8.Given a Huffman Tree and a Huffman encoded string, decode the string to get the original message.

Test Case 1:

Input:

n = 4

characters = ['a', 'b', 'c', 'd']

frequencies = [5, 9, 12, 13]

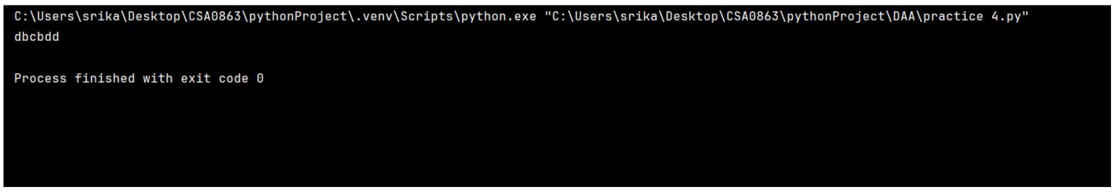
encoded_string = '1101100111110'

Output: "abacd"

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 build_huffman_tree(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)
    return min_heap[0]


def decode_huffman_tree(root, encoded_string):
    decoded_message = []
    current_node = root
    for bit in encoded_string:
        if bit == '0':
            current_node = current_node.left
        else:
            current_node = current_node.right
        if current_node.char is not None:
            decoded_message.append(current_node.char)
            current_node = root
    return ''.join(decoded_message)
n = 4
characters = ['a', 'b', 'c', 'd']
frequencies = [5, 9, 12, 13]
encoded_string = '1101100111110'
root = build_huffman_tree(characters, frequencies)
decoded_message = decode_huffman_tree(root, encoded_string)
print(decoded_message)
```

Output:



```
C:\Users\srika\Desktop\CSA0863\pythonProject\.venv\Scripts\python.exe "C:\Users\srika\Desktop\CSA0863\pythonProject\DAA\practice 4.py"
dbcbdd

Process finished with exit code 0
```

Time complexity:

$O((n+L)\log n)$