Q6. Implement Huffman Coding. Analyze its time complexity.

TIME COMPLEXITY: O(L + N * log(N))

SPACE COMPLEXITY: O(N)

where L is the length of the input file and N is the number of the unique characters.

```
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <map>
#include <cmath>


using namespace std;



class minHeapNode
{
public:
    char data;
    int freq;
    char representative;
    minHeapNode * left;
    minHeapNode * right;
    minHeapNode(char data, int freq) : data(data), freq(freq),
     representative(data), left(NULL), right(NULL) {}
};

class myComparator
{
public:
    int operator() (const minHeapNode * p1, const minHeapNode * p2)
    {
        if(p1 -> freq == p2 -> freq)
            return p1 -> representative >= p2 -> representative; // ASCII
             cmoparison
        else return p1 -> freq > p2 -> freq;
    }
};


minHeapNode * buildHuffmanTree(map<char, int> & file)
{
    priority_queue<minHeapNode* , vector<minHeapNode*>, myComparator>
     minHeap;
    map<char, int> :: iterator it;
    for (auto it : file)
    {
        minHeap.push(new minHeapNode(it.first, it.second));
```

```
    }

    while (minHeap.size() != 1)
    {
        minHeapNode * first = minHeap.top();
        minHeap.pop();
        minHeapNode * second = minHeap.top();
        minHeap.pop();
        minHeapNode * N1 = new minHeapNode('$', first -> freq + second ->
         freq);
        N1 -> left = first;
        N1 -> right = second;
        N1 -> representative = min({N1 -> left -> representative, N1 ->
         right -> representative});
        minHeap.push(N1);
    }

    return minHeap.top();
}

void getCodes(minHeapNode * root, string code, map<char, string> & codes)
{
    if(!root)
    {
        return;
    }
    if (root -> data != '$')
    {
        codes.insert({root -> data, code});
    }
    getCodes(root -> left, code + '0', codes);
    getCodes(root -> right, code + '1', codes);
}

string decode_file(minHeapNode * root, string code)
{
    string ans = "";
    minHeapNode * curr = root;
    for (int i = 0; i < code.size(); i++)
    {
        if (code[i] == '0')
        {
            curr = curr -> left;
        }
        else curr = curr -> right;
        if (!curr -> left && !curr -> right)
        {
            ans += curr -> data;
            curr = root;
        }
    }
    return ans + '\0';
}

void levelOrderTraversal(minHeapNode * root)
```

```
{
    queue<minHeapNode*> q;
    q.push(root);
    q.push(NULL);
    while (1)
    {
        minHeapNode * top = q.front();
        q.pop();
        if(!top)
        {
            cout << endl;
            if(q.empty())
                break;
            q.push(NULL);
        }
        else
        {
            cout << top -> data << " : " << top -> representative << "\t";
            if(top -> left)
                q.push(top -> left);
            if(top -> right)
                q.push(top -> right);
        }
    }
}

int main()
{
    string text = "I love programming!";
    map<char, int> file;
    for (int i = 0; i < text.size(); i++)
    {
        if (file.count(text[i]) == 0)
        {
            file[text[i]] = 1;
        }
        else file[text[i]]++;
    }
    for(auto it : file)
    {
        cout << it.first << ": " << it.second << " " << int(it.first) <<
         endl;
    }
    minHeapNode * root = buildHuffmanTree(file);
    map<char, string> codes;
    getCodes(root, "", codes);

    for(auto it : codes)
    {
        cout << it.first << ": " << it.second << "  " << (int)it.first <<
         endl ;
    }
    string encoded = "";
    for (int i = 0; i < text.size(); i++)
    {
```

```
        encoded += codes.at(text[i]);
    }
    float size = float(encoded.size());

    cout << "Average Length: " << round (1.0 * size * 100/ text.size())/100
     << " bits/symbol" << endl;
    cout << "Encoded file : " << encoded << endl;
    cout << "File Decoded : " << decode_file(root, encoded) << endl;
}
```

OUTPUT:

```
Input Text: I love programming!

 : 2 32
!: 1 33
I: 1 73
a: 1 97
e: 1 101
g: 2 103
i: 1 105
l: 1 108
m: 2 109
n: 1 110
o: 2 111
p: 1 112
r: 2 114
v: 1 118
 : 1101  32
!: 11100  33
I: 11101  73
a: 11110  97
e: 11111  101
g: 000  103
i: 0010  105
l: 0011  108
m: 010  109
n: 0110  110
o: 100  111
p: 0111  112
r: 101  114
v: 1100  118
Average Length: 3.79 bits/symbol
Encoded file :
 1110111010011100110011111111010111101100000101111100100100010011000011100
File Decoded : I love programming!
Program ended with exit code: 0
```