1st chapter

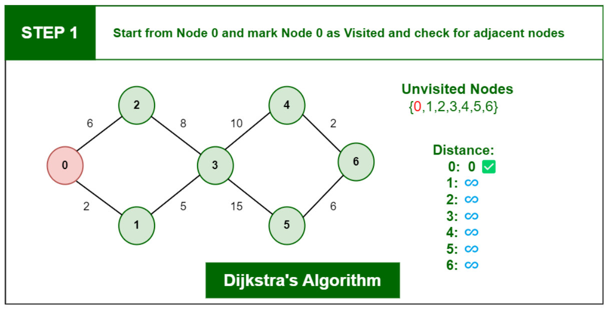
Tags Done✔️

❓ ***1. 2-1 | Give an example of an application that requires algorithmic content at the application level, and discuss the function of the algorithms involved.***

🚗 Imagine you find yourself in a city divided by a river, and your gas tank is perilously low. Your mission is to navigate the network of bridges to reach a specific destination efficiently. There are 10 bridges connecting the two sides of the city, each spanning different distances and neighborhoods.

The goal is clear: find the fastest and most economical route that ensures you don't run out of gas before reaching the gas stations or your destination.

🔎 This is a classic *Shortest Path* problem. We could use a *Dijkstra algorithm* to solve the problem. Dijkstra's algorithm is a well-suited algorithm for finding the shortest path in a weighted graph. In this context, each bridge is represented as a vertex in the graph, and the distances between the bridges are the weights of the edges. The gas stations and the destination point are also represented as vertices.



Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduc
Welcome! If you've always wanted to learn and understand Dijkstra's algorithm, then this article is for you. You will see how it works behind the scenes with a step-by-step graphical explanation. You will learn: * Basic
https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/

> 🔧 For example, we could use graphs to model a transportation network where nodes would represent facilities that send or receive products and edges would represent roads or paths that connect them — Estefania Navone, freeCodeCamp.

❓ ***1. 2-2 | Suppose that for inputs of size n on a particular computer, insertion sort runs in $8n^2$ steps and merge sort runs in $64nlogn$ steps. For which values of n does insertion sort beat merge sort?***

Insertion: $8n^2$steps

Merge: $64nlogn$ steps

- *Let's use an inequality*

We want to find the values of n for which $8n^2$ is less than 64*nlogn*:

$8n^2 < 64nlogn$ → *When insertion requires less time than merge.*

Divide both sides by 8 to simplify:

$n^2 < 8nlogn$

Divide both sides by n (assuming n is positive) to further simplify:

$n < 8logn$

❓ ***1.2-3 | What's the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?***

$100n^2 < 2^n$

$n^2 < \frac{2^n}{100}$

$n < \frac{\sqrt{2^n}}{100}$

$100n^2 < 2^n$

It seems we need, I dunno, a graph? Something more jazzy? Jazzier? So, I found a hint in one of Numerade's solutions, and I'll try it on my own. Breathe in and out, all right… A good first step is to design an equation that will represent the relationship between the running time of these two algorithms.

Let's use logarithmic properties

$log(100n^2) < log(2n)$

$log(100) + log(n^2) < n \cdot log(2)$

$2 + 2 \cdot log(n) < n \cdot log(2)$

$2 \cdot log(n) < n \cdot log(2) - 2$

Dividing both sides by $log(2)$:

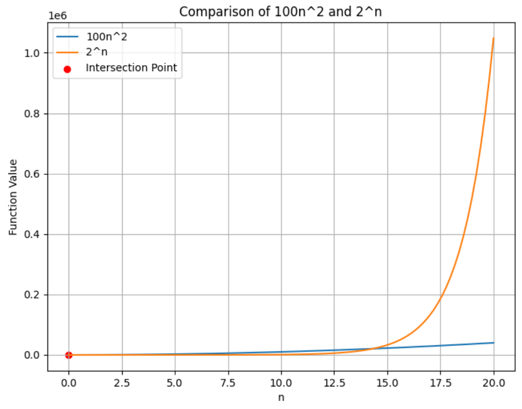
$2 \cdot \frac{log(2)}{log(n)} < \frac{n - log(2)}{2}$

$2 \cdot log2(n) < \frac{n - log(2)}{2}$

Hello darkness, my old friend…

People writing codes that… So I code a possible solution:

```
import matplotlib.pyplot as plt
import numpy as np


# Define the functions
n = np.linspace(0, 20, 100)
function1 = 100 * n**2
function2 = 2**n

# Plot the functions
plt.figure(figsize=(8, 6))
plt.plot(n, function1, label='100n^2')
plt.plot(n, function2, label='2^n')

# Mark the intersection point
intersection_point = np.argwhere(np.diff
(np.sign(function1 - function2))).[0]
plt.scatter(n[intersection_point], functi
on1[intersection_point], color='red', lab
el='Intersection Point')

# Add labels and legend
plt.xlabel('n')
plt.ylabel('Function Value')
plt.title('Comparison of 100n^2 and 2^n')
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



🤔 ***Additional doubts | What does influence the running time of an algorithmic structure?***

The specific architecture of the network, the nature of the problem being solved, and the hardware on which the algorithm is implemented.

And for SNNs?

SNNs are often implemented on specialized hardware, like neuromorphic chips, to take advantage of their parallel processing capabilities and better emulate biological neural systems. Since they model the spiking behavior of neurons, incorporating aspects of time and temporal dynamics into the neural computation, their running time is significantly impacted. That is, running time can be a bit elusive and nuanced when dealing with neural networks.

📖 The book’s recommendations: "For less technical material, see the books by **Christian and Griffiths, Cormen, Erwig, MacCormick, and Vöcking et al.**"