DESIGN AND ANALYSIS OF ALGORITHMS

LAB WORKBOOK WEEK – 8

NAME: MARADANA SRIJA

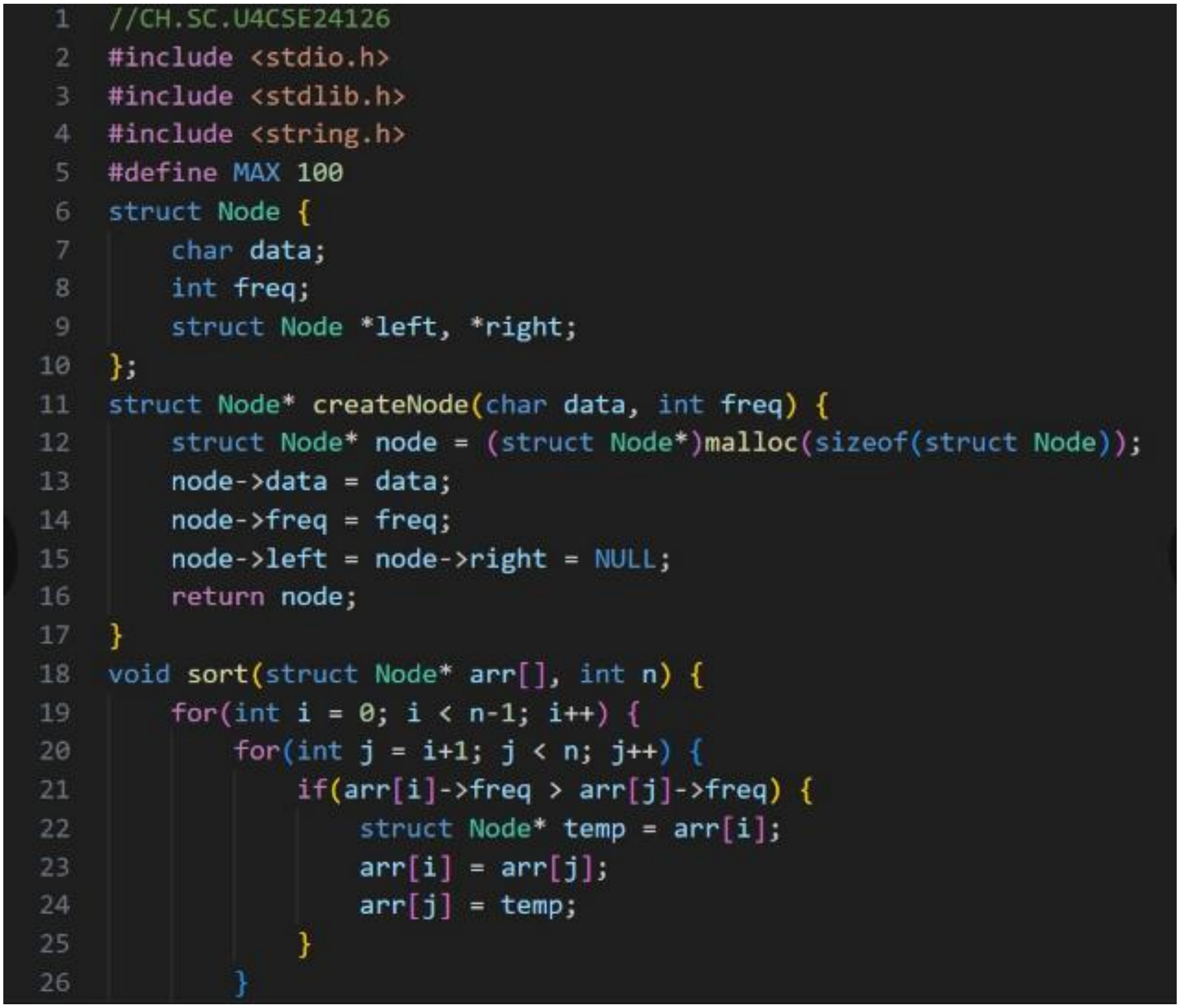
ROLL NUMBER: CH.SC.U4CSE2412R

CLASS: CSE-B

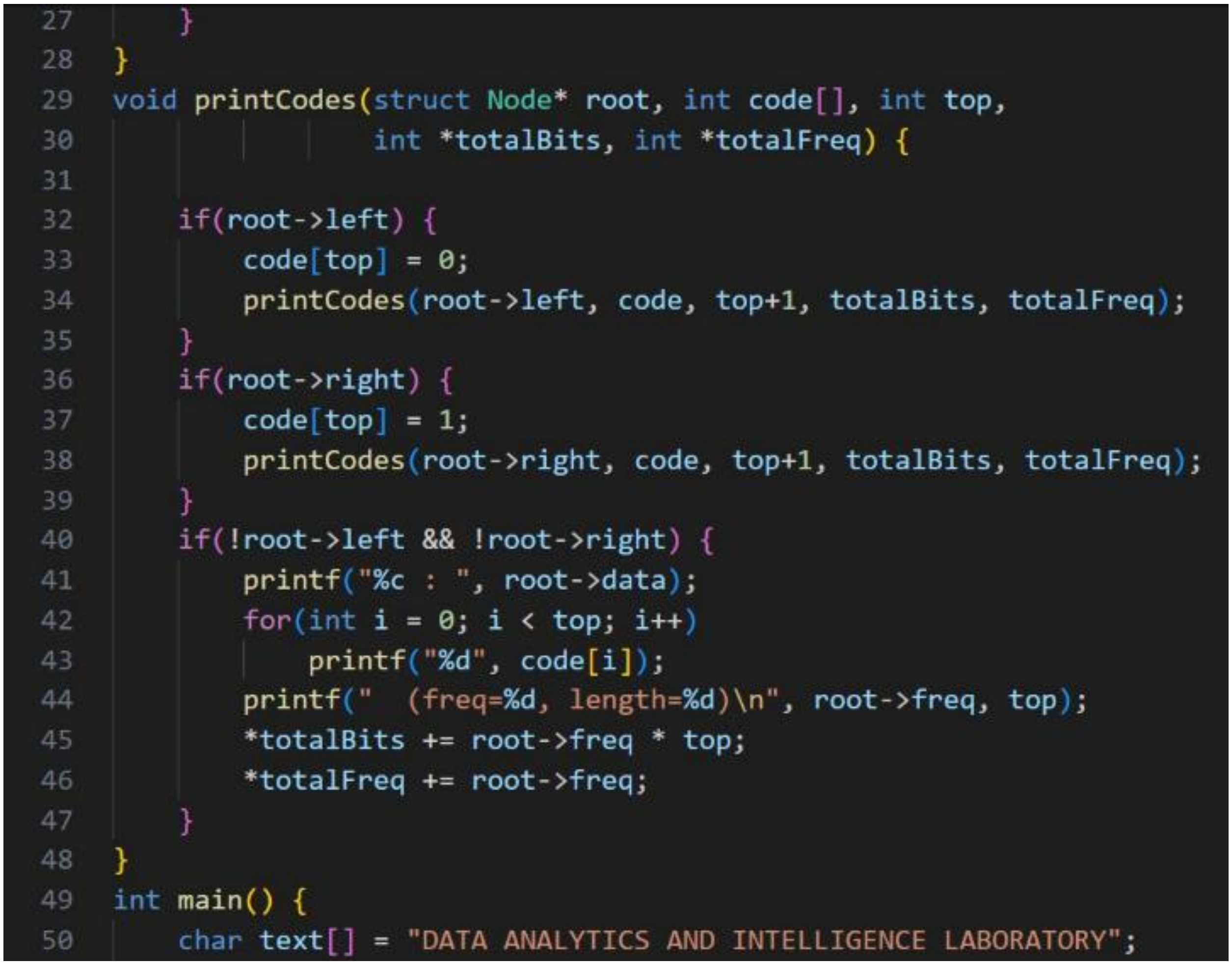
Huffman Coding:

DATA ANALYTICS AND INTELLIGENCE LABORATORY

Code:

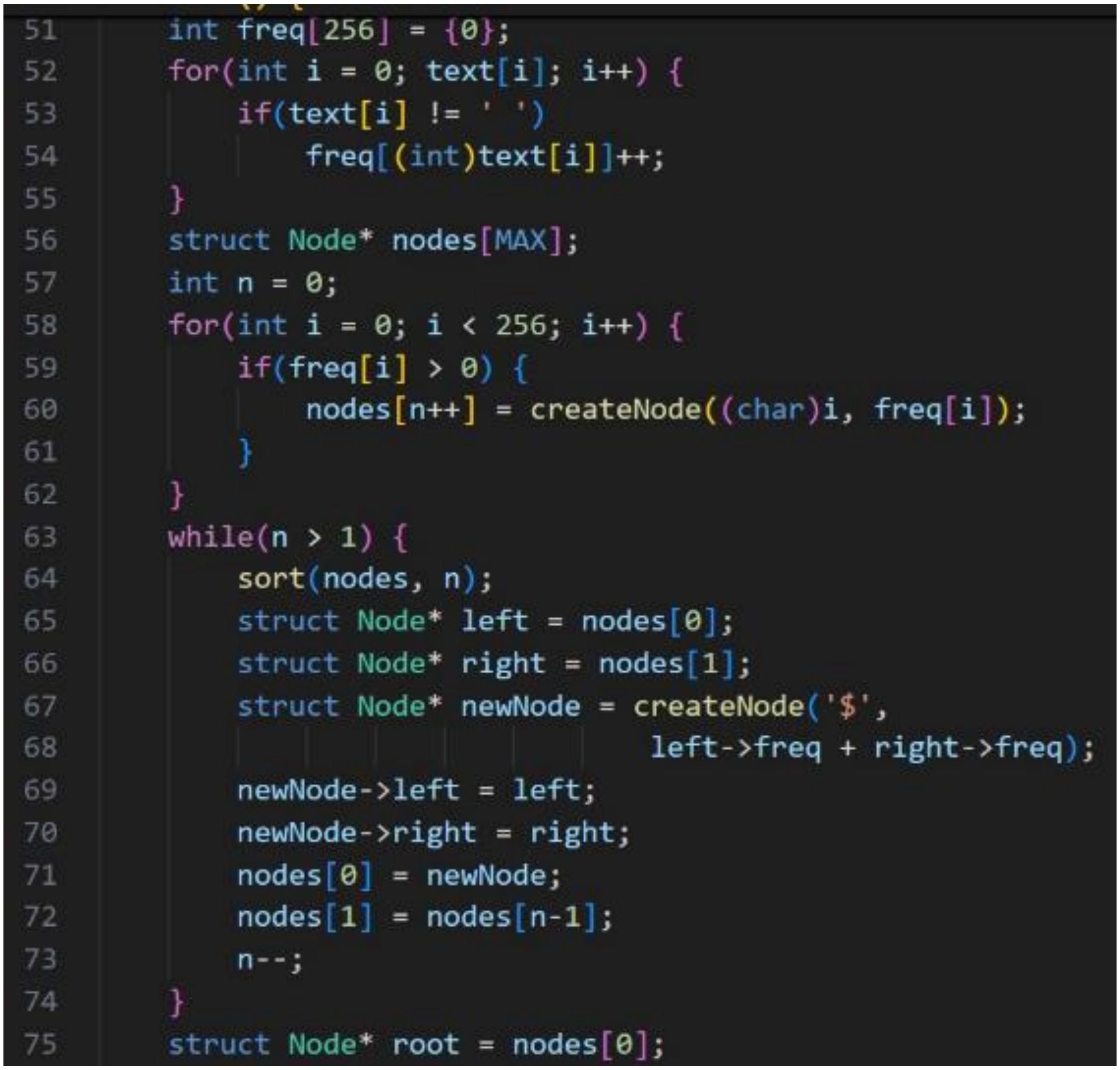


```c
//CH.SC.U4CSE24126
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
struct Node {
    char data;
    int freq;
    struct Node *left, *right;
};
struct Node* createNode(char data, int freq) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = 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;
            }
        }
```

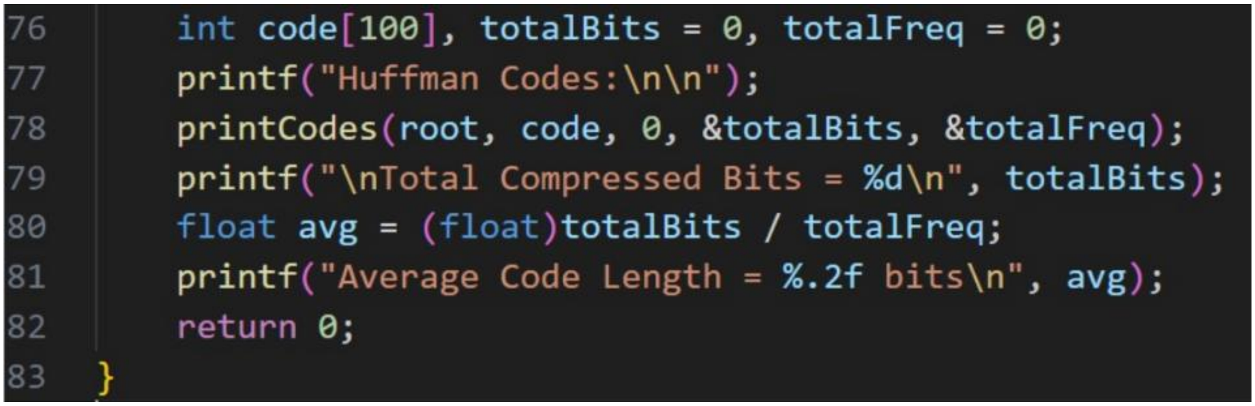


```c
    }
}
void printCodes(struct Node* root, int code[], int top,
                int *totalBits, int *totalFreq) {

    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";
```

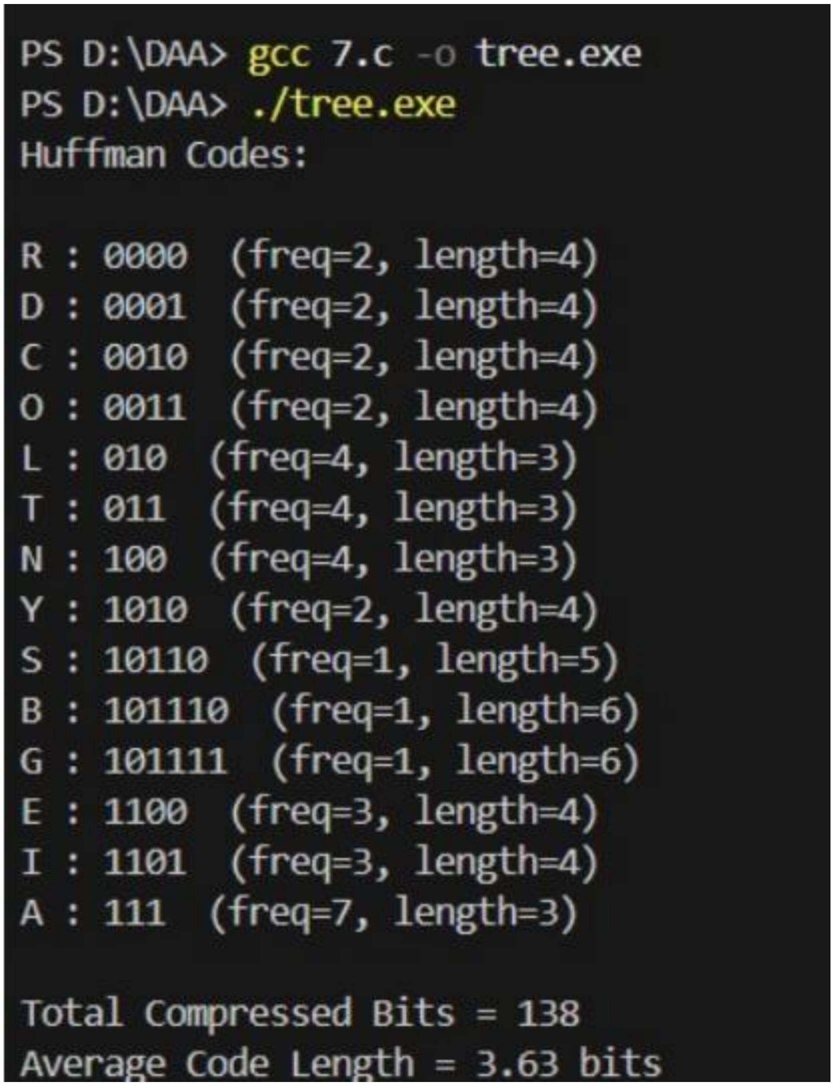


```c
    int freq[256] = {0};
    for(int i = 0; text[i]; i++) {
        if(text[i] != ' ')
            freq[(int)text[i]]++;
    }
    struct Node* nodes[MAX];
    int n = 0;
    for(int i = 0; i < 256; i++) {
        if(freq[i] > 0) {
            nodes[n++] = createNode((char)i, freq[i]);
        }
    }
    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];
```



```c
    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:



```
PS D:\DAA> gcc 7.c -o tree.exe
PS D:\DAA> ./tree.exe
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:

Hufman Coding

Data Analystics and Intelligence Laboratory

Algorithm :-

1) Write characters & frequency in tabular form

2) Now write in ascending order

3) Add first two least frequency their sum is in root node & first no. is left child second one is right child

4) Check if there are in ascending order

if same frequency

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.

$$\text{compressed bits} = \Sigma \text{ codelength} \times \text{frequency}$$

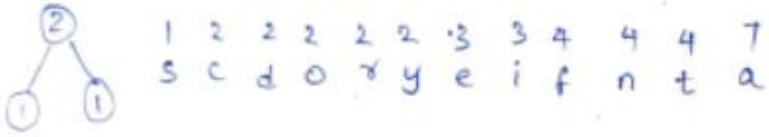
$$\text{Average HC length per character} = \frac{\Sigma(\text{code length} \times \text{frequency})}{\Sigma(\text{frequency})}$$

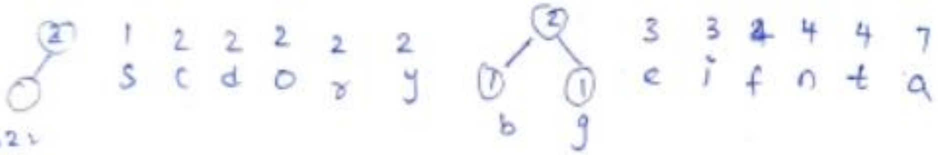
Character	d	a	t	n	f	y	i	c	s	e	g	b	o	r
Frequency	2	7	4	4	4	2	3	2	1	3	1	1	2	2

Write in ascending order:

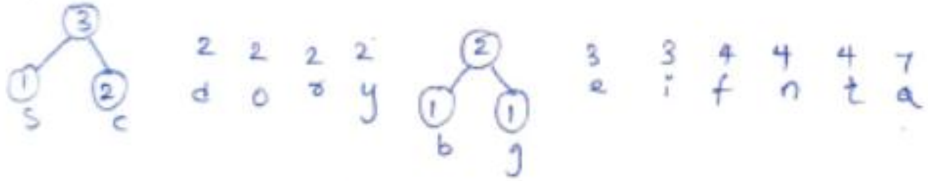
Step 1:-

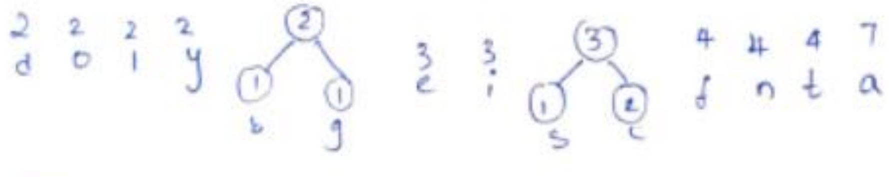
1	1	1	2	2	2	2	2	3	3	4	4	4	7
b	g	s	c	d	o	r	y	e	i	f	n	t	a



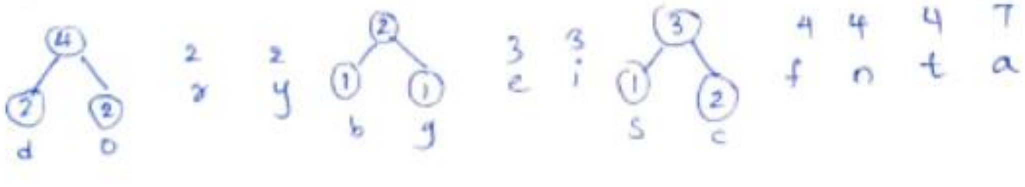


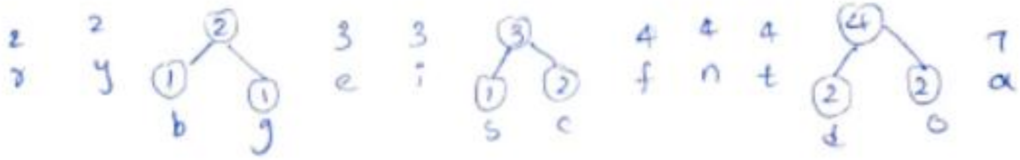
Step 2:-



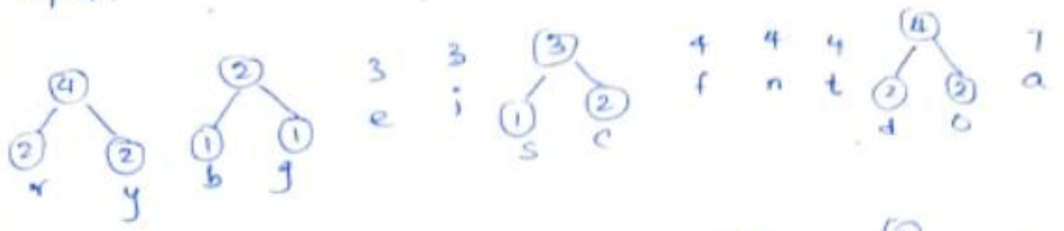


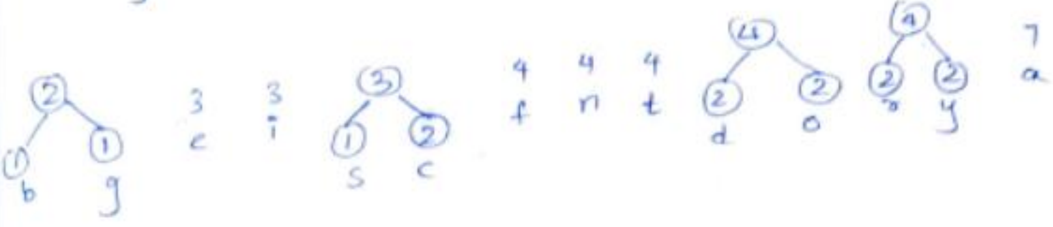
Step 3:-



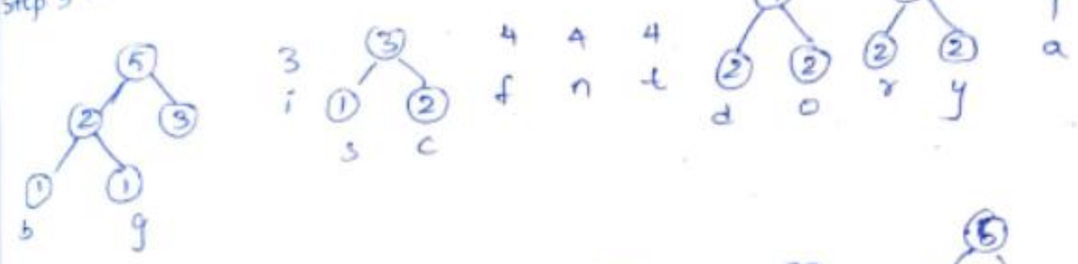


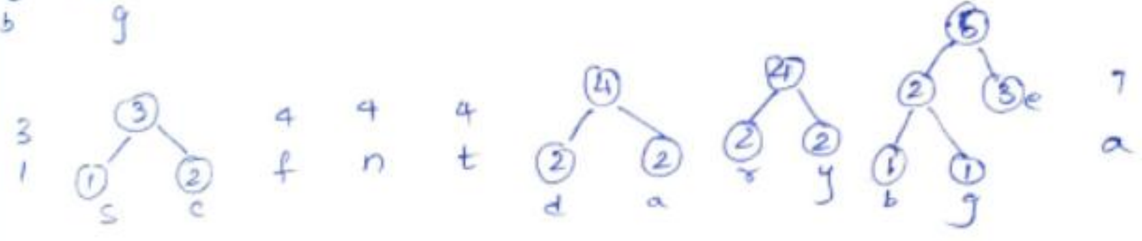
Step 4 :-



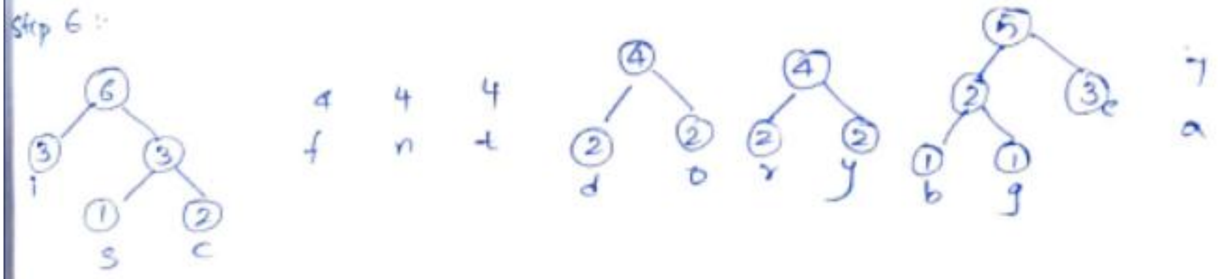


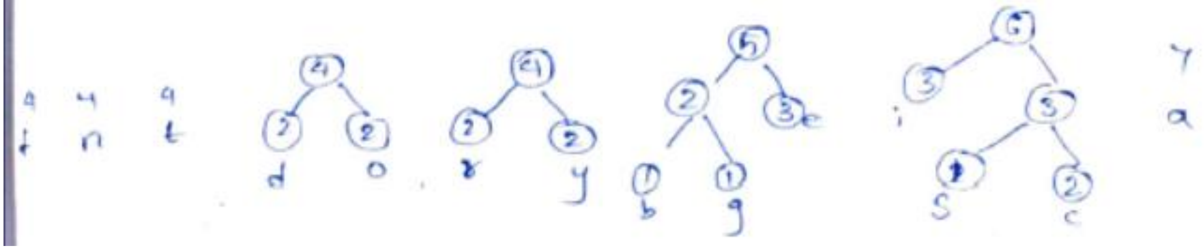
Step 5 :-



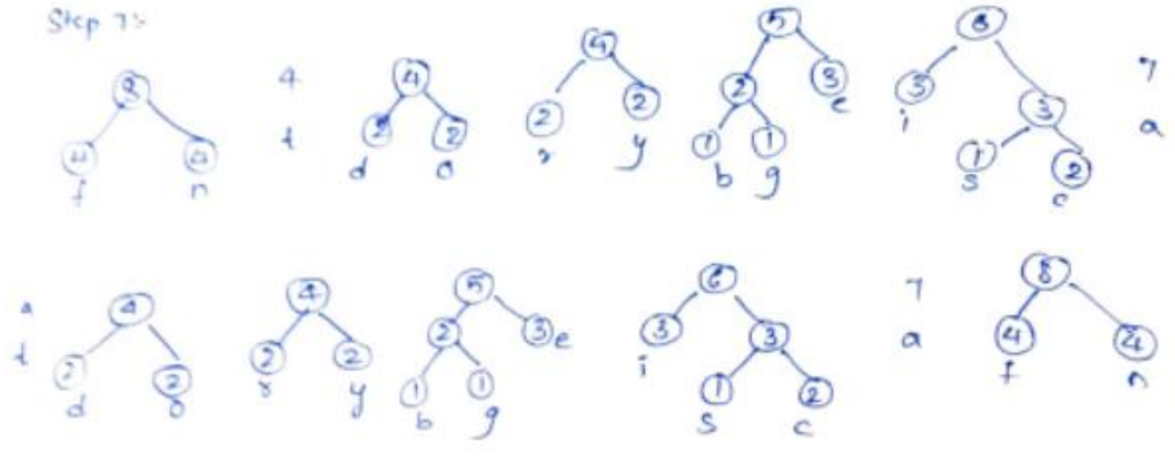


Step 6 :-

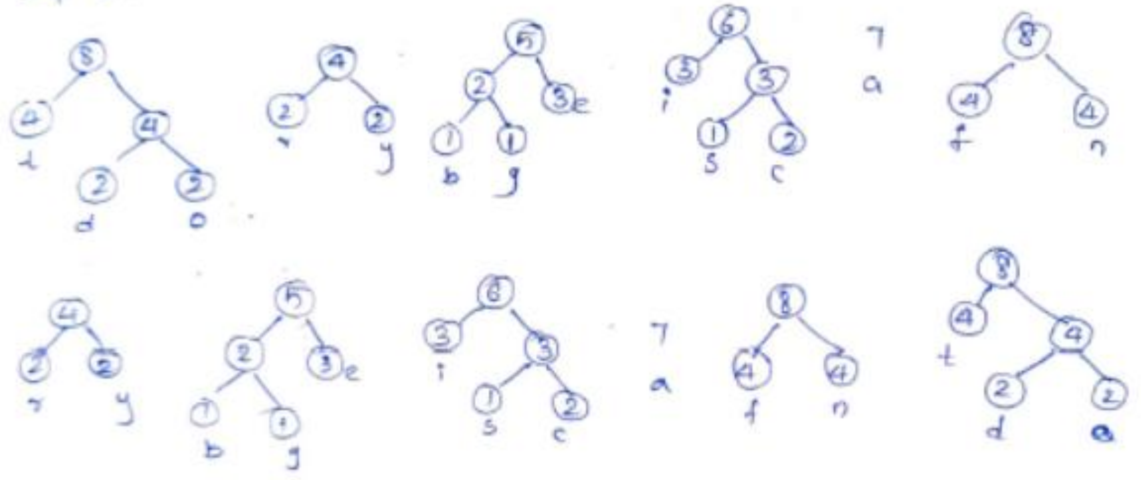




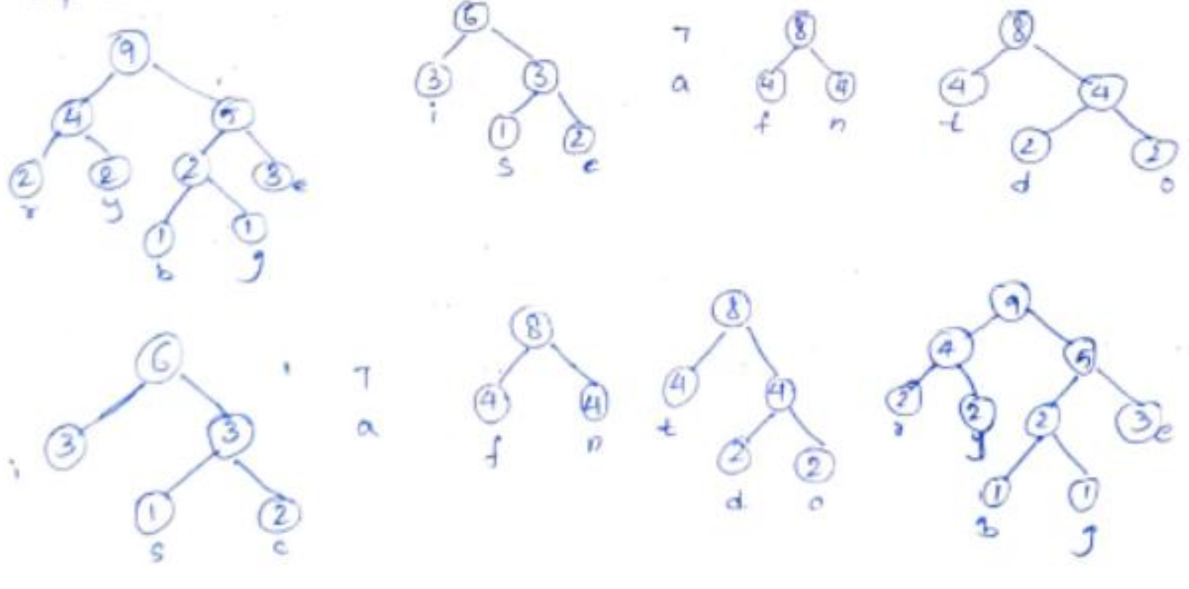
Step 7:-



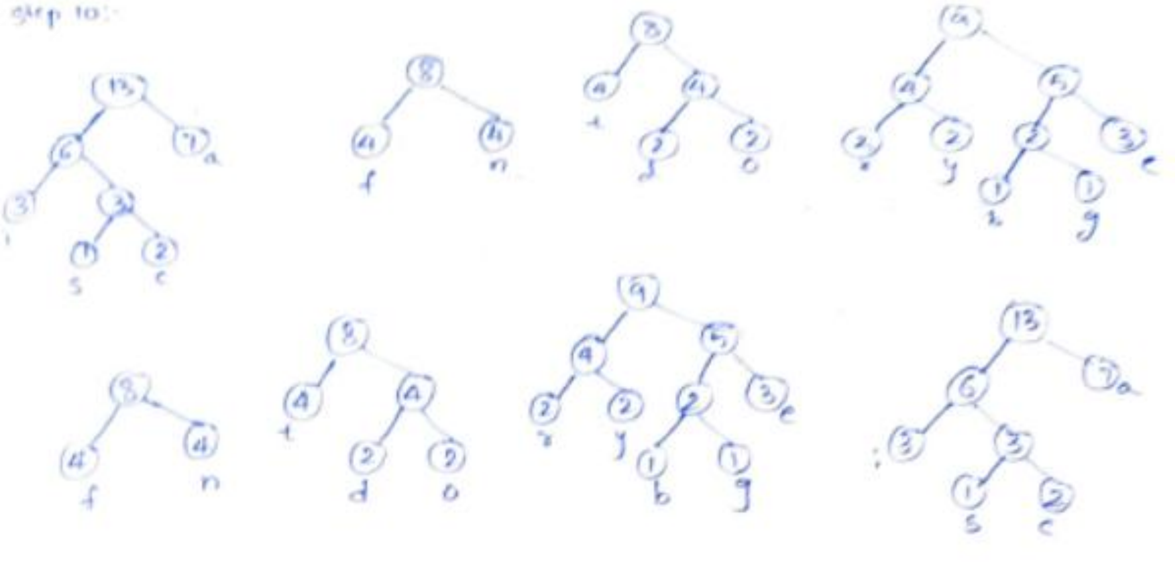
Step 8:-



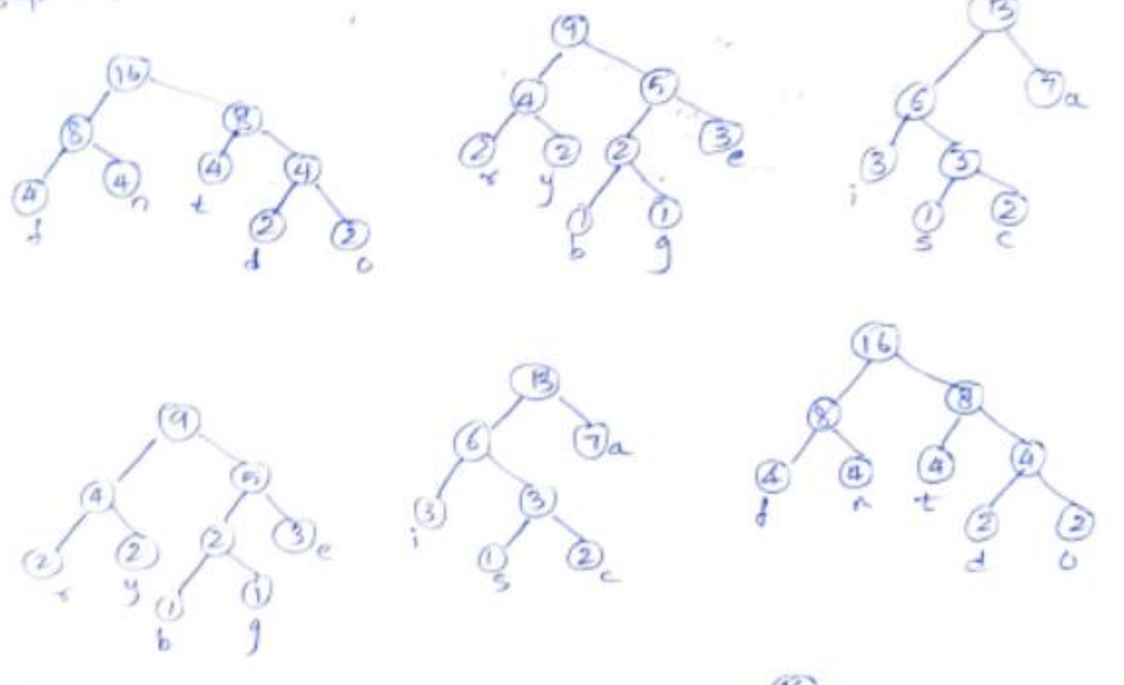
Step 9:-



step 10:-

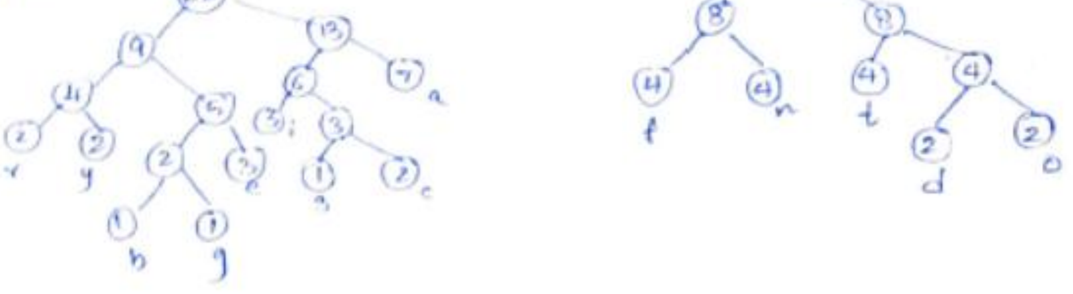


step 11 :-

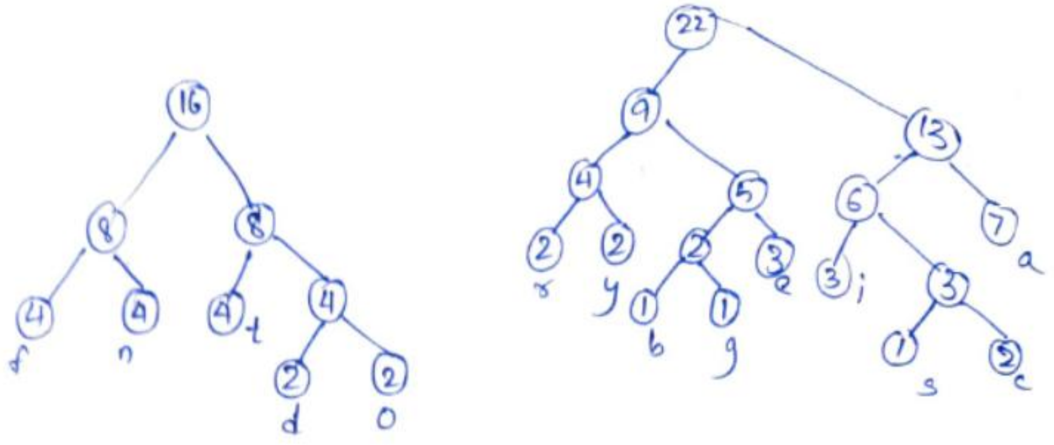


16 8 4 f 4 n 8 4 t 4 2 d 2 o
9 4 5 2 r 2 y 2 3 e 1 b 1 g
13 6 7 a 3 i 3 1 s 2 c
9 4 5 2 r 2 y 2 3 e 1 b 1 g
13 6 7 a 3 i 3 1 s 2 c
16 8 4 f 4 n 8 4 t 4 2 d 2 o

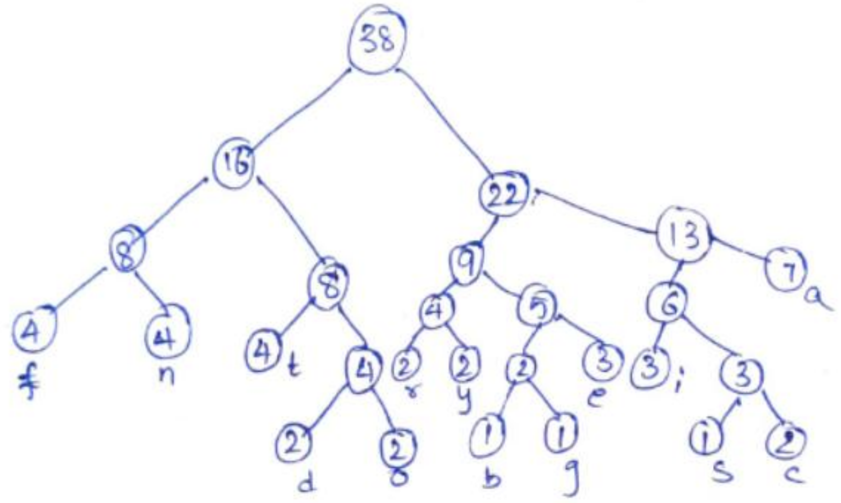
step 12:-



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


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

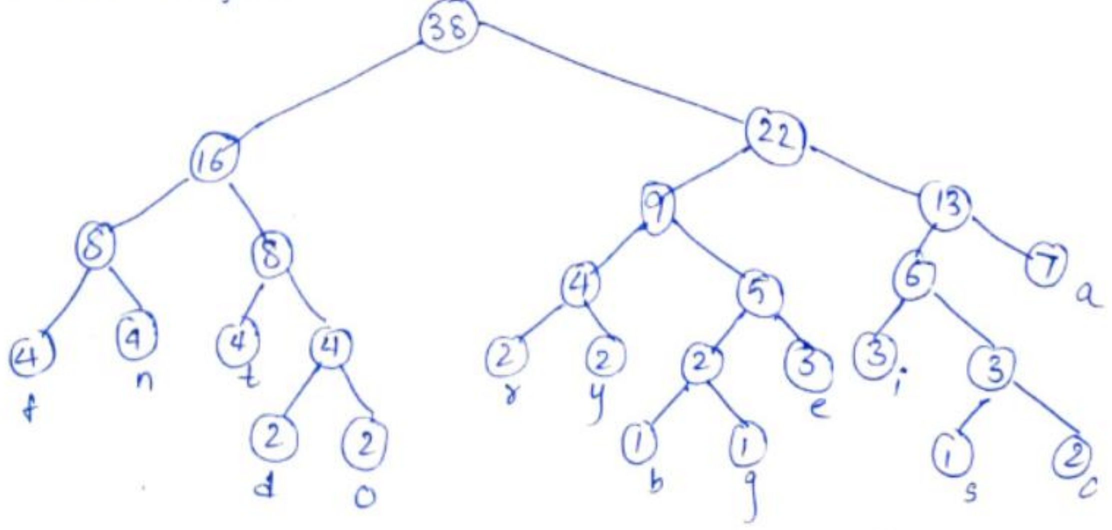
step 13:-

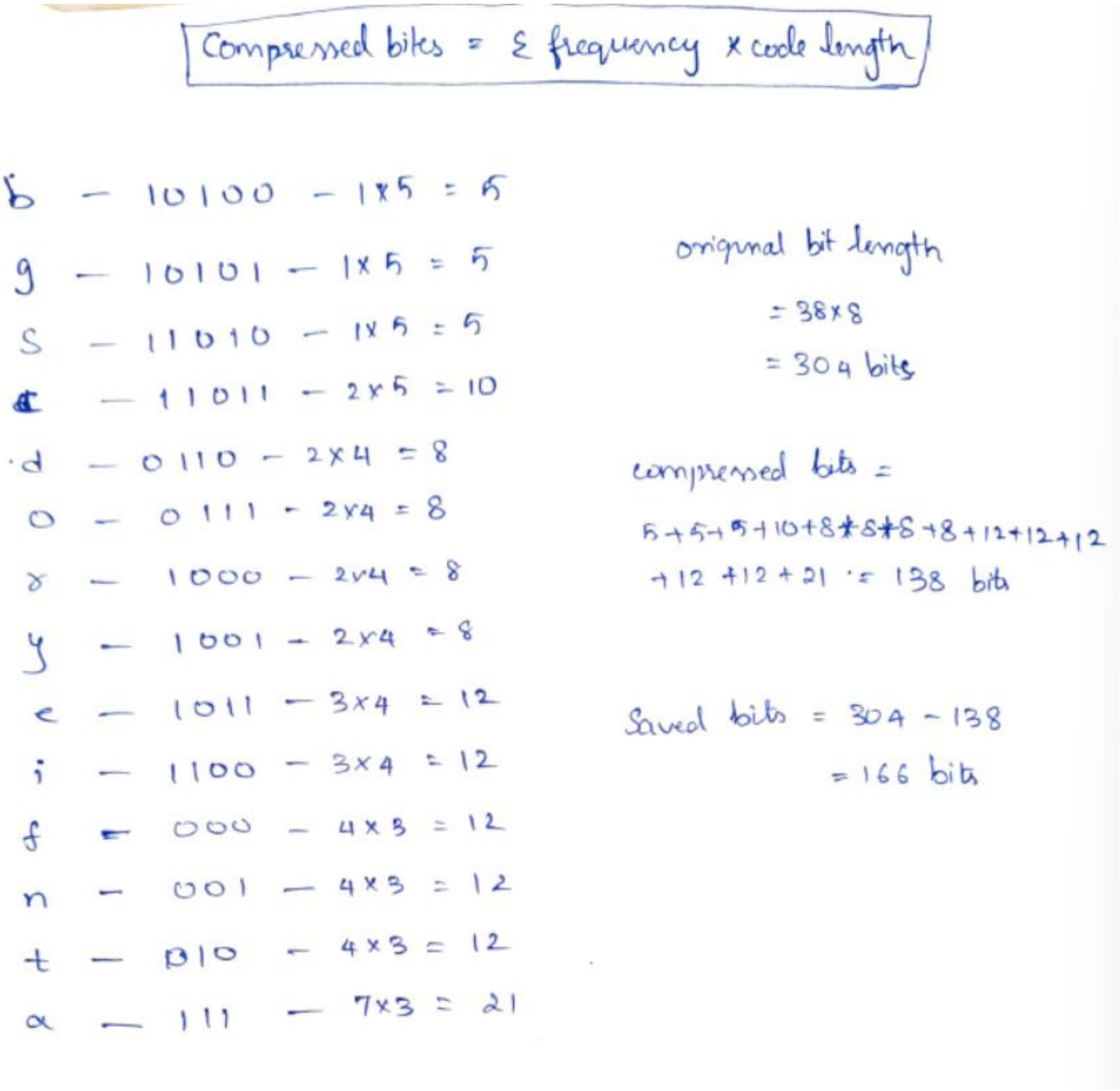

38
16
22
8
8
9
13
7
a
4
4
4
t
4
4
5
6
f
n
2
2
2
3
3
3
r
y
e
i
2
2
1
1
1
2
d
o
b
g
s
c

Right side – 1

left side – 0

Final tree diagram :-





Compressed bites = Σ frequency x code length

b	10100	1x5	= 5
g	10101	1x5	= 5
s	11010	1x5	= 5
c	11011	2x5	= 10
d	0110	2x4	= 8
o	0111	2x4	= 8
r	1000	2x4	= 8
y	1001	2x4	= 8
e	1011	3x4	= 12
i	1100	3x4	= 12
f	000	4x3	= 12
n	001	4x3	= 12
t	010	4x3	= 12
a	111	7x3	= 21

original bit length

= 38x8

= 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

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)