Google Code Jam Cheat Sheet

<http://blog.benoitblanchon.fr/> - v1.0

# Algorithm design

## Ask yourself before coding

* Is a suboptimal solution accepted, i.e. can I use a heuristic?
* If so, what input would fool my heuristic?
* Can I divide the problem into smaller ones?
* Can I build the answer from a base case, like N=1?
* Would it be easier if the data was sorted?

## Classic algorithmic techniques

1. Brute force
2. Divide and conquer
3. Base case and build
4. Dynamic programming

## Your implementation is too slow?

1. Can I save the intermediate result to avoid computing the same thing again and again?
2. Can I pre-compute some useful data?
3. Can I quickly eliminate a whole set of candidate solution (early fail)?

## Maximum problem size, depending on complexity

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Complexity** |  |  |  |  |  |  |  |  |
| **N max** |  |  |  |  |  |  |  |  |

## Data type limits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int | long | BigInteger | doubles | decimal |
|  |  |  | , 15 digits | , 28 digits |

# Mathematics

## Quadratic equation

|  |  |  |
| --- | --- | --- |
|  |  |  |

## Combinations or Sets (order is ignored)

|  |  |  |
| --- | --- | --- |
| *k* elements among *n* | Number of pairs | With repeated elements: |

## Permutations (order counts)

|  |  |
| --- | --- |
| *k* elements among *n* | Number of permutations |

## Probability

|  |  |  |
| --- | --- | --- |
| A and B | A or B | A knowing B (Bayes theorem) |

## Workaround the accuracy of floating point values

|  |  |  |
| --- | --- | --- |
|  |  | Don’t do strict comparisons.  Accept a small error (epsilon). |

## Prime numbers

|  |  |  |
| --- | --- | --- |
| function gcd(a, b)  while b ≠ 0  t := b  b := a mod b  a := t  return a |  | Coprime |

# Stuck?

1. Read the problem again, make sure you leverage every piece of information
2. Try to solve by hand
3. Watch carefully the results and look for a pattern
4. Still nothing? Keep 30 to 60 minutes for the brute force implementation

# Before you send your solution

1. Read the problem again, make sure there is no misunderstanding
2. Look at the statistics.   
   If some people fails (< 90% success), you should look for the pitfall.  
   If many people fails (< 70% success), you may consider skipping the problem.
3. Forget about the O(n!), it’s doomed to fail
4. Assert everything (input, output and intermediate values) if something goes wrong you’ll detect and fix the bug within the 4 minutes
5. Add samples to the input file: singular case (zero, negative…) and extreme cases
6. Remove all unused code and clean everything that stays in your program