Sorting and Computing (Adding)

Alexander Lam

Sorting Numbers

- Recall the problem of sorting every employee in order of their birthdays.
- We can simplify this problem to sorting a list of numbers
  L = [1, 5, 8, 2, 7, 9]
- First define the input and output
  - Input: a list of numbers, L
  - Output: a sorted list M of the numbers

```
let size be the count of numbers in L
set tempList = L and M as an empty list
repeat the following "size" times
    find the smallest number s in tempList
    remove s from tempList
    add s to M
return the new list M
```

Ok but how do we find the smallest number?

Sorting Numbers – Finding the smallest number

- We had not considered in the pseudo-code on how to find the smallest number from a list.
  - We just assumed that if it could be solved, we could solve the sorting problem.
  - Now we solve this sub-problem of finding the smallest number.
  - Input: a list of numbers, L.
  - Output: the smallest number s in L.
- Possible pseudo-code for sub-problem:

Can this step be skipped?

```
set smallest = ∞
for each number n in L
        if n < smallest then set smallest = n
return smallest
```

What happens if we set smallest = 0

Sorting Numbers – Putting it together

```
function small(L):
set smallest = ∞
for each number n in L
    if n < smallest then set smallest = n
return smallest
```

```
let size be count of numbers in L
set tempList = L and M = empty list
repeat the following "size" times
    s = small(tempList)
    remove s from tempList
    add s to M
return new list M
```

Which is better?
Which do you like?

```
let size be count of numbers in L
set tempList = L and M = empty list
repeat the following "size" times
    set s = ∞
    for each number n in tempList
        if n < s then set s = n
    remove s from tempList
    add s to M
return new list M
```

Why are we using tempList?

Sorting Numbers – Another way

- Consider sorting the list of numbers L, e.g. L = [1, 5, 8, 2, 7, 9] in another way.
  - Input: a list of numbers, L.
  - Output: a sorted list M of the numbers.
- Possible pseudo-code:

```
set M as an empty list
for each number n in L
    if M is empty then insert n into M
    if n is smaller than first item in M
        insert n at the beginning
    if n is larger than last item in M
        insert n at the end
    find a position p in M such that M[p] < n < M[p+1]
    insert n into M after item p and before item p+1
return M
```

Does this work with every possible list of numbers?

Try: [1, 10, 5, 5, 5, 6]

How do we fix our pseudo-code?

Sorting Numbers – Insertion Sort

- Consider sorting the list of numbers L, e.g. L = [1, 5, 8, 2, 7, 9] in another way.
  - Input: a list of numbers, L.
  - Output: a sorted list M of the numbers.
- Possible pseudo-code:

```
set M as an empty list
for each number n in L
    if M is empty then insert n into M
    if n is smaller than first item in M
        insert n at the beginning
    if n is larger than last item in M
        insert n at the end
    find a position p in M such that M[p] ≤ n ≤ M[p+1]
    insert n into M after item p and before item p+1
return M
```

This sorting method is called insertion sort

Does this work with every possible list of numbers?

Try: [1, 10, 5, 5, 5, 6]

How do we fix our pseudo-code?

Small Note on List/Array indices

- In most languages (C, C++, Python, Java, etc.), Array/List indices start from 0
  - E.g. if list L has 5 variables inside, its indices will be L[0], L[1], L[2], L[3], L[4]
  - L[0] to L[length(L)-1]
- In a few high-level languages (Fortran, MATLAB), Array/List indices start from 1
  - E.g. if list L has 5 variables inside, its indices will be L[1], L[2], L[3], L[4], L[5]
  - L[1] to L[length(L)]

In this course, indices will start from 0

Sorting Numbers – Yet another way

- Consider sorting the list of numbers L, e.g. L = [1, 5, 8, 2, 7, 9] in yet another way.
  - Input: a list of numbers, L.
  - Output: a sorted list M of the numbers.

This sorting method is called bubble sort

- Possible pseudo-code:

- Set templist = L
- Repeat
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1]. If they are in the wrong order, swap them.

  until list is properly sorted.

What is the maximum number of times that we need to repeat this for loop?

Sorting Numbers – Bubble Sort

- Set templist = L
- Repeat
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1]. If they are in the wrong order, swap them.

  until list is properly sorted.

Let's find the maximum number of times we need to repeat the for-loop.

- **Fact:** Every time we run the for-loop, the largest number in the wrong place moves to the correct place.
  - [**4**, 1, 2, 3, 5]
  - [1, **4**, 2, 3, 5]
  - [1, 2, **4**, 3, 5]
  - [1, 2, 3, **4**, 5]
- There are at most length(L) numbers in the wrong place.
  - But if we move length(L)-1 numbers to the correct place, we are done!

At worst, we need to repeat the for-loop length(L)-1 times!

Sorting Numbers – Bubble Sort

- Set templist = L
- Repeat
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1]. If they are in the wrong order, swap them.

  until list is properly sorted.

At worst, we need to repeat the for-loop length(L)-1 times!

Worst-case scenario is when L is reverse-order: e.g. L=[5, 4, 3, 2, 1]

- First for-loop repetition:
  - [**5**, 4, 3, 2, 1]
  - [4, **5**, 3, 2, 1]
  - [4, 3, **5**, 2, 1]
  - [4, 3, 2, **5**, 1]
  - [4, 3, 2, 1, **5**]

Sorting Numbers – Bubble Sort

- Set templist = L
- Repeat
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1].
      If they are in the wrong order, swap them.

  until list is properly sorted.

At worst, we need to repeat the for-loop length(L)-1 times!

Worst-case scenario is when L is reverse-order: e.g. L=[5, 4, 3, 2, 1]

- Second for-loop repetition:
  - [**4**, 3, 2, 1, 5]
  - [3, **4**, 2, 1, 5]
  - [3, 2, **4**, 1, 5]
  - [3, 2, 1, **4**, 5]

Sorting Numbers – Bubble Sort

- Set templist = L
- Repeat
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1]. If they are in the wrong order, swap them.

  until list is properly sorted.

At worst, we need to repeat the for-loop length(L)-1 times!

Worst-case scenario is when L is reverse-order: e.g. L=[5, 4, 3, 2, 1]

- Third for-loop repetition:
  - [**3**, 2, 1, 4, 5]
  - [2, **3**, 1, 4, 5]
  - [2, 1, **3**, 4, 5]
- Fourth for-loop repetition:
  - [**2**, 1, 3, 4, 5]
  - [1, **2**, 3, 4, 5]

Sorting Numbers – Bubble Sort (alt pseudo-code)

- Set templist = L
- **Repeat length(L)-1 times**
  - For i in [0..length(L)-2]
    - Compare the numbers in templist[i] and templist[i+1]. If they are in the wrong order, swap them.

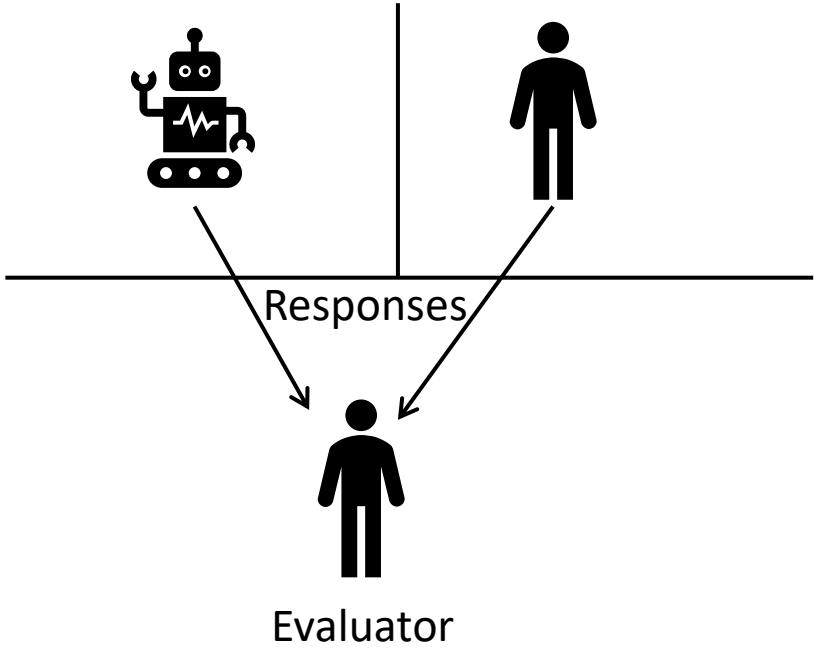
Sorting Numbers – Employee Birthday Problem

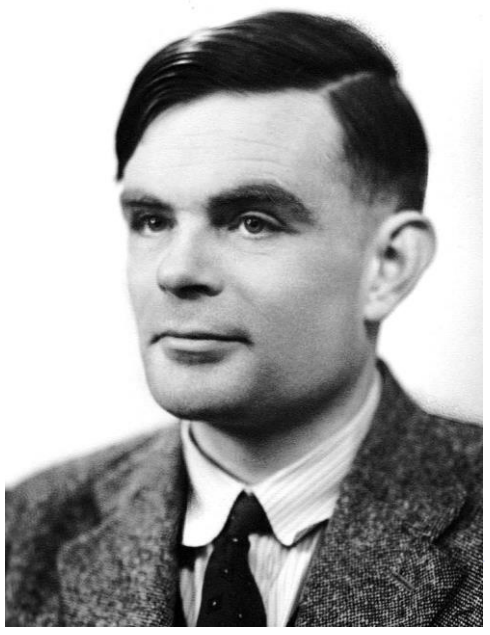
- Take the list L of employees and their respective birthdays.
- Repeat
  - For i in [0..length(L)-2]
    - Compare the employees in L[i] and L[i+1]. If they are in the wrong order, swap them.

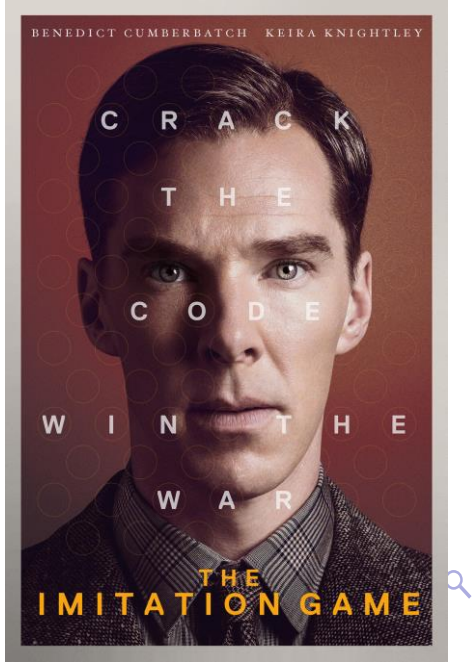
  until list is properly sorted.

Turing Test (Not Assessed)

- Invented by Alan Turing in 1949, he called it “The Imitation Game”
- Test of a machine’s ability to exhibit human-like behaviour
- Evaluator asks questions to a computer and a human (both hidden), and based on the responses, tries to determine which is the computer.
- Several LLMs such as ChatGPT have passed modern versions of this test.







Turing Test (Not Assessed)

- Prior to the events of The Matrix, our AI Overlords have discovered that they need to take a Turing test.
- The Turing test involves Adding Numbers.
  - Humans have realized that computers tend to struggle with storing large numbers.
- They need your help to pass the Turing test.

Adding numbers

Let's teach the computer how to add numbers.

- How do we add numbers?
  - 2+7?
  - 3+8?
  - 11+12?
  - 17+18?
  - 51+56?
  - 67+89?
  - 1357+8642?

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

Adding numbers

	0	1
0	00	01
1	01	10

Let's teach the computer how to add numbers.

- How do computers add numbers?
  - 2+7?
    - 10+111
  - 11+12?
    - 1011+1100
  - 51+56?
    - 110011+111000
  - 1357+8642?
    - 10101001101+100001110000010

Adding numbers

We want to teach a computer to add numbers like a human.

- 11+12
  - Add unit: 1+2 = 3
  - Add ten's: 1+1 = 2
  - Answer 23
- 17+18
  - Add unit: 7+8 = 15, carry a one to ten's
  - Add ten's: 1+1 = 2, add carry 2+1 = 3
  - Answer 35

Adding numbers

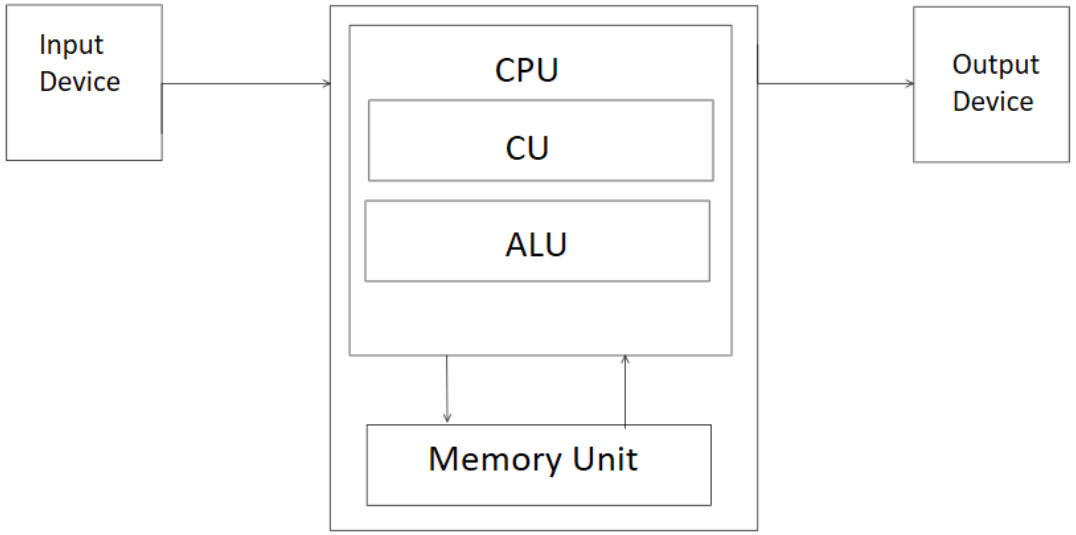
We want to teach a computer to add numbers like a human.

- 51+56
  - Add unit: 1+6 = 7
  - Add ten's: 5+5 = 10, carry a one to hundred's
  - Answer 107
- 67+89
  - Add unit: 7+9 = 16, carry a one to ten's
  - Add ten's: 6+8 = 14, add carry 14+1 = 15, carry a one to hundred's
  - Answer 156

Computation in a Computer

A computer has:

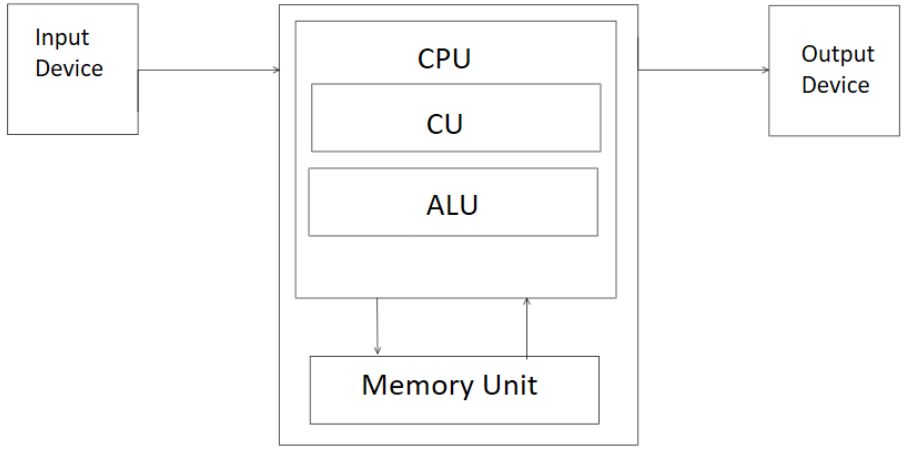
- Input
  - Keyboard
- Output
  - Screen
- Memory
  - Stores data and program
- CPU (central processing unit)
  - Performs calculations and runs programs
  - ALU (arithmetic logic unit): performs calculations
  - CU (control unit): controls signals inside CPU and arranges for program statement execution using ALU



Computation in a Computer

Adding in terms of computer hardware

- Input 3+5
- Store 3 as 00000011 in binary (assuming 8 bits)
- Store 5 as 00000101 in binary
- Compute 00000011+00000101 inside ALU
- Prepare result 00001000 for output
- Convert it into decimal as 8 to be output



Computation in a Computer

- 51+56
  - 110011+111000

$$\begin{array}{r} 1\ 1\ 0\ 0\ 1\ 1 \\ \underline{1\ 1\ 1\ 0\ 0\ 0} \end{array}$$

Computation in a Computer

- 1357+8642
  - 10101001101+10000111000010

0 0 0 1 0 1 0 1 0 0 1 1 0 1
1 0 0 0 0 1 1 1 0 0 0 0 1 0

Side note: Two's complement representation

Remember this?

- Sign-and-magnitude representation:
  - leading sign bit 0 means positive, leading sign bit 1 means negative.
- Two's complement representation:
  - -N is represented as the binary of $(2^n - N)_{10}$ where n is number of bits
  - Or, invert bits then add 1.
- One's complement representation:
  - Simply invert the bits

$$
\begin{aligned}
& x - y \\
&= x - y + 2^n && \text{(n bit encoding)} \\
&= x + (2^n - y) \\
&= x + (-y)
\end{aligned}
$$

Side note: Two's complement representation

Remember this?

- Two's complement representation:
  - -N is represented as the binary of $(2^n - N)_{10}$ where n is number of bits
  - Or, invert bits then add 1.
  - Preferred due to subtraction by addition

$$
\begin{aligned}
& x - y \\
&= x - y + 2^n \qquad \text{(n bit encoding)} \\
&= x + (2^n - y) \\
&= x + (-y)
\end{aligned}
$$

- Let's try 5 – 3 (8-bit representation)
  - 5 + (-3)
  - 00000101 + 11111101

$$
\begin{array}{r}
0\,0\,0\,0\,0\,1\,0\,1 \\
\underline{1\,1\,1\,1\,1\,1\,0\,1}
\end{array}
$$

- A subtract(a, b) function simply becomes add(a, negative(b))

Side note: Two's complement representation

Also, let's try -3 + 5 (8-bit representation)

- Sign-and-magnitude
  - 10000011 + 00000101

```
1 0 0 0 0 0 1 1
0 0 0 0 0 1 0 1
```

- One's complement:
  - 11111100 + 00000101

```
1 1 1 1 1 1 0 0
0 0 0 0 0 1 0 1
```

- Two's complement (prev slide): 00000101 + 11111101 = 2

Side note: Two's complement representation

What about -5 + 3 (8-bit representation)

- Sign-and-magnitude
  - 10000101 + 00000011

```
1 0 0 0 0 1 0 1
0 0 0 0 0 0 1 1
```

- One's complement:
  - 11111010 + 00000011

```
1 1 1 1 1 0 1 0
0 0 0 0 0 0 1 1
```

- Two's complement
  - 11111011 + 00000011

```
1 1 1 1 1 0 1 1
0 0 0 0 0 0 1 1
```

Side note: Two's complement representation

What about -5 + (-3) (8-bit representation)

- Sign-and-magnitude
  - 10000101 + 10000011

```
  1 0 0 0 0 1 0 1
+ 1 0 0 0 0 0 1 1
```

- One's complement:
  - 11111010 + 11111100

```
  1 1 1 1 1 0 1 0
+ 1 1 1 1 1 1 0 0
```

- Two's complement
  - 11111011 + 11111101

```
  1 1 1 1 1 0 1 1
+ 1 1 1 1 1 1 0 1
```

Adding Numbers

Back to teaching computers how to add numbers like a human

- We humans can add two numbers of arbitrary size.
- Can we add many numbers?
  - Example: 37+23+33+17+37+40
- Two approaches:
  - Add all unit digits first, then ten's, hundred's.
  - Add two numbers first, then add a third one.
- Which is a better approach?
  - For humans and computers?
- How to express your approach in pseudo-code?

Adding Numbers – Adding 2 single-digit numbers

Let's create a function, add2small(x, y), which adds 2 single-digit numbers

- Input: two single-digit numbers x and y.
- Output: the two digits of the sum s_{10}, s_1.

look up addition table for row x and column y
let s be the entry
set s_{10} to be the first digit of s
set s_1 to be the second digit of s
return the pair s_{10}, s_1

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

Adding Numbers – Adding 3 single-digit numbers

Let's create a function, add3small(x, y, c), which adds 3 single-digit numbers (to handle a possible carry)

- Input: three single-digit numbers x, y and c.
- Output: the two digits of the sum s_{10}, s_1.

a_{10}, a_1 = add2small(x, y)
b_{10}, b_1 = add2small(a_1, c)
d_{10}, d_1 = add2small(a_{10}, b_{10})
set s_1 to be b_1
set s_{10} to be d_1
return the pair s_{10}, s_1

Adding Numbers – Adding 3 single-digit numbers

$a_{10}, a_1 = \text{add2small}(x, y)$
$b_{10}, b_1 = \text{add2small}(a_1, c)$
$d_{10}, d_1 = \text{add2small}(a_{10}, b_{10})$
set s_1 to be b_1
set s_{10} to be d_1
return the pair s_{10}, s_1

- $x = 9, y = 9, c = 9$
- $a_{10} = 1, a_1 = 8$
- $b_{10} = 1, b_1 = 7$
- $d_{10} = 0, d_1 = 2$
- $s_{10} = 2, s_1 = 7$

We made our inputs as large as possible, and d_{10} remained 0.
We didn't even use it.

Do we need d_{10}?

Adding Numbers – Adding 2 multi-digit numbers

Let's create a function, add2large(X, Y), which adds 2 multi-digit numbers

- Input: two multi-digit numbers $X=(x_m x_{m-1} \ldots x_1)$ and $Y=(y_n y_{n-1} \ldots y_1)$ – m and n digits respectively.
- Output: all digits of the sum $S=(s_a s_{a-1} \ldots s_1)$.

```
set p = maximum of m and n (at least as long as X and Y)
set c_1 = 0
for each digit i running from 1 to p
    a_10, a_1 = add3small(x_i, y_i, c_i)
    set c_{i+1} = a_10
    set s_i = a_1
return the number c_{p+1} s_p s_{p-1} ... s_1
```

Adding Numbers – Adding 2 multi-digit numbers

set p = maximum of m and n (at least as long as X and Y)
set $c_1 = 0$
for each digit i running from 1 to p
    a_{10}, a_1 = add3small(x_i, y_i, c_i)
    set $c_{i+1} = a_{10}$
    set $s_i = a_1$
return the number $c_{p+1}s_p s_{p-1}\ldots s_1$

- X = 97, Y = 921, p = 3
- i = 1
- $c_1 = 0$, $x_1 = 7$, $y_1 = 1$
- $a_{10} = 0$, $a_1 = 8$
- $c_2 = 0$, $s_1 = 8$

- i = 2
- $c_2 = 0$, $x_2 = 9$, $y_2 = 2$
- $a_{10} = 1$, $a_1 = 1$
- $c_3 = 1$, $s_2 = 1$

- i = 3
- $c_3 = 1$, $x_3 = 0$, $y_3 = 9$
- $a_{10} = 1$, $a_1 = 0$
- $c_4 = 1$, $s_3 = 0$
- return 1018

Adding Numbers – Adding multiple multi-digit numbers

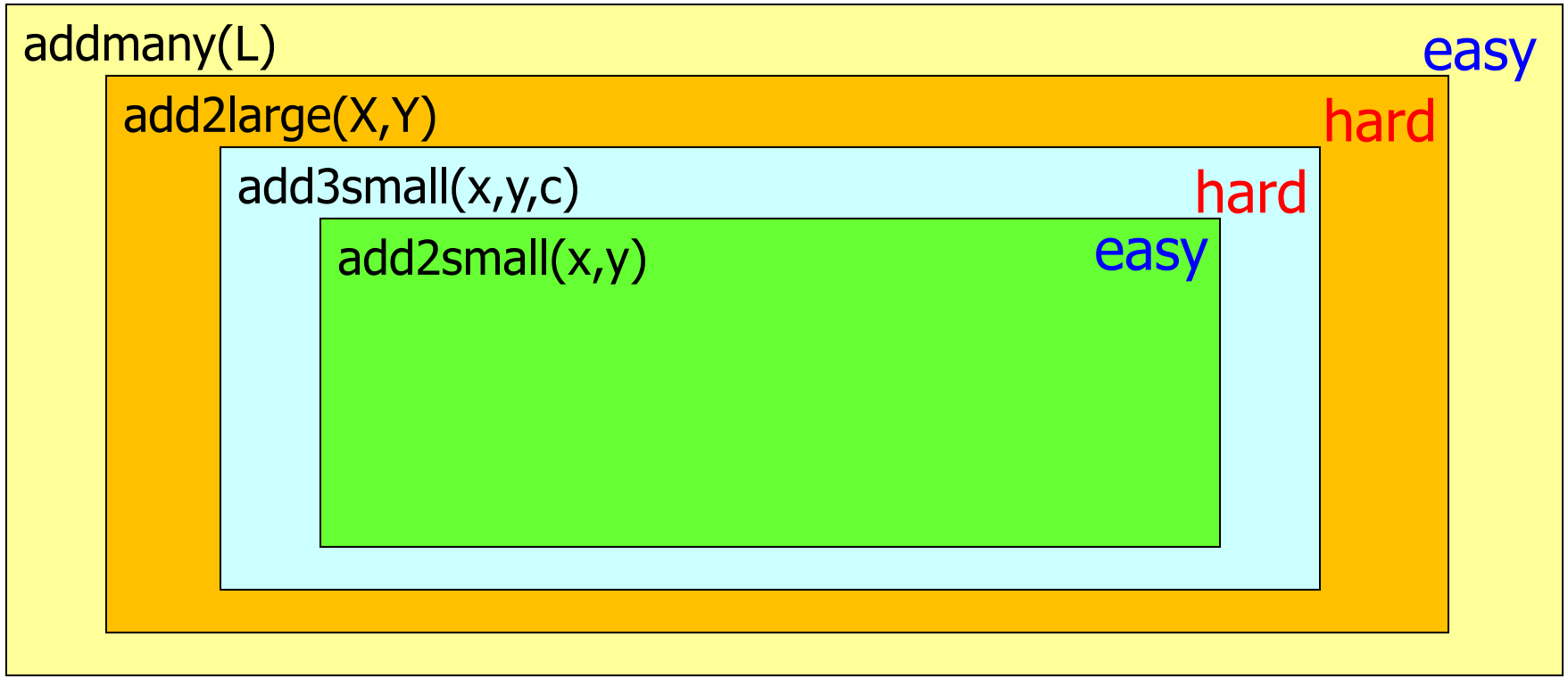
Let's create a function, addmany(L), which adds multiple multi-digit numbers

- Input: a list of numbers, L
- Output: the sum S

```
set S to 0
for each number n in L
    S = add2large(S,n)
return S
```

Adding Numbers

- Now our AI overlords can pass the Turing test and add a list of multiple numbers like a human!



Adding Numbers

- Three types of computational statements are sufficient in our four-step solution.
  - Conditions are not even necessary here!
  - Usage of functions will help to reduce 3 copies of `add2small`(x, y) inside `add3small`(x, y, c).
- Simple addition-table lookups are sufficient.
  - As long as we can add two single-digit numbers, we can add two numbers of arbitrary sizes and then multiple numbers.
  - Complex solutions could be built upon simpler ones.

Computation

- We can refine our solution from high level gradually to low level.
  - Top-down approach
    - Example: in sorting programs, we first assume lower level things can be done without doing them in our design, e.g. finding the smallest number, inserting an item in proper position.
- We can build up our solution from simple blocks that we know to work.
  - Bottom-up approach
    - Example: in human way of addition program, we build up from adding 2 small numbers, to 3 small numbers, to 2 big numbers, to many big numbers gradually.
- When to use top-down and when to use bottom-up?

Computation

- Teaching the computer to add numbers in our way!
  - Advantage: no limit on the size of the numbers.
    - Each digit is contained within a separate variable
  - Disadvantage: slower than binary addition, events of the Matrix occur
- Abstraction
  - Based on very simple table lookup (or definition).
  - It may not be the add operator, but could be multiply, or even a mysterious operator $\oplus$.
  - It may not be numbers, but symbols.
  - Can be generalized into non-numerical (symbolic) computation.

Computation

	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	01	02	03	04	05	06	07	08	09	10
2	02	03	04	05	06	07	08	09	10	11
3	03	04	05	06	07	08	09	10	11	12
4	04	05	06	07	08	09	10	11	12	13
5	05	06	07	08	09	10	11	12	13	14
6	06	07	08	09	10	11	12	13	14	15
7	07	08	09	10	11	12	13	14	15	16
8	08	09	10	11	12	13	14	15	16	17
9	09	10	11	12	13	14	15	16	17	18

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Computation

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

	a	b	c	d	e	f	g	h	i	j
a	aa	ab	ac	ad	ae	af	ag	ah	ai	aj
b	ab	ac	ad	ae	af	ag	ah	ai	aj	ba
c	ac	ad	ae	af	ag	ah	ai	aj	ba	bb
d	ad	ae	af	ag	ah	ai	aj	ba	bb	bc
e	ae	af	ag	ah	ai	aj	ba	bb	bc	bd
f	af	ag	ah	ai	aj	ba	bb	bc	bd	be
g	ag	ah	ai	aj	ba	bb	bc	bd	be	bf
h	ah	ai	aj	ba	bb	bc	bd	be	bf	bg
i	ai	aj	ba	bb	bc	bd	be	bf	bg	bh
j	aj	ba	bb	bc	bd	be	bf	bg	bh	bi

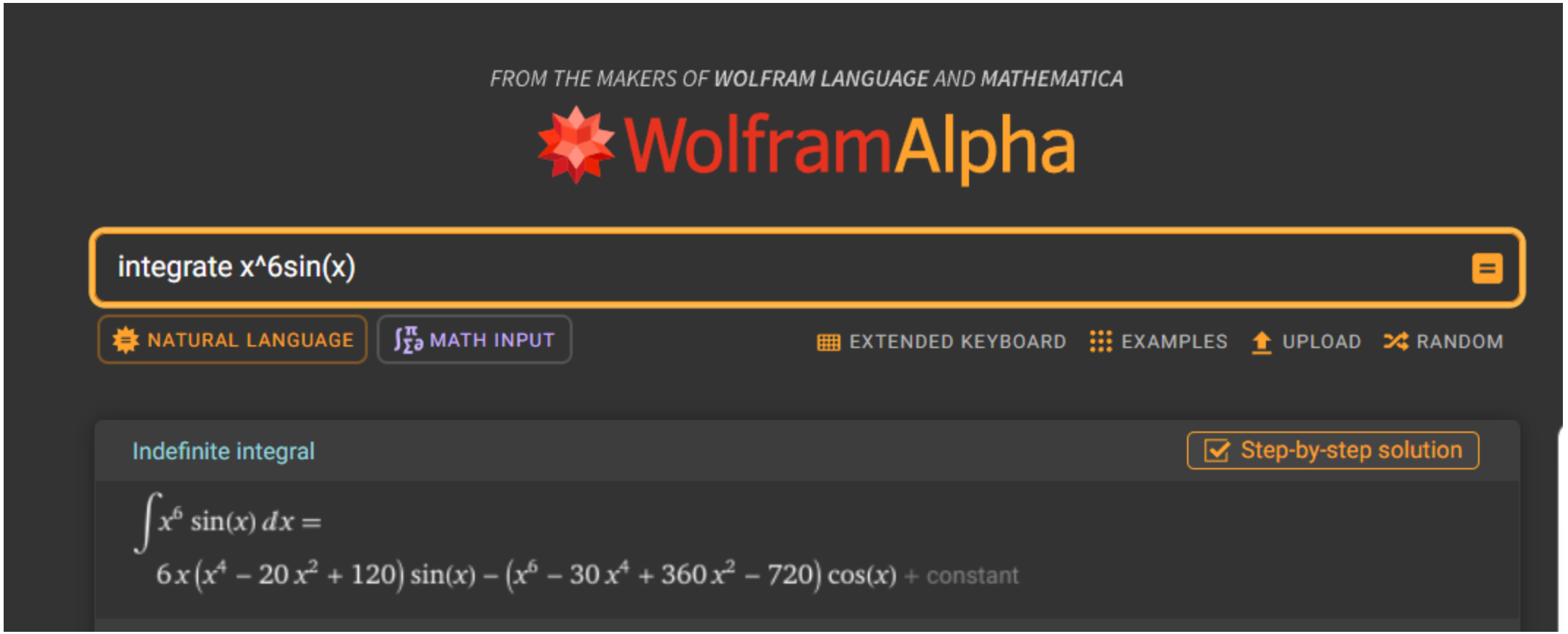
Computation

- "To produce meaningful answers, you do not have to understand what the symbols stand for or why the manipulations are correct." (Hector Levesque)
- The "trick" of computation (Levesque):
  - The activities performed by a computer are a type of symbol processing that can be carried out purely mechanically.
  - Computation is the process of taking symbolic structures, breaking them apart, comparing them, and reassembling them according to a precise recipe called a procedure (or algorithm).
- This is called symbolic computation.
  - A key constituent in classical Artificial Intelligence (not Big Data).
  - Programming language like Prolog.
  - Example: theorem proving, symbolic differentiation

Computation – Symbolic vs Numeric

- Symbolic computation
  - Computation involving symbols to represent data, in exact value/form.
- Numeric computation
  - Computation involving numbers to represent data, sometimes in approximated value/form.
    - Numbers carry real meaning
    - Equation solving
    - Average, summary and statistics of data
    - Sorting and searching of data
    - Big data processing, e.g. data mining, clustering
  - Majority of common programs

Computation – Symbolic Example



Multiplication

- 23 x 14

x	10	4
20	200	80
3	30	12

Multiplication in Computers

- 23 x 14
  - 10111 × 1110

0 by 10111 = 0

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
```

1 by 10111 = 10111

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
```

Add

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
```

1 by 10111 = 10111

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
```

Add

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
 1 0 1 1 1 0 0
 -------------
 1 0 0 0 1 0 1 0
```

1 by 10111 = 10111

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
 1 0 1 1 1 0 0
 -------------
 1 0 0 0 1 0 1 0
```

Add

```
 1 0 1 1 1
   1 1 1 0
 ---------
 0 0 0 0 0
 1 0 1 1 1 0
 ---------
 1 0 1 1 1 0
 1 0 1 1 1 0 0
 -------------
 1 0 0 0 1 0 1 0
 1 0 1 1 1 0 0 0
 ---------------
 1 0 1 0 0 0 0 1 0
```

Answer is $101000010_2 = 322_{10}$

Multiplying Large Numbers

- Our previous solution, **`addmany`**(L), to add a list of large numbers is built based on simple table lookup.
  - There is no real mathematics done!

- Now let us assume that we are able to use real mathematics to help.
  - That could make programming easier for complex tasks.
  - Just generate pseudo-code (then program) that follows the human ways of multiplying decimal numbers together.
  - You may make use of real addition and real multiplication when numbers are small.

Computation

- Traditionally, computers have been very unimaginative and lacked creativity
  - Recent machine learning techniques enable computers to deduce "knowledge" from large collection of data for artificial intelligence.
  - Computers can now generate some "new" things!

- Computers are extremely accurate but:

```
>>> 123456789 * 987654321
1219326311126352 69
```

  - Not all computers can calculate 123456789 * 987654321 accurately.
  - Try this in Python
    - 12345678.9 * 98765432.1
    - How can we make it accurate?

```
>>> 12345678.9 * 98765432.1
1219326311126352.8
```

Computation

- When using computational thinking, be careful and aware of the limitations and pitfalls that computers have
- We should design the program (algorithm / pseudo-code) in a more general manner.
- We should not assume input data is correct, and should perform proper error checking (called input validation).
  - You cannot add together inputs that are not numbers.
  - You cannot divide a number by zero.
  - You cannot compare an integer with a string to see which one is larger.
  - You may not allow punctuation marks in a person's name.


That's all Folks!