DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 8

NAME: Konda Bhavani

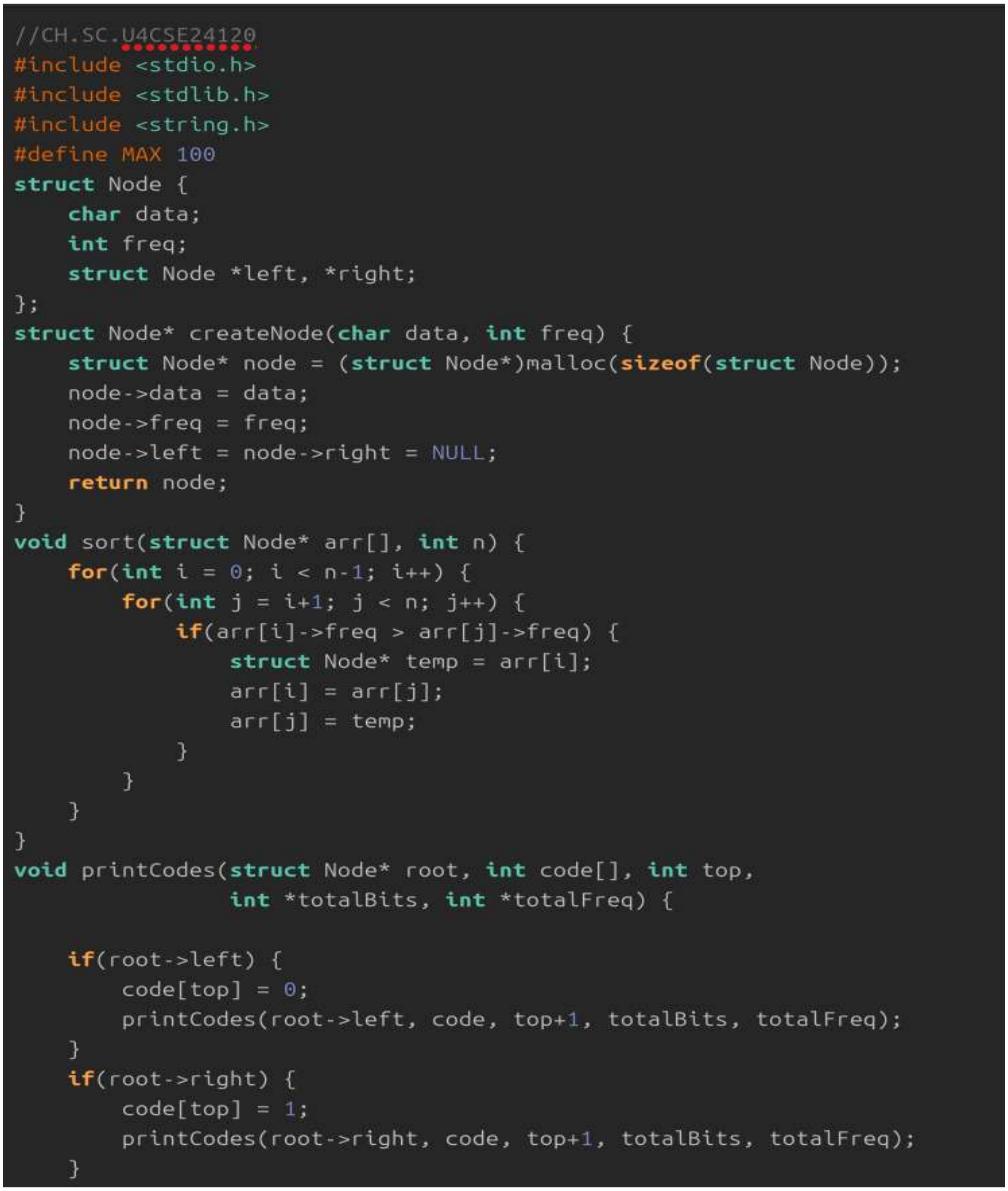
ROLL NUMBER: CH.SC.U4CSE24120

CLASS: CSE-B

Huffman Coding:

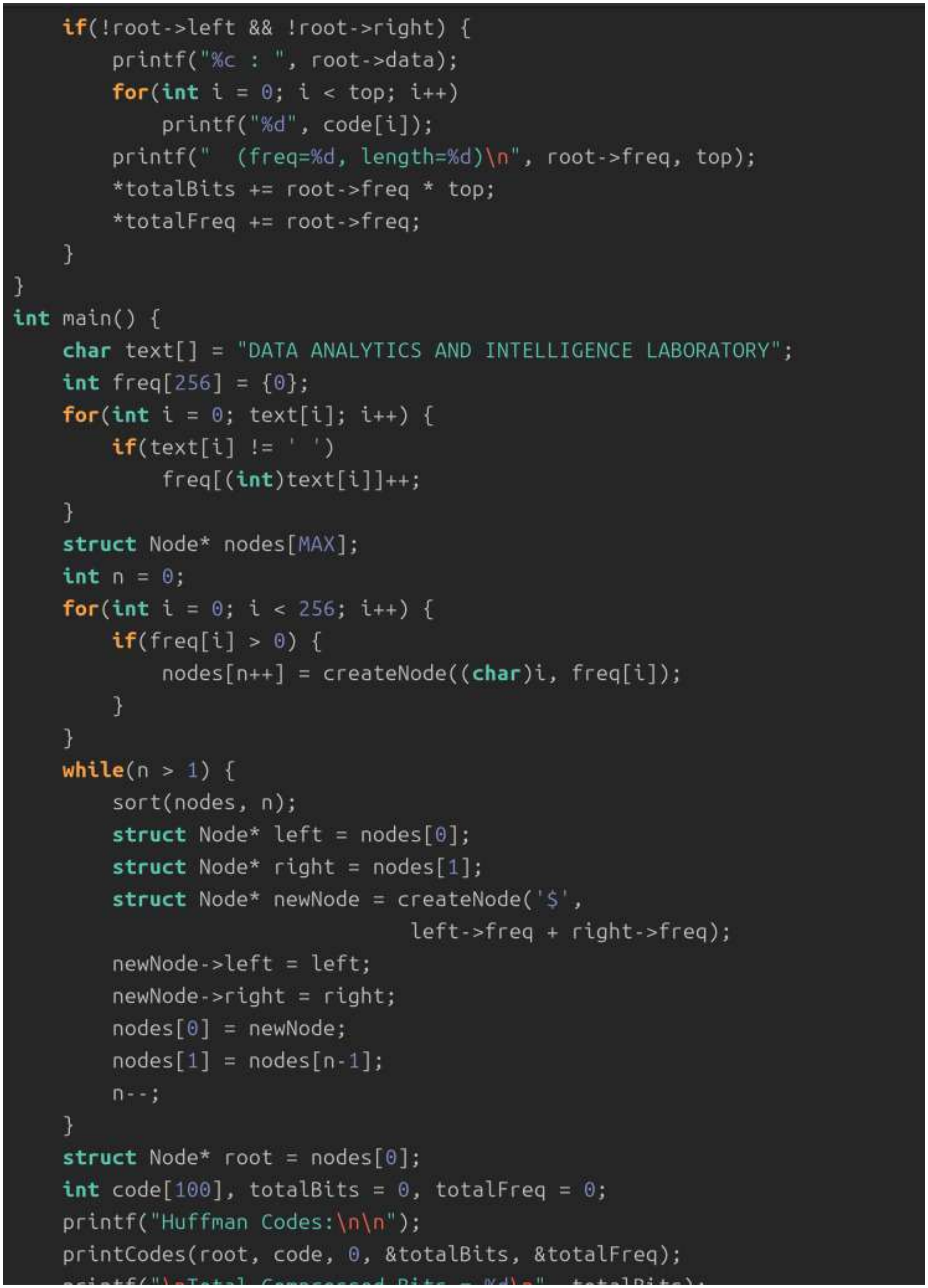
DATA ANALYTICS AND INTELLIGENCE LABORATORY

Code:

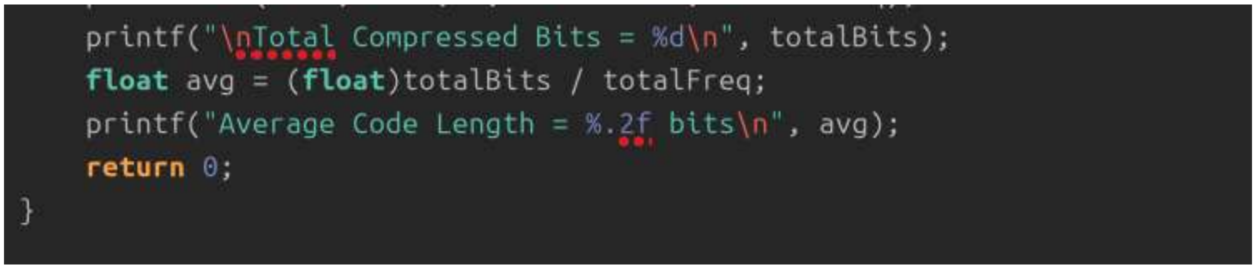


```c
//CH.SC.U4CSE24120
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
struct Node {
    char data;
    int freq;
    struct Node *left, *right;
};
struct Node* createNode(char data, int freq) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = 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);
    }
```

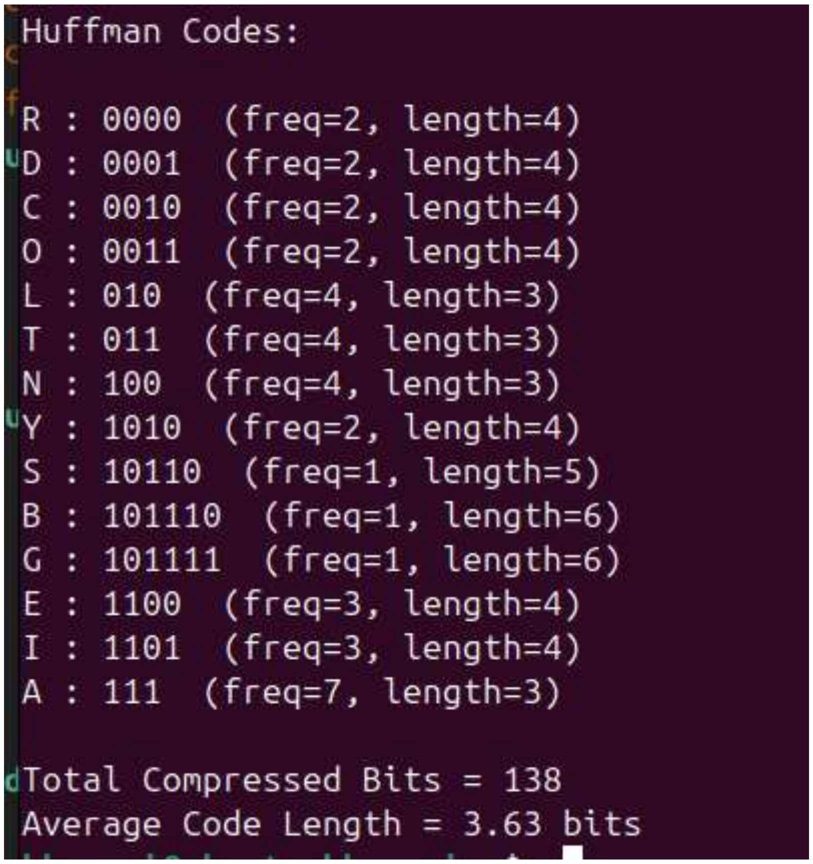


```c
    if(!root->left && !root->right) {
        printf("%c : ", root->data);
        for(int i = 0; i < top; i++)
            printf("%d", code[i]);
        printf("  (freq=%d, length=%d)\n", root->freq, top);
        *totalBits += root->freq * top;
        *totalFreq += root->freq;
    }
}
int main() {
    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);
```



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

Output:

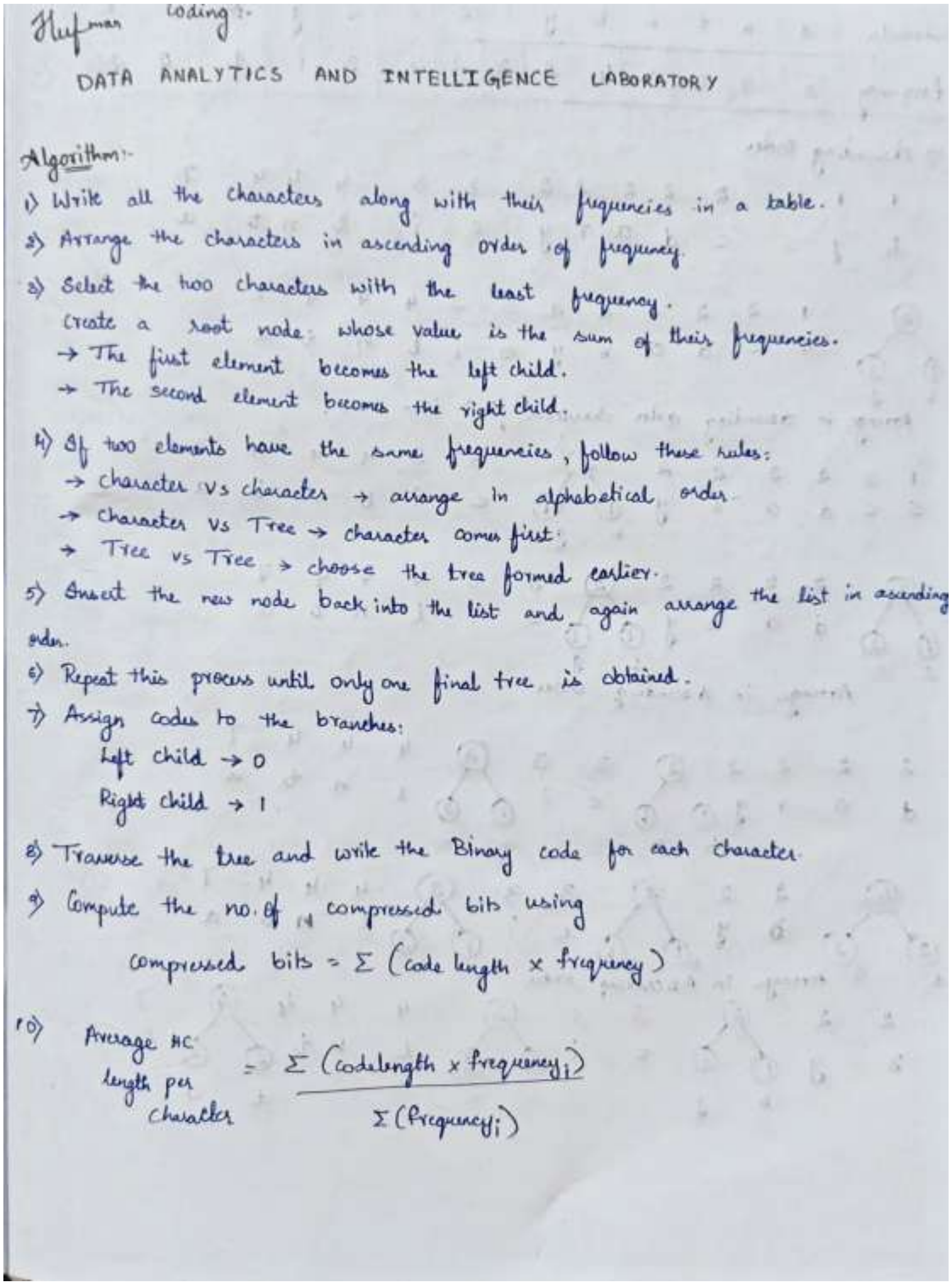


```
Huffman Codes:

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:



Huffman coding:-

DATA ANALYTICS AND INTELLIGENCE LABORATORY

Algorithm:-

1) Write all the characters along with their frequencies in a table.

2) Arrange the characters in ascending order of frequency.

3) Select the two characters with the least frequency.
Create a root node whose value is the sum of their frequencies.
→ The first element becomes the left child.
→ The second element becomes the right child.

4) If two elements have the same frequencies, follow these rules:
→ character vs character → arrange in alphabetical order
→ Character vs Tree → character comes first
→ Tree vs Tree → choose the tree formed earlier.

5) Insert the new node back into the list and again arrange the list in ascending order.

6) Repeat this process until only one final tree is obtained.

7) Assign codes to the branches:
Left child → 0
Right child → 1

8) Traverse the tree and write the Binary code for each character

9) Compute the no. of compressed bits using

$$\text{compressed bits} = \Sigma\,(\text{code length} \times \text{frequency})$$

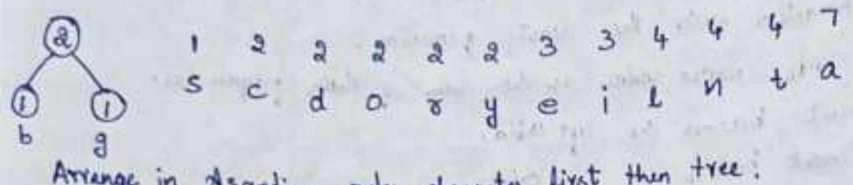
10) Average HC length per character

$$= \frac{\Sigma\,(\text{codelength} \times \text{frequency}_i)}{\Sigma\,(\text{Frequency}_i)}$$

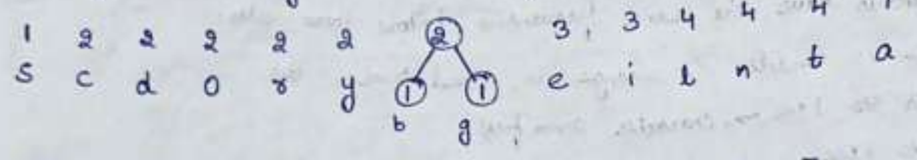
Character	d	a	t	n	l	y	i	c	s	e	g	b	o	[illegible]
frequency	2	7	4	4	4	2	3	2	1	3	1	1	2	2

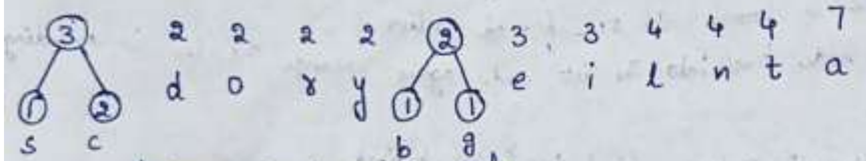
⇒ Ascending order

1	1	1	2	2	2	2	2	3	3	4	4	4	7
b	g	s	c	d	o	r	y	e	i	l	n	t	a

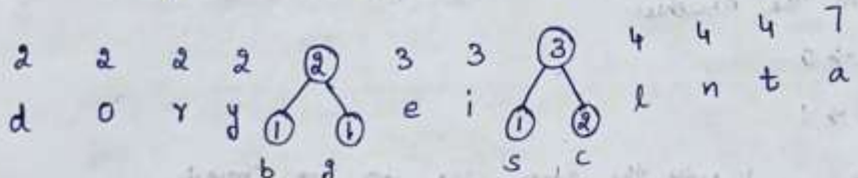


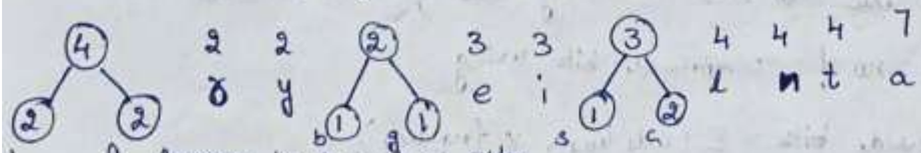
Arrange in Ascending order character first then tree.



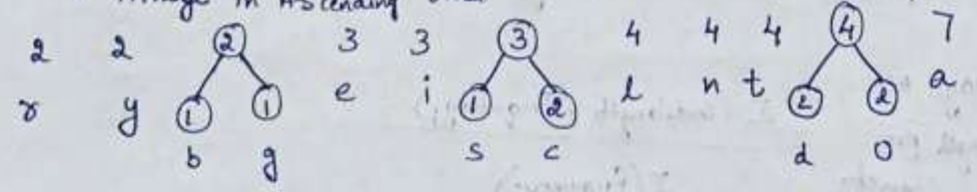


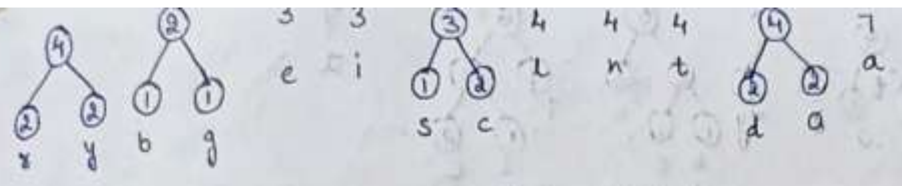
Arrange in Ascending order



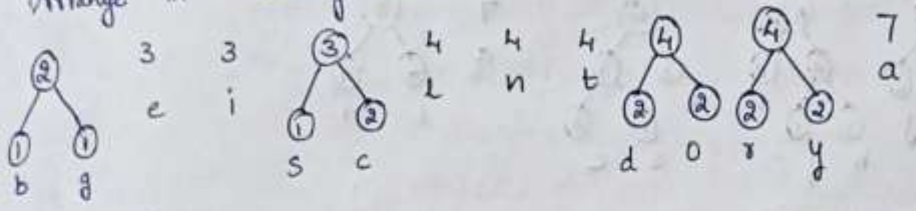


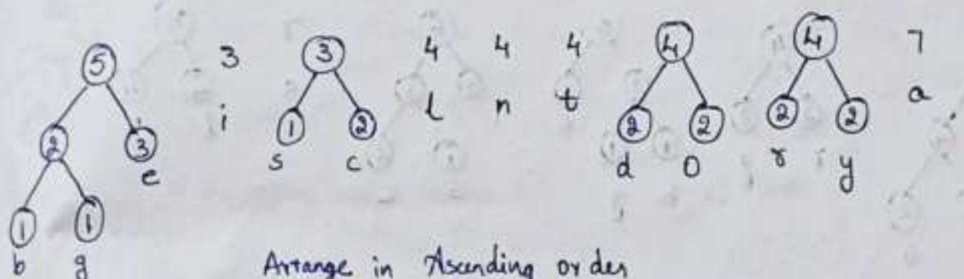
Arrange In Ascending order



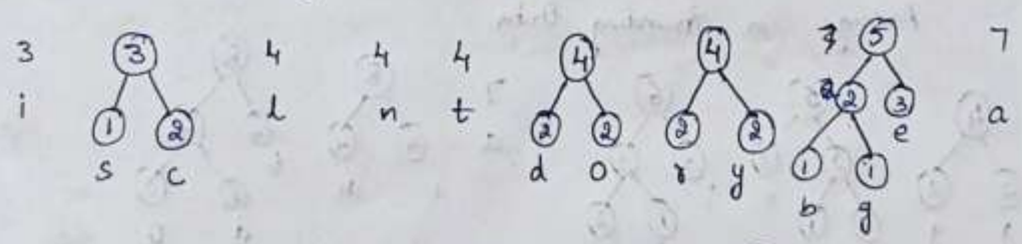


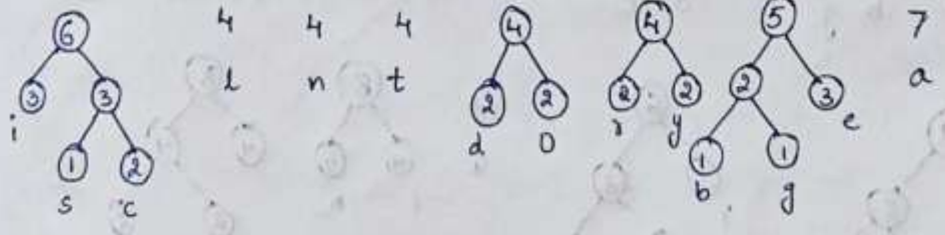
Arrange in Ascending order



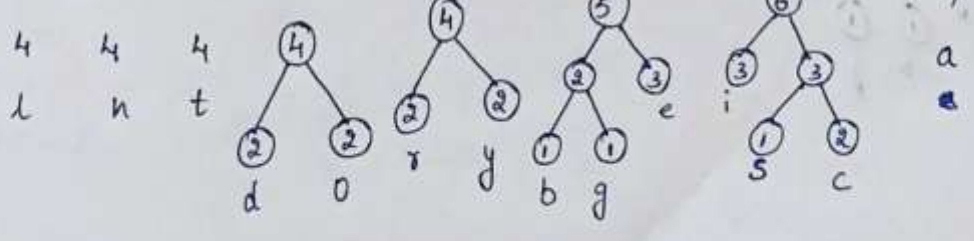


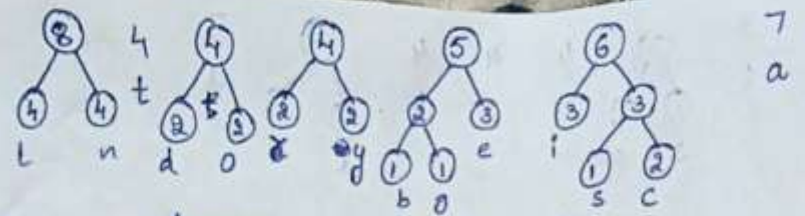
Arrange in Ascending order



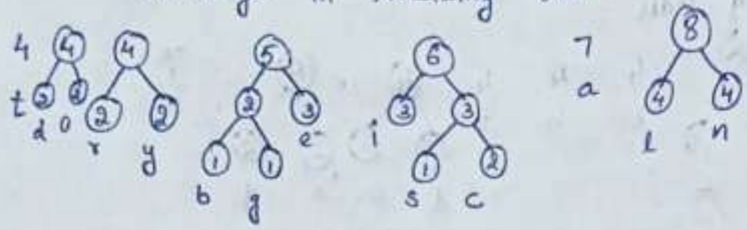


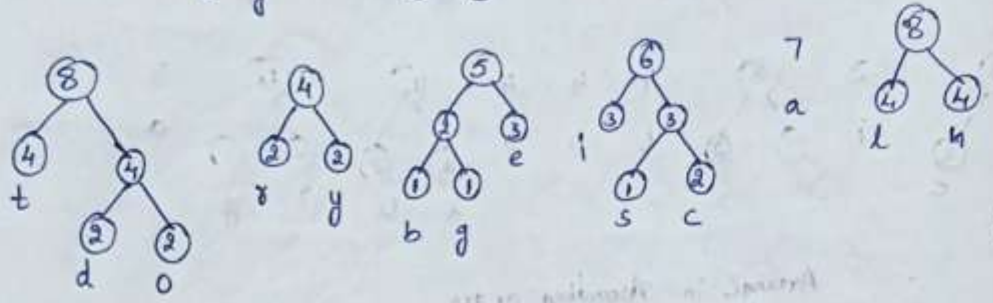
Arrange in Ascending order



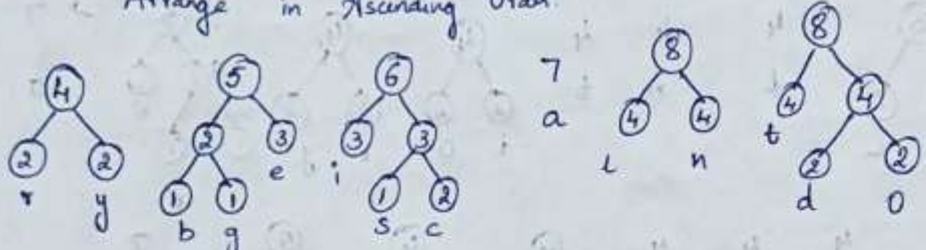


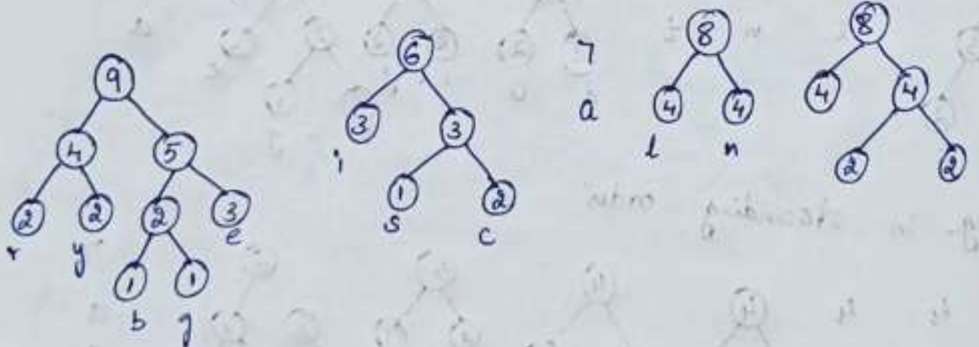
Arrange in Ascending Order



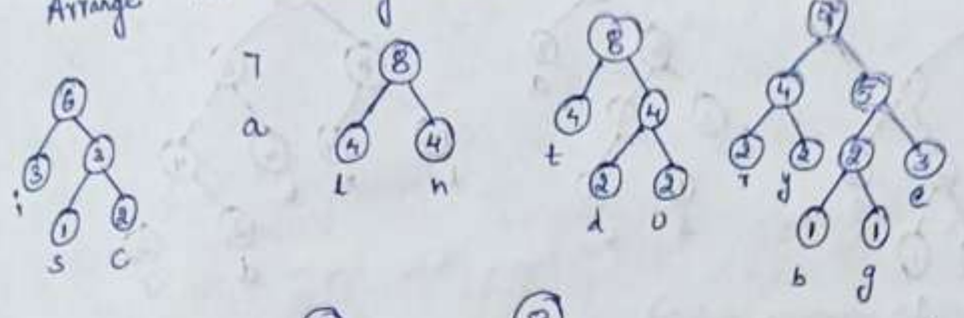


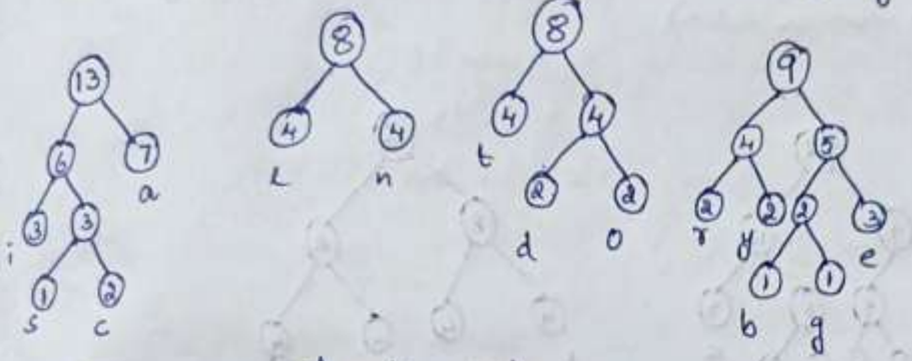
Arrange in Ascending Order



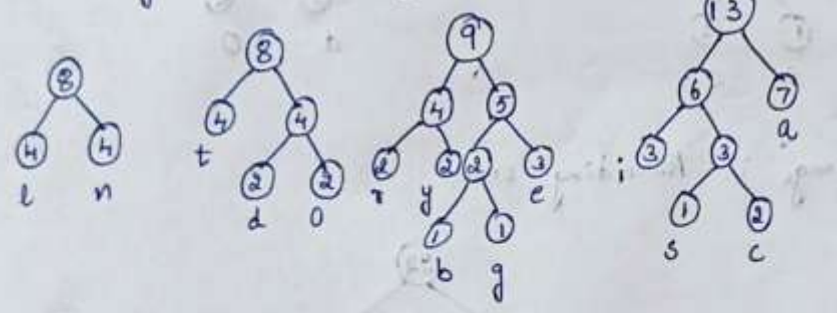


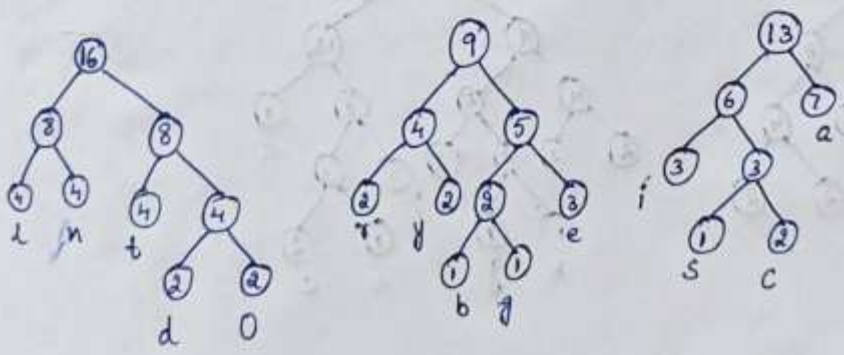
Arrange in ascending order



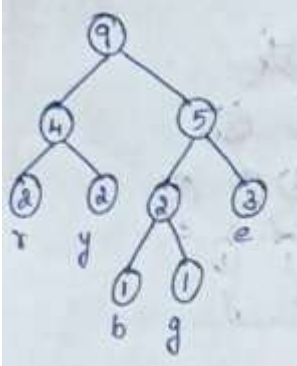


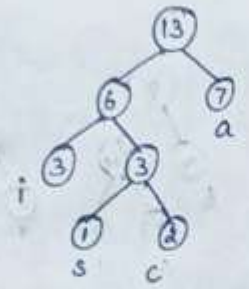
Arrange in Ascending order.

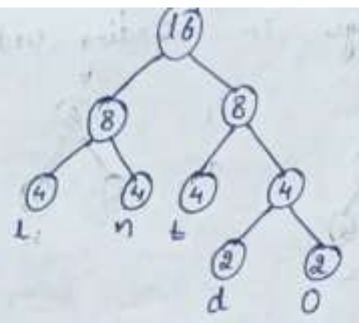


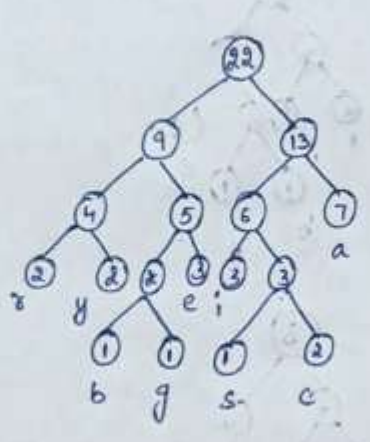


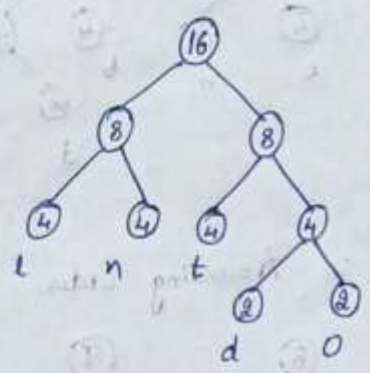
Arrange in Ascending order.



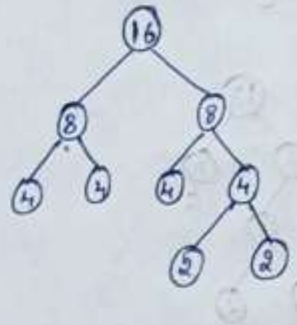


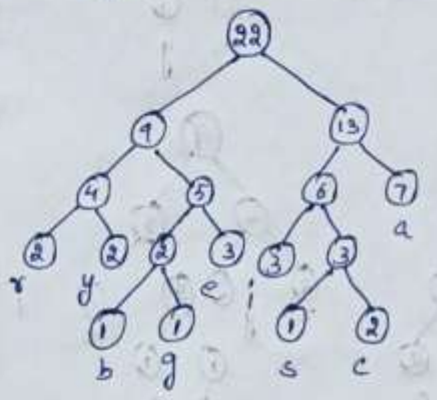




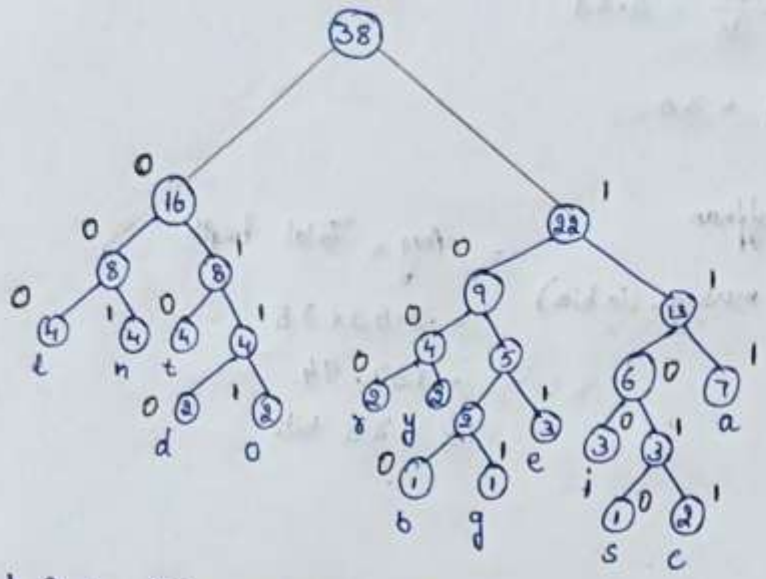


Arrange in Ascending order.





Final Tree



b	-	1 0 1 0 0	=	5 x 1	= 5
g	-	1 0 1 0 1	=	5 x 1	= 5
s	-	1 1 0 1 0	=	5 x 1	= 5
c	-	1 1 0 1 1	=	5 x 2	= 10
d	-	0 1 1 0	=	4 x 2	= 8
o	-	0 1 1 1	=	4 x 2	= 8
r	-	1 0 0 0	=	4 x 2	= 8
y	-	1 0 0 1	=	4 x 2	= 8
e	-	1 0 1 1	=	4 x 3	= 12
i	-	1 1 0 0	=	4 x 3	= 12
l	-	0 0 0	=	3 x 4	= 12
n	-	0 0 1	=	3 x 4	= 12
t	-	0 1 0	=	3 x 4	= 12
a	-	1 1 1	=	3 x 7	= 21

$$\text{Avg} = \frac{5+5+5+10+8+8+8+8+12+12+12+12+12+21}{1+1+1+2+2+2+2+2+3+3+4+4+4+7}$$

$$= \frac{138}{38} = 3.63$$

Total length = 38

Length of Hoffman encoded message (in bits) = Avg x Total length

= 3.63 x 38

= 137.94

≃ 138 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 = $O(n)$