AMRITA SCHOOL OF COMPUTING

DESIGN AND ANALYSIS OF ALGORITHMS

(23CSE211)

Name: Y.V.S.Likesh

Roll No.: CH.SC.U4CSE24152

Class: BTech (CSE-B)

School: Amrita School of Computing,
Chennai Campus.

LAB-7

1) Hoffman Coding for the given sentence.

“Data Analytics and Intelligence Laboratory”

Analysis:

Given sentence contains:

No. of “**D**”: 2

No. of “**A**”: 7

No. of “**T**”: 4

No. of “**N**”: 4

No. of “**L**”: 4

No. of “**Y**”: 2

No. of “**I**”: 3

No. of “**C**”: 2

No. of “**S**”: 1

No. of “**E**”: 3

No. of “**G**”: 1

No. of “**B**”: 1

No. of “**O**”: 2

No. of “**R**”: 2

Code:

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
struct Node{
    char data;
    int freq;
    struct Node *left, *right;
};
struct MinHeap{
    int size;
    struct Node* array[MAX];
};
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 swap(struct Node** a, struct Node** b){
    struct Node* temp = *a;
    *a = *b;
    *b = temp;
}
```

```c
void heapify(struct MinHeap* heap, int i){
    int smallest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;
    if (left < heap->size &&
        heap->array[left]->freq < heap->array[smallest]->freq)
        smallest = left;
    if (right < heap->size &&
        heap->array[right]->freq < heap->array[smallest]->freq)
        smallest = right;
    if (smallest != i) {
        swap(&heap->array[i], &heap->array[smallest]);
        heapify(heap, smallest);
    }
}
```

```c
struct Node* extractMin(struct MinHeap* heap){
    struct Node* temp = heap->array[0];
    heap->array[0] = heap->array[heap->size - 1];
    heap->size--;
    heapify(heap, 0);
    return temp;
}
void insertHeap(struct MinHeap* heap, struct Node* node){
    heap->size++;
    int i = heap->size - 1;
    while (i && node->freq < heap->array[(i - 1)/2]->freq){
        heap->array[i] = heap->array[(i - 1)/2];
        i = (i - 1)/2;
    }
    heap->array[i] = node;
}
```

```c
struct Node* buildHuffman(char data[], int freq[], int n){
    struct MinHeap heap;
    heap.size = 0;
    for (int i = 0; i < n; i++)
        heap.array[heap.size++] = createNode(data[i], freq[i]);
    for (int i = (heap.size - 2)/2; i >= 0; i--)
        heapify(&heap, i);
    while (heap.size > 1) {
        struct Node* left = extractMin(&heap);
        struct Node* right = extractMin(&heap);
        struct Node* top = createNode('$', left->freq + right->freq);
        top->left = left;
        top->right = right;
        insertHeap(&heap, top);
    }
    return extractMin(&heap);
}
```

```c
void printCodes(struct Node* root, int arr[], int top){
    if (root->left){
        arr[top] = 0;
        printCodes(root->left, arr, top + 1);
    }
    if (root->right){
        arr[top] = 1;
        printCodes(root->right, arr, top + 1);
    }
    if (!(root->left) && !(root->right)){
        printf("%c : ", root->data);
        for (int i = 0; i < top; i++)
            printf("%d", arr[i]);
        printf("\n");
    }
}
int main(){
    char data[] = {'D','A','T','N','L','Y','I','C','S','E','G','B','O','R'};
    int freq[]  = { 2 , 7 , 4 , 4 , 4 , 2 , 3 , 2 , 1 , 3 , 1 , 1 , 2 , 2};
    int n = sizeof(data)/sizeof(data[0]);
    struct Node* root = buildHuffman(data, freq, n);
    int arr[MAX], top = 0;
    printf("Huffman Codes:\n");
    printCodes(root, arr, top);
    return 0;
}
```

Output:

```
y-v-s-likesh@y-v-s-likesh-VirtualBox:~/Documents$ ./a.out
Huffman Codes:
C : 0000
D : 0001
T : 001
O : 0100
Y : 0101
N : 011
L : 100
R : 1010
B : 10110
S : 101110
G : 101111
I : 1100
E : 1101
A : 111
```

Space Complexity:

The space complexity of this program is O(n) because it needs space to store the heap and the Huffman tree nodes.

Time Complexity:

The time complexity of this program is O(n log n) because we repeatedly extract and insert nodes into min-heap and each heap operation costs (log n).