"Help, my problem is NP-complete"

Sometimes you may need to solve an NP-complete problem. Here are several approaches to try.

① Exhaustive Search

Try all solutions. (Best for small problem sizes)

```
Backtrack(X):
  If X is complete then
    return X
  bestX ← ∅
  For each expansion X' of X do
    bestX ← best(bestX, Backtrack(X'))
  return bestX
```

```
Algorithm BranchAndBound(X, b):
  If X is complete then
    return X
  If X cannot beat b then
    return ∅
  For each expansion X' of X do
    b ← best(b, BranchAndBound(X, b))
  return b
```

② SAT / ILP Solvers

Many problems can be efficiently transformed into SAT or ILP problems, which have very good solvers.

Even though no solver is sub-exponential in the worst case, many practical instances can be solved quickly.

③ Meta-Heuristics

In practice, near-optimal solutions are often good enough. These can be found much more quickly than the optimal solution, by using metaheuristics. A metaheuristic efficiently samples the solution space to find near-optimal solutions.

Algorithm SimulatedAnnealing(n):

```
X ← random solution
for i ← 0 to n do
  X' ← random neighbour of X
  if X' is better than X then
    X ← X'
  else with probability p(i)
    X ← X'
return X
```

Other examples of metaheuristics include: hill climbing, evolution strategies, genetic algorithms, and ant colony optimization.

(4) Approximation Algorithms

For some problems, it is possible to find solutions that are no more than a fixed constant factor away from the optimum.

Algorithm ApproxVertexCover(G):

```
V' ← ∅
while there is an uncovered edge e do
    add both endpoints of e to V'
return V'
```

The size of V' is at most twice the size of the smallest possible vertex cover.

Trade-offs:

	Quality	Speed	Simplicity	
Exhaustive	+	−	+	
SAT/ILP	+	−	+/o	
Metaheur.	−	+	+/o	
Approx.	o	+	−	

Final Exam

1. Short-answer questions
   - Solve recurrences
   - Give running times
   - Basic questions about P/NP

2. Divide and Conquer
   - Given code, prove correctness
   - Show termination
   - Analyze running time

3. Dynamic Programming
   - Prove optimal substructure
   - Give a recurrence
   - Give the algorithm (pseudo-code)
   - Analyze running time

4. Graph Problem
   - Give algorithm
   - Prove Correctness
   - Analyze running time

5. NP-completeness
   - Prove in NP
   - Reduce from known NP-complete problem