7.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')]

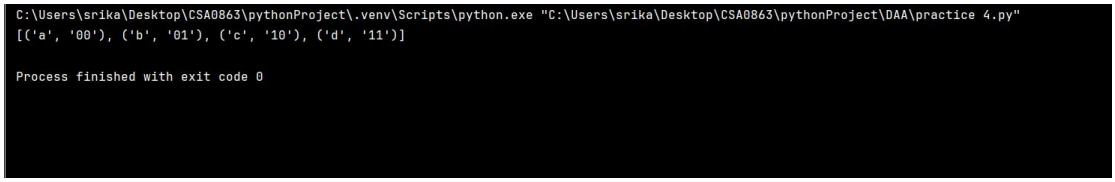
Program:

```
import heapq

class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None
    def __lt__(self, other):
        return self.freq < other.freq
def huffman_codes(characters, frequencies):
    min_heap = []
    for char, freq in zip(characters, frequencies):
        heapq.heappush(min_heap, Node(char, freq))
    while len(min_heap) > 1:
        left = heapq.heappop(min_heap)
        right = heapq.heappop(min_heap)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
```

```
        heapq.heappush(min_heap, merged)
    root = min_heap[0]
    huffman_code = {}
    def generate_codes(node, code):
        if node is not None:
            if node.char is not None:
                huffman_code[node.char] = code
            generate_codes(node.left, code + '0')
            generate_codes(node.right, code + '1')
    generate_codes(root, "")
    return sorted(huffman_code.items())
n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
print(huffman_codes(characters, frequencies))
```

Output:



Time complexity:

O(nlogn)