

Spaghetti Sort

An algorithm every Italian and Italy lover and algorithms lover should know.

My implementation using bitwise operations... yep no arithmetic or comparisons.

GitHub inside

Non-trivial bitwise implementation inside

Running this on your machine causes the following:

- 1) Full stomach: Memory eager
- 2) Demanding digestion: CPU intense

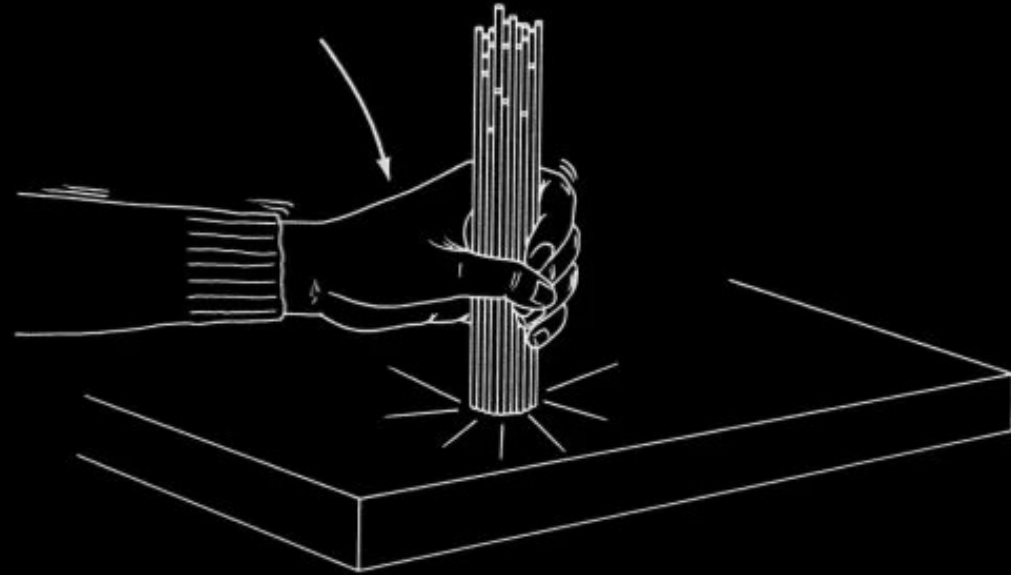


From Wikipedia:

Spaghetti sort is a linear-time, analog algorithm for sorting a sequence of items, introduced by A. K. Dewdney in his Scientific American column. **This is possible because the hand, the spaghetti rods and the table work as a fully parallel computing device.**

The sorting method is illustrated using uncooked rods of spaghetti:

- 1) For each number x in the list, obtain a rod of length x .
- 2) Once you have all your spaghetti rods, take them loosely in your fist and lower them to the table, so that they all stand upright, resting on the table surface.
- 3) Now, for each rod, lower your other hand from above until it meets with a rod—this one is clearly the longest, take it. Repeat until all rods have been removed.



My Misson

Freaky Implementation of Spaghetti Sort: Binary Spaghetti Sort

```
# Spaghetti sort implementation using binary operations. Because of its
# nature, it sorts N+ numbers {1, 2, 3, ...}
#
# Key points:
# 1) Mimic the spaghetti sort main steps
#   1 - Spaghetti creation
#   2 - Spaghetti alignments according to length
#   3 - Hand moving down to collect spaghetti by length
# 2) Don't use imports
# 3) Use as many bitwise operations as possible
# Note: Instead of one hand we will be using a mask of bits
```

Functions we need

```
def bitwiseRoundedLog2(x):
    # Calculates rounded log2
    k = -1
    while x > 0:
        x = x >> 1
        k += 1
    return k
```

Example

```
x = 6
log2(x) = 2.585
x >> 1 = 3
3 >> 1 = 1
1 >> 1 = 0 } 3
```

```
def bitwiseBooleanToInteger(bitlist):
    # Turns a list of bit into an integer
    v = 0
    for bit in bitlist:
        v = (v << 1) | bit # sets the right most bit
    return v
```

Example

```
bitlist = [1, 0, 1]
v = 0 << 1 | 1 = 1
v = 1 << 1 | 0 = 10 = 2
v = 10 << 1 | 1 = 101 = 5
```

Step 1

Take array column = [0, 0, 1, 0, 0] make it a number.
 $S = \text{bitwiseBooleanToInteger}(\text{Column}) = 4 = 100$
 $S = S \wedge \text{mask} = 100$
 $\text{mask} = S \mid \text{mask} = 100$

Processing bits in S 100

$K = \text{bitwiseRoundedLog2}(S) = 2$

Take value at index $\text{length} + (-k - 1)$ 2 \Rightarrow 5

Updating S, setting processed bit to zero

$S = S \wedge 1 \ll K = 0$

Step 2

Take array column = [1, 0, 1, 1, 0] make it a number.
 $S = \text{bitwiseBooleanToInteger}(\text{Column}) = 22 = 10110$
 $S = S \wedge \text{mask} = 10010$
 $\text{mask} = S \mid \text{mask} = 10110$

Processing bits in S 10010

$K = \text{bitwiseRoundedLog2}(S) = 4$

Take value at index $\text{length} + (-k - 1)$ 0 \Rightarrow 4

Updating S, setting processed bit to zero

$S = S \wedge 1 \ll K = 10$

Processing bits in S 10

$K = \text{bitwiseRoundedLog2}(S) = 1$

Take value at index $\text{length} + (-k - 1)$ 3 \Rightarrow 4

Updating S, setting processed bit to zero

$S = S \wedge 1 \ll K = 0$

Step 3

Take array column = [1, 1, 1, 1, 0] make it a number.
 $S = \text{bitwiseBooleanToInteger}(\text{Column}) = 30 = 11110$
 $S = S \wedge \text{mask} = 1000$
 $\text{mask} = S \mid \text{mask} = 11110$

Processing bits in S 1000

$K = \text{bitwiseRoundedLog2}(S) = 3$

Take value at index $\text{length} + (-k - 1)$ 1 \Rightarrow 3

Updating S, setting processed bit to zero

$S = S \wedge 1 \ll K = 0$

Step 4

Take array column = [1, 1, 1, 1, 1] make it a number.
 $S = \text{bitwiseBooleanToInteger}(\text{Column}) = 31 = 11111$
 $S = S \wedge \text{mask} = 1$
 $\text{mask} = S \mid \text{mask} = 11111$

Processing bits in S 1

$K = \text{bitwiseRoundedLog2}(S) = 0$

Take value at index $\text{length} + (-k - 1)$ 4 \Rightarrow 2

Updating S, setting processed bit to zero

$S = S \wedge 1 \ll K = 0$

END

sorted array [5, 4, 4, 3, 2]

```
def freakyBinarySpaghettiSort(values):  
  
    length, maximum, minimum, mask = len(values), max(values), min(values), 0  
    range_limit = maximum-minimum+1  
    sorted, spaghetti = [0]*length, [0]*length  
    spaghetti = [(1<<(n-minimum+1))-1<<(maximum-n) for n in values]  
  
    index = 0  
    for x in range(0, range_limit):  
        xor = bitwiseBooleanToInteger([(s >> x) & 1 for s in spaghetti])  
        xor = xor ^ mask  
        mask = xor | mask  
        while xor > 0:  
            k = bitwiseRoundedLog2(xor)  
            sorted[index] = values[-k-1]  
            bin_pos = 1 << k  
            xor = xor ^ bin_pos  
            index += 1  
    return sorted
```

Full implementation available at:

<https://github.com/Sinnefa/binary-spaghetti-sort-using-bitwise-operations>