Faster Evaluation of Subtraction Games

David Eppstein

9th International Conference on Fun With Algorithms
(FUN 2018)

La Maddalena, Italy, June 2018

Nim

Start with several piles of matches (or other objects)

Each turn: take any number of matches from one pile

Win by emptying the last pile

Featured in Last Year at Marienbad (1960)

Nim strategy

Aim to make bitwise xor of pile values become zero

If it is already zero, your opponent is winning

Subtract-a-square

Can only take a square number of objects per turn

Can be interesting even with only a single pile

Golomb (1966) credits its invention to Richard A. Epstein

Hot position

You want to move, because
you can win if you make the right move

Cold position

You don’t want to move, because
your opponent is already winning

Sieving algorithm for telling hot from cold

Mark all positions as cold

For $i = 0, 1, 2 \ldots n$:

  if i is still marked cold:

    Mark all $i + j^2$ as hot

Evaluates first n positions in time $O(c_n\sqrt{n})$

where $c_n = \#$ cold positions

Les cribleuses de blé [the grain sifters], Gustave Courbet, 1854

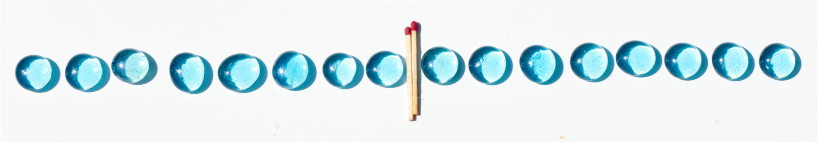
Divide-and-conquer for telling hot from cold

(Outside the recursion): Mark all positions as cold

Recursively evaluate the first half of the positions

Mark as hot all positions $i + j^2$ in the second half
such that i is a cold position in the first half

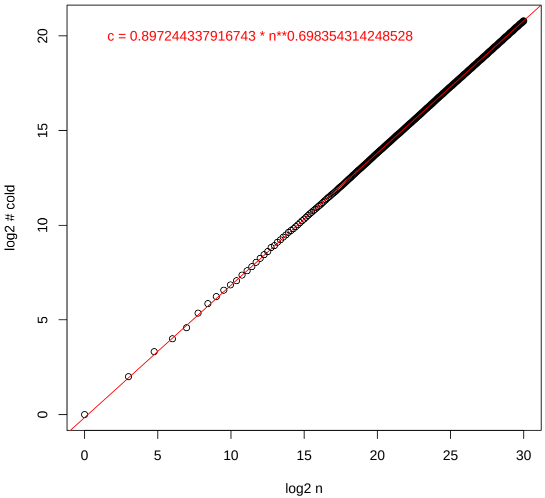
Recursively evaluate the second half of the positions



The middle "conquer" step is Boolean convolution, $O(n \log n)$ time

So the whole algorithm takes $O(n \log^2 n)$ time

Which algorithm is faster?



Experimentally, sieving takes $O(n^{1.2})$ time

Divide-and-conquer is faster in theory, but only for $n > 10^{18}$

What about multiple piles?

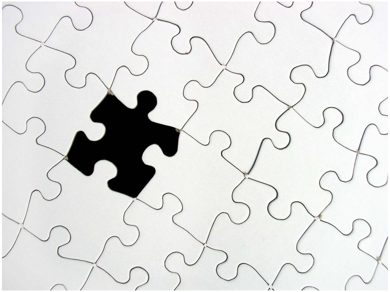
Sprague–Grundy theorem:
Every position is equivalent to a position in standard nim

Strategy: Move to make xor of nim-values become zero

Dynamic programming for nim-values

For each position $i = 0, 1, 2, \ldots n$:

- Look up nim-values of all positions $i - j^2$
- Value for i is the smallest value that is not in this set



File:Puzzle_black-white_missing.jpg on Wikimedia commons, by Willi Heidelbach

Time: $O(n^{3/2})$

Divide-and-conquer for nim-values

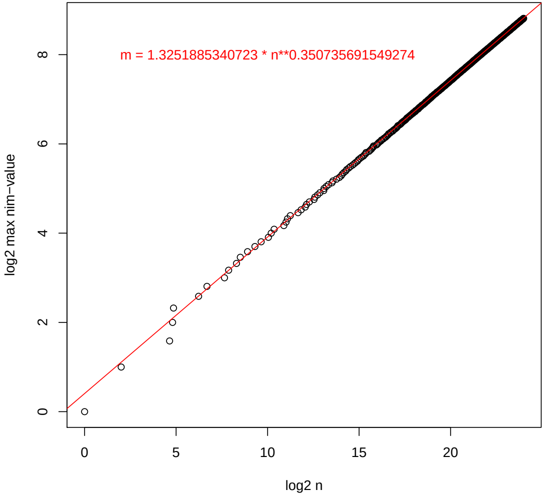
For each nim-value $v = 0, 1, 2, \ldots$:

- Mark positions with value $< v$ as hot, others as cold
- Apply the divide-and-conquer hot-cold algorithm
- Set the value of the remaining cold positions to v

Thermochromic mugs. File:Hot_Cold_mug.jpg on Wikimedia commons, by Damianosullivan.

Time: $O(mn \log^2 n)$
where $m =$ max nim-value encountered

Which algorithm is faster?



Experimentally, divide-and-conquer takes $O(n^{1.35} \log^2 n)$, versus $O(n^{1.5})$ for dynamic programming

Divide-and-conquer is faster in theory, but only for $n > 10^{26}$

Conclusions

New divide-and-conquer algorithms for subtract-a-square

Extends to any similar subtraction game

Improvement in theory but not in practice

Connections to deep results in number theory
Furstenberg–Sárközy theorem: $\#$ cold $= o(n)$

Time comparison is experimental rather than proven;
can we prove $n^{1/2+\epsilon} \leq \#$ of cold positions $\leq n^{1-\epsilon}$
or $n^{\epsilon} \leq$ max nim-value $\leq n^{1/2-\epsilon}$?