Amrita Vishwa Vidhyapeetham

Chennai Campus

Data Analysis And Algorithms

Week-6

Greedy Algorithms

Job Sequencing

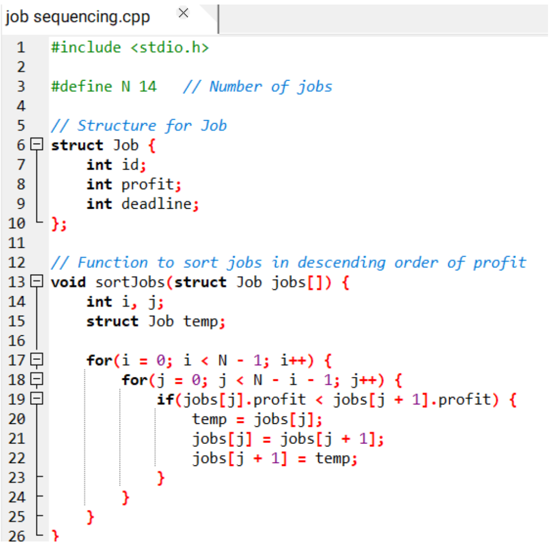
B.S Chenthil Hari

CH.SC.U4CSE24103

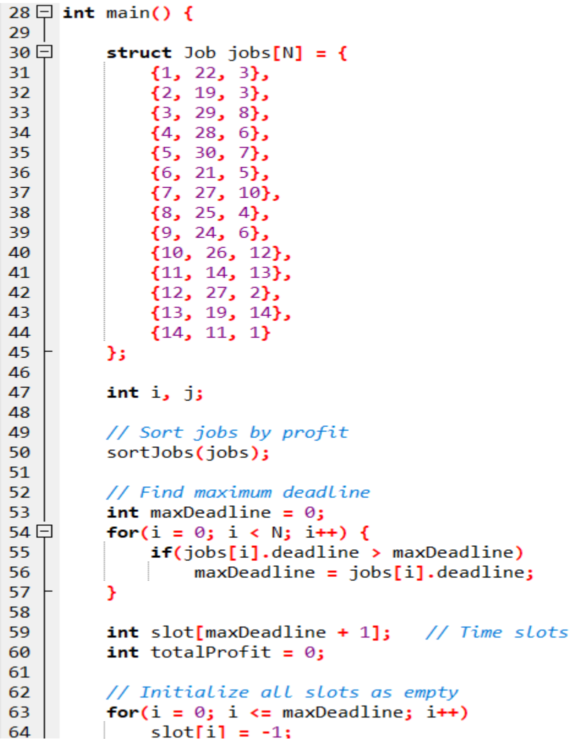
Ch.sc.u4cse24103

Job sequencing:

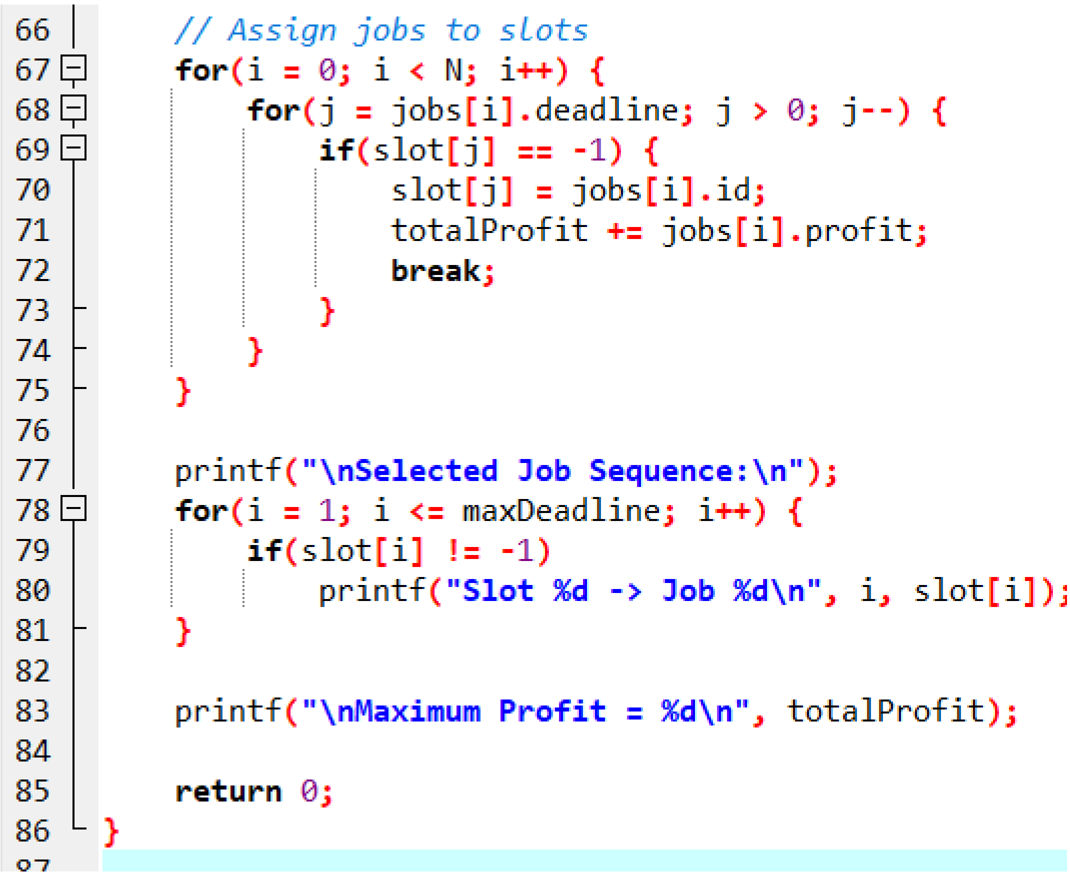
1.) let there be 14 jobs with the profit of (22,19,29,28,30,21,27,25,24,26,14,27,19,11) deadline(3,3,8,6,7,5,10,4,6,12,13,2,14,1) do job sequencing: Solution:



```
#include <stdio.h>

#define N 14    // Number of jobs

// Structure for Job
struct Job {
    int id;
    int profit;
    int deadline;
};

// Function to sort jobs in descending order of profit
void sortJobs(struct Job jobs[]) {
    int i, j;
    struct Job temp;

    for(i = 0; i < N - 1; i++) {
        for(j = 0; j < N - i - 1; j++) {
            if(jobs[j].profit < jobs[j + 1].profit) {
                temp = jobs[j];
                jobs[j] = jobs[j + 1];
                jobs[j + 1] = temp;
            }
        }
    }
}
```

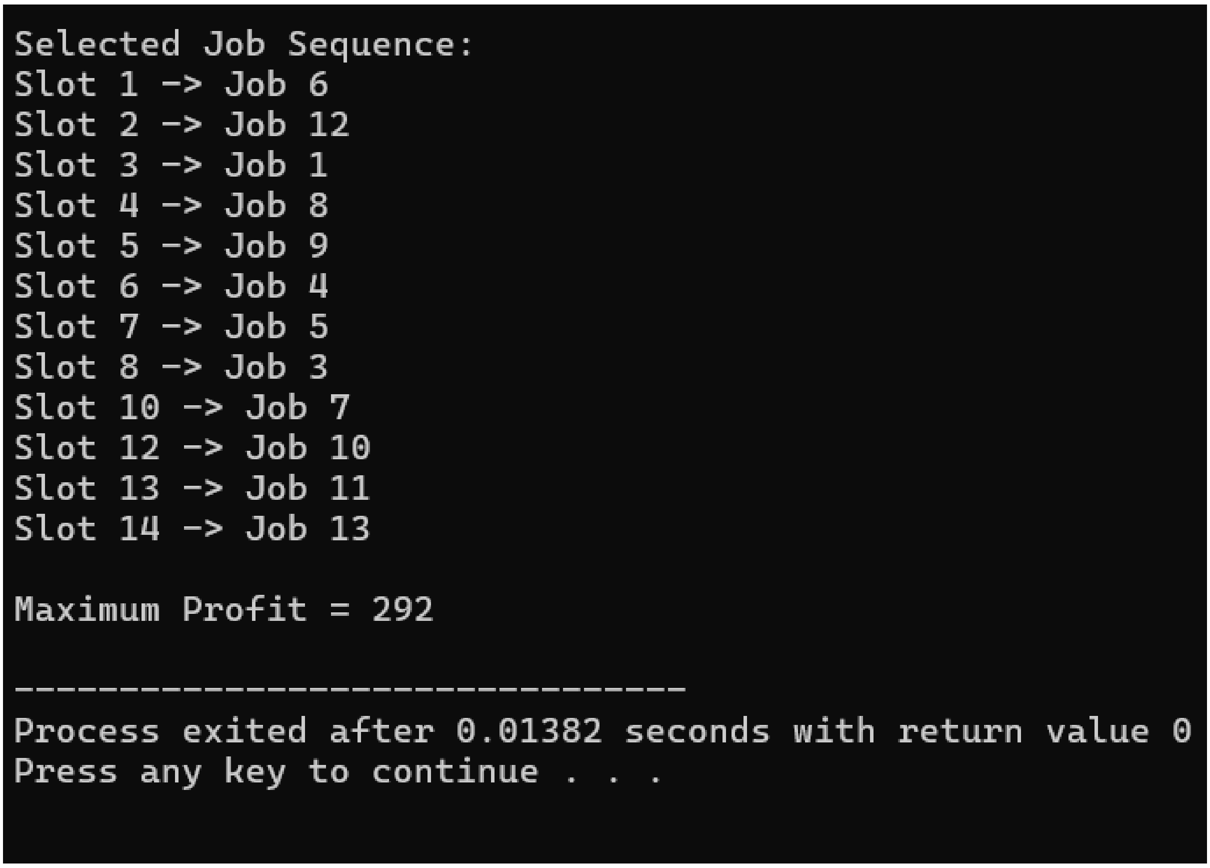


```
int main() {

    struct Job jobs[N] = {
        {1, 22, 3},
        {2, 19, 3},
        {3, 29, 8},
        {4, 28, 6},
        {5, 30, 7},
        {6, 21, 5},
        {7, 27, 10},
        {8, 25, 4},
        {9, 24, 6},
        {10, 26, 12},
        {11, 14, 13},
        {12, 27, 2},
        {13, 19, 14},
        {14, 11, 1}
    };

    int i, j;

    // Sort jobs by profit
    sortJobs(jobs);

    // Find maximum deadline
    int maxDeadline = 0;
    for(i = 0; i < N; i++) {
        if(jobs[i].deadline > maxDeadline)
            maxDeadline = jobs[i].deadline;
    }

    int slot[maxDeadline + 1];    // Time slots
    int totalProfit = 0;

    // Initialize all slots as empty
    for(i = 0; i <= maxDeadline; i++)
        slot[i] = -1;
```



```
    // Assign jobs to slots
    for(i = 0; i < N; i++) {
        for(j = jobs[i].deadline; j > 0; j--) {
            if(slot[j] == -1) {
                slot[j] = jobs[i].id;
                totalProfit += jobs[i].profit;
                break;
            }
        }
    }

    printf("\nSelected Job Sequence:\n");
    for(i = 1; i <= maxDeadline; i++) {
        if(slot[i] != -1)
            printf("Slot %d -> Job %d\n", i, slot[i]);
    }

    printf("\nMaximum Profit = %d\n", totalProfit);

    return 0;
}
```

Output:



```
Selected Job Sequence:
Slot 1 -> Job 6
Slot 2 -> Job 12
Slot 3 -> Job 1
Slot 4 -> Job 8
Slot 5 -> Job 9
Slot 6 -> Job 4
Slot 7 -> Job 5
Slot 8 -> Job 3
Slot 10 -> Job 7
Slot 12 -> Job 10
Slot 13 -> Job 11
Slot 14 -> Job 13

Maximum Profit = 292

--------------------------------
Process exited after 0.01382 seconds with return value 0
Press any key to continue . . .
```

Time Complexity:

Time Complexity of Job Sequencing with Deadlines:

- If jobs are sorted first → sorting takes **O(n log n)**
- Assigning jobs to slots takes **O(n^2)** in worst case

Overall Time Complexity = O(n^2)

(Where *n* is the number of jobs.)

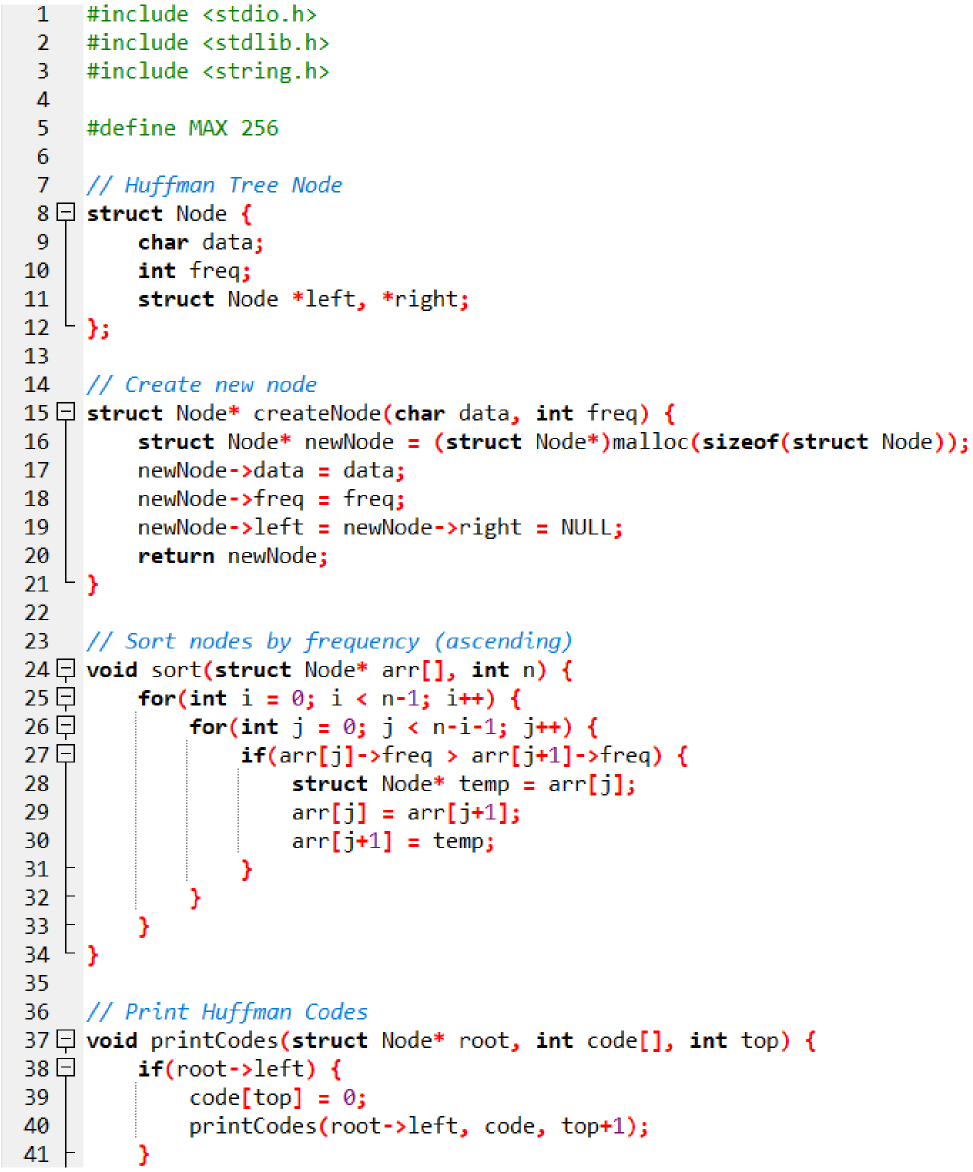
If advanced methods like Disjoint Set (Union-Find) are used, it can be reduced to:

O(n log n).

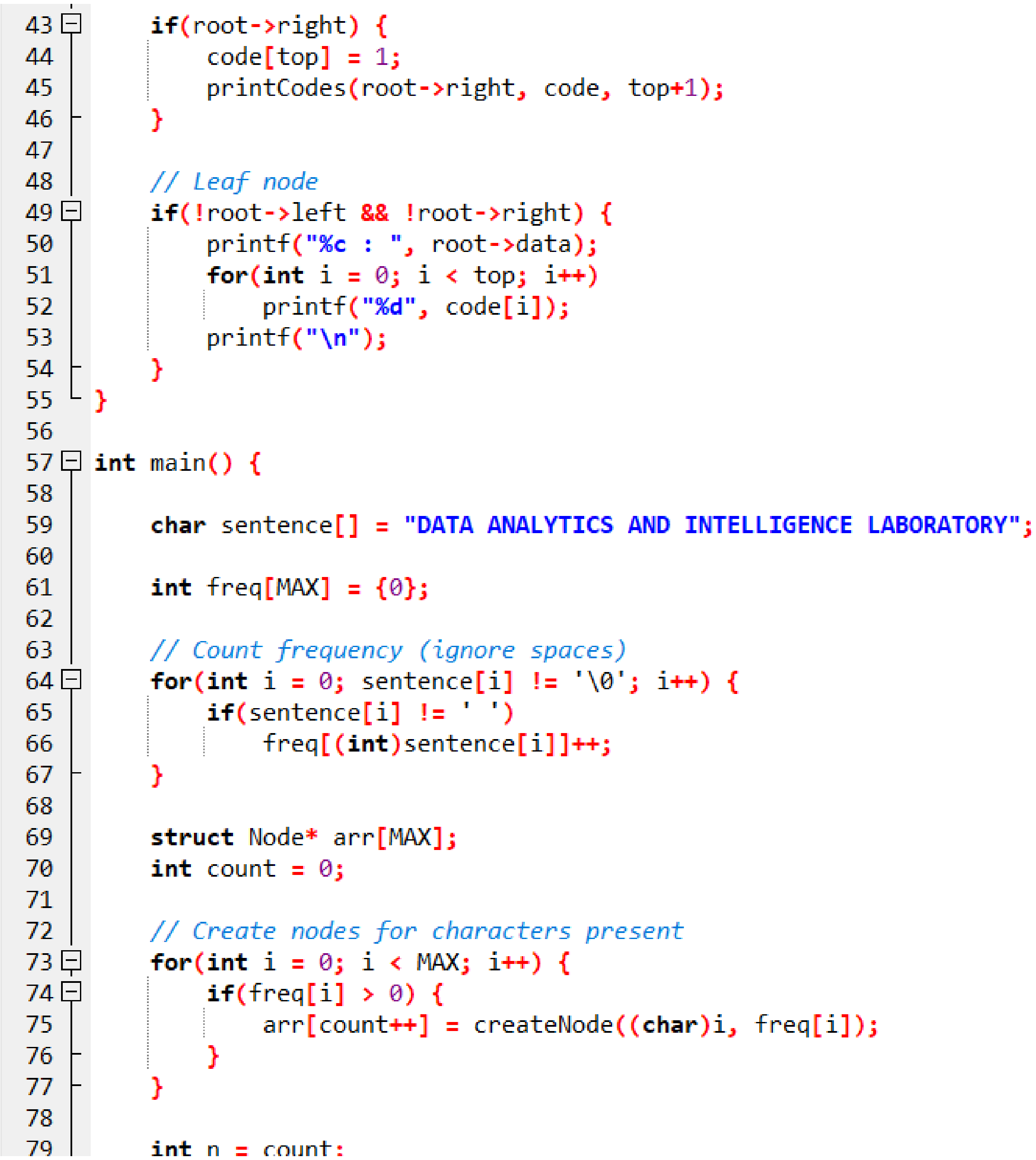
Huffman Coding:

Do DATA ANALYTICS AND INTELLIGENCE LABORATORY in Huffman Coding

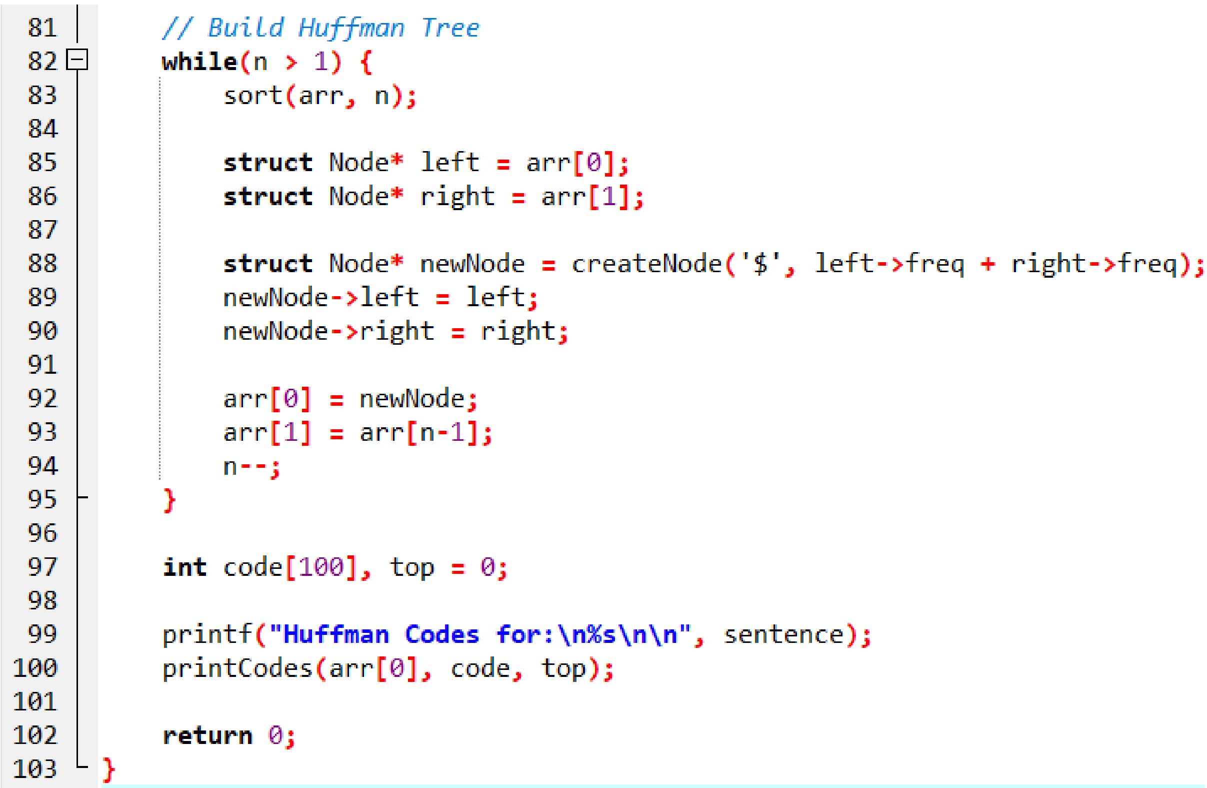
Solution:



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

#define MAX 256

// 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* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->freq = freq;
    newNode->left = newNode->right = NULL;
    return newNode;
}

// Sort nodes by frequency (ascending)
void sort(struct Node* arr[], int n) {
    for(int i = 0; i < n-1; i++) {
        for(int j = 0; j < n-i-1; j++) {
            if(arr[j]->freq > arr[j+1]->freq) {
                struct Node* temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}

// Print Huffman Codes
void printCodes(struct Node* root, int code[], int top) {
    if(root->left) {
        code[top] = 0;
        printCodes(root->left, code, top+1);
    }
```

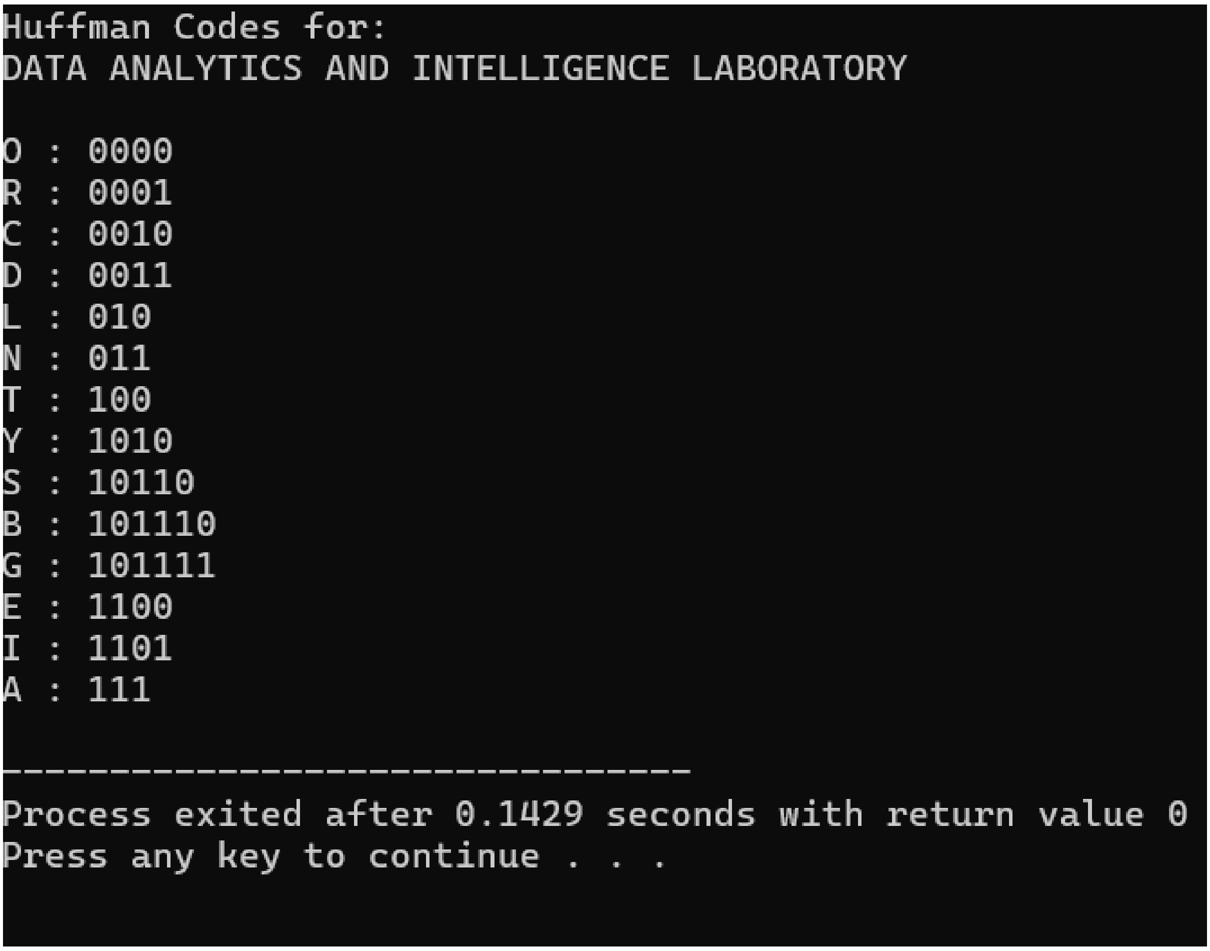


```c
    if(root->right) {
        code[top] = 1;
        printCodes(root->right, code, top+1);
    }

    // Leaf node
    if(!root->left && !root->right) {
        printf("%c : ", root->data);
        for(int i = 0; i < top; i++)
            printf("%d", code[i]);
        printf("\n");
    }
}

int main() {

    char sentence[] = "DATA ANALYTICS AND INTELLIGENCE LABORATORY";

    int freq[MAX] = {0};

    // Count frequency (ignore spaces)
    for(int i = 0; sentence[i] != '\0'; i++) {
        if(sentence[i] != ' ')
            freq[(int)sentence[i]]++;
    }

    struct Node* arr[MAX];
    int count = 0;

    // Create nodes for characters present
    for(int i = 0; i < MAX; i++) {
        if(freq[i] > 0) {
            arr[count++] = createNode((char)i, freq[i]);
        }
    }

    int n = count;
```



```c
    // Build Huffman Tree
    while(n > 1) {
        sort(arr, n);

        struct Node* left = arr[0];
        struct Node* right = arr[1];

        struct Node* newNode = createNode('$', left->freq + right->freq);
        newNode->left = left;
        newNode->right = right;

        arr[0] = newNode;
        arr[1] = arr[n-1];
        n--;
    }

    int code[100], top = 0;

    printf("Huffman Codes for:\n%s\n\n", sentence);
    printCodes(arr[0], code, top);

    return 0;
}
```

Output:



```
Huffman Codes for:
DATA ANALYTICS AND INTELLIGENCE LABORATORY

O : 0000
R : 0001
C : 0010
D : 0011
L : 010
N : 011
T : 100
Y : 1010
S : 10110
B : 101110
G : 101111
E : 1100
I : 1101
A : 111

--------------------------------
Process exited after 0.1429 seconds with return value 0
Press any key to continue . . .
```

Time Complexitty:

Time Complexity of Huffman Coding:

- If implemented using a **Min Heap (Priority Queue)** → **O(n log n)**
- If implemented using **simple sorting (like bubble sort)** → **$O(n^3)$**

(Where *n* is the number of distinct characters.)

Time Complexity = O(n log n).