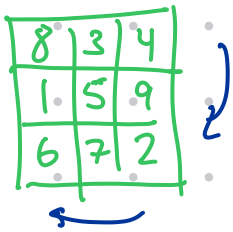
27 Feb. 2024

MAGIC SQUARES

- How easy to find them using simulated annealing?
- Which transition functions worked best?
- What questions do you have?

3×3: 9! total arrangements

8 are magic squares — all equivalent up to rotation and reflection



8 rotations/reflection

1 unique 3×3 magic square

$\frac{8}{9!} = \frac{1}{\frac{9!}{8}}$ = proportion of 3×3 arrangements that are magic

4×4: 16! total arrangements, or $\frac{16!}{8}$ up to rotation/reflection

880 unique 4×4 magic squares, up to rotation/reflection

$\frac{880}{\frac{16!}{8}}$ = proportion

5×5: 25! arrangements

275, 305, 224 unique 5×5 magic squares up to rotation/reflection

6×6: 36! arrangements

~ 17, 753, 889, 189, 701, 385, 264 unique magic squares up to rotation/reflection

RUNTIME DEMO: How do we quantify complexity of algorithms?

Example 1: $f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ Complexity: $O(n)$

↑ n addition operations

runtime is proportional to number of additions

observe: linear runtime

Example 2: $f(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} i \cdot j$ $O(n^2)$

↑ n^2 additions and multiplications

observe quadratic runtime: proportional to n^2

Example 3: selection sort $O(n^2)$

list of n items:

look for min item ↑ ↑ ↑ ↑ --- ↑] n comparison operations

put it in position 1

look for next smallest item, put it in position 2] $n-1$ comparisons

repeat until list is sorted $O(\frac{n^2}{2})$

total number of comparisons:

$n + (n-1) + (n-2) + \cdots + 2 + 1 \approx \frac{n^2}{2}$

Example 4: quicksort

Choose a pivot

Sort values so that smaller values come before the pivot, larger values come after

] n comparisons and swaps

apply quicksort to left and right sublists

Number of operations for quicksort

- $\log_2 n$ subdivisions
- for each subdivision level, roughly n comparisons

→ about $n \cdot \log_2(n)$ total operations $O(n \cdot \log n)$

ASYMPTOTIC NOTATION

Big-O notation: "big-oh"

If $f(x)$ and $g(x)$ are functions, $(\mathbb{R} \to \mathbb{R})$

then $f(x) = O(g(x))$ as $x \to \infty$ if there are some constants C and x_0 such that $|f(x)| < C \cdot g(x)$ for all $x > x_0$.

"$f(x)$ is big-oh of $g(x)$"

$f(x)$ doesn't grow faster than $g(x)$ for large x

Examples:

$x^2 = O(x^3)$

↳ $|x^2| < 1 \cdot x^3$ for $x > 1$

$5x^2 = O(x^2)$

↳ $|5x^2| < 6x^2$ for $x > 0$

$\log(x) = O(x)$

↳ $\log(x) < 1 \cdot x$ for $x >$ something

$x^2 + 5x + 1000 = O(x^2)$

↳ $x^2 + 5x + 1000 < C \cdot x^2$ for $x > x_0$

$x^2 + 5x + 1000 \neq O(x)$

rule of logs: $\log_b x = \frac{\log_c x}{\log_c b} = C \cdot \log_c(x)$

constant

$$\log_b(x) = O(\log_c(x))$$

Theorem: for sorting algorithms that involve comparison and swaps, complexity $O(n \cdot \log n)$ is optimal

Hierarchy of functions by growth order:

$x^{2}!$ — $n^2!$ number of square arrangements of $(1, 2, \ldots, n^2)$

exponential:
- $x!$
- 5^x
- e^x
- 2^x ← 2^n = number of subsets of n items

polynomial:
- x^3
- x^2
- x^1
- $x^{0.5}$
- $x^{0.1}$

log:
- $\log(x)$
- $\log(\log(x))$
- $\log(\log(\log x))$