Reading problem statement

1. Read carefully, make sure no statements conflicts.
2. Rewrite important details /mark it .
3. If text is not small, Re-read the problem statement again. Make sure you have the full picture.
4. Extract constraints info.
    - Never ignore any constraints, especially unusual one (e.g. 2*(a+b) < c).
    - Sometimes constraints are not direct.
      Find triangle angle with 2 precision -> 360 * 10^2 //brute force
5. Trace Samples as long as they are traceable.
6. Think in Missed cases, smallest boundaries & largest boundaries & Especial cases.
7. Write it on paper (to test it in your idea).
8. **don't make assumptions.**
9. Revise carefully output section and output formats.

Investigate

Analysis

- Problem Constraints
- Problem domain(s)
- Search Space Size (size of **unique** solutions)
- Nature of target Function
- Output Bounding

Problem Abstraction

never to drop the original problem, sometimes your abstraction drop some important domain consideration

Problem Simplification

- Adhoc
- Problem to Sub-Problems: **you are JUST solving a sub-problem you invented.**
- Incrementally: think in a special problem/case, and then try to update the solution for general problem/case.
- Simplification by Assumptions
- think in general case

Problem Reverse

- Adhoc
- f(x) = y , search (x)
- Property Reverse
    - probability(x) = 1 - probability(!x)
    - Subset with x = total - (subset without x)
    - Min = total – max\ Min = max -ai
    - atmost vs exect

Problem Domain re-interpretation

Thinking

- Think on papers not in pc
- The Brute Force solution
    - Think in BF ITERATIVE and RECURSIVE.
    - Think about Search space & Search State
- Problem Domain re-interpretation
- Concretely, Symbolically, Pictorially
- Forward and backward
- Brainstorm - Rank – Approach
- Divide & Conquer problem

Observations Discovery

used pc if better (SIMPLE code only, don't take a lot of time)

Some popular properties:

- number of states
- Symmetry
- Inference
- Redundancy
- Independency
- IO re-representation
    - Graph
        - Think in bipartite graph
        - DAG: topological sort
        - Tree is a bipartite graph
- Canonical Form
- Cycle tricks

- Input Function Nature
- Tricks
    - Dp
        - Inference value
        - convert to table:
            - Applied data structures
            - Adhoc trick
        - Dp optimizations (D&C, Knuth, Convex Hull)
- Patterns
- Cyclic Function (repeated after X step)

Stuck?

1. Try to re-state the problem definition in 2 or 3 different ways and see if this helps.
2. Still Stuck? Do BF & Observe. Write the impractical BF solution, and try to find a pattern for answer / useful fact
3. Still Stuck? Iterate on your algorithms list, see if it could be the solution.
4. **Be careful in analysis for solutions. You may discard a correct solution**
    - **E.g., Calculated O(n^3) although it is O(n^2logn)**
    - **E.g., Calculated recursion depth wrongly.**

BIG order

- Check Exact # of operation
- Check for Reduced variables & Constrained input combinations
- Check duplicates and unique values
- **SQRT tricks**
- Preprocessing
    - Think in all kinds of precomputation and try to utilize any of them.
    - **Next array**
    - **Calc on machine a small temp array that help you in run time.**
- In query problems
    - **Solve them offline if that help**
- **Order of loops and data such that loops break so early as much as possible.**
- Reference of locality tricks
    - Order loops such that: loops don't pass over arrays is in depth. Watch out from Col-order accessing
    - Maybe duplicate some memory to switch some col order to row order.

Solution Verification

But before you verify, you should remember **(keep it simple)**! Could we find something simpler! verifying the solution requires:

1. Test cases Verification (Ones in statement, boundaries, yours)
2. Solution Order: Time & Memory
3. Full logic & intuitive Revision
4. Is valid (BS/TS)?
5. Correctness, at least good intuitive - Assumption's validations
6. In case recursive code, does depth fit?
7. In case (*+^) operations, Any overflow (intermediate & output)
8. In case Double /, **Is precision fine**? Using long double instead of double?
9. In case Double /, **can we not use it??**

Solution Implementation

If you don't have a full picture: back to paper and think again

After Implementation

1. Revise code order & logic. Make sure it matches what you intended
2. Challenge every block of code. **Never to read in a way that drop even ONE character**
3. Think how this block of code maybe fail.
4. Revise data types
5. Double comparison, precision of +- numbers [(int)(a +/- EPS)]
6. **Return statement in functions**

Test special cases. Testing the boundaries + revise SPECIAL CASES.
Failed? **Check Error Inspection List**

Error Inspection

General

- Do you read all input file?
- Initialization (between test cases)
- **TYPO, variable names**
- Conditions \function base case\return statement in function
- Overflow
- avoid double operations if possible
- Corner cases
- Arrays boundary

Wrong Answers

- Review constrains
- Review code again
- Re-read the problem statement
    - Tricky text description
- Geometry
    - Double precision
    - Are there duplicate points? Does it matter? Co-linearity?
    - Polygon: convex? concave?
- Graph
    - Connected or disconnected?
    - Directed or Undirected?
    - Self Loops?
    - Multiple edges
- Precision
    - Watchout -0.0
    - int x = (int)(a +/- EPS) depends on a > 0 | a < 0.

Time Limit Exceed

- May be bug and just infinite loop
- Can we precompute the results?
- Function calls may need reference variables.
- % is used extensively?
    - If mod is 2^p-1, use bitwise
- What is blocks of code that represent order? Do we just need to optimize it?
- Big Input file
    - Need scanf & printf?
    - Optimize code operations
    - Switch to arrays and char[]
- DP Problems
    - **Do you really need to clear each time?**
    - **The base case order is not O(1)**
    - Use effective base conditions
        - E.g. If you are sure Dp(0, M) is X, do not wait until Dp(0,0)
    - **Cyclic recurrence?**
- Backtracking
    - If you have different ways to do it, try to do what minimize stack depth

Run time error

- Make sure to have correct array size.
- Make sure no wrong indexing < 0 || x >= n
- In DP, check you access dimensions correctly
- **Stack overflow**
- /0, %0
- Using incorrect compare function (e.g. return that return (A, B) same answer as (B, A))

Algorithm's list

Data Structures	Dynamic Programming	Graphs
STL / ordered set	DP basics (0\1, ranges)	Basic Algorithms
bitmask / bitset	DP (bitmask , SOS (sum of submasks))	dfs , bfs
monoqueue	Speed-up: matrix power	Topological Sort
BIT	Speed-up: Convex Hull Optimization	Graph Edge Types
segment tree	Speed-up: Divide and Conquer Optimization	SCC Trijan
lazy	Speed-up: Knuth Optimization	Articulation Points and Bridges
dynamic	**Techniques**	MST (Kruskal, prim)
merge sort	two pointers	Shortest Path
persistent	State-Space Search	bfs
wavelet tree	Meet-in-the-middle	dijkstra
sparse table	Backtracking	Bellman-Ford's (SPFA)
SQRT Decomposition	Binary Search	floyd
MO	ternary search	2SAT
Treaps	divide and conquer	bellman difference constraints
Graph DS	**String Algorithms**	Flows
DSU	Trie	maximum bipartite matching
LCA	KMP/ Aho-Corasick	Max flow Algorithms
HLD	hashing	hungarian algorithm
Centroid Decomposition	suffix array/automaton	Min Cost Max Flow

Math
Combinatorics
Fibonacci Numbers
NCR , NPR
Catlan Numbers
Number Theory
Prime Numbers
Prime Factorization
Sieve (linear sieve)
matrix exponential
divisors
GCD and LCM
Diophantine
CRT
Totient
Moebius
Game Theory
mirror technique
Nim
Greandy number

Geometry
Basic Geometry
Points and Lines
Vectors
Line Segments
Triangles
Circles
Polygons
Convex Hull
Polygon Cut
Polygon Centroid
Point in Polygon O(log n)
Line Sweep