A Crash Course on Computation, Linear Algebra, and Quantum Computing

Tanay Biradar - Archbishop Mitty High School Computer Science Club

Information and Computation

1. Read up to "Entropy and Compression" in The Essence of Computation (in the QC Canvas module)
2. Read "Booleans, Operations, and Logic" in The Essence of Computation to "Classical Computation"

Big O Notation - Algorithm Complexity

An **algorithm** is a method of getting a task done. It can be the steps required for a computer to sort a list of numbers or find the position of a given number.

Let us consider the list $A = \{3, 1, 4, 1, 5, 9, 2, 6\}$ for the following:

Unstructured Search

I want to know which position in the list the number 9 is located in. In programming, *we always start counting at 0*, so the 0th element is 3, the 1st element is 1, the 2nd element is 4, and so on.

How do I find which location in the array the number 9 is located in? Since the list is not sorted, we need to check through every element in the list. Since we have 8 elements, we'd have to do 8 operations in the *worst-case scenario*. Of course, we can get lucky and find the element in the 0th position, but we as computer scientists are worried about the worst case.

If we had n elements, we'd need to make n operations in the worst case. We therefore say that algorithm this takes "on the order of" n operations; in other words, this is an $O(n)$ algorithm.

Some algorithms might take $4n$ operations instead of exactly n, but we don't really care about that constant coefficient. We're worried about how the algorithm *behaves* as we increase our input size.

In Python, searching for a list looks like this. Python is pretty English-like, so hopefully this makes some sense to you even if you haven't coded before:

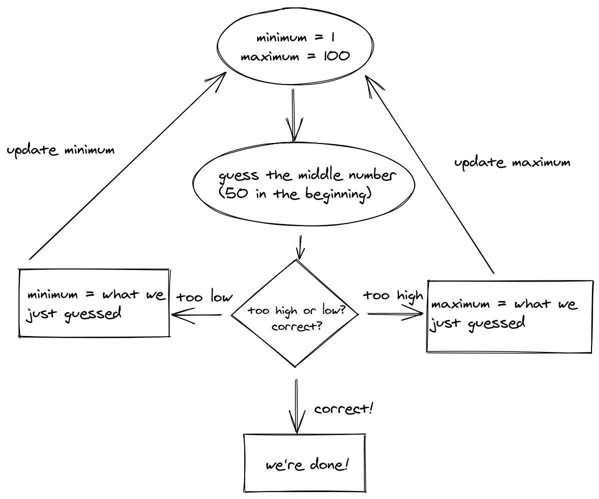
```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
target_element = 9 # I want to know where this element is located

# For every number in the interval [0, length of my_list)
for i in range(len(my_list)):
    if my_list[i] == target_element:
        print(i) # My target number is located here!
        break # we're done
```

Binary Search

Now imagine I sorted A to form the list B. Now, $B = \{1, 1, 2, 3, 4, 5, 6, 9\}$

How could a computer tell which location the number 84 is in? It's the same strategy you could use if I told you to guess the number I'm thinking of (and it's between 1 and 100). I'll tell you if it's higher or lower.


minimum = 1
maximum = 100
update minimum
update maximum
guess the middle number (50 in the beginning)
minimum = what we just guessed
too low
too high or low? correct?
too high
maximum = what we just guessed
correct!
we're done!

Here's a Khan Academy summary

With each guess, you halve the number of possible correct answers. Therefore, with our list B, we'd make a maximum of 3 guesses before we found the correct answer. For a list of n elements, you would make a maximum of $log_2(n)$ guesses.

Binary search is therefore an $O(log(n))$ algorithm. In computer science, logarithms tend to be base 2. But using the change of base formula, we can change this into whatever base we want. It's just a constant factor we multiply our base 2 log by, so it doesn't make a difference as far as Big O notation is concerned.

This Khan Academy article explains the running time effectively

Linear Algebra

Linear algebra has been extensively covered by online sources, and Khan Academy does an excellent job with many of the topics we need to know. If you can, *make sure to go through the following videos listed here*

1. Heading: Vectors
    1. Vector intro for linear algebra
    2. Real coordinate spaces
    3. Adding vectors algebraically & graphically
    4. Multiplying a vector by a scalar

        5. Vector examples
    2. Heading: Linear combinations and spans
        1. Linear combinations and spans
    3. Heading: Vector dot and cross products
        1. Vector dot product and vector length
    4. Heading: Null space and column space
        1. Matrix vector products

You can go through some of the exercises if you find that helpful, too. Let us know if you have any questions on the videos, too. This seems like a lot of information to take in, but do your best to at least get familiar with it—as we use it in projects, your understanding will solidify.

Note that Khan Academy deals with "real spaces" ($\mathbb{R}$), the range of vectors and matrices that have real numbers as their entries.

In quantum mechanics and quantum computing, we'll be dealing with **Hilbert Spaces**, whose vectors and matrices can have complex numbers as well. This is a direct consequence of quantum mechanics: We'll talk about how why we need to have complex numbers to describe the quantum world—it's really cool!

Note: A 2-dimensional vector (a vector with 2 entries, such as $\begin{bmatrix}1\\0\end{bmatrix}$) of real numbers is said to belong to $\mathbb{R}^2$. A 2-dimensional vector of complex numbers (such as $\begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{i}{\sqrt{2}}\end{bmatrix}$) is said to belong to $\mathbb{C}^2$. We'll talk about why both are divided by $\sqrt{2}$.

Quantum Computing

Remember that the bit is the fundamental unit of information. On a classical (regular) computer, that can only be a 0 or a 1.

On a quantum computer, we can use **qubits**: quantum bits. Using the principles of quantum mechanics (which we'll get into), we can also represent combinations of 0 and 1—something that seemingly makes no sense!

Let's represent a qubit using the language of linear algebra. First, let's define the 0 and 1 state for a qubit. We'll be using **Dirac Notation** (AKA **bra-ket notation**) to assign names to vectors:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \quad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$$

The $|0\rangle$ and $|1\rangle$ are known as **kets**, as they are column vectors. A **bra vector** is a row vector, such as $\begin{bmatrix}1 & 0\end{bmatrix}$. We could write this vector in Dirac Notation like so: $\langle 0|$. We mostly deal with kets.

We'll be seeing these two everywhere, so you'll end up memorizing them sooner or later. A handy trick is to remember that the 0th element of the $|0\rangle$ state (qubit) is a 1, and the 1st element of the $|1\rangle$ state is a 1.

Because quantum computers also allow us to represent combinations of 0 and 1, the following is also a valid qubit state:

$$|\psi\rangle = \begin{bmatrix}\frac{1}{\sqrt{2}}\\\frac{i}{\sqrt{2}}\end{bmatrix}$$

This is **superposition**, and we'll talk about it along with the rest of quantum mechanics soon. Note that $|\psi\rangle$ is used to denote an arbitrary qubit state, much like x is the default variable in algebra.

Also note that the square of the norm of this vector, $|||\psi\rangle||^2$ is 1. There is a reason, and we'll talk about that, too.