

Introduction

Name : Joshit Vadela

Name of Professor : Aravind Kumar Chilakamarri

Name of course : CS50x Harvard Course

About Myself : I am a diligent and enthusiastic grade 9 student with a fervent passion about Computer Science and Programming. Through self-study, online courses and my professor “Aravind Kumar Chilakamarri”, I have gained a satisfactory foundation on the Programming language Python. One of my key interests include the CS50x Harvard course which I'm currently following with great enthusiasm and determination.

Objective : As a dedicated learner, I've made it my mission to provide clear, organised, and comprehensive notes on “CS50x” to help fellow students navigate through this complex subject matter.

Conclusion : Thank you for visiting my GitHub repository and considering my notes. Together, we can make the learning journey through CS50x Harvard more engaging and informative.

Lecture 0 - Scratch

Binary

What is computer Science?

Programming is a big part of computer science, where you write code to express ideas and solve actual problems involving data, but computer science itself is the study of information.

What does it mean to solve a problem?

Problem solving is where you get some input (which represents the problem you want to solve) and then there will be an output (which represents your goal which is to solve the problem) and then there will be a secret recipe in the middle which will help you solve the problem.

What is Unary and Binary?

Unary is a type of system which would help us solve problems which include numbers (for ex. taking attendance at the beginning of class). In other words, unary is a very simple system of using a single symbol to solve some problems. Unary, in mathematical and technical terms is called "Base-1" meaning that the number that you are operating with has 1 digit in it.

Computers use a language other than Unary, this language is called Binary. Bi means 2, so computers have 2 digits which they would use to communicate with or solve problems within.

The technical term bit comes from the word binary digit. A bit is 0 & 1. So in simple words, us humans, in our everyday life use the Unary system to solve problems, meaning that we use the digits from 0 to 9. On the other hand computers use the Binary system to solve problems, meaning that they use the digits 0 or 1.

How do computers only speak in Binary?

The simplest way in which we could understand this by taking the light bulb as an example. So if humans were a computer, to represent the number 0 you would keep the light switch off & to represent the number 1 you would switch the light bulb on.

Now, this is relevant to computers because inside a computer there are thousands, even millions of tiny switches (we could metaphorically think of these switches as light bulbs). These switches are called transistors.

Just like a light bulb, these transistors could be switched on to represent 1 and switched off to represent 0.

Obviously, with this ability of computers they could count 0 to 1 but it turns out that they could count even higher with a little bit more electricity.

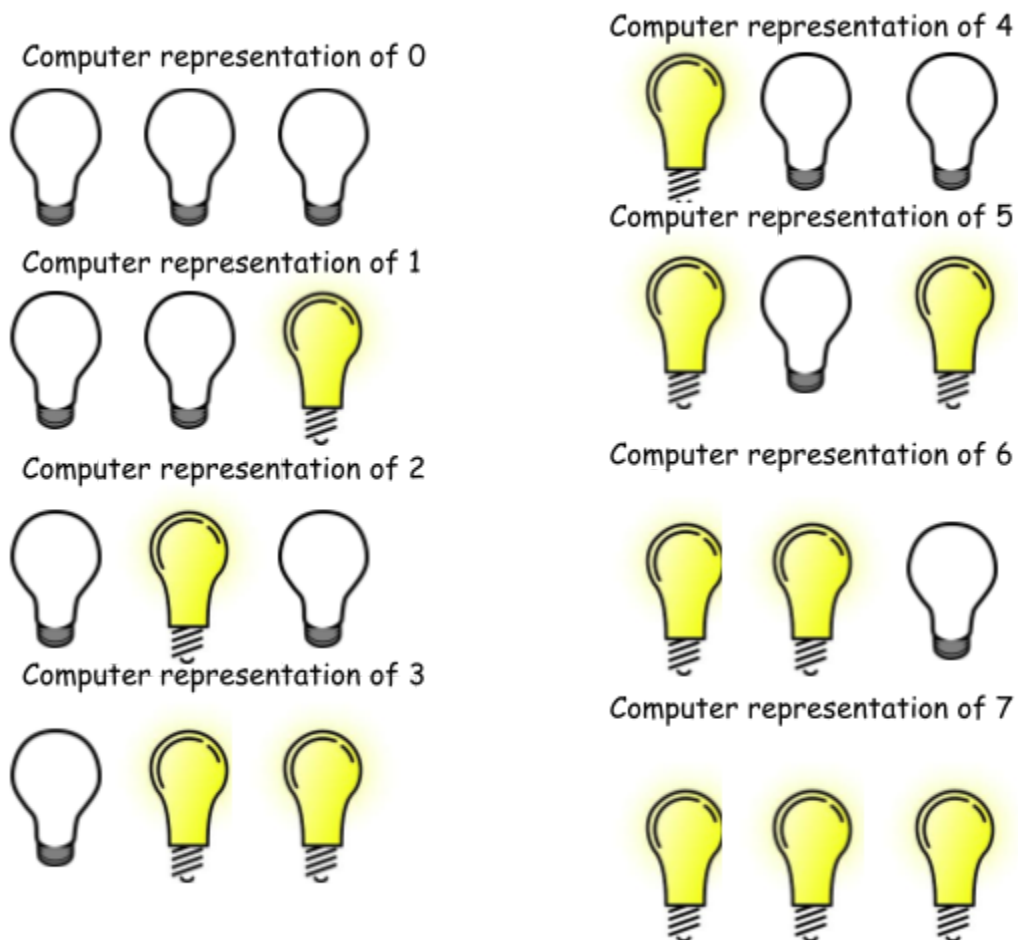
Based on the above concept, when we take a singular light bulb or a transistor and switch it on, it represents 1 and when we switch it off, it represents 0. So to represent more numbers we would need to use more light bulbs.

To understand this concept better, let us take 3 light bulbs which are all switched off. We know that when a light bulb is turned off it represents 0.

So we would think that with these 3 light bulbs (when they are switched on) could represent 0, 1, 2, 3, but when we consider the combination of bulbs being on and off the computer might be able to represent more than 0, 1, 2, 3.

For example we could say that the computer stores the number 0 when all 3 light bulbs are off. When the first light bulb is on but the other two are off, this could be how the computer stores the number 1. Similarly, when the second light bulb is turned on and the other two are off, this could be how the computer stores the number 2.

Following the above example, with 3 light bulbs the computer can store the number 0, 1, 2, 3, 4, 5, 6 & 7.



What we have to understand from this example is that the way the computer stores these number values is by memorising the patterns in which these transistors or light bulbs turn on or off.

We cannot represent more than 0, 1, 2, 3, 4, 5, 6, 7 with 3 light bulbs because the pattern or the combinations in which all three of these lights could uniquely be arranged is only in 8 ways. Thus, we can only represent 8 numbers. The same method could be used for representing numbers with multiple light bulbs.

In the above example, the computer used base 2 (Binary) to represent numbers 0 through 7. Binary, just like the decimal system that we use, follows certain types of rules.

Let's recall back to how decimal place values work,

=> 123

When we look at the above number "123" our mind reads it as "one hundred and twenty three". This is because we know that the rightmost number (3) is in the 1's place, the middle digit (2) is in the 10's place and the leftmost digit (1) is in the 100's place. Now we just do quick mental maths, which would lead to " $100 \times 1 + 10 \times 2 + 1 \times 3$ " which is " $100 + 20 + 3$ " gives us the number we know as "123".

We could also use base terminology to represent these places. For example the 1's place is 10^0 , the 10's place is 10^1 , the 100's place is 10^2 and so forth.

Now, if we take computers, they obviously only have access to electricity on or off (0 or 1). So they tend to use a different base. 2^0 , 2^1 , 2^2 and so forth, a.k.a 1's place, 2's place, 4's place and if we keep going 8's place,

16's place and so on. At the end of the day, the idea for both the decimal system and the binary system is fundamentally the same.

4	2	1		4	2	1		4	2	1
0	0	0		0	1	1		1	1	0
4	2	1		4	2	1		4	2	1
0	0	1		1	0	0		1	1	1
4	2	1		4	2	1				
0	1	0		1	0	1				

Finally, this is the computer representation of the above diagram of light bulbs into numbers.

Here, as we could see

=> "off off off" represents 0 because $4 \times 0 + 2 \times 0 + 1 \times 0$ is 0

=> "off off on" represents 1 because $4 \times 0 + 2 \times 0 + 1 \times 1$ is 1

=> "off on off" represents 2 because $4 \times 0 + 2 \times 1 + 1 \times 0$ is 2, and so on

Now if the computer wanted to represent 8, it would just move to the 8's place (another transistor) just like how we move to the thousands place when we want to represent a number in the thousand's column.

Essentially at the end of the day the binary system used by computers and the decimal system used by humans are similar but instead of base 10 for decimal system, we use base 2 for binary system and only the numbers 0 and 1 are used.

Now, singular “Bits” are not that useful as they cannot be used to store memory. This is why “Bytes” is a more useful measurement unit to use when talking about data or memory. Generally, there are 8 “Bits” in a Byte.

For example, when a picture or a video is downloaded, the memory that they take up is measured in a larger number of Bytes such as Kilobytes, Megabytes or even Gigabytes. This just means that there are lots of combinations of 8 Bits in the computer's hard drive.

****Fun Fact: A Computer can count up to 256 with 8 Bits or 1 Byte ****

Representation

How do Computers represent letters?

Obviously, we use computers for more than just mathematics. They are used to send messages or texts or emails and so on. At the end of the day, every device has millions of transistors that they use in units of Bytes (8 Bits) to represent numbers using various combinations and patterns of 0's and 1's.

“So then how might a computer represent the alphabets?”

We could possibly assign a number to every letter. That would lead to (for simplicity purposes) A represented by 0, B 1, C 2 and so on.

Now how would we represent 0 or 1 or 2?

How might we be able to differentiate between the numbers representing the letters and the numbers representing numbers themselves?

Before we think about the solution to that, computers, phones etc actually represent the letter "A" by the number 65 (the numbers 0 to 64 have been used to represent special and control characters). It turns out that the letter "B" is going to be 66, "C" being 67 and so forth.

[Example:

Representation of "A" by computers etc - 01000001

(1 being in the 1's place and another 1 being in the 64's place)]

In the above example, we can visually see how the computer represents the letter A.

So now how do we represent all of the other letters?

It turns out that Americans, years ago, came up with ASCII (American Standard Code for Information Interchange). ASCII is an acronym that describes what is being explained above. Representing letters (A, B, C ...) with binary numbers starting from 65.

ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(null)	NUL	032	(space)	064	@	096	
001	☉	SOH	033	!	065	A	097	a
002	☼	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♣	EOT	036	\$	068	D	100	d
005	♠	ENQ	037	%	069	E	101	e
006	♣	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	■	BS	040	(072	H	104	h
009	(tab)	HT	041)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♪	SO	046	.	078	N	110	n
015	☉	SI	047	/	079	O	111	o
016	▶	DLE	048	0	080	P	112	p
017	▶	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	!!	DC3	051	3	083	S	115	s
020	=	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↕	ETB	055	7	087	W	119	w
024	↕	CAN	056	8	088	X	120	x
025	↕	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[123	{
028	(cursor right)	FS	060	<	092	\	124	
029	(cursor left)	GS	061	=	093]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	☐

Now whenever you receive a text or a message you could see under the hood what patterns on 0's and 1's were sent.

[Example:

Text Message you received in Binary Code -

01001000 01001001 00100001
72 73 33]

In the above example we could see that the text message received is "HI!". 72 representing "H", 73 representing "I" and 33 representing "!" (sometimes you would need a cheat sheet like the one above to know the binary code for certain characters).

Now back to the original question, if 65 represents A, then how would you represent 65 the number or 66 the number?

When the chart above is clearly observed, we could see that there are **other** numbers that represent numbers 0 through 9.

048	0
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9

As we can see above, if you hit "5" on your computer, the computer does not store 5, it would store 53.

How do Computers represent other Characters?

In the above ASCII chart, one can clearly notice that the letters used are of the English language. This is a problem as the ASCII chart does not include characters from other languages and we might want to send messages in a language other than English, this could include accented or Asian characters/alphabets.

If we recall back, ASCII uses a byte (7 bits to be specific) to represent English letters. Meaning that ASCII could represent up to 256 total characters (256 based on the possibilities of patterns formed using 8 bits). This is obviously not enough characters to represent all of the letters in every language in this world.

Therefore, to represent all of these various characters including Asian characters, we would just have to add another byte to create more patterns. So instead of using 1 byte, we could use 2 or 3 or even 4 bytes to represent all of these characters.

The solution then to ASCII, is Unicode which is also a mapping of numbers and letters just like ASCII but in many different languages. People from various countries, companies and cultures came together to agree on the specific Binary representation of each character of each and all languages of humans.

“Now where do emojis come in?”

If we start using 32 bits, which is about 4 bytes, we could represent as many as 4 billion patterns which is way more than we need to represent

all human languages in the world. Now, we could utilise these patterns to represent more playful things such as emojis.

[Example:

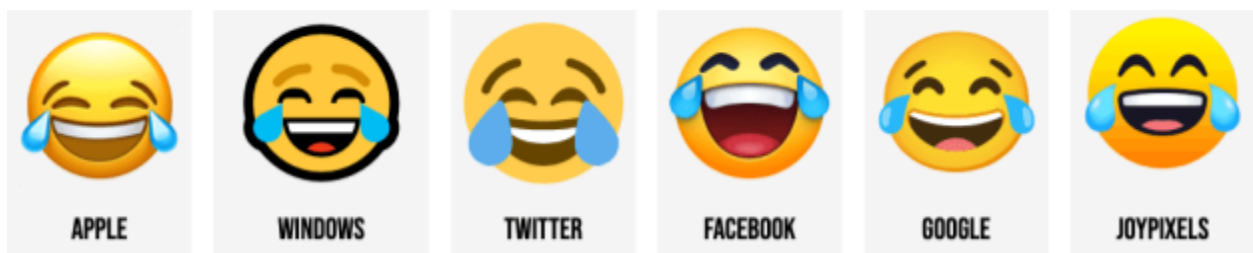
Emoji you received in Binary Code -

11110000100111111001100010000010



The above is the Binary code that the people of Unicode decided to represent the popular emoji “Tears of Joy”.

Now sometimes we notice that the same emojis look different on IOS or Android or Windows or other softwares. Even though we look at these emojis as images they're really characters and IOS, Android and other softwares just have different fonts just like how there are different fonts we use in the English language. So that same pattern of 0's and 1's might just slightly be different.



Tears of Joy emoji representation in different softwares.

Every year, the people at Unicode add more emojis and other characters.

How do Computers represent Emoji customisation?

As discussed above, emojis are commonly used text messages. Different emojis would be trending at different times.

Although a lot of the emojis are the default Yellow colour, some emojis have the option to pick specific skin tones. If it's supported by the software and Unicode we could actually interact with the emoji and customise it which modifies the display.

Now, if a specific emoji has 5 different skin tones, it doesn't make sense to have five different identical patterns of 0's and 1's except for the change in skin tones because it's just repeated code which takes up more bits than actually required.

“Now how then, did companies like Apple and Google implement support for emoji skin tones without wasting data?”

The way Unicode solves this problem is the first byte or bytes (patterns of 0's and 1's) that you would receive in text just represents the structure of the emoji, the default Yellow version, then if the text is immediately followed by a certain reusable pattern of bits which are standardised to represent each of the skin tones, the phone, computer, Mac will change the default colour into the apt human skin tone.

This does not take up that much data or information because we are only using twice as many bits and not five times as many bits.

Again, there are some emojis these days, which are combinations of 2 or more emojis.

[Example:

One Emoji made with combinations of 2 emojis

Example 1:



Example 2:



These above emojis, are examples of a single emoji which is made of 3 or 2 other emojis, the bike; the person; the mountains.

Now, the way computers represent this emoji (example 1) is that it has a specific pattern of 0's and 1's for "the bike" emoji, another pattern for "the person" and finally a pattern for "the mountains". At the end, to assemble this emoji together the computer brings these different patterns of 0's and 1's together.

Obviously, if the computer wants to represent the other emoji (example 2), it would just reuse the same patterns of 0's and 1's for "the bike" and "the person" instead of creating a new pattern every time.

The same would apply to other complex emojis too, the pattern's of 0's and 1's for a specific character would be reused again and again based on the use of the character in different emojis.

{Additional Notes: This is the imperfection of humans, where years ago they built a system (ASCII) that was completely American stranded, no characters, emojis or the like, which has evolved.

So that's an important detail to keep in mind that computing is always evolving, and programming languages such as Python, C++, Ruby etc, those too are evolving as well with new features and versions}

How do Computers represent images and videos?

In computers, images are an assembly of some amount of red, some amount of green and some amount of blue (RGB). Although there are other forms of representations, RGB is still very common.

“What does the representation of images by RGB mean?”

To represent every dot on your phone, TV, laptop or desktop, there is a specific number representing how much red, green and blue the dot should show respectively.

[Example:

Any three number values used to represent RGB-

72

73

33]

In the above example, the numbers [72, 73, 33] in text format would represent “HI!”.

The same number combination in photoshop or other graphics designer would represent [72] medium amount of red, [73] medium amount of green and [33] low amount of blue.

[Medium and low because the lowest value in a byte is 0 and the highest value in a byte is 255. Therefore, [72] and [73] could be considered medium and [33] could be considered low]

Those 3 colours are combined (with the respective numbers) in such a way that it would represent a sort of murky shade of yellow dot on your screen.

[Example:

Murky shade of yellow dot represented by RGB-

Initial representation -  72  73  33

Combined representation - ]

This is how (with the RGB combination of [72], [73] & [33]) the computer would store the above colour.


We have actually seen this representation (RGB) before. When using the emoji "Face with Tears of Joy", generally it looks like "😂", but when zoomed in, one could notice that this emoji is a combination of pixels, which is just a dot on the screen. When we further zoom into the emoji, we can see every single pixel or dot that composes the emoji.

Now, this is why images and photographs get so big (because each pixel is being represented by a number and there are millions of pixels in each image).

If this is how images are represented, by a pattern of bits each assigned to some amount of red, green or blue for an output.

"How are videos represented?"

A video, if at the end of the day all we have is 0's and 1's, are pixels changing values over time or a sequence of images that are changing on the screen (like a flipbook).

 I made this @EdSheeran #flipbook FOUR years ago! Will it go #viral a...

A flipbook is a great example to understand the concept of a video because the artist drew on hundreds of pieces of individual papers with almost identical images where the ink from their pen was slightly moving in each drawing.

If we digital this, such that each of the pen marks are dots or pixels on a screen that's really what a video is. A sequence of all individual images flying across the screen.

A typical Hollywood movie is 24 FPS (Frames per Second). This means that each second of video shows 24 distinct still images.

Therefore, it's not actually a motion picture, it's a sequence of pictures. Our brain interprets it as smooth movement, when in reality we're just seeing a lot of pictures really fast.

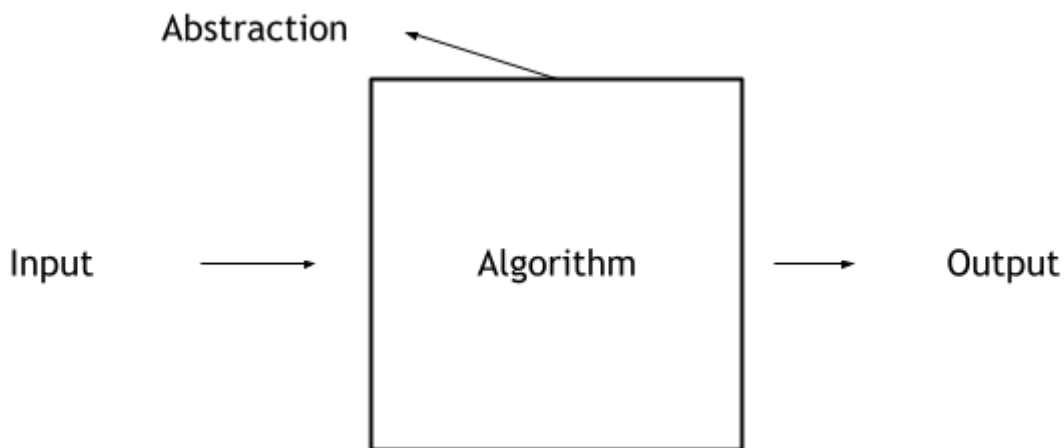
In the same format, music can also be quantised digitally, by assigning a specific frequency value (numbers measured in Hertz). A second number could be used to represent the loudness and a third number could represent the duration and etc.

{Additional Notes: To summarise the topics discussed so far, humans standardised ways to represent inputs and outputs of alphabets, characters, sound and many alike.

They don't always agree, that's why there are different file formats for Apple, Microsoft, Google etc}

Algorithms

What are Abstractions?



So far, we have talked about how humans have standardised the ways to represent the inputs and outputs to and from problems.

Now, we would be focusing on this box (above diagram) in the middle.

This box could be termed as abstraction.

Abstraction is a term that is used all over the place in Computer Science and problem solving. This term refers to the simplification of something so that we don't focus on the lower implementation details but on the high process or goals.

For example, driving a car, as far as you are concerned, could possibly be an abstraction. This is because most of us really don't care about how the engine and other parts of a car work. To you it's just a way of getting from point A to point B.

Someone, the mechanic, knows the lower implementation details. But if you had to understand every time you use a car how it works, that would be a slow process. Therefore, we would just think about the higher level of abstraction.

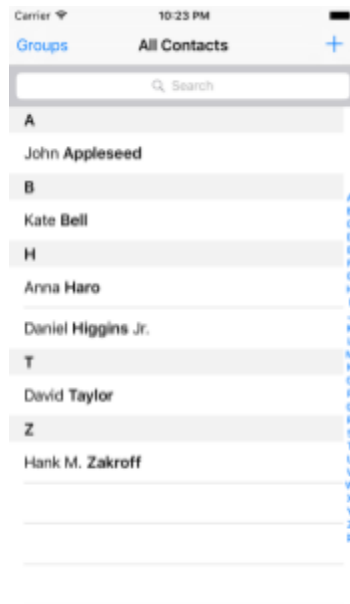
{Additional Notes (What are Algorithms?): "So what then is in the above box, this abstraction at the moment?"}

Generally, it's what a computer scientist would call an algorithm. Algorithms are step-by-step instructions for solving problems.}

What are Implementation details?

Now, let's consider the implementation details, that is how we solve a problem.

For example if we take the contacts app on your phone which is used for the contact of family members, friends or the like, you would have noticed that all of these contacts are alphabetized from “A” to “Z” and we would click on the search and use auto complete to find the contact that we are looking for.



Contacts app alphabetized by last name.

When we start typing autocomplete, you would notice that when “H” is typed in you only see the contacts “Anna Haro” and “Daniel Higgins Jr.” (refer to the picture above), and when “H-A” is typed in one would only see “Anna Haro”.

Now, during this process one could possibly notice that autocomplete happens very fast, you type in the letter and you have the contact instantly.

“So how does this happen?”

A computer could start at the top of the list and run through it to the bottom, searching for all of the "H's" or "H-A's" but this algorithm is not ideal for larger sets of data as it would be a long process to search through everything.

Back in the day, when iphone contacts were not a thing, there used to be phonebooks. When you want to search for someone, say "James Harvard", one could do so by looking through each of the individual pages of the phonebook, but looking through each page is not the most time efficient process or algorithm.

Other options might suggest going through 2 pages at a time for efficiency, but this algorithm/method is not correct because you would be skipping pages and the contact you are looking for "James Harvard" might be on one of the pages that was skipped.

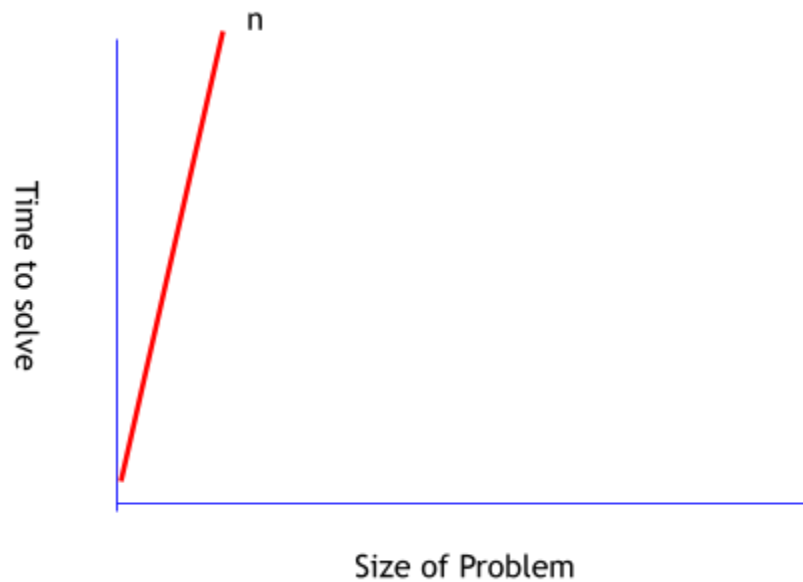
Now, the efficient way of doing this, I would argue, is to open roughly to the middle of book, based on this one could easily notice the letter section on the book (For e.x "M" section or "N" section) and based on that move to the right or (in this case of "James Harvard") to the left of the book, completely ignoring the other half as it is confirmed that "H" is to the left of "M".

Again, in this half of the book (left half) following the same process or algorithm of finding the middle, deciding whether to search on the left or the right and ignoring the side that is not being considered.

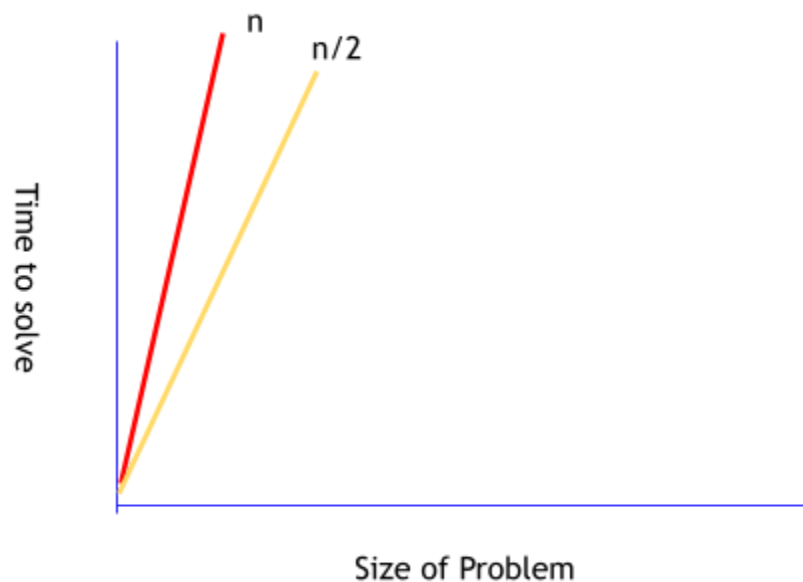
We would keep doing this until we find the "H" section, where "James Harvard" would be found.

“What is the implication of this performance?”

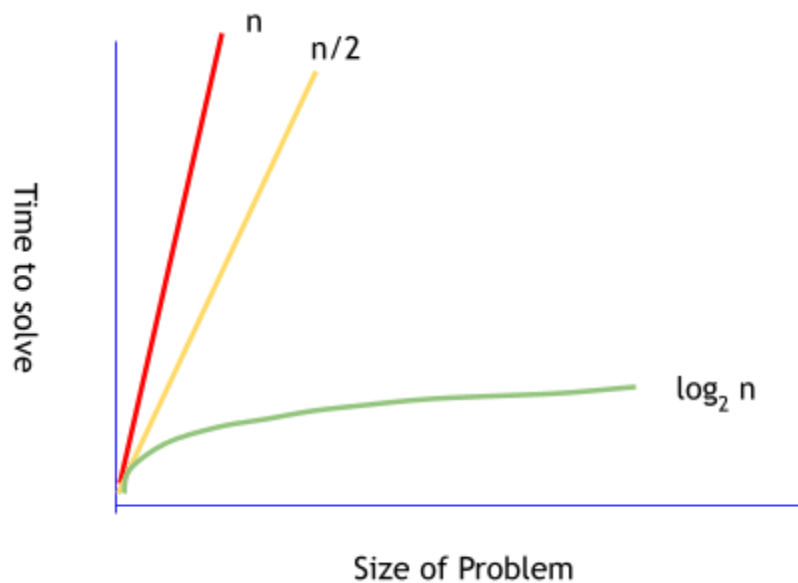
Let's look at each of the above processes used to find the contact “James Harvard” in a graph.



Now, this is the graph for the first algorithm (red line), here one can notice that it is represented using a straight line because if there's "n" pages in the phone book, it would take "n" units of time to find the required name.



This is the graph for the second algorithm (yellow line), again, one could notice that the yellow line is represented as a straight line and it was twice as fast (2 pages at a time). Here “ $n/2$ ” means there’s half as many pages here, so time to solve would also be half the units of time.



This is the graph for the final algorithm (green line), here uniquely, one can notice that it's a fundamentally different representation of the algorithm. The green line is going up but at a much slower rate of increase. So even if the phonebook is doubled next year, it would take one more step to find “James Harvard”.

{Additional Notes: The above example of the phonebook and contacts are an example of implementing details and not abstraction because we are looking at the minor details of how to solve the problem and not the final goal or output. }

All of the above explanations relate to the topic because your computer or phone is essentially doing the same thing under the hood with large

sets of data when searching for a specific character in contacts or the like.

What is Pseudocode?

Pseudocode is an English-like syntax that programmers use to start solving a problem before using Python, C or even Scratch. They don't start typing in code in C or Python or the like, they would use their human language to summarise or jot down the outline of what they want to code.

Pseudocode of the "Phone Book" example -

1. Pick up phone book
2. Open to middle of phone book
3. Look at page
4. If person is on page
5. Call person
6. Else if person is earlier in the book
7. Open to the middle of left half of book
8. Go back to line 3
9. Else if person is later in book
10. Open to middle of right half of book
11. Go back to line 3
12. Else
13. Quit

Above is the Pseudocode example of how one could summarise the code of finding a given person in a phone book.

Lines 12 and 13 say "Else; Quit". This is because if the given person is not in the phone book, instead of the program crashing, we are giving it a catch error.

Hence, there is well-defined behaviour for every possible scenario of the four possibilities.

1. Pick up phone book
2. Open to middle of phone book
3. Look at page
4. If person is on page
5. Call person
6. Else if person is earlier in the book
7. Open to the middle of left half of book
8. Go back to line 3
9. Else if person is later in book
10. Open to middle of right half of book
11. Go back to line 3
12. Else
13. Quit

Taking a look at the Pseudocode again, some of these words are highlighted in Yellow, a defined pattern could be noticed amongst these words. This is because all of the highlighted words here are English verbs.

In programming, these verbs are to be called “functions”. So when something is being programmed or being coded, you would want the program or the computer to do something, some action or verb, these actions or verbs are referred to as “functions”.

If we look at the words highlighted in Light Blue, these could be called “conditional”, where you have more than one option to carry out, do this thing or that thing.

And you would decide which of those things to do based on what's highlighted in Red. These words could be referred to as “Boolean

expressions", which is just a question with a yes/no, a true/false, a 1 or a 0 answer.

One can notice that lines 5, 7, 8, 10 and 11 are indented. This is important because this is telling the program or the computer that line 5 should only be carried out if line 4 is a yes or a true.

Finally, the lines highlighted in Light Grey represent "loops", these result in doing the same thing again and again. Now, this algorithm/Pseudocode the "loop" would eventually stop because "John Harvard" would be found on the page or, he won't be at all.