, balanserte søketrær (Red-Black, AVL, Splay)

- Representation change

Omgjør til en annen *representasjon* av problemet.
F.eks: Heaper, Heapsort, balanserte søketrær (2-3-4 trær, B-trær), Horner's regel/binær eksponentiering

- Problem reduction

Omgjør til et annet/lignende problem, der allerede har algoritme/svaret.
F.eks: Minste felles multippel, sti-telling i graf, reduksjon av optimaliserings (min/max) problemer

Space and Time Tradeoffs (Plass-tid avveining)

- Input Enhancement / Preconditioning / Preprocessing

Går først gjennom (deler av) input slik at får viktig hjelpe-informasjon.
F.eks: Distribuert telling, streng-søk (Boyer-Moore / Knuth-Morris-Pratt)

- Prestructuring

Bruker ekstra plass for å få raskere/mer fleksibel adgang til dataene.
F.eks: Hashing, indeksering vha. B-trær

- Dynamic programming/planning

Prøver å minimere antall rekursive kall ved å lagre mellom-svar/-beregninger i en (eller flere) tabell(er).

- Bottom-up

Henter/kombinerer verdier i tabellen til delsvar på vei mot løsningen. F.eks: Rekursiv Fibonacci, Transitive Closure(Warshall) og alle-korteste-stier (Floyd), optimale binære søketrær

- Top-down (memory functions)

Lagrer unna kun relevante delsvar, etter hvert som de genereres, for å effektivisere ved å bruke dem senere.
F.eks: Dijkstra/A*, Knapsack Problem, Memory function

Greedy Technique

Bygger løsningen bit for bit ved stadig å gjenta samme enkle steg (som er gjennomførbart, lokalt optimalt og ugjenkallelig).
F.eks: MST (Prim/Kruskal), Union-Find, SP (Dijkstra), A*, Huffman (trær/koding)

Iterative Improvement

Starter med et "svar" og bygger videre mot en løsning vha. enkle steg.
F.eks: Nettverksflyt (Max-Flow, Ford-Fulkerson), Simplex metoden, Stable Marriage Problem

"Limitations of Algorithm Power" / State-space-tree teknikker

Ofte vrient algoritmisk, da antall muligheter stiger eksponentielt.
Bruker state-space-trær og rekursjon for å prøve mange/alle muligheter/ alternativer, og så evt. avbryte når *en* løsning er funnet.
(jfr. beslutningstrær, P/NP/NP-komplette problemer, nummeriske alg.)

- Backtracking

Optimalisering er underordnet.
F.eks: Dronningplassering (Eks_14), hestehopp (gml. oppg.11), labyrint (gml.H95-4), 4-farge land (gml.oppg.12), pinnespill (gml.eks. 11)

- Branch-and-bound

Traverseringsrekkefølgen er uvesentlig.
F.eks: Tildeling(er), Knapsack Problem, Traveling Salesman, NP problemer, ikke-lineære ligninger

NB: Ikke uvanlig at algoritmer er basert på kombinasjon av flere av teknikkene.

Bygd på (begge av Anany Levitin):

- *Introduction to The Design and Analysis of Algorithms - 2nd Edition, Pearson/Addison Wesley, 2007*
- *A New Road Map of Algorithm Design Techniques, Dr.Dobb's Journal #311 April 2000 page 23-28*