200) Given a set of characters and their corresponding frequencies, construct the Huffman Tree and generate the Huffman Codes for each character.

Test Case 1:
Input:
n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
Output: [('a', '110'), ('b', '10'), ('c', '0'), ('d', '111')]
Test Case 2:
Input:
n = 6
characters = ['f', 'e', 'd', 'c', 'b', 'a']
frequencies = [5, 9, 12, 13, 16, 45]
Output: [('a', '0'), ('b', '101'), ('c', '100'), ('d', '111'), ('e', '1101'), ('f', '1100')]

AIM: To write a python program for corresponding frequencies, construct the Huffman Tree and generate the Huffman Codes for each character.

PROGRAM:

```
import heapq
from collections import defaultdict

class Node:
    def __init__(self, freq, char=None, left=None, right=None):
        self.freq = freq
        self.char = char
        self.left = left
        self.right = right

    def __lt__(self, other):
        return self.freq < other.freq

def calculate_frequencies(text):
    frequencies = defaultdict(int)
    for char in text:
        frequencies[char] += 1
    return frequencies

def build_huffman_tree(frequencies):
    heap = [Node(freq, char) for char, freq in frequencies.items()]
    heapq.heapify(heap)
```

```
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.freq + right.freq, left=left, right=right)
        heapq.heappush(heap, merged)

    return heap[0]

def generate_huffman_codes(node, prefix='', codebook={}):
    if node.char is not None:
        codebook[node.char] = prefix
    else:
        generate_huffman_codes(node.left, prefix + '0', codebook)
        generate_huffman_codes(node.right, prefix + '1', codebook)
    return codebook

def huffman_encoding(text):
    frequencies = calculate_frequencies(text)
    huffman_tree = build_huffman_tree(frequencies)
    huffman_codes = generate_huffman_codes(huffman_tree)
    return huffman_codes

def main():
    text = "this is an example for huffman encoding"
    huffman_codes = huffman_encoding(text)
    for char, code in huffman_codes.items():
        print(f"Character: {char} | Huffman Code: {code}")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Character: n | Huffman Code: 000
```

TIME COMPLEXITY :O(n+mlog m)