DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 8

NAME: CHANTATI SAI PURNISHA MAHI

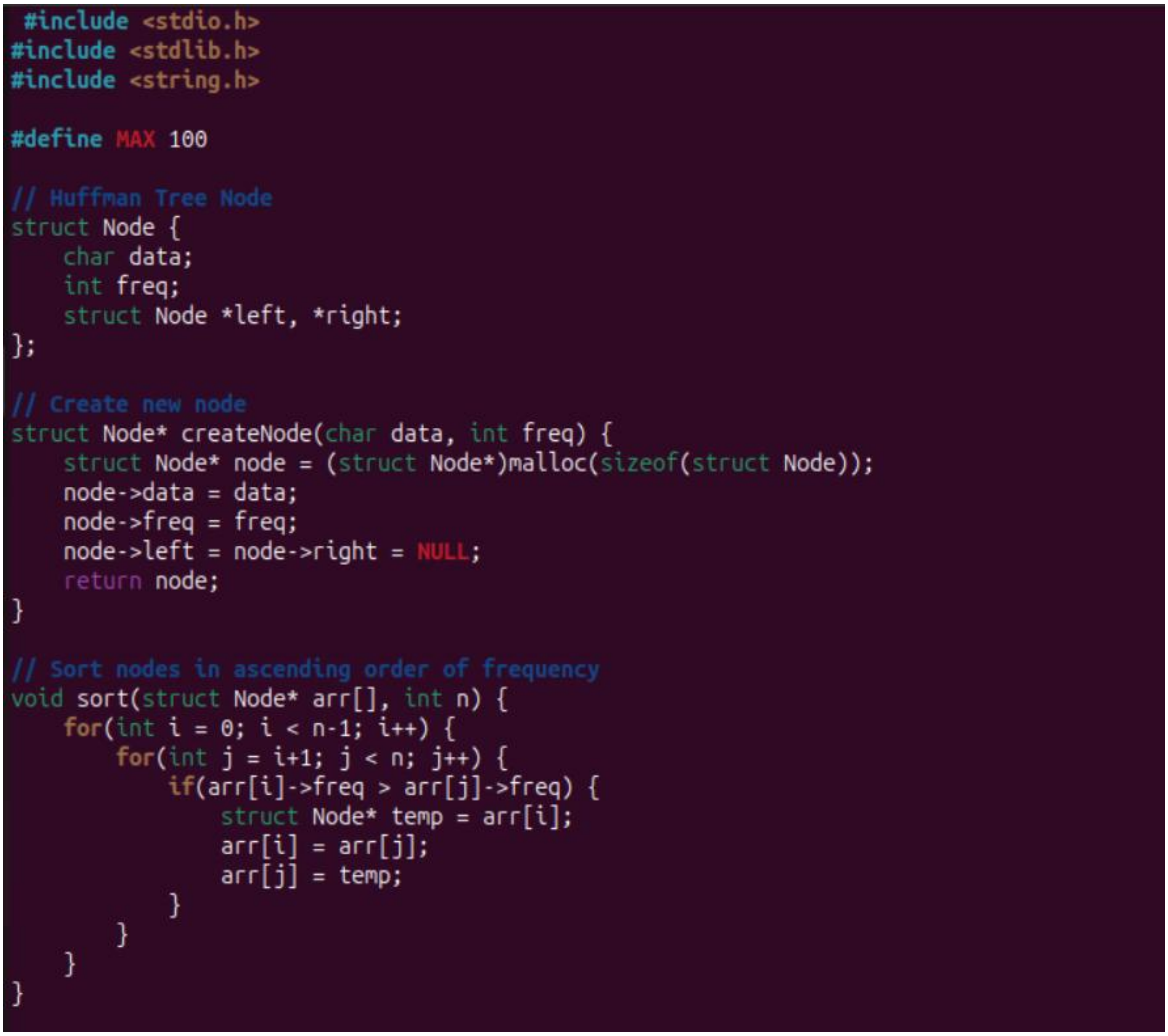
ROLL NUMBER: CH.SC.U4CSE24156

CLASS: CSE-B

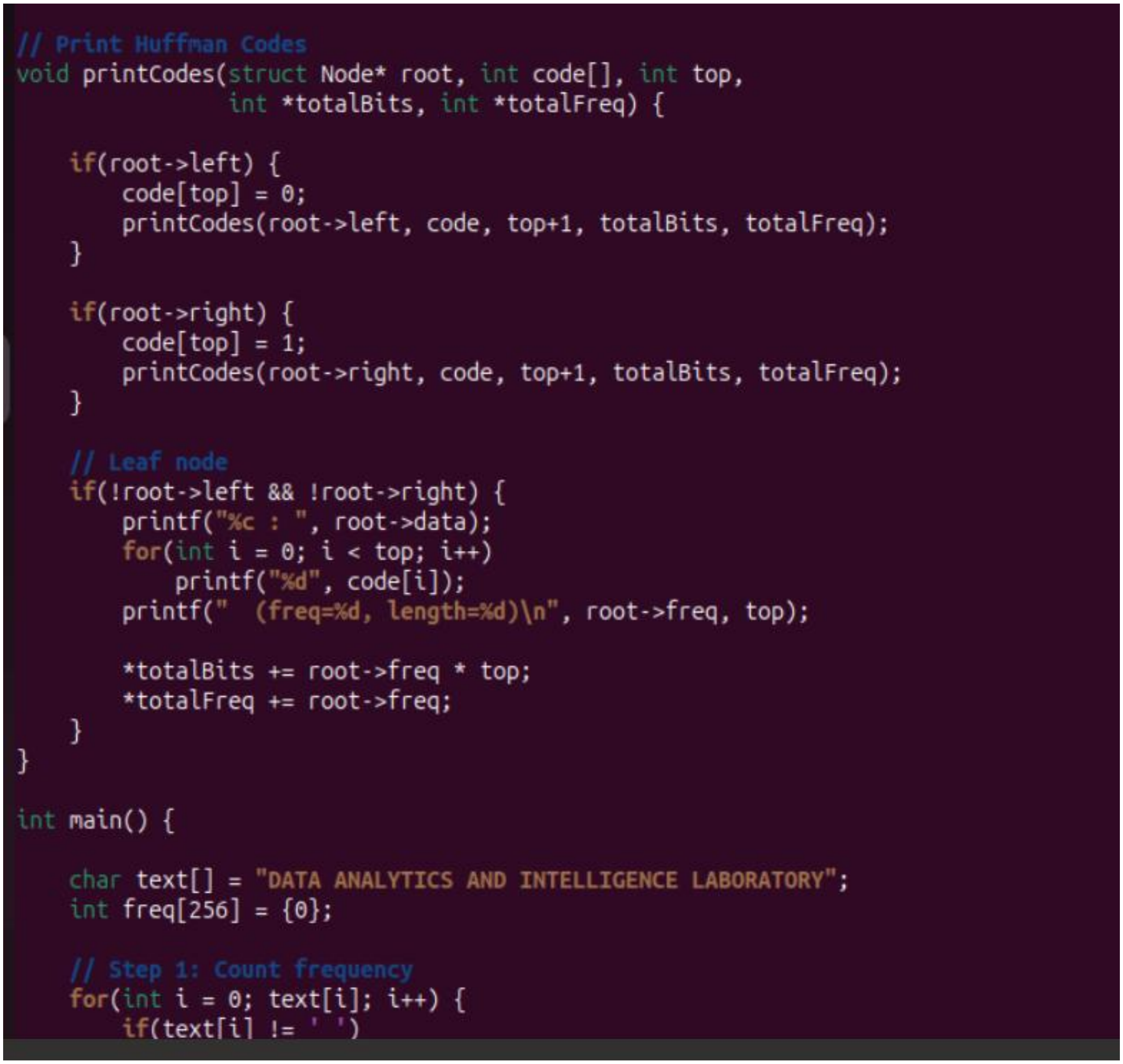
Huffman Coding:

DATA ANALYTICS AND INTELLIGENCE LABORATORY

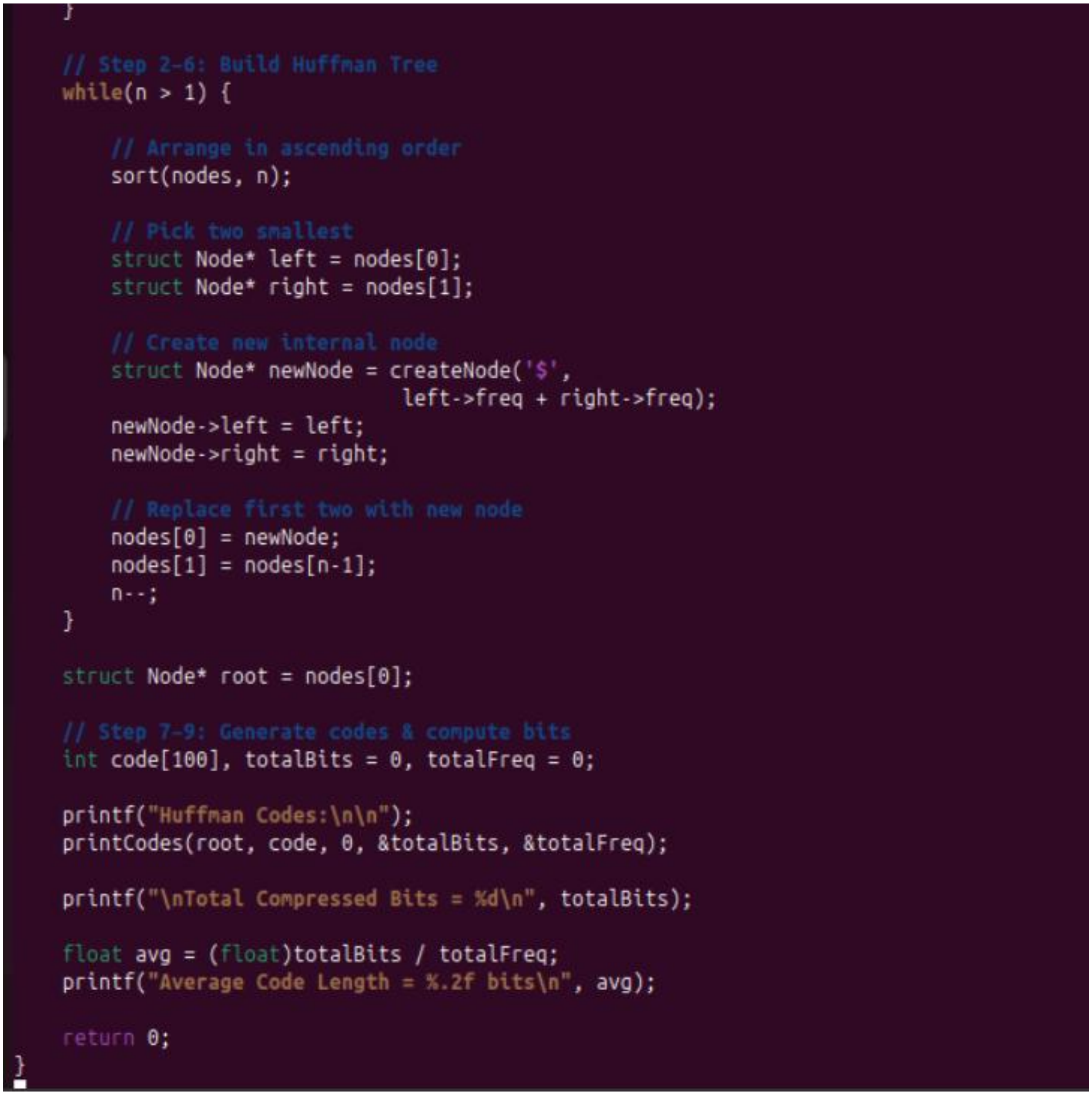
Code:



```c
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 100

// Huffman Tree Node
struct Node {
    char data;
    int freq;
    struct Node *left, *right;
};

// Create new node
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;
}

// Sort nodes in ascending order of frequency
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;
            }
        }
    }
}
```

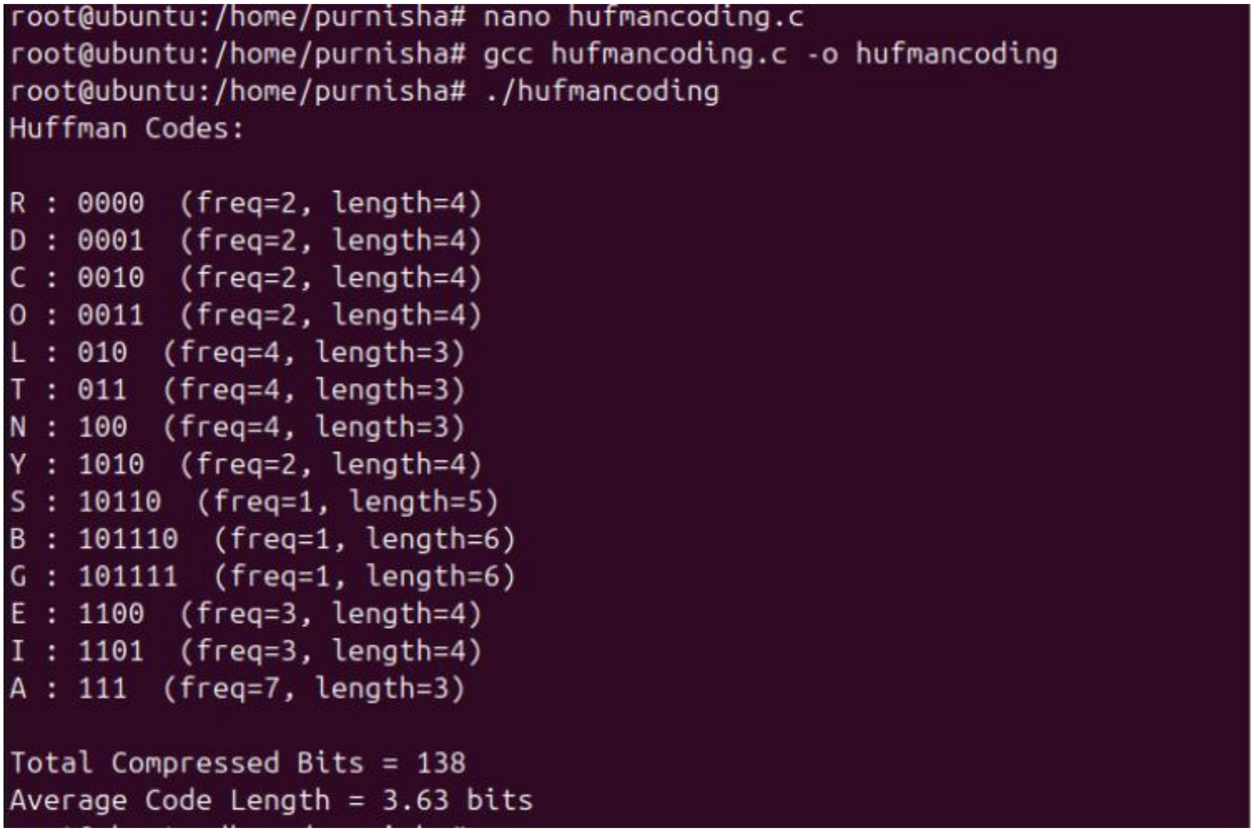


```c
// Print Huffman Codes
void printCodes(struct Node* root, int code[], int top,
                int *totalBits, int *totalFreq) {

    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);
    }

    // Leaf node
    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};

    // Step 1: Count frequency
    for(int i = 0; text[i]; i++) {
        if(text[i] != ' ')
```



```
    }

    // Step 2-6: Build Huffman Tree
    while(n > 1) {

        // Arrange in ascending order
        sort(nodes, n);

        // Pick two smallest
        struct Node* left = nodes[0];
        struct Node* right = nodes[1];

        // Create new internal node
        struct Node* newNode = createNode('$',
                                left->freq + right->freq);
        newNode->left = left;
        newNode->right = right;

        // Replace first two with new node
        nodes[0] = newNode;
        nodes[1] = nodes[n-1];
        n--;
    }

    struct Node* root = nodes[0];

    // Step 7-9: Generate codes & compute bits
    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:



```
root@ubuntu:/home/purnisha# nano hufmancoding.c
root@ubuntu:/home/purnisha# gcc hufmancoding.c -o hufmancoding
root@ubuntu:/home/purnisha# ./hufmancoding
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
```

TIME COMPLEXITY:- O(m + n^3) ≈ O(n^3)

JUSTIFICATION:-

m = length of input text

n = number of distinct characters

Frequency conting loop runs for each character → O(m)

Creating nodes for ASCII range (256) → constant → O(1)

While loop runs (n−1) times to build tree

Each time sorting n nodes using bubble sort → O(n^2)

Total tree building time → (n−1) × O(n^2) → O(n^3)

Tree traversal to print codes visits each node once → O(n)

Dominant term → O(n^3)

SPACE COMPLEXITY:- O(n)

JUSTIFICATION:-

int freq[256] → 256 × 4 bytes = 1024 bytes

One Node contains:

- char → 1 byte
- int → 4 bytes
- left pointer → 8 bytes
- right pointer → 8 bytes
  Total ≈ 21 bytes → padded ≈ 24 bytes per node

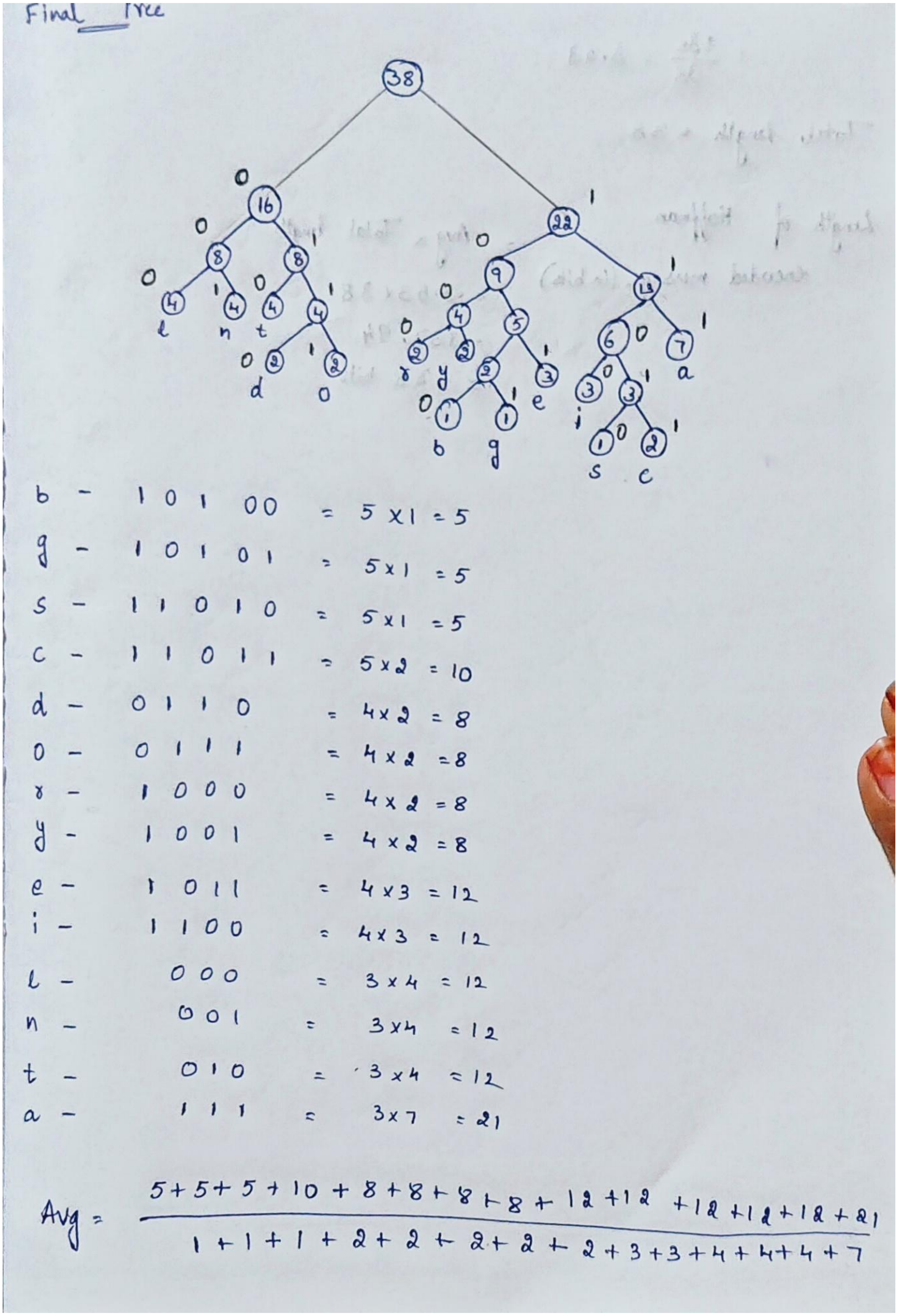
Nodes array Node* nodes[100] → 100 × 8 bytes = 800 bytes

Code array int code[100] → 100 × 4 bytes = 400 bytes

Recursion stack for tree traversal proportional to tree height → O(n)

Memory mainly grows with number of nodes → O(n)

Final Tree


38
0
1
16
22
0
1
0
1
8
8
9
13
0
1
0
1
0
1
0
1
4
4
4
4
4
5
6
7
l
n
t
a
0
1
0
1
0
1
0
1
2
2
2
2
2
3
3
3
d
o
r
y
e
i
0
1
0
1
1
1
1
2
b
g
s
c

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

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

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

Total length = 38

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

$= 3.63 \times 38$

$= 137.94$

$\simeq 138$ 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)

Hufman Coding

Data Analystics and Intelligene labarotary.

Algorithm:-

1. Write characters & frequeny in tabulor form.
2. Now write in ascending order.
3. Add first two least frequency ~~and~~ their sum is in root node & first no is left child second one is right childe
4. Check if there are in ascending order if same frequeny.
   - i, Character vs character → Alphabetical order
   - ii, Character vs Tree → character first
   - iii, Tree vs Tree → Earlier formed tree first
5. Repeat this process until you get one final Tree.
6. After this, left child is 0 right child is 1 do it for full tree.
7. Write each letter codes. (In 0 or 1)
8. Then find Compressed bits using formulae.

$$\text{Compressed bits} = \Sigma \text{ codelength} \times \text{frequeny}.$$

9. Average bit length per Character

$$= \frac{\Sigma(\text{code length} \times \text{frequency}_i)}{\Sigma(\text{frequency}_i)}$$

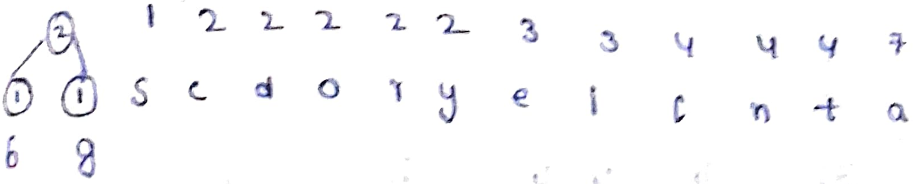
Sol:-

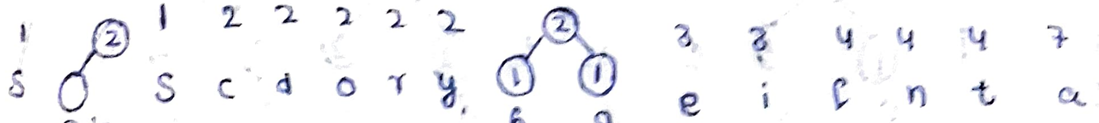
character	d	a	t	n	f	y	i	c	s	e	g	b	o	r
frequency	2	7	4	4	4	2	3	2	1	3	1	1	2	2

write in asciending order

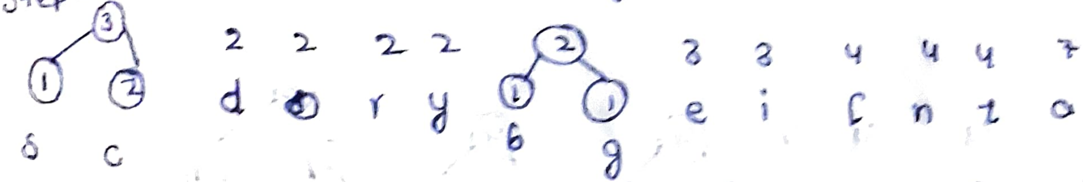
~~Bss~~ Step 1:-

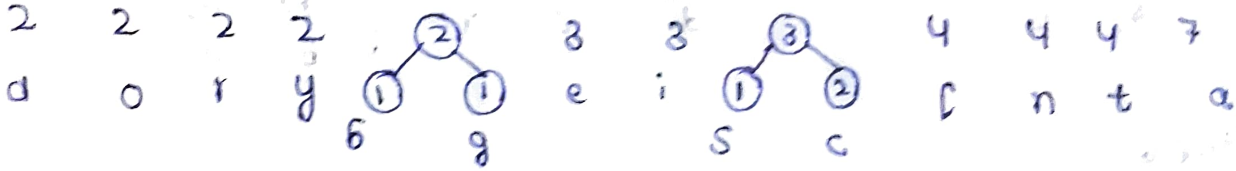
1	1	1	2	2	2	2	2	3	3	4	4	4	7
b	g	s	c	d	o	r	y	e	i	f	n	t	a



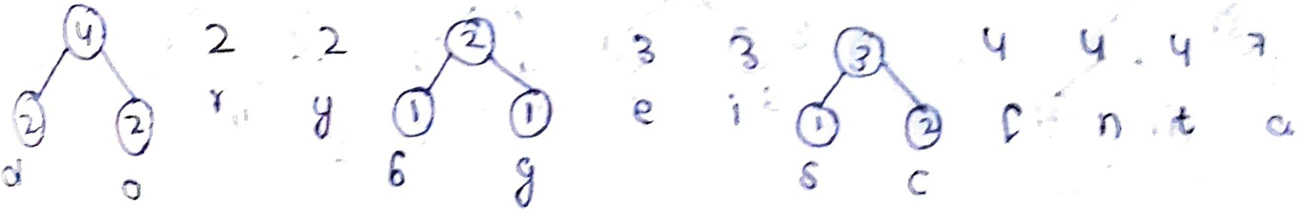


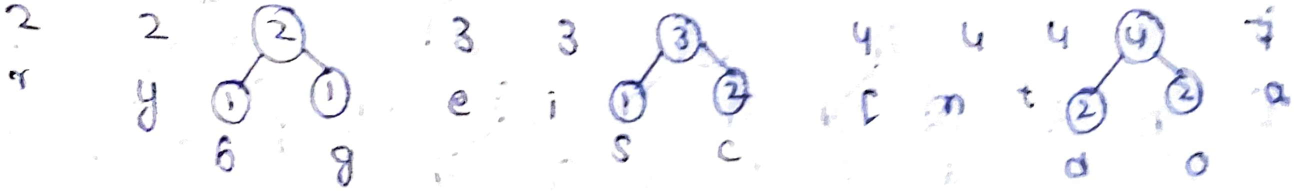
Step 2:-



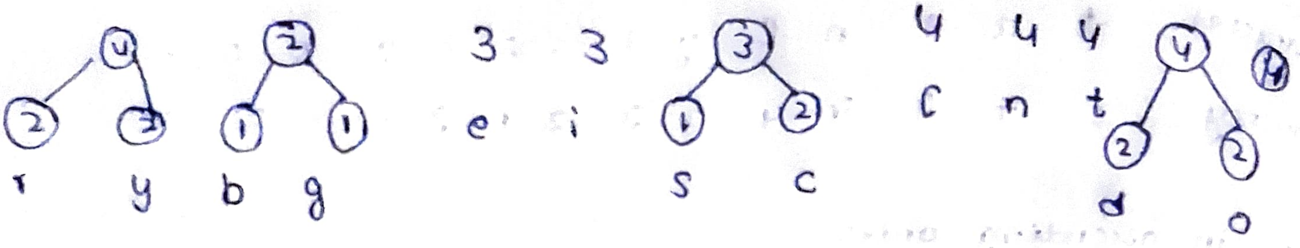


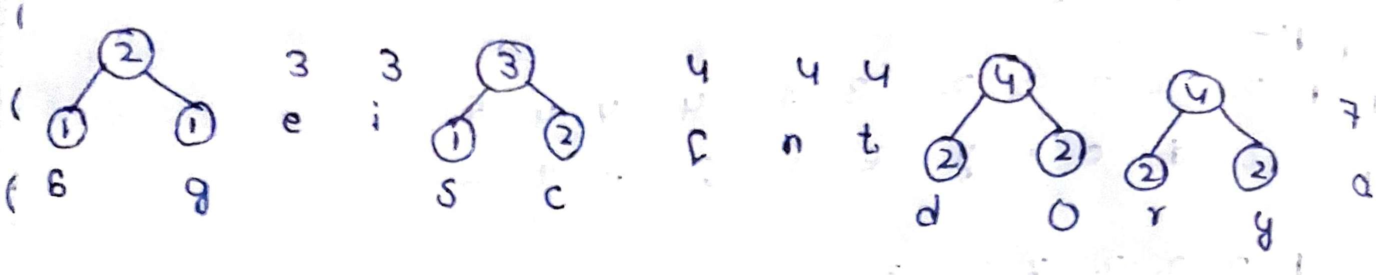
Step 3:-



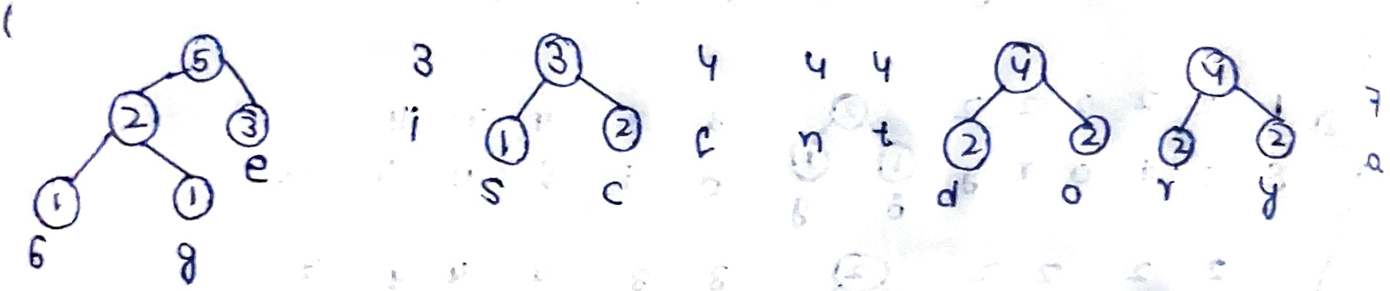


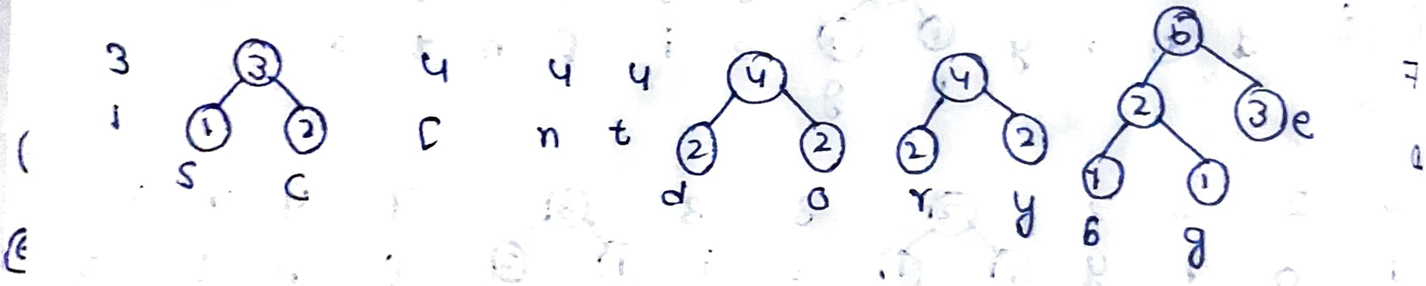
Step 4:-



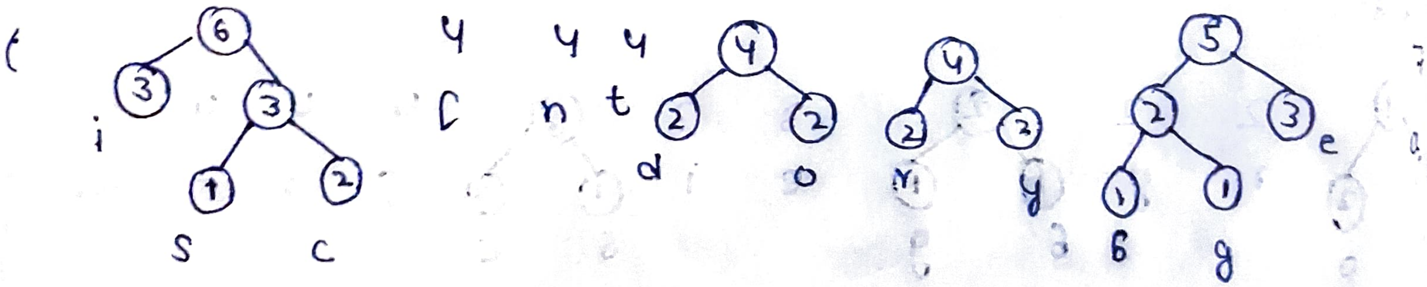


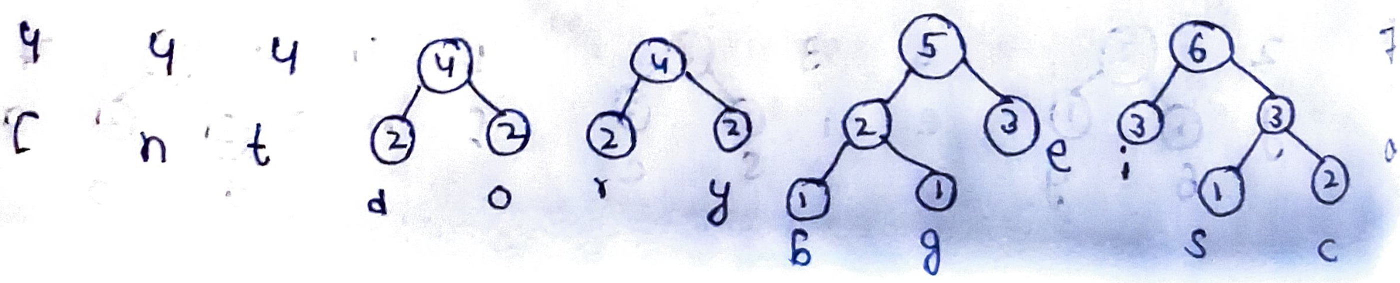
Step 5:-



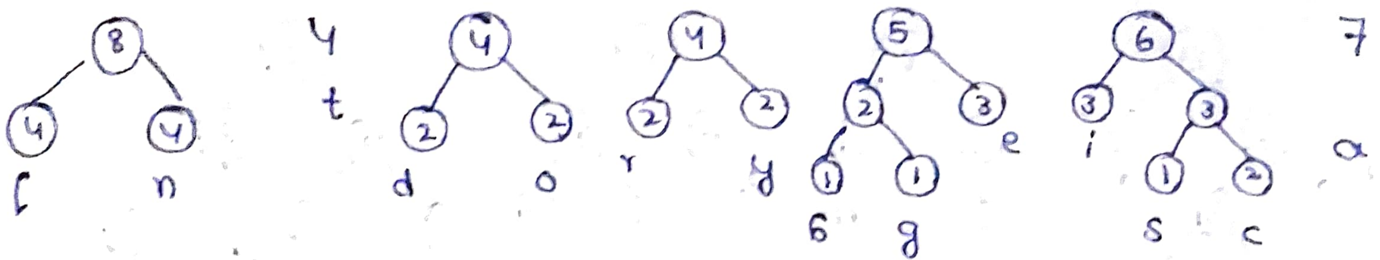


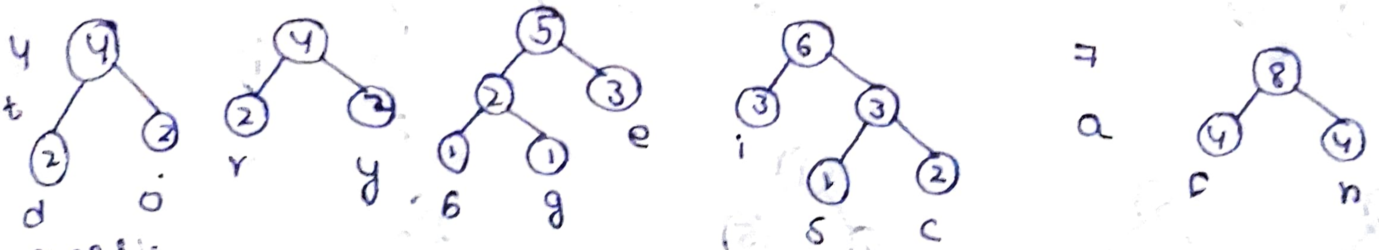
Step 6:-



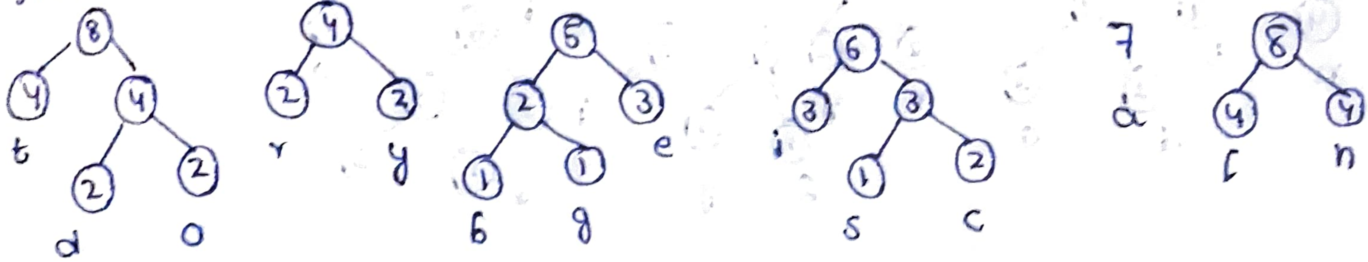


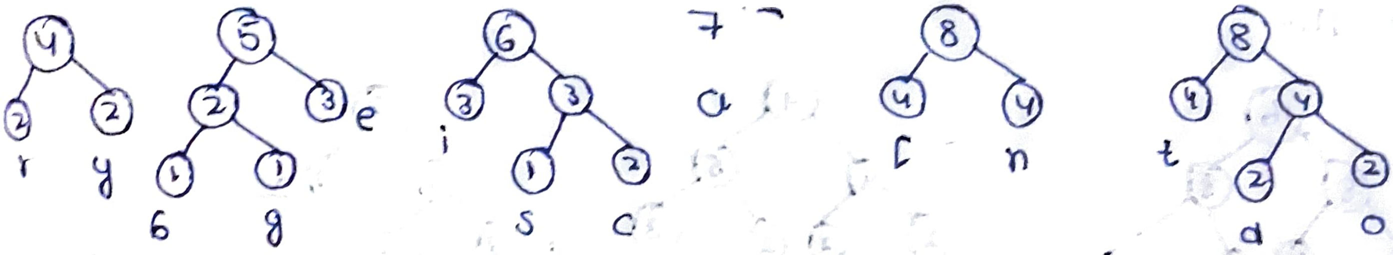
Step 7:-



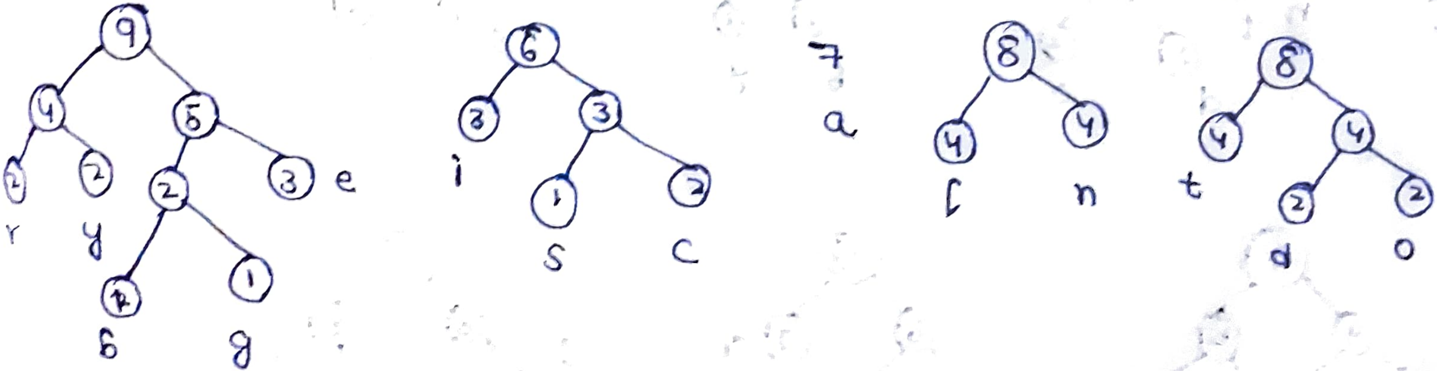


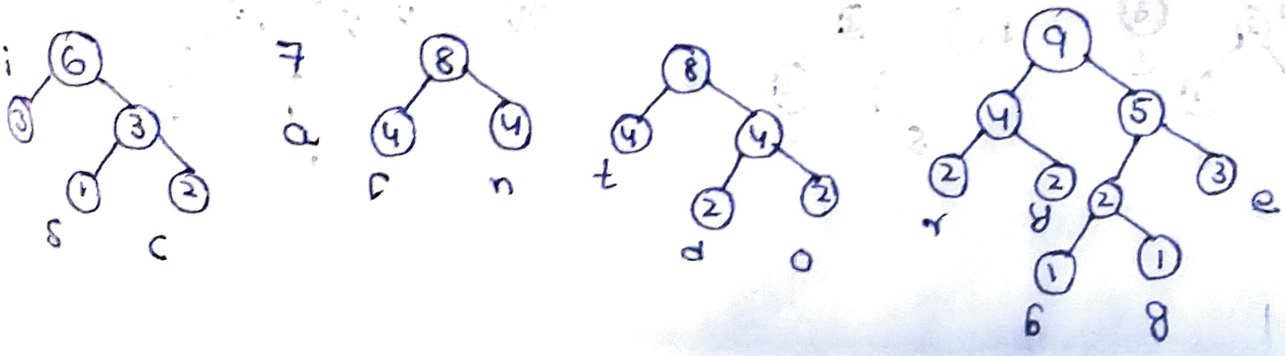
Step 8:-



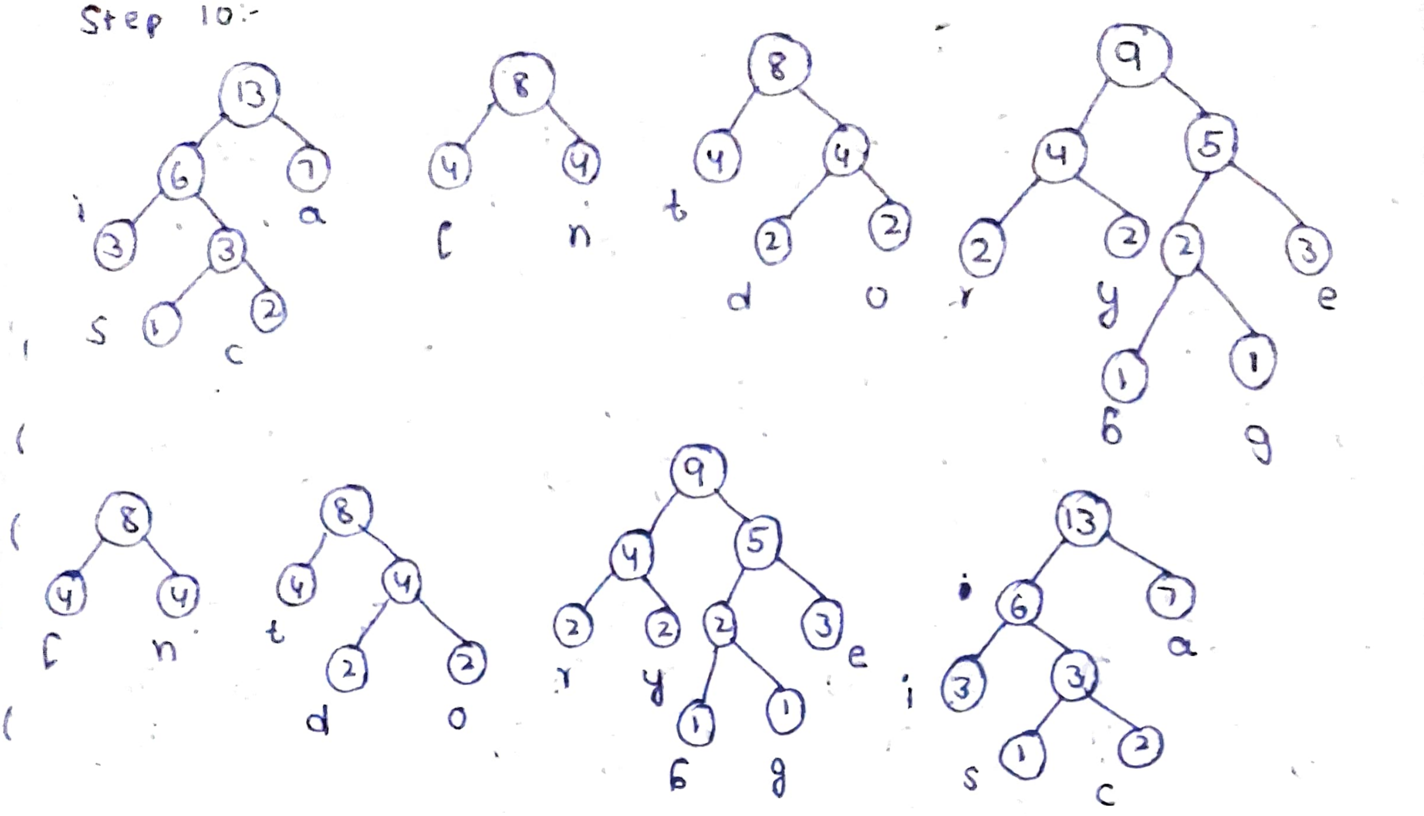


Step 9:-

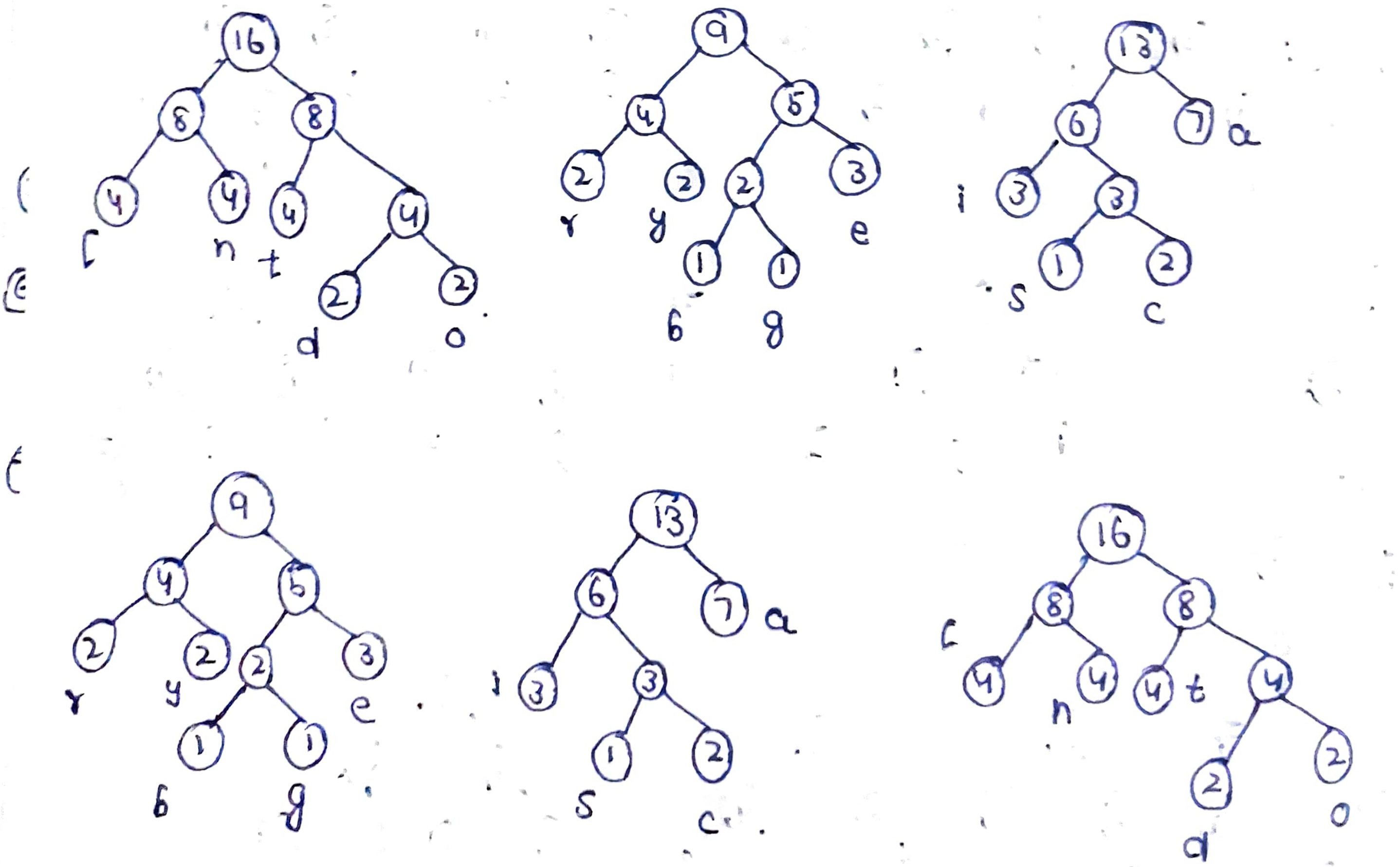




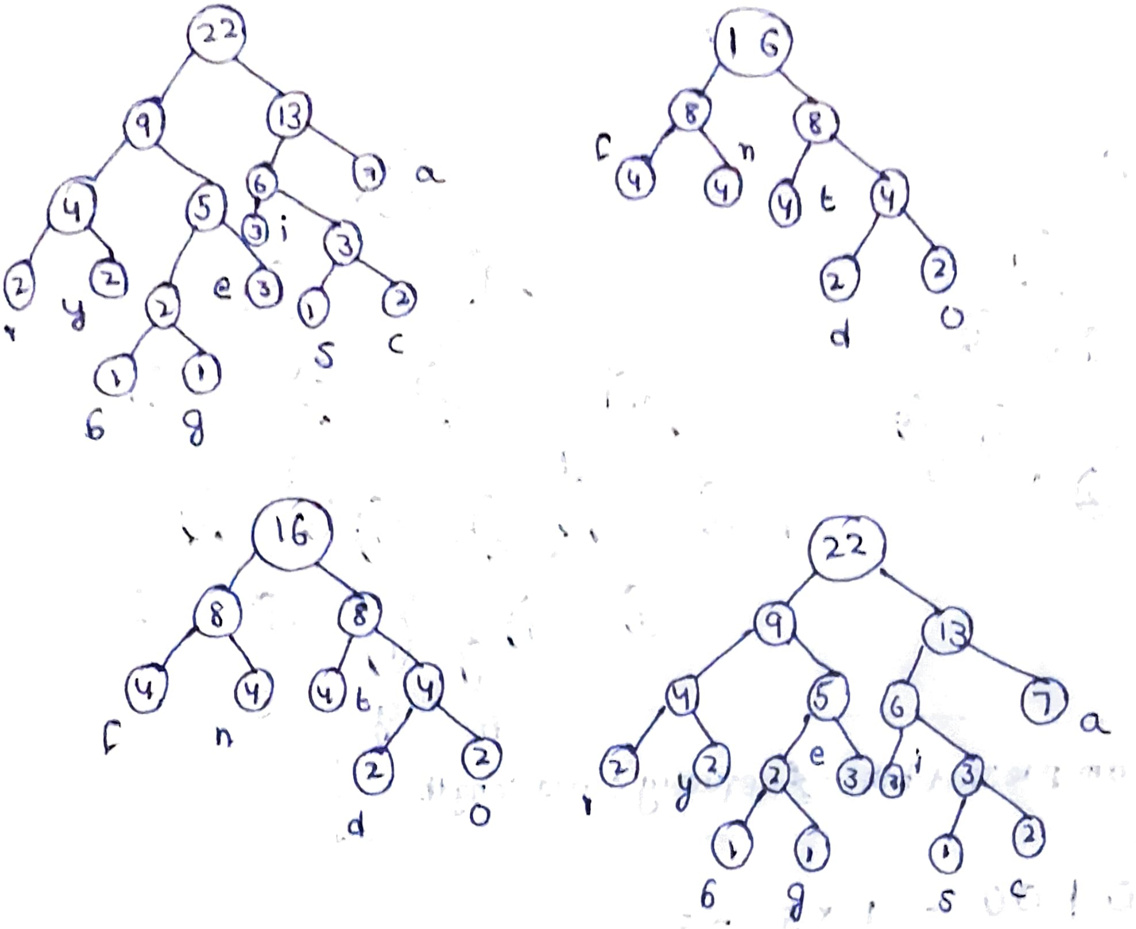
Step 10:-



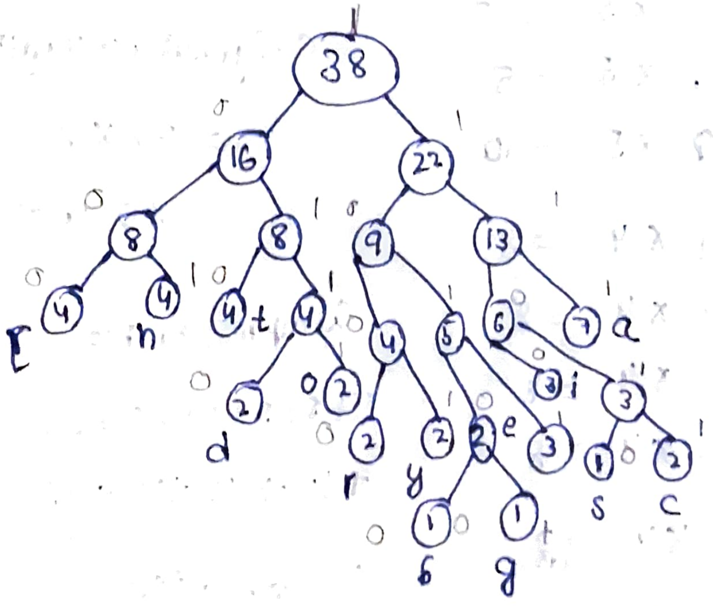
Step 11:-



Step 12:-



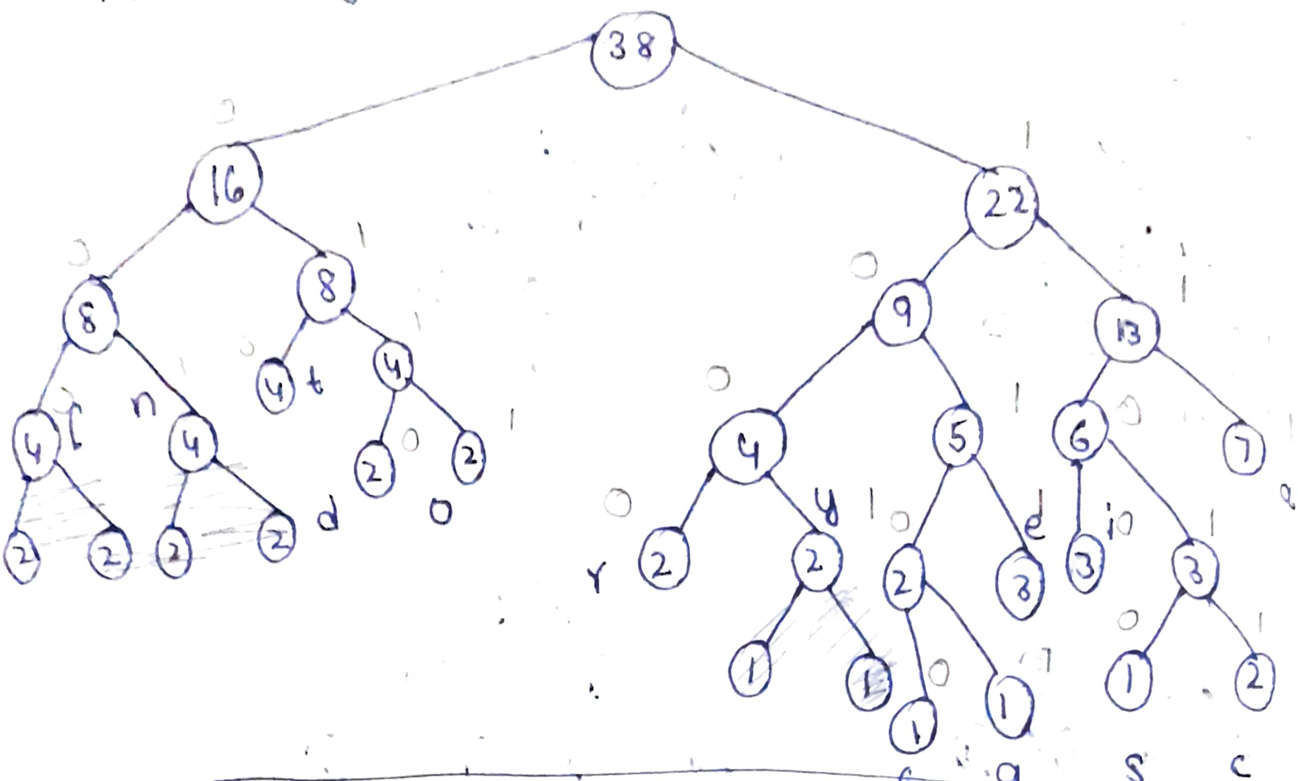
Step 13:-



Right side - 1

Left side - 0

Final Tree diagram:-



Compressed bits = Σfrequency x code length.

6	10100	1 x 5	= 5
g	10101	1 x 5	= 5
s	11010	1 x 5	= 5
c	11011	2 x 5	= 10
d	0110	2 X 4	= 8
o	0111	2 x 4	= 8
r	1000	2 x 4	= 8
y	1001	2 x 4	= 8
e	1011	3 x 4	= 12
i	1100	3 x 4	= 12
[	000	4 x3	= 12
n	001	4 x3	= 12
t	010	4 x 3	= 12
a	111	7 x 3	= 21

Original bit length

= 38 X 8

= 304 bits

Compressed bits =

5+5+5+10+8+8+8+8+12+
+12+12+12+12+21

= 138 bits.

Saved bits = 304-138

= 166 bits.

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

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

Total length $= 38$

length of Hoffman encoded message $= \text{Avg} \times \text{Total length}$

$$= 3.63 \times 38$$

$$= 137.94$$

$$\simeq \underline{\underline{138 \text{ bits}}}$$