**Huffman Encoding**

**Presentation Portion**

**What is Huffman Encoding?**

This process of Huffman Encoding was established by David A. Huffman in his 1952 MIT paper “A Method for the Construction of Minimum-Redundancy Codes”. This paper addresses the problem of finding the most efficient binary code. Huffman Encoding remains an efficient method for lossless data compression. Lossless compression can be explained as minimizing the amount of data without losing any of the details. The data can be decompressed into the exact same as it was before compression.

It functions by attaching some optimal length code (prefix code) to each input value. Values that are more common will evaluate to a shorter-bit prefix code, while codes for less common inputs will consist of more characters. The length of this code represents frequency of occurrence. Huffman Encoding uniquely eliminates ambiguity when decoding. The process of Huffman Encoding requires two major steps: building a Huffman Tree from input characters and traversing through the tree to assign codes to each input value.

**How can Huffman Encoding be implemented?**

Huffman Encoding can be implemented to reduce the size of virtually anything. It is most often implemented to reduce file size. Digital data compression is useful for minimizing the space that files occupy on a hard drive and reducing the time needed to transfer such files. This remains a crucial aspect of technology. Almost everything done digitally is compressed. For example, once opening a website browser, the web server that delivers the web page most often compresses the data. When the browser received the compressed version, it uncompressed it. Every image, audio, and video file is compressed. Most current encoding involves a version of this process!

**What is the space usage for Huffman Encoding?**

Huffman Encoding efficiently encodes some input in terms of space. The space required to store the prefix codes is dependent on the frequency distribution of the values in the input data. The space complexity is usually proportional to the size of the data (the number of unique values in it). Therefore, the most frequent space complexity remains O(n). However, there remains a rare possibility that each input character has identical frequency, which would result in a worst-case space complexity of O(nlogn). Huffman Encoding is widely used as it exhibits excellent compression efficiency!

**What are the runtimes for Huffman Encoding?**

The time complexity for Huffman Encoding remains O(nlogn) for the number of unique characters, n. This is because each node is being accessed (O(n) runtime) and rearranged (O(logn) runtime). The worst case for Huffman Encoding occurs when the frequency distribution follows the Fibonacci sequence or when the probability of a symbol exceeds 0.5. In the very unlikely best case, the input array is already sorted which the algorithm handles in linear time, O(n).

**\*\* Should I include this information in the slides and where?**

Huffman Encoding is considered a greedy algorithm and remains one of the most successful algorithms. Greedy algorithms attempt to find a global minimum/maximum through small-grained or local minimal/maximal choices. This process leads to optimal character encoding.

**What are our sources?**

[34-Huffman-Encoding (stanford.edu)](https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1132/handouts/34-Huffman-Encoding.pdf)

[Huffman Coding | Greedy Algo-3 - GeeksforGeeks](https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/)

[Huffman Code | Brilliant Math & Science Wiki](https://brilliant.org/wiki/huffman-encoding/)

[Huffman Coding Algorithm (programiz.com)](https://www.programiz.com/dsa/huffman-coding)