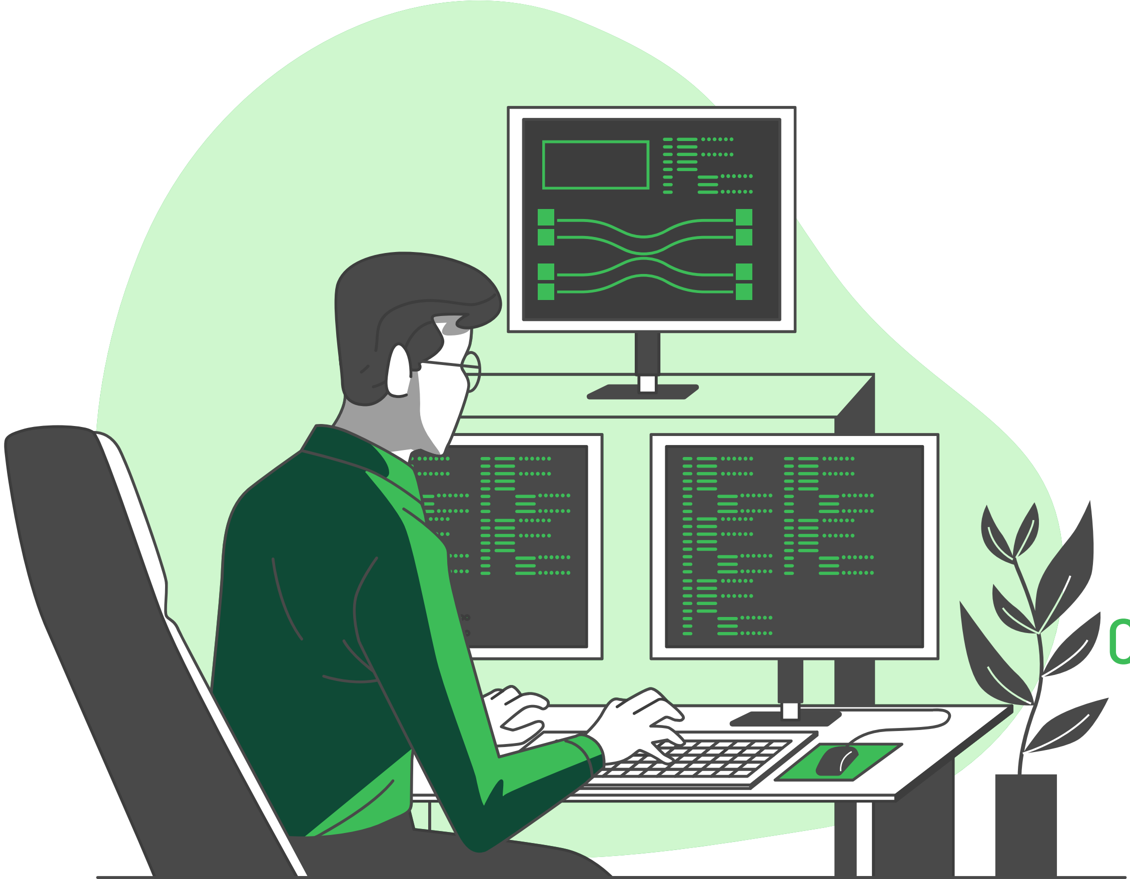


Runtime Expressions

CS 251 - Data Structures and Algorithms

Note:
Slides complement the discussion in class

Table of Contents

01

Definitions

Let’s formalize a few things

Algorithms

*"Informally, an **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**. An algorithm is thus a sequence of computational steps that transform the input into the output."*

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. Introduction to Algorithms (The MIT Press) (p. 5). The MIT Press. Kindle Edition.

Algorithms Are:

Effective

Return correct results

Precise

"No more, no less"

Finite

Run in a limited amount of time/steps

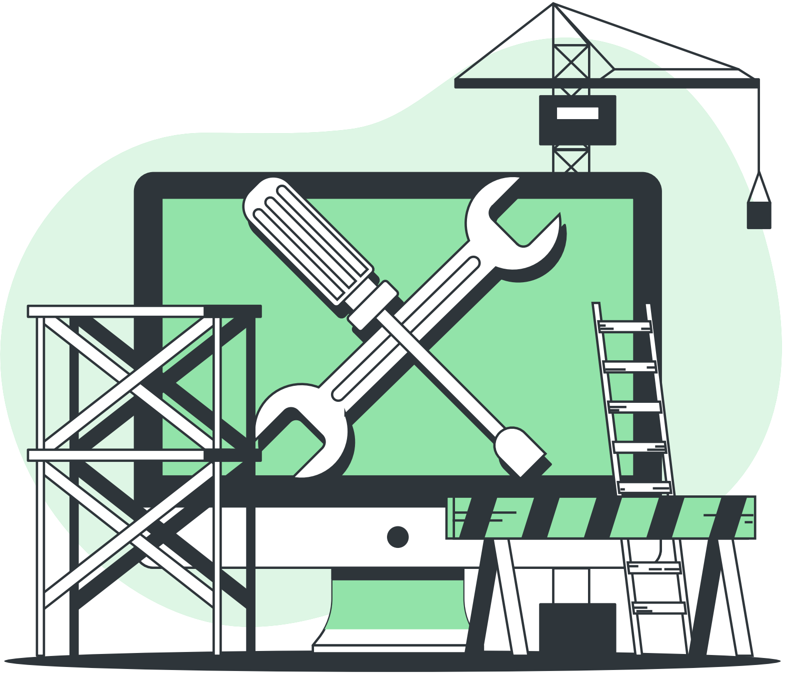
Should an algorithm be efficient?

```
algorithm isprime1(n:integer) → boolean

   if n <= 1 then
      return false
   end if

   for i from 2 to n-1 do
      if n mod i = 0 then
         return false
      end if
   end for

   return true

end algorithm
```

```
algorithm isprime2(n:integer) → boolean

   if n <= 1 then
      return false
   end if

   for i from 2 to sqrt(n) do
      if n mod i = 0 then
         return false
      end if
   end for

   return true

end algorithm
```

Both **isprime1** and **isprime2** are algorithms. Yet, **isprime2 is more efficient than isprime1** in terms of the number of steps required to reach an answer for the same input value.

Data Structures



*"A **data structure** is a collection of **data values**, the **relationships** among them, and the **functions or operations** that can be applied to the data. If any one of these three characteristics is missing or not stated precisely, the structure being examined does not qualify as a data structure."*

- Peter Wegner and Edwin D. Reilly. 2003. Data structures. Encyclopedia of Computer Science. John Wiley and Sons Ltd., GBR, 507–512.

A Few Data Structure Examples

Array type:
Bit array
Bit field
Bitboard
Bitmap
Circular buffer
Control table
Image
Dope vector
Dynamic array
Gap buffer
Hashed array tree
Lookup table
Matrix
Sorted array
Sparse matrix
...

Linked List type:
Doubly linked list
Array list
Linked list
Association list
Self-organizing list
Skip list
Unrolled linked list
VList
Conc-tree list
Xor linked list
Zipper
Doubly connected edge list
Difference list
Free list
...

Tree type:
AA tree
AVL tree
Binary search tree
Binary tree
Cartesian tree
Conc-tree list
Order statistic tree
Pagoda
Red–black tree
Rope
Scapegoat tree
Splay tree
T-tree
Tango tree
Treap
...

Graph type:
Adjacency list
Adjacency matrix
Graph-structured stack
Scene graph
Decision tree
Binary decision diagram
And-inverter graph
Directed graph
Directed acyclic graph
Multigraph
Hypergraph
...

NIST
National Institute of Standards and Technology

Dictionary of Algorithms and Data Structures

This web site is hosted by the Software and Systems Division, Information Technology Laboratory, NIST. Development of this dictionary started in 1998 under the editorship of Paul E. Black.

This is a dictionary of algorithms, algorithmic techniques, data structures, archetypal problems, and related definitions. Algorithms include common functions, such as Ackermann's function. Problems include traveling salesman and Byzantine generals. Some entries have links to implementations and more information. Index pages list entries by area and by type. The two-level index has a total download 1/20 as big as this page.

Don't use this site to cheat. Teachers, contact us if we can help.

Currently we do not include algorithms particular to business data processing, communications, operating systems or distributed algorithms, programming languages, AI, graphics, or numerical analysis: it is tough enough covering "general" algorithms and data structures. If you have suggestions, corrections, or comments, please get in touch with Paul Black.

Some terms with a leading variable, such as *n*-way, *m*-dimensional, or *p*-branching, are under *k*-. You may find useful entries in A Glossary of Computer Oriented Abbreviations and Acronyms.

To look up words or phrases, enter them in the box, then click the button.

Google SafeSearch [Google Search] Web DADS

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

α: see inverse Ackermann function
ω
Ω
ρ-approximation algorithm
~
Θ

A

absolute performance guarantee
abstract data type
(a,b)-tree
accepting state
Ackermann's function
active data structure
acyclic directed graph: see directed acyclic graph
acyclic graph
adaptive heap sort
adaptive Huffman coding
adaptive k-d tree
adaptive sort
address-calculation sort
adjacency-list representation
adjacency-matrix representation

M

Malhotra-Kumar-Maheshwari blocking flow
Manhattan distance
many-one reduction
map: see dictionary
Markov chain
Marlena
marriage problem: see assignment problem
Master theorem
matched edge
matched vertex
matching
matrix
matrix-chain multiplication problem
matrix multiplication
max-heap property
maximal independent set
maximally connected component
Maximal Shift
maximum bipartite matching: see bipartite matching
maximum-flow problem
MBB: see minimum bounding box

https://xlinux.nist.gov/dads/

Traditional Data Structure Functions

Read

at (key) ⟶ item
find (key) ⟶ item
get (key) ⟶ item
search (key) ⟶ item
lookup (key) ⟶ item
exists (key) ⟶ boolean

Write

add (key [,item])
insert (key [,item])
edit (key [,item])
update (key [,item])
delete (key)
swap (key1, key2)

Could we design algorithms without data structures?

Could we design data structures without algorithms?

We Analyze DS and Algos Because:

01 Predict Performance

"How much time/space does it require?"

02 Comparison

"Which one is [objectively] better?"

03 Provide Guarantees

"Will it always work?"

04 Theoretical Basis

"What can we built on top of them?"

02

Runtime Expressions

Counting instructions in terms of the input size

Analysis Challenges

Time

Why is my program so slow?

Space

Why does my program run out of memory?

Idea

$$T(n) = \sum_i c_i n_i$$

Where i is an operation, c_i is the cost of operation i, $n = |x|$ is the input size, x is the input

Total running time: sum of cost x frequency for each operation i
Need to determine the set of operations
Costs depends on machine/compiler
Frequency depends on algorithm and input data
…

Runtime Expression $T(n)$

A function that "counts" the number of steps or instructions an algorithm runs in terms of the input size.

Example: $T(n)$ for the cost of printing "Peekabo" n times:

```
for i from 1 to n do
   print("Peekaboo")
end for
```

Let c_p be the cost of a print statement.

$$T(n) = \sum_{i=1}^{n} c_p = c_p n$$

Note: You should remember this topic from CS 182.

A single pseudocode snippet could have multiple runtime expressions. It depends on which operation/task you focus on. Here are two $T(n)$ for the pseudocode snippet below.

```
for i from 0 to n-1 do
   print(i)
end for
```

1. Closed-form expression for the **print calls**. Let c_p be the cost of a print call:

$$T(n) = \sum_{i=0}^{n-1} c_p = c_p(n - 1 + 1) = c_p n$$

2. Closed-form expression for the **variable declarations**. Let c_d be the cost of a variable declaration:

$$T(n) = c_d$$

Q: When does it declare a variable?
A: When defining the for loop (it needs variable `i`)

Here is another pseudocode snippet:

```
sum ← 0
for i from 0 to n do
   sum ← sum + i
end for
```

Closed-form expression for the **number of variable assignments**. Let c_a be the cost of an assignment:

$$T(n) = c_a + \sum_{i=0}^{n} 2c_a + c_a = 2c_a + 2c_a(n+1) = 2c_a n + 4c_a$$

`sum ← 0`

Assigning new values to `i` and `sum` for all iterations of the for loop

`i ← n+1`
(exiting the for loop)

```
sum ← 0
for i from 0 to n do
   sum ← sum + i
end for
```

<u>Closed-form expression for the compares</u>. Let c_c be the cost of a compare:

$$T(n) = \sum_{i=0}^{n+1} c_c = c_c(n + 1 + 1) = c_c n + 2c_c$$

<u>Closed-form expression for the value of sum in terms of n</u> after running the code snippet:

$$T(n) = \sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

Note that variable increments could be different than 1:

```
for i from 0 to n by 2 do
   print(i)
end for
```

How do we get the **closed-form expression for the print calls**?

Strategy: Since i increases by 2, express its sequence in terms of increments by 1:

$$i = 0, 2, 4, 6, 8, \ldots, n = 2 \cdot 0, 2 \cdot 1, 2 \cdot 2, 2 \cdot 3, 2 \cdot 4, \ldots, 2 \cdot \frac{n}{2}$$

$$T(n) = \sum_{k=0}^{\frac{n}{2}} c_p = c_p \sum_{k=0}^{\frac{n}{2}} 1 = c_p \left(\frac{n}{2} + 1\right) = \frac{1}{2} c_p n + c_p$$

Note that variable increments could be different than 1:

```
for i from 1 to n by multiplying by 2 do
   print(i)
end for
```

How do we get the **closed-form expression for the print calls**?

Strategy: Express the sequence in terms of increments by 1:

First iteration: $i = 1 = 2^0$
Second iteration: $i = 2 = 2^1$
Third iteration: $i = 4 = 2^2$
Fourth iteration: $i = 8 = 2^3$
Fifth iteration: $i = 16 = 2^4$
...
Last iteration: $i = 2^k \leq n \Rightarrow k \leq \log_2(n)$

$$T(n) = \sum_{k=0}^{\log_2(n)} c_p = c_p(\log_2(n) + 1) = c_p \log_2(n) + c_p$$

Note that loop structures could be nested:

```
for i from 0 to n do
   for j from 1 to n-1 do
      print(i + j)
   end for
end for
```

How do we get the **closed-form expression for the print calls**?

$$T(n) = \sum_{i=0}^{n} \sum_{j=1}^{n-1} c_p = \sum_{i=0}^{n} c_p(n-1) = c_p(n-1)(n+1) = c_p n^2 - c_p$$

Note that loop structures could be nested:

```
for i from 0 to n by 2 do
   for j from 1 to n-1 do
      foo(i + j, n)
   end for
end for
```

Let n be the cost of a single foo call. The **closed-form expression for the foo calls** using the pseudocode snippet above is:

$$T(n) = \sum_{k=0}^{\frac{n}{2}} \sum_{j=1}^{n-1} n = \sum_{k=0}^{\frac{n}{2}} n(n-1) = n(n-1)\left(\frac{n}{2} + 1\right) = \frac{1}{2}n^3 + \frac{1}{2}n^2 - n$$

Try it on Wolfram Alpha: sum(sum n for j from 1 to n-1) for k from 0 to n/2

Note that loop structures could be nested:

```
for i from 0 to n do
   for j from i to n-1 do
      foo(i + j, n)
   end for
end for
```

Let n be the cost of a single foo call. The **closed-form expression for the foo calls** using the pseudocode snippet above is:

$$T(n) = \sum_{i=0}^{n} \sum_{j=i}^{n-1} n = \sum_{i=0}^{n} \left(\sum_{j=0}^{n-1} n - \sum_{j=0}^{i-1} n \right) = \sum_{i=0}^{n} (n(n-1+1) - n(i-1+1))$$

$$= \sum_{i=0}^{n} n^2 - in = \sum_{i=0}^{n} n^2 - \sum_{i=0}^{n} in = n^2(n+1) - n\frac{n(n+1)}{2} = n^2(n+1) - \frac{1}{2}n^2(n+1)$$

$$= \frac{1}{2}n^2(n+1) = \frac{1}{2}n^3 + \frac{1}{2}n^2$$

Try it on Wolfram Alpha: sum(sum n for j from i to n-1) for i from 0 to n

Note that loop structures could be nested:

```
for i from 0 to n by 2 do
   for j from i to n-1 do
      foo(i + j, n)
   end for
end for
```

Let n be the cost of a single foo call. The closed-form expression for the foo calls using the pseudocode snippet above is:

$$T(n) = \sum_{k=0}^{\frac{n}{2}} \sum_{j=2k}^{n-1} n = \sum_{k=0}^{\frac{n}{2}} \left(\sum_{j=0}^{n-1} n - \sum_{j=0}^{2k-1} n \right) = \sum_{i=0}^{\frac{n}{2}} n^2 - 2kn = \sum_{i=0}^{\frac{n}{2}} n^2 - \sum_{i=0}^{\frac{n}{2}} 2kn$$

$$= n^2 \left(\frac{n}{2} + 1\right) - 2n \frac{\frac{n}{2}\left(\frac{n}{2} + 1\right)}{2} = \frac{n^3}{2} + n^2 - \frac{n^3}{4} - \frac{n^2}{2} = \frac{1}{4} n^3 + \frac{1}{2} n^2$$

Try it on Wolfram Alpha: sum(sum n for j from 2k to n-1) for k from 0 to n/2

Note that, when working with logarithms, it is fine to give approximate results by skipping the base. We can do this thanks to the properties of logarithms (i.e., change of base):

```
for i from 1 to n by multiplying by 3 do
   foo(i + j, n)
end for
```

Let $\log(n)$ be the cost of a single foo call. The **closed-form expression for the foo calls** using the pseudocode snippet above is:

$$T(n) = \sum_{k=0}^{\log_3(n)} \log(n) = \log(n)\,(\log_3(n) + 1) \approx \log(n)\log(n) + \log(n)$$

$$= \log^2(n) + \log(n)$$

Note that, when working with logarithms, it is fine to give approximate results by skipping the base. We can do this thanks to the properties of logarithms (i.e., change of base):

```
for i from 1 to n by i do
   foo(i + j, n)
end for
```

Let n be the cost of a single foo call. The **closed-form expression for the foo calls** using the pseudocode snippet above is:

$$T(n) = \sum_{k=0}^{\log_2(n)} n = n(\log_2(n) + 1) = n\log_2(n) + n \approx n\log(n) + n$$

```
algorithm isprime1(n:integer) → boolean

   if n <= 1 then
      return false
   end if

   for i from 2 to n-1 do
      if n mod i = 0 then
         return false
      end if
   end for

   return true

end algorithm
```

Closed-form expression for the **divisions** made by isprime1 for a prime number n:

$$T(n) = \sum_{i=2}^{n-1} 1 = \sum_{i=1}^{n-1} 1 - 1 = (n-1) - 1$$
$$= n - 1 - 1 = n - 2$$

```
algorithm isprime2(n:integer) → boolean

   if n <= 1 then
      return false
   end if

   for i from 2 to sqrt(n) do
      if n mod i = 0 then
         return false
      end if
   end for

   return true

end algorithm
```

Closed-form expression for the **divisions** made by isprime2 for a prime number n:

$$T(n) = \sum_{i=2}^{\sqrt{n}} 1 = \sum_{i=1}^{\sqrt{n}} 1 - 1 = \sqrt{n} - 1$$

```
algorithm ThreeSum(A:array) → integer
   let n be the length of A
   count ← 0

   for i from 0 to n-1 do
      for j from i+1 to n-1 do
         for k from j+1 to n-1 do
            if A[i] + A[j] + A[k] = 0 then
               count ← count + 1
            end if
         end for
      end for
   end for

   return count
end algorithm
```

Closed-form expression for the **array accesses** made by ThreeSum:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n-1} 3$$

...after lots of steps...

$$T(n) = \frac{1}{2}n^3 - \frac{3}{2}n^2 + n$$

Type in Wolfram Alpha: sum (sum (sum 3, k=j+1 to n-1), j=i+1 to n-1), i=0 to n-1

Example: Selection Sort

```
algorithm SelectionSort(A:array) → array
   let n be the length of A

   for i from 0 to n-2 do
      idx ← i
      for j from i + 1 to n-1 do
         if A[j] < A[idx] then
            idx ← j
         end if
      end for
      if i ≠ idx then
         swap(A, i, idx)
      end if
   end for

   return A
end algorithm
```

Algorithm: select the smallest item to put into the current slot.

Let A be an array of length n storing unique items.

Q: What if A is already sorted?
#compares: $T(n) = \frac{1}{2}n^2 - \frac{1}{2}n$
#swaps: $T(n) = 0$

Q: What if A is sorted in descending order?
#compares: $T(n) = \frac{1}{2}n^2 - \frac{1}{2}n$
#swaps: $T(n) = \left\lfloor \frac{1}{2}n \right\rfloor$

Think about an array A for which the number of swaps is larger than $\left\lfloor \frac{1}{2}n \right\rfloor$.

Example: Insertion Sort

```
algorithm InsertionSort(A:array) → array
   let n be the length of A

   for i from 1 to n-1 do
      j ← i - 1
      while j >= 0 and A[j] > A[j + 1] do
         swap(A, j, j + 1)
         j ← j - 1
      end while
   end for

   return A
end algorithm
```

Algorithm: Insert the current item into the proper slot in the sorted part of the array (left).

Let A be an array of length n storing unique items.

Q: What if A is <u>already sorted</u>?
#compares: $T(n) = n - 1$
#swaps: $T(n) = 0$

Q: What if A is <u>sorted in descending order</u>?
#compares: $T(n) = \frac{1}{2}n^2 - \frac{1}{2}n$
#swaps: $T(n) = \frac{1}{2}n^2 - \frac{1}{2}n$

Example: Matrix Multiplication

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{pmatrix}$$

#multiplications = 8, #additions = 4

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}\begin{pmatrix} j & k & l \\ m & n & o \\ p & q & r \end{pmatrix} = \begin{pmatrix} aj + bm + cp & ak + bn + cq & al + bo + cr \\ dj + em + fp & dk + en + fq & dl + eo + fr \\ gj + hm + ip & gk + hn + iq & gl + ho + ir \end{pmatrix}$$

#multiplications = 27, #additions = 18

```
algorithm mult(A:matrix, B:matrix) → matrix

   assume A and B are of dimension nxn
   let C be a matrix of dimension nxn

   for i from 0 to n-1 do
      for j from 0 to n-1 do

         C[i][j] ← 0

         for k from 0 to n-1 do
            C[i][j] ← C[i][j] + A[i][k] * B[k][j]
         end for

      end for
   end for

   return C

end algorithm
```

Closed-form expression for the additions and multiplications made by mult. Keep track of these numbers by using constants a and m, respectively.

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1}(m+a)$$
$$= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(m+a)n = \sum_{i=0}^{n-1}(m+a)n^2$$
$$= (m+a)n^3 = mn^3 + an^3$$

More additions than expected! ☹

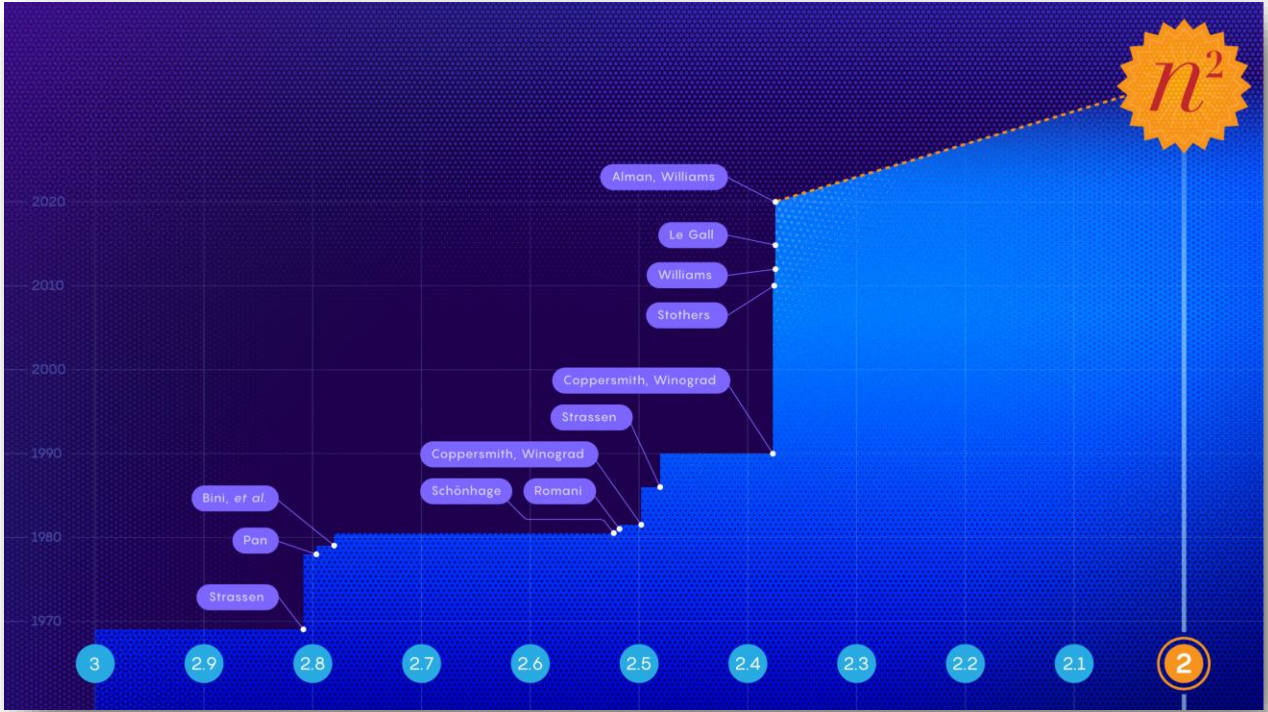
```
algorithm mult(A:matrix, B:matrix) → matrix

   assume A and B are of dimension nxn
   let C be a matrix of dimension nxn

   for i from 0 to n-1 do
      for j from 0 to n-1 do

         C[i][j] ← A[i][0] * B[0][j]

         for k from 1 to n-1 do
            C[i][j] ← C[i][j] + A[i][k] * B[k][j]
         end for

      end for
   end for

   return C

end algorithm
```

<u>Closed-form expression for the additions and multiplications</u> made by mult. Keep track of these numbers by using constants a and m, respectively.

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left(m + \sum_{k=1}^{n-1}(m+a)\right) \\
&= \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\big(m + (m+a)(n-1)\big) \\
&= \sum_{i=0}^{n-1}\big(m + (m+a)(n-1)\big)n \\
&= \big(m + (m+a)(n-1)\big)n^2 \\
&= mn^2 + mn^3 - mn^2 + an^3 - an^2 \\
&= mn^3 + an^2(n-1)
\end{aligned}
$$

The number of additions we expected! ☺
Still a cubic number of multiplications ☹

Matrix Multiplication Algorithm Goal



https://www.quantamagazine.org/mathematicians-inch-closer-to-matrix-multiplication-goal-20210323/

That's Enough Counting

Do you have any questions?