DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 7

NAME: Chepoori Sai Vivek

ROLL NUMBER: CH.SC.U4CSE24108

CLASS: CSE-B

Huffman Coding:

DATA ANALYTICS AND INTELLIGENCE LABORATORY

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
struct Node {
    char data;
    int freq;
    struct Node *left, *right;
};
struct Node* createNode(char data, int freq) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->freq = freq;
    node->left = node->right = NULL;
    return node;
}
void sort(struct Node* arr[], int n) {
    for(int i = 0; i < n-1; i++) {
        for(int j = i+1; j < n; j++) {
            if(arr[i]->freq > arr[j]->freq) {
                struct Node* temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
void printCodes(struct Node* root, int code[], int top,
                int *totalBits, int *totalFreq) {

    if(root->left) {
        code[top] = 0;
        printCodes(root->left, code, top+1, totalBits, totalFreq);
    }
    if(root->right) {
        code[top] = 1;
        printCodes(root->right, code, top+1, totalBits, totalFreq);
    }
```

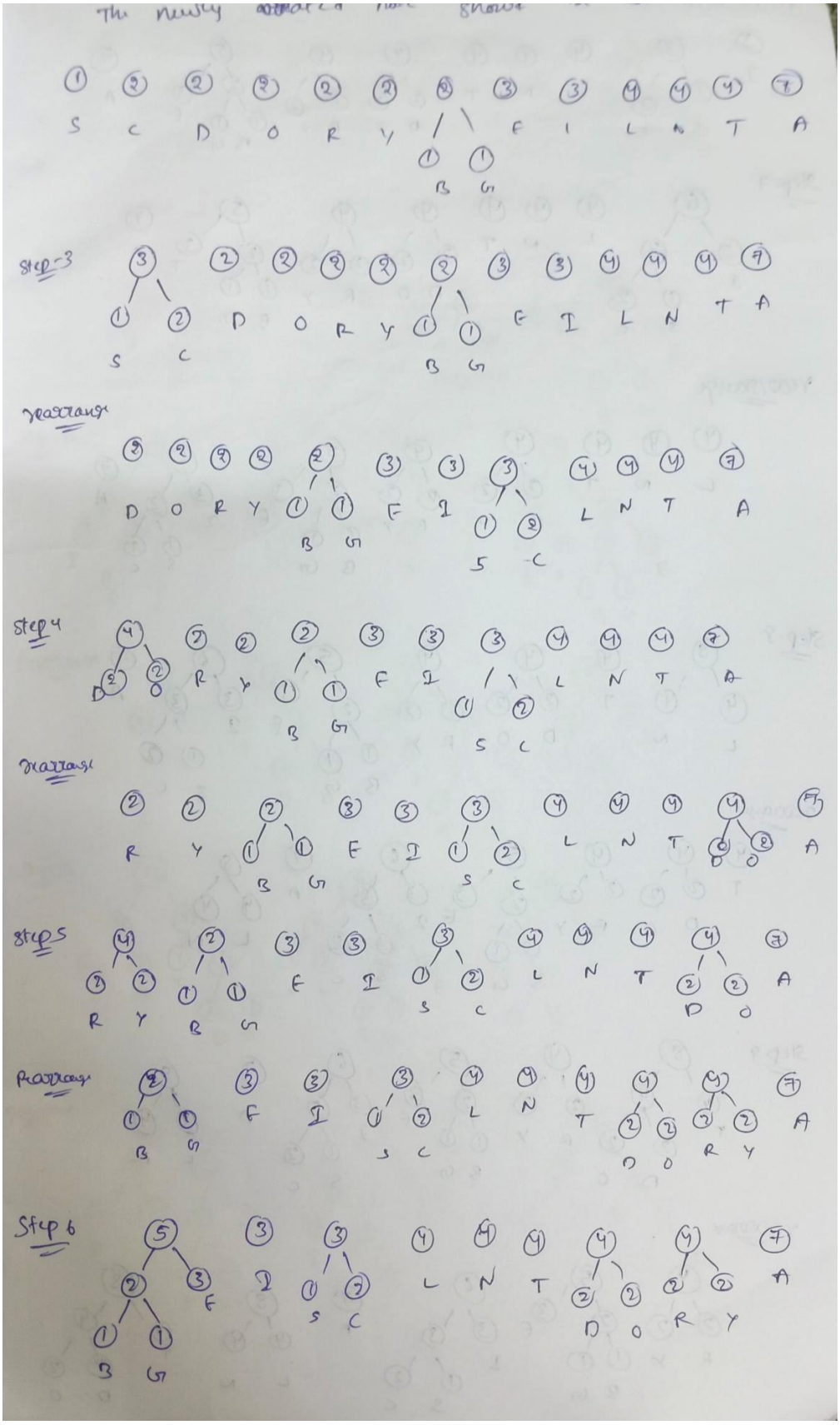
```
    if(!root->left && !root->right) {
        printf("%c : ", root->data);
        for(int i = 0; i < top; i++)
            printf("%d", code[i]);
        printf("  (freq=%d, length=%d)\n", root->freq, top);
        *totalBits += root->freq * top;
        *totalFreq += root->freq;
    }
}
int main() {
    char text[] = "DATA ANALYTICS AND INTELLIGENCE LABORATORY";
    int freq[256] = {0};
    for(int i = 0; text[i]; i++) {
        if(text[i] != ' ')
            freq[(int)text[i]]++;
    }
    struct Node* nodes[MAX];
    int n = 0;
    for(int i = 0; i < 256; i++) {
        if(freq[i] > 0) {
            nodes[n++] = createNode((char)i, freq[i]);
        }
    }
    while(n > 1) {
        sort(nodes, n);
        struct Node* left = nodes[0];
        struct Node* right = nodes[1];
        struct Node* newNode = createNode('$',
                                left->freq + right->freq);
        newNode->left = left;
        newNode->right = right;
        nodes[0] = newNode;
        nodes[1] = nodes[n-1];
        n--;
    }
    struct Node* root = nodes[0];
    int code[100], totalBits = 0, totalFreq = 0;
    printf("Huffman Codes:\n\n");
    printCodes(root, code, 0, &totalBits, &totalFreq);
```

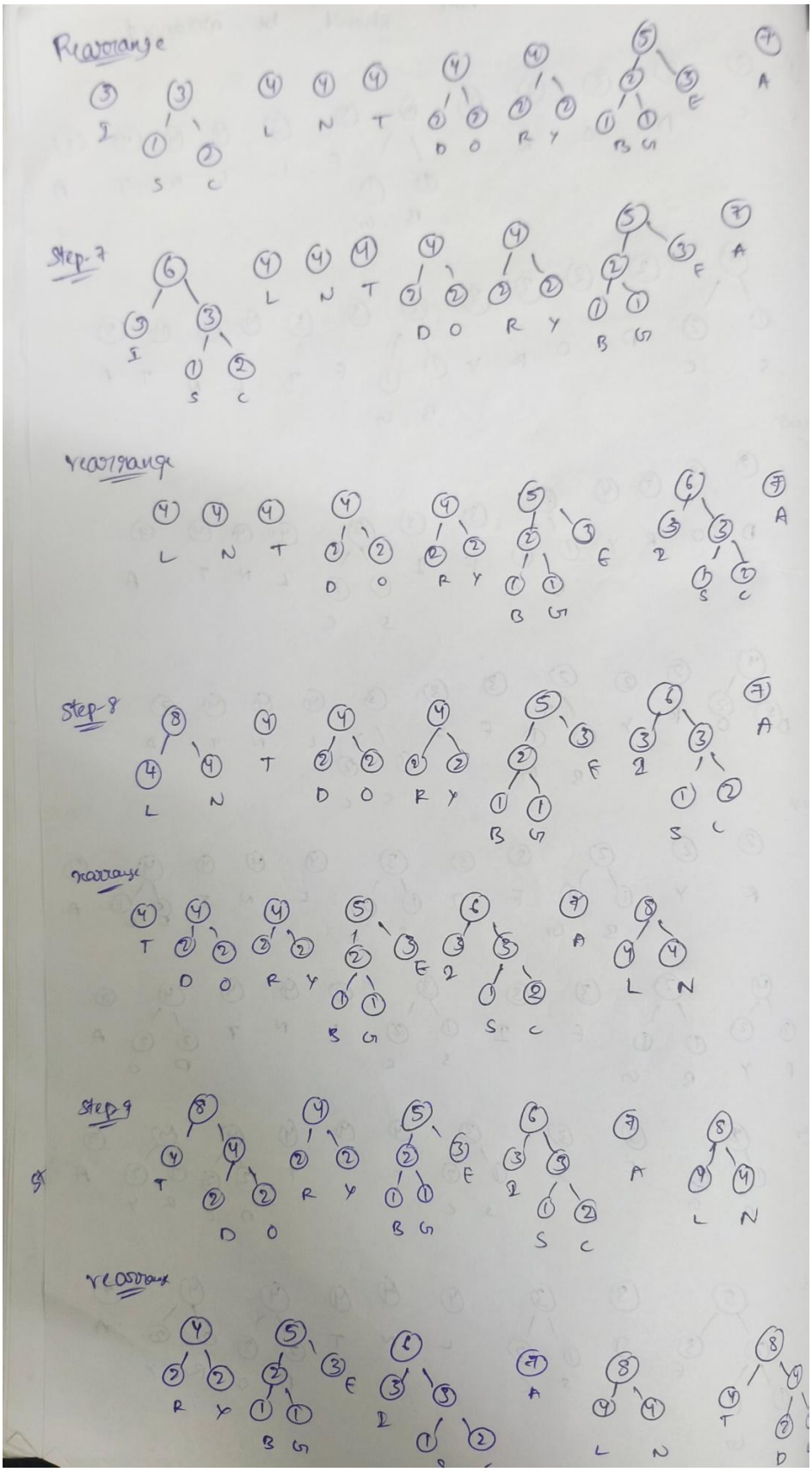
```
    printf("\nTotal Compressed Bits = %d\n", totalBits);
    float avg = (float)totalBits / totalFreq;
    printf("Average Code Length = %.2f bits\n", avg);
    return 0;
}
```

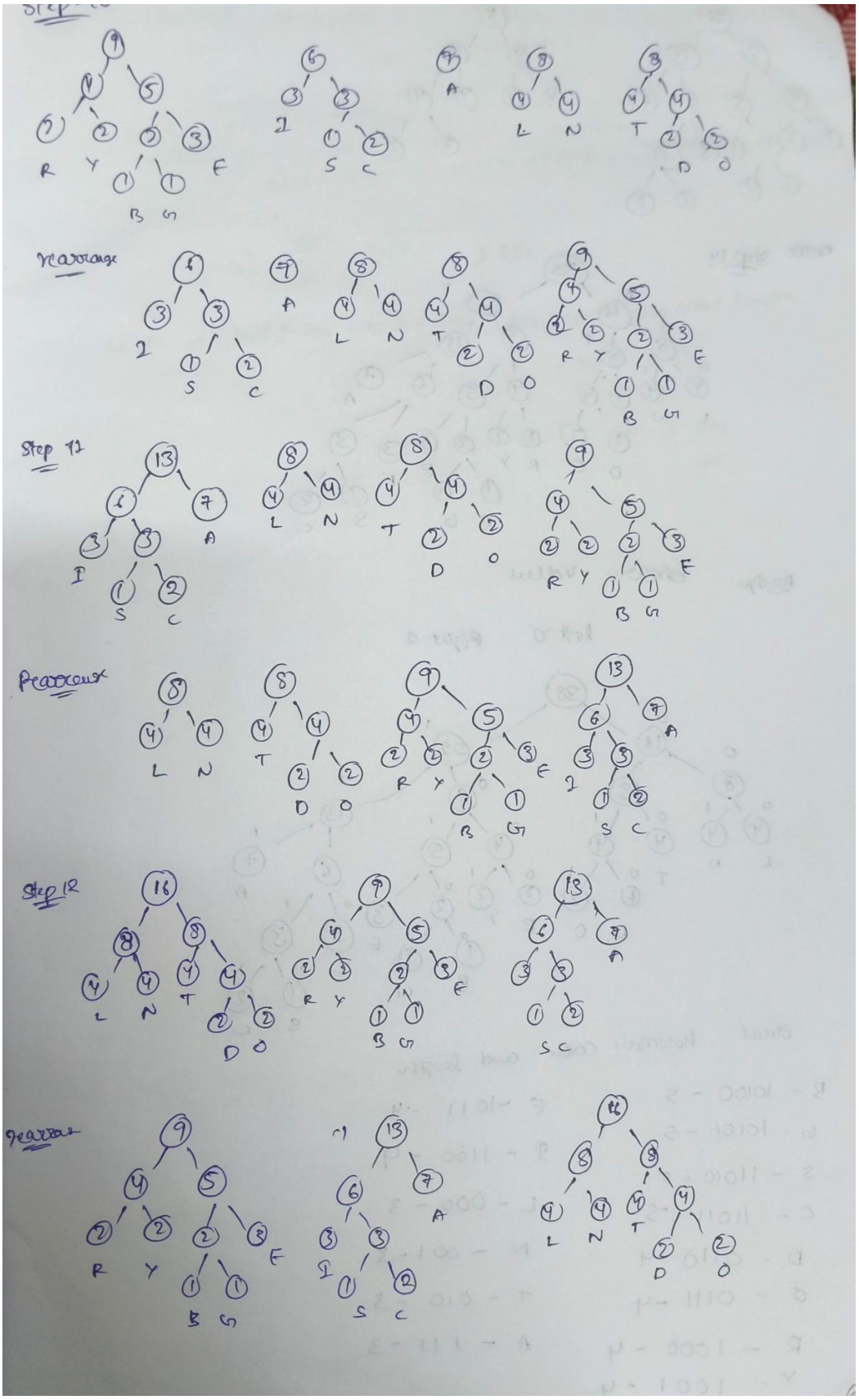
Output:

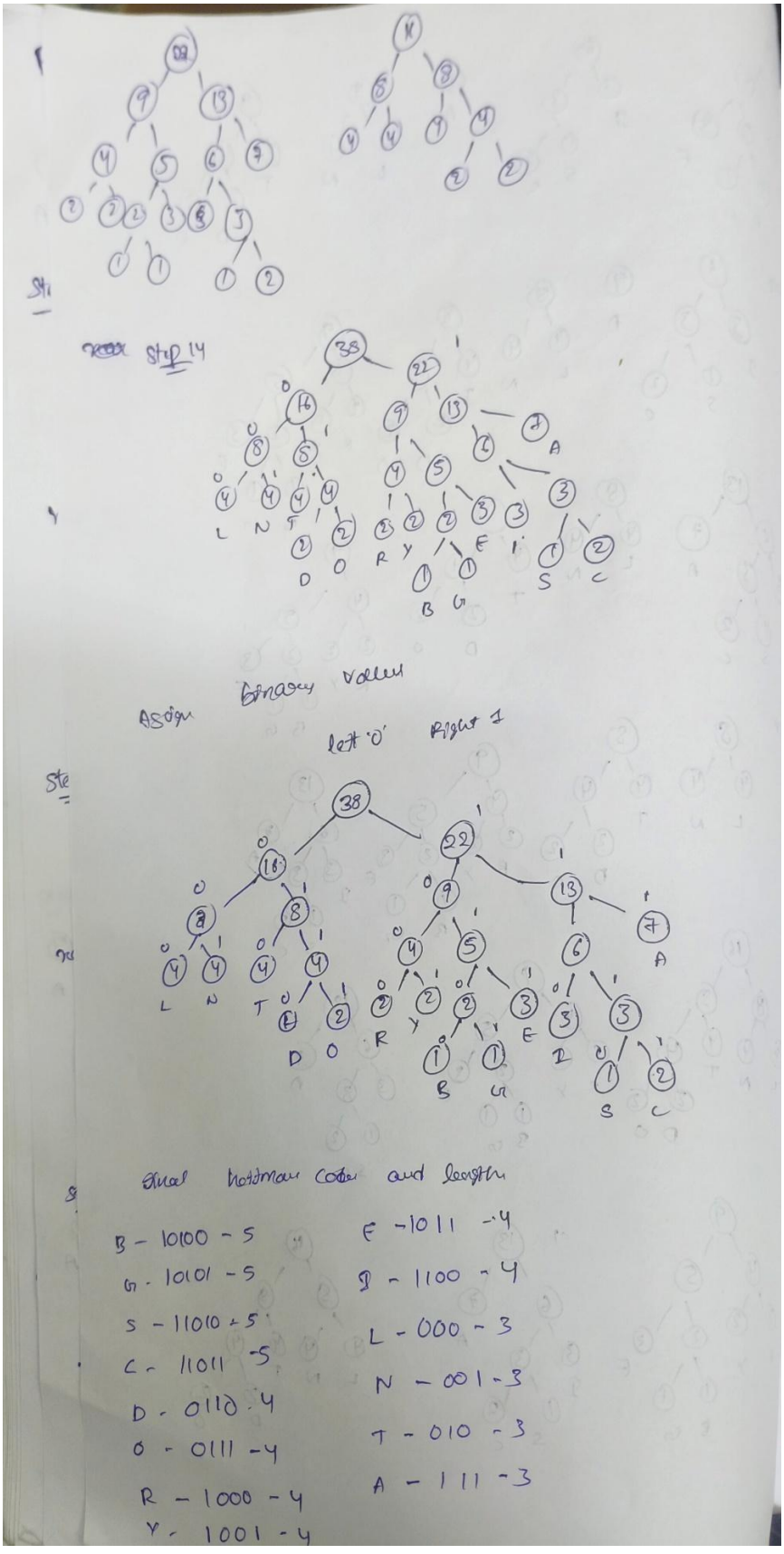
```
Huffman Codes:

R : 0000  (freq=2, length=4)
D : 0001  (freq=2, length=4)
C : 0010  (freq=2, length=4)
O : 0011  (freq=2, length=4)
L : 010  (freq=4, length=3)
T : 011  (freq=4, length=3)
N : 100  (freq=4, length=3)
Y : 1010  (freq=2, length=4)
S : 10110  (freq=1, length=5)
B : 101110  (freq=1, length=6)
G : 101111  (freq=1, length=6)
E : 1100  (freq=3, length=4)
I : 1101  (freq=3, length=4)
A : 111  (freq=7, length=3)

Total Compressed Bits = 138
Average Code Length = 3.63 bits
```

Working:




Rearrange
Step-7
rearrange
Step-8
rearrange
Step 9
rearrange


Step
R Y B S F
I S C
A
L N
T D O
Rearrange
Step 11
Rearrange
Step 12
Rearrange


Step 14
Assign binary values
left '0' Right 1
Final Huffman Codes and lengths
B - 10100 - 5
G - 10101 - 5
S - 11010 - 5
C - 11011 - 5
D - 0110 - 4
O - 0111 - 4
R - 1000 - 4
Y - 1001 - 4
E - 1011 - 4
I - 1100 - 4
L - 000 - 3
N - 001 - 3
T - 010 - 3
A - 111 - 3

Average code-length = $\frac{\sum (f_i \times l_i)}{\sum f_i}$

$$= \frac{(1\times5) + (1\times5) + (1\times5) + (2\times5) + 2(4) + 2(4) + 2(4) + 3(4) + 3(3) + 4(3) + 4(3) + 4(3) + 4(3) + 3(3)}{1+1+1+2+2+2+2+3+3+4+4+4+4+3}$$

$$= \frac{138}{38} = 3.631 \approx 3.631$$

Length of not many encoded msg = total freq × avg code length

$$= 38 \times 3.631$$

$$= 137.97$$

$$\approx 138 \text{ bits}$$

=

Time Complexity:

The algorithm repeatedly sorts the nodes in ascending order and merges the two smallest nodes.
Since Bubble Sort is used inside a loop, sorting is done multiple times.

- Best / Average Case = $O(n^3)$
- Worst Case = $O(n^3)$

Space Complexity:

Space is required for storing the Huffman tree and node list.
Recursion is used to generate codes.

- Average Case = $O(n)$
- Worst Case = $O(n)$