92. huffman codeing problem
Program:

```
# Python program for Huffman Coding
import heapq

class Node:
    def __init__(self, symbol=None, frequency=None):
        self.symbol = symbol
        self.frequency = frequency
        self.left = None
        self.right = None

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

def build_huffman_tree(chars, freq):

    # Create a priority queue of nodes
    priority_queue = [Node(char, f) for char, f in zip(chars, freq)]
    heapq.heapify(priority_queue)
    while len(priority_queue) > 1:
        left_child = heapq.heappop(priority_queue)
        right_child = heapq.heappop(priority_queue)
        merged_node = Node(frequency=left_child.frequency + right_child.frequency)
        merged_node.left = left_child
        merged_node.right = right_child
        heapq.heappush(priority_queue, merged_node)

    return priority_queue[0]

def generate_huffman_codes(node, code="", huffman_codes={}):
    if node is not None:
        if node.symbol is not None:
            huffman_codes[node.symbol] = code
        generate_huffman_codes(node.left, code + "0", huffman_codes)
        generate_huffman_codes(node.right, code + "1", huffman_codes)

    return huffman_codes
chars = ['a', 'b', 'c', 'd', 'e', 'f']
freq = [4, 7, 15, 17, 22, 42]

# Build the Huffman tree
root = build_huffman_tree(chars, freq)
huffman_codes = generate_huffman_codes(root)
for char, code in huffman_codes.items():
    print(f"Character: {char}, Code: {code}")
```

Output:

```
Character: f, Code: 0
Character: a, Code: 1000
Character: b, Code: 1001
Character: c, Code: 101
Character: d, Code: 110
Character: e, Code: 111

=== Code Execution Successful ===
```

Time complexity:O(nlogn)